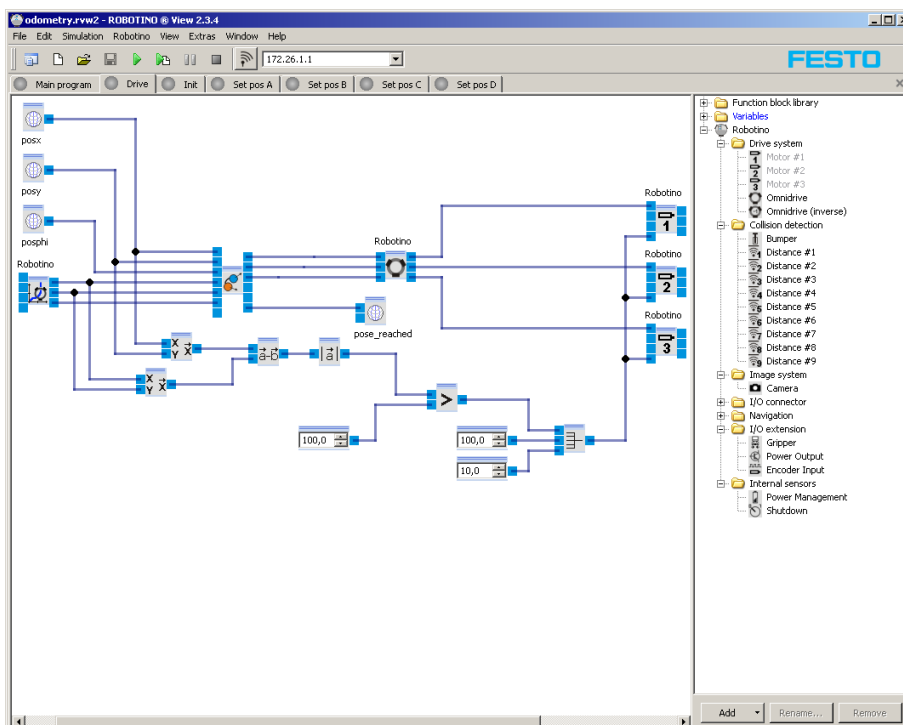


FESTO

Robotino® View2



Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts verboten, soweit nicht ausdrücklich gestattet. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte vorbehalten, insbesondere das Recht, Patent-, Gebrauchsmuster- oder Geschmacksmusteranmeldungen durchzuführen.

The copying, distribution and utilisation of this document as well as the communication of its contents to others without express authorisation is prohibited. Offenders will be held liable for the payment of damages. All rights reserved, in particular the right to carry out patent, utility model or ornamental design registration.

Sin nuestra expresa autorización, queda terminantemente prohibida la reproducción total o parcial de este documento, así como su uso indebido y/o su exhibición o comunicación a terceros. De los infractores se exigirá el correspondiente resarcimiento de daños y perjuicios. Quedan reservados todos los derechos inherentes, en especial los de patentes, de modelos registrados y estéticos.

Toute diffusion ou reproduction du présent document, de même que toute exploitation ou communication de son contenu sans l'accord express de l'auteur est proscrite. Toute contravention pourra donner lieu à des demandes de dommages et intérêts. Tous droits réservés, notamment en termes de demande de brevet, de modèle déposé et de protection par dessin ou modèle.

© Festo Didactic GmbH & Co. KG, 73770 Denkendorf, Germany, April 2010
Internet: www.festo-didactic.com
e-mail: did@de.festo.com

Inhalt / Contents / Contenido / Sommaire

1	Welcome	8
1.1	Improvements	8
1.2	Installation, update and de-installation	8
1.3	Changing language	9
2	Familiarisation with the workspace	9
2.1	Structure and concept of user interface	9
2.1.1	Tool bar	11
2.1.2	Function block library	12
2.2	Terminology	13
3	Using Robotino® View	13
3.1	Create a new project	13
3.2	Load an existing project	14
3.3	Insert function blocks into sub-programs	14
3.4	Interlink function blocks	14
3.5	Global variables	15
3.6	Execute a sub-program	16
3.7	Execute the main program	17
3.8	Connect to Robotino®	18
3.9	Keyboard shortcuts	19
3.10	Type conversion	20
3.11	Updates	20
3.12	Upload projects to Robotino and execute them	20
3.12.1	Browse Robotino	21
3.12.2	Upload and execute	22
3.13	Upgrade Robotino packages	23
3.13.1	Robotino firmware installation	25
3.13.2	Interna	26
4	Examples	26
4.1	Control programs	26
4.1.1	Tutorial 2	27
4.2	Logic	34
4.2.1	Multiplexer	34
4.2.2	FlipFlop	34
5	Function block library	34
5.1	Logic	35
5.1.1	Counter up	35
5.1.2	Counter down	38
5.1.3	Multiplexer	39
5.1.4	Demultiplexer	40
5.1.5	AND	41
5.1.6	AND_FL	43
5.1.7	NAND	45
5.1.8	NAND_FL	47
5.1.9	OR	48
5.1.10	XOR	49
5.1.11	NOT	50
5.1.12	NOR	51

5.1.13	Latching relay	52
5.1.14	Sample and hold element	53
5.2	Mathematics	54
5.2.1	Arithmetic operations	54
5.2.2	Comparison Operations	58
5.2.3	Functions	60
5.2.4	Arrays	67
5.3	Vector analysis	69
5.3.1	Vector operations	69
5.3.2	Element operations	73
5.3.3	Transformations	74
5.4	Display	77
5.4.1	Oscilloscope	77
5.4.2	Laser range finder data display	78
5.5	Image processing	79
5.5.1	Segmenter	79
5.5.2	Segment Extractor	83
5.5.3	Line Detector	85
5.5.4	ROI	87
5.5.5	Image Information	89
5.5.6	Colorspace conversion	91
5.6	Generators	91
5.6.1	Arbitrary waveform generator	92
5.6.2	Constant	93
5.6.3	Timer	94
5.6.4	Random generator	95
5.7	Filter	95
5.7.1	Mean filter	96
5.8	Navigation	96
5.8.1	Position Driver	96
5.8.2	Constant pose	101
5.8.3	Pose composer	102
5.8.4	Pose decomposer	103
5.8.5	Path composer	104
5.8.6	Path decomposer	104
5.8.7	Path driver	106
5.8.8	Obstacle avoidance	114
5.9	Input Devices	116
5.9.1	Control Panel	116
5.9.2	Slider	117
5.10	Data exchange	118
5.10.1	Image Reader	118
5.10.2	Image Writer	120
5.11	Variables	121
6	Devices	121
6.1	Add and edit	121
6.2	Show dialogs	122
6.3	Robotino	123
6.3.1	Toolbar	123
6.3.2	Dialog	124
6.3.3	Function blocks	124
6.4	Joystick	150
6.4.1	Dialog	150
6.4.2	Function blocks	150

6.5	Local camera	151
6.5.1	Dialog	152
6.5.2	Function blocks	153
6.6	OPC Client	154
6.6.1	Dialog	155
6.6.2	Function blocks	156
6.7	Data exchange	158
6.7.1	Server	158
6.7.2	Client	161
6.7.3	Function blocks	165
6.8	UDP data exchange	165
6.8.1	Protocol	165
6.8.2	Dialog	169
6.8.3	Function blocks	170
6.8.4	Example	171
7	Programming	171
7.1	My function blocks	171
7.1.1	Tutorial 1	173
Index	175

1 Welcome

Robotino® View is the intuitive graphic programming environment for Robotino®. Robotino® View enables you to create and execute control programs for Robotino®.

1.1 Improvements

Robotino® View 2 combines modern operational concepts, extensibility by the user and intuitive usage. All innovations preserve the many positive aspects known from Robotino® View 1. The user being familiar with Robotino® View 1 will recognize many features from the previous version. These are for example the function block library or the toolbar by which the connection to Robotino is established. At first sight Robotino® View 2 looks very similar to its predecessor.

Kept feature?

- Programs are designed as data flow control programs. The function block library contains the units the data flow graph is build from.
- The connection to Robotino is established via the toolbar.

Whats new?

- The sequence control program is replaced by a "real" control program known from PLC programming following DIN EN 61131.
- Robotino® View 2 is not only able to control Robotino, but any device and in unlimited quantities. I.e. an arbitrary number of Robotinos can be controlled simultaneously from within one Robotino View project.
- Subprograms can be reused within the main program.
- Subprograms can be imported from different projects.
- The user can design and implement custom function blocks which are loaded into the function block library at runtime.
- The user can design and implement custom devices and load them as Plugin into the device management.

Changes within this version:

- The device manager had been integrated into the function block library. See [Add and edit devices](#)^[127].
- Projects can be uploaded to Robotino and also be started on Robotino (requires Robotino CF card Version 2.0). See [upload projects](#)^[20].
- New devices for data exchange over network. See [Devices for data exchange](#)^[158].

1.2 Installation, update and de-installation

You must be in possession of the administrator rights to be able to install Robotino® View.

To install Robotino® View follow the instructions in the dialog boxes.

If users without administrator rights are to use Robotino® View,

they will need to include the programs released from the restrictions under Windows® XP

in the security centre for the setting for the firewall of Robotino® View (Port 80 and 8080).

1.3 Changing language

Robotino® View automatically recognizes the language set in your Windows® System and selects the corresponding translation of Robotino® View.

You can change the automatic setting at any time via menu item Extras ▶ languages. The new settings immediately take effect.

2 Familiarisation with the workspace

Once you have familiarised yourself with the workspace and the designations used in Robotino® View, it will be easier for you to follow the remainder of the documentation.

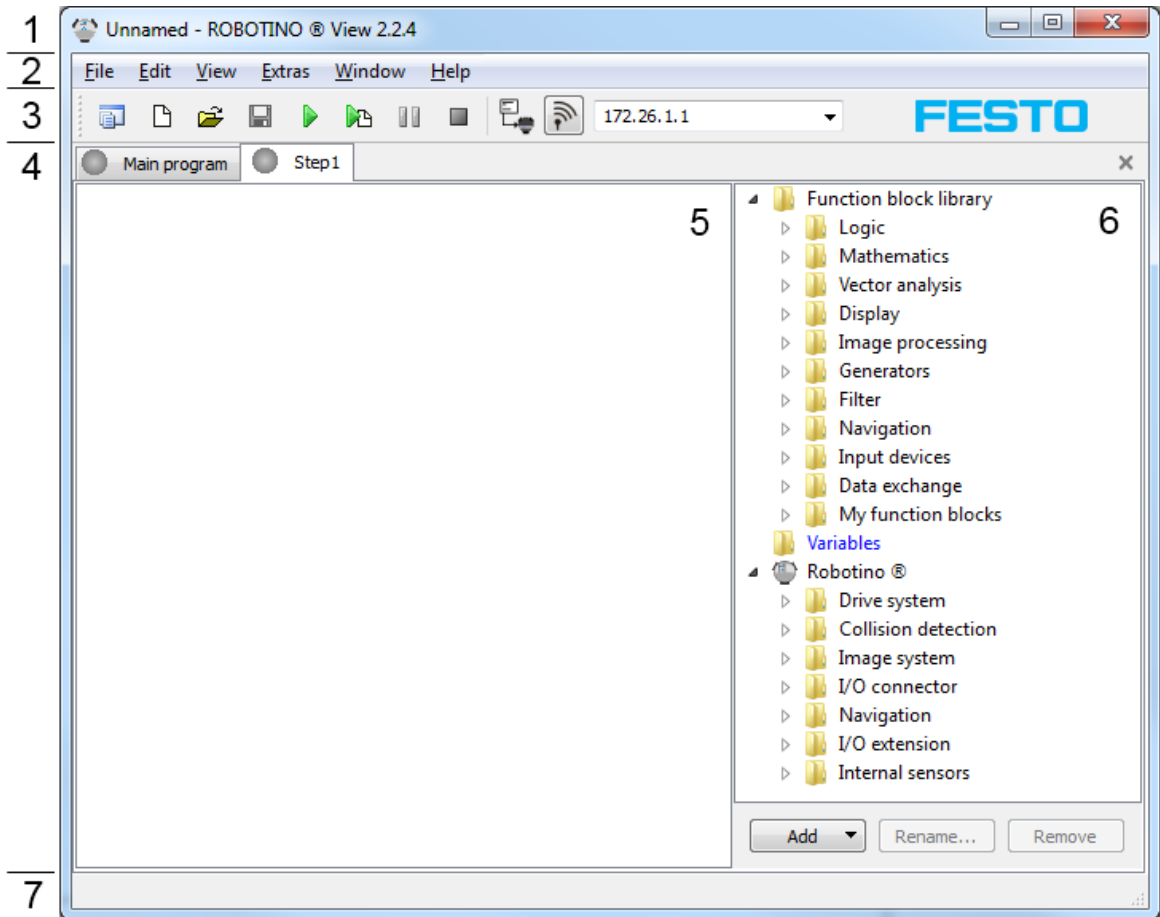
In this section you will learn more about:

- The design and concept of the user interface,
- The terminology used in Robotino® View.

2.1 Structure and concept of user interface

When starting up Robotino® View an empty project with a single "Robotino" device is opened. The complete workspace is taken up by the project.

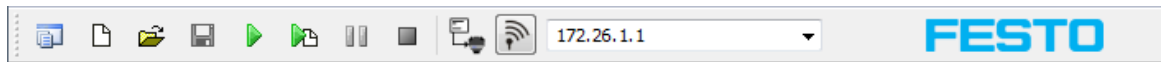
Familiarisation with the workspace



Number	Name	Description
1	Title bar	<ul style="list-style-type: none"> Shows the name of the current project (Unnamed). If there are unsaved changes within the project, the project name is followed by a *. Next to the project name the application name and application version is shown (here Robotino View version 2.2.4). Default buttons to minimize, maximize and closing.
2	Menu bar	Menus to load/save, edit, view ...
3	Tool bar	<ul style="list-style-type: none"> Quickly accessible buttons to the function from the menus. Buttons to start and stop the simulation. Input box for Robotini's IP address and connect button (see Robotino tool bar (123)). Festo Logo with Link to the Festo homepage.
4	Program selector	Here you can switch between the main program and the subprograms of a project. The subprogram "Step1" is visible at present.
5	Program workspace	Here the program is viewed and edited. Obviously, the subprogram "Step1" is empty.

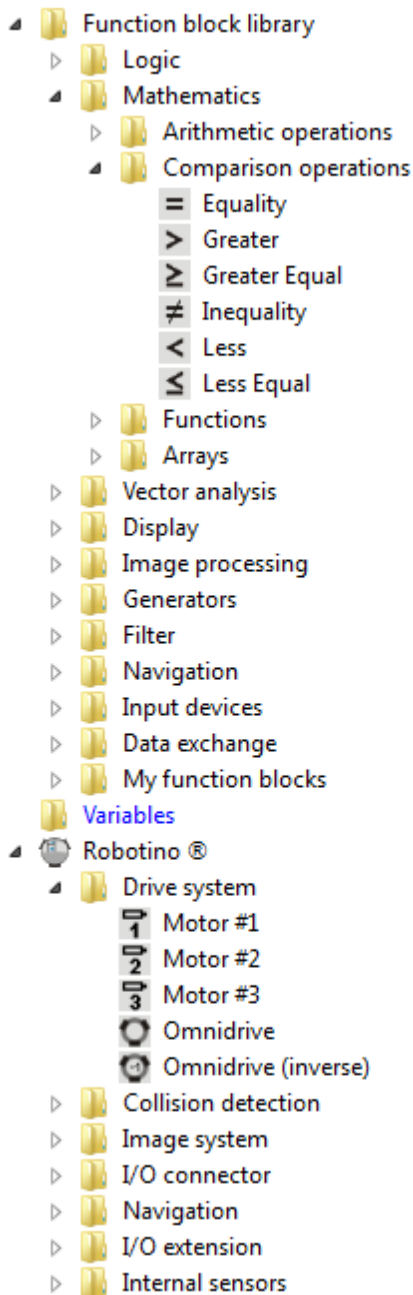
6	Function block library 121	The function blocks available for programming are displayed here.
7	Status bar	Shows information about project and application status.

2.1.1 Tool bar



	Create a new project
	Create a new sub-program
	Open an existing project
	Save current project
	Start main program
	Start the currently visible program
	Pause simulation
	Stop simulation
	Upload the project to Robotino
IP address input and connect button	see Robotino tool bar 1231
	Festo-Logo with link to Festo Homepage

2.1.2 Function block library



The folder function block library contains function blocks that are available in every project. Currently visible are the function blocks Equality, Greater ... Less Equal from the sub folder Comparison operations.

The folder Robotino® contains function blocks that are provided by the "[Robotino®](#)" device. A new project always contains one "[Robotino®](#)". Currently visible are the function blocks "[Motor1](#)" to "[Omnidrive \(inverse\)](#)" from the folder "[Drive system](#)".

The folder Variables contains function blocks to read and write global Variables.

You can add function blocks via Drag&Drop to the current sub-program.

Function blocks of devices are bound to concrete hardware resources. "[Motor1](#)^[125]" is available once on Robotino. Therefore you can add "[Motor1](#)^[125]" only once to a sub-program. If "[Motor1](#)^[125]" had been added to a sub-program already, the icon of "[Motor1](#)^[125]" in the function block library is gray.

2.2 Terminology

Function block	Smallest function unit a subprogram consists of. By networking several function blocks it is possible to realise complex robot behaviour.
Subprogram	Function blocks are interlinked by networks in a sub-program
Main program	A control program written in sequential function chart connecting the subprograms.
Project	A project consists of a main program and several subprograms. Projects can be loaded and saved.
Network	Function blocks are linked by one or several networks.
Network point	Network points are within a network and enable the structuring and graphic representation of a network. A new sub-network can be started from a network point.


3 Using Robotino® View

Robotino® View is used to create the control programs for Robotino®. In this section you will learn how to:

- create a new project
- load an existing project
- insert function blocks into a sub-program
- interlink function blocks by networks
- execute a sub-program and the main program
- establish a connection to Robotino®


3.1 Create a new project

There are two possibilities to create a new project:

- Via the menu bar File ▶ New
- Via the [tool bar](#)^[11] with the button „Create a new project“ 

3.2 Load an existing project

There are three possibilities to load an existing project:

- Via the menu bar File ▶ Open
- Via the [tool bar](#)^[11] with the button "Load a project from file" 
- Via the keyboard shortcut Ctrl + O

Saved projects do have the file extension .rvw2

3.3 Insert function blocks into sub-programs

After creating a new project or after loading an existing project from file you can start developing your own control program or modifying the existing one.

Example:

Make sure a sub-program is shown in the current view. With newly created project there is always the sub-program "Step1". The sub-program "Step1" is shown after creating a new project. The [function block library](#)^[12] is only visible when looking at a sub-program.

Expand the folder "[Logic](#)^[35]" in the function block library. Drag "[Counter Up](#)^[35]" from the function block library and drop it in the sub-program.

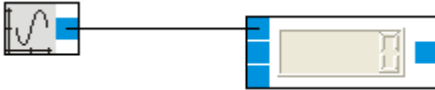
Expand the folder "[Generators](#)^[91]" in the function block library. Drag "[Arbitrary Waveform Generator](#)^[92]" and drop it left to the "[Counter Up](#)^[35]".



3.4 Interlink function blocks

By connecting output connectors to input connectors of function blocks, data is passed from the function blocks output to the others function block input. The connection is visualized by a line called network. A network is always connected to exactly one output connector and at least one input connector.

The current sub-program example contains an "[Arbitrary Waveform Generator](#)^[92]" and a "[Counter Up](#)^[35]" function block. Connect the "[Arbitrary Waveform Generator](#)^[92]" output to the upper input of "[Counter Up](#)^[35]".



Do a left mouse click on the "[Arbitrary Waveform Generator](#)^[92]" output connector. By this you are creating a net line which is attached to the "[Arbitrary Waveform Generator](#)^[92]" output and with the other end to the mouse pointer.

By clicking with the left mouse button somewhere in the sub-program you can create net point. To create the network click on the upper input of "[Counter Up](#)^[35]".

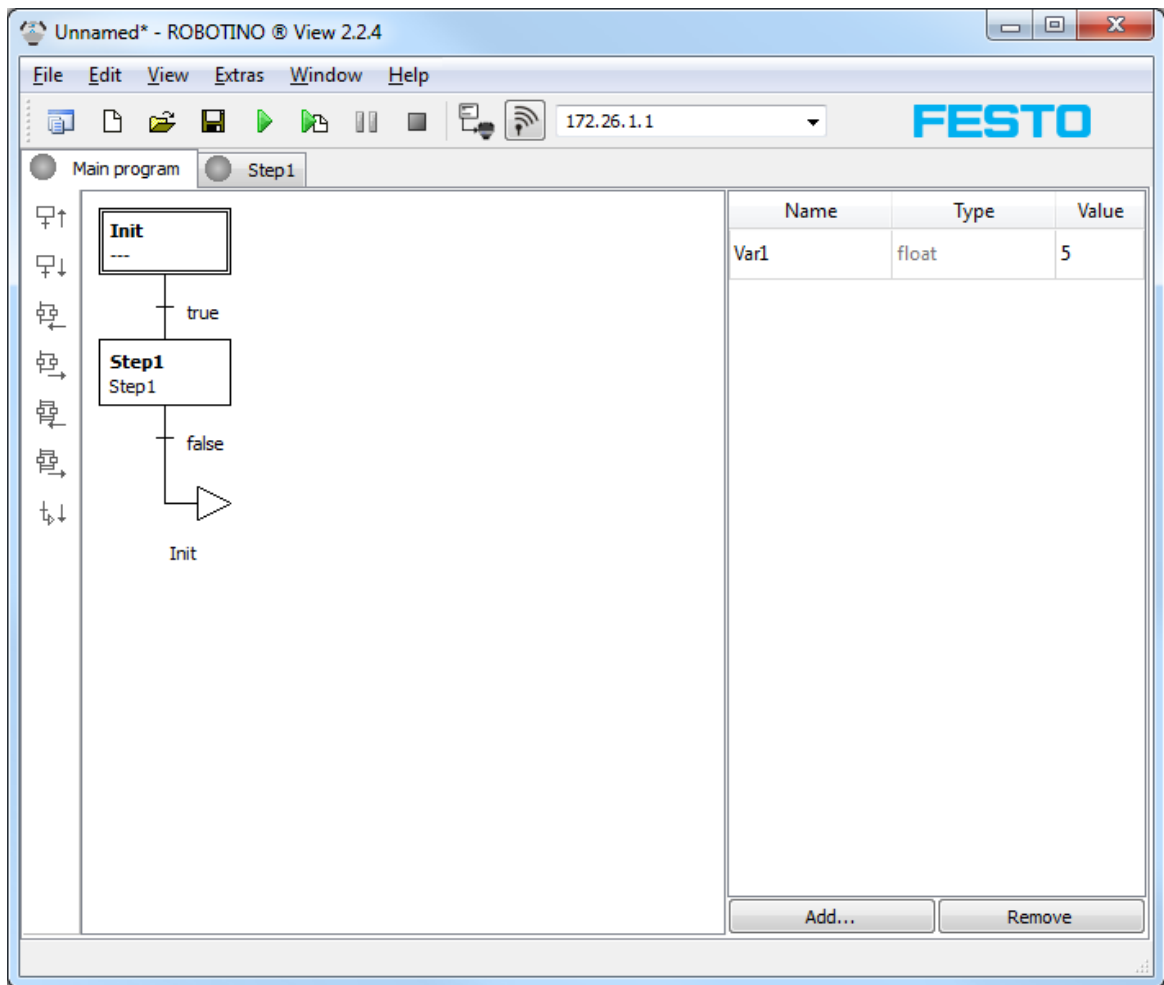
To delete a net point mark it by doing a left click on it and press **Del**.

To erase a net line mark it by doing a left click on it and press **Del**. This might erase the whole network.

3.5 Global variables

Global variables can be read and written in every subprogram of a project; in the main program they can be used in the transition conditions.


In the main program view the variable management is located on the right side. It enables you to add, remove and rename variables and to assign initial values to them.



Main program with variable management

Global variables store floating point numbers only. Support for other data types will be included in future versions of Robotino View. After creating a variable function blocks for reading and writing the variable are available in the function block library.

3.6 Execute a sub-program

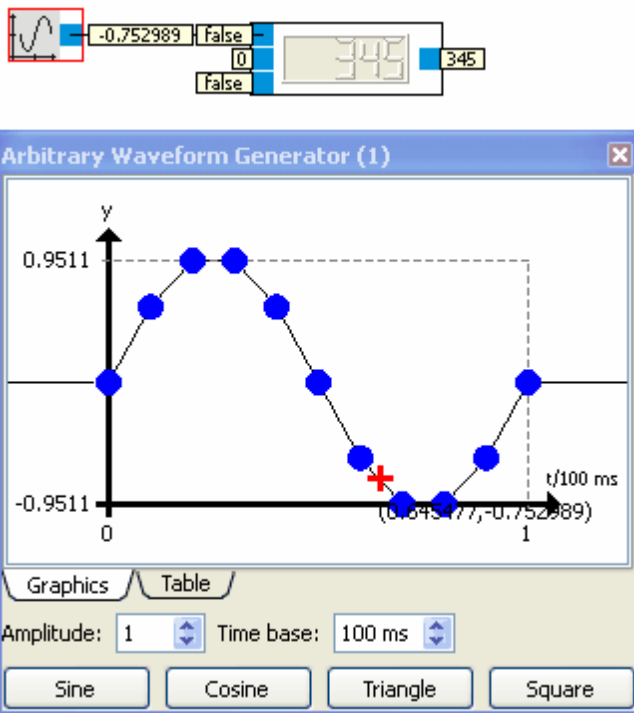
After connecting the "[Arbitrary Waveform Generator](#)^[92]" to the "[Counter Up](#)^[35]" you can start simulation of the sub-program by clicking „Start"  shown in the [tool bar](#)^[11].

You can display the values generated by the "[Arbitrary Waveform Generator](#)^[92]" and "[Counter Up](#)^[35]" by selecting View ▶ Show Connector Values or by pressing **Ctrl + D**.




You can see the "[Arbitrary Waveform Generator](#)^[92]" generating values between 0 and 10. "[Counter Up](#)^[35]" increments its output when the input changes from false (0) to true (≠0). At the moment this only happens when starting the sub-program. See [type conversion](#)^[20] to read how floating point numbers are converted to boolean. Furthermore it is very unlikely that the "[Arbitrary Waveform Generator](#)^[92]" output matches exactly to 0.

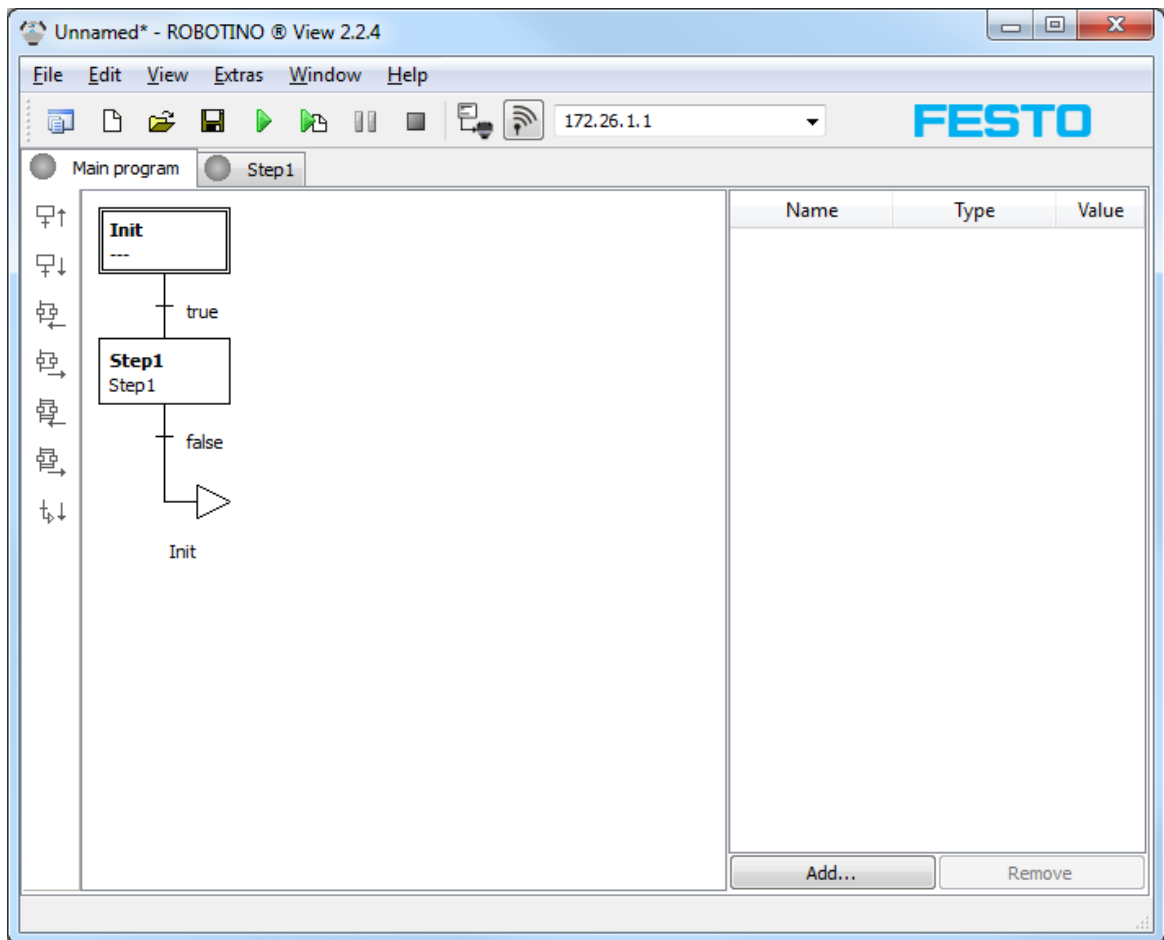
To see the counter counting, select square from the "[Arbitrary Waveform Generator](#)^[92]" dialog. The generated output is now in the range -1 to 1.





3.7 Execute the main program

By clicking on „Start“  in the [tool bar](#)^[11], simulation of the currently visible program is started. If the sub-program "Step1" is visible only "Step1" is simulated. "Step1" is part of the main program, which can be simulated as well.

Use the [Program selector](#)^[9] to make the main program the current program.











By clicking on „Start"  in the [tool bar](#)^[11] simulation of the main program is started. The Init step is run only once, because the transition condition following the Init step is true. As the transition condition following Step1 is constantly false, Step1 and the sub-program assigned to it (also called Step1) is executed.

You can always start simulation of the main program no matter which via is currently visible, by clicking the "Start main program" button  in the [tool bar](#)^[11].

3.8 Connect to Robotino®

Enter Robotino's IP address in the IP address input field in the [tool bar](#)^[11]. The default address is 172.26.1.1. Click onto the connection button left to the address input field. If the connect button changes from gray to green, the connection is established and data between Robotino and Robotino View is exchanged.

3.9 Keyboard shortcuts

Function	Keyboard shortcut
Open file	Ctrl + O
Save file	Ctrl + S
Save file as	Shift + Ctrl + S
Quit Robotino® View	Ctrl + Q
Undo	Ctrl + Z
Redo	Ctrl + Y or Shift + Ctrl + Z
Delete selection	Del
Cut selection	Ctrl + X
Copy selection	Ctrl + C
Paste selection	Ctrl + V
Move object up	
Move object down	
Move object left	
Move object right	
Move view up	Ctrl + 
Move view down	Ctrl + 
Move view left	Ctrl + 
Move view right	Ctrl + 
Clear selection	Esc
Select all	Ctrl + A
Demagnify view	F3
Magnify view	Shift + F3
Magnify grid	F4
Demagnify grid	Shift + F4
Toggle function block library visibility	Ctrl + L
Toggle function block connector values' visibility	Ctrl + D
Toggle function block connector descriptions' visibility	Ctrl + T

3.10 Type conversion

Data type	implicit conversion to	Description
int	float, bool	Conversion to bool will result in true if the value is not 0.
float	int, bool	Conversion to bool will result in true if the value is not 0.
bool	int, float	True results in 1, false results in 0.
pose	path	A pose is converted to a path with length 1.
path	pose	The result of the conversion of a path to a pose is the path's first pose. If the path is empty, the conversion results in an invalid pose.
float	float array	A floating point number is converted to a float array with length 1.

3.11 Updates


Robotino View has an online update feature. To check the availability of a new software version, select "Check For Updates" in the "Extras" menu. This check is also done automatically after the application has been launched. If a new Version is available, it can be downloaded and installed automatically.


The behaviour of the update feature can be configured in the preferences dialog ("Extras" ▶ "Preferences..."). If the Internet can only be accessed via a proxy, address, port, user name and password can be entered here. But in enterprise networks, using the Internet Explorer settings will mostly be the easiest way.

3.12 Upload projects to Robotino and execute them

Since Robotino View version 2.1.0 and Robotino flash card 2.0 it has been possible to upload projects to Robotino via FTP and directly execute them from Robotino View. This function is accessible via Robotino ▶ Upload project.

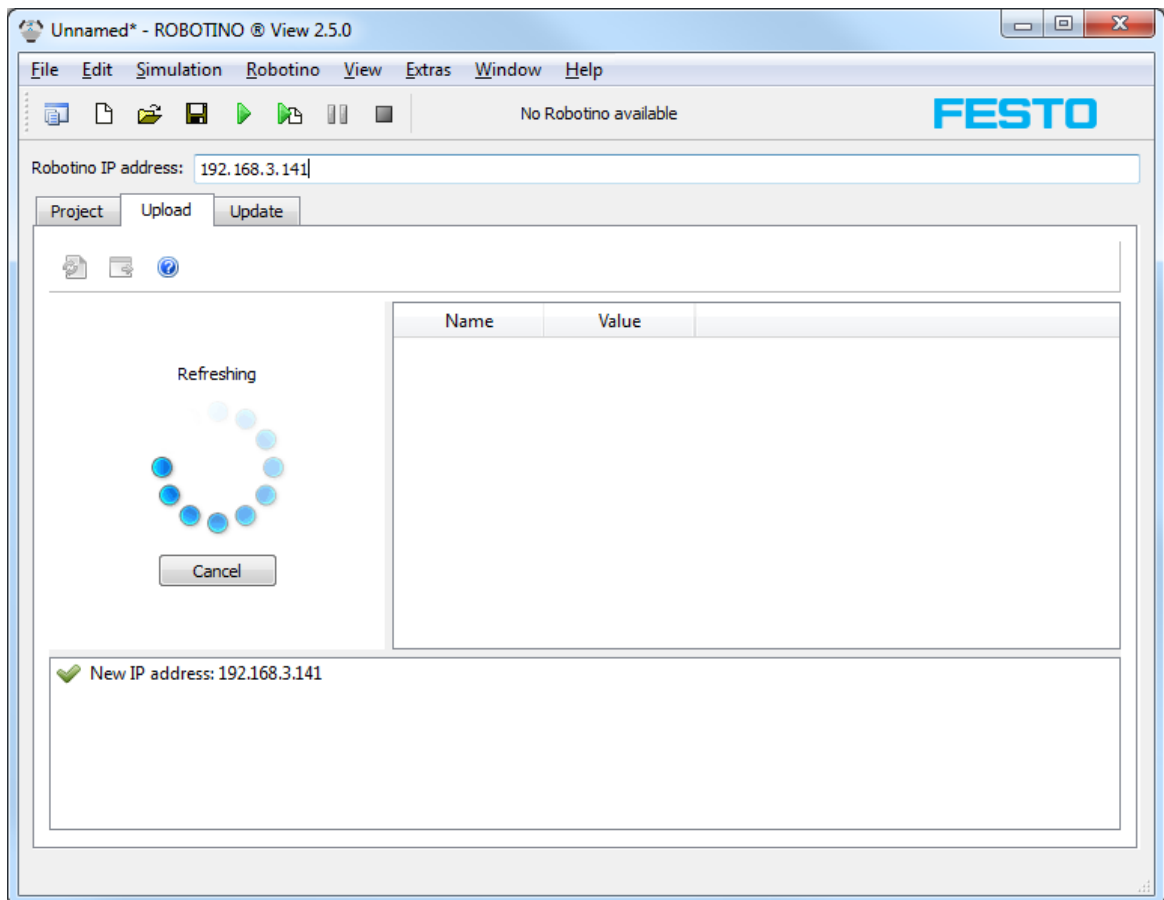
The first time the upload dialog is called the first Robotino device's current IP address will be entered into the "Robotino IP address" input field. If there is no Robotino device in the current project, the input field remains empty.

When the dialog is opened, the directory view in Robotino will be updated. The execution of an action is displayed by an animation. The view update can also be invoked via the button . Further details about browsing the directory structure on Robotino can be found in the section [Browse Robotino](#) ^[21].

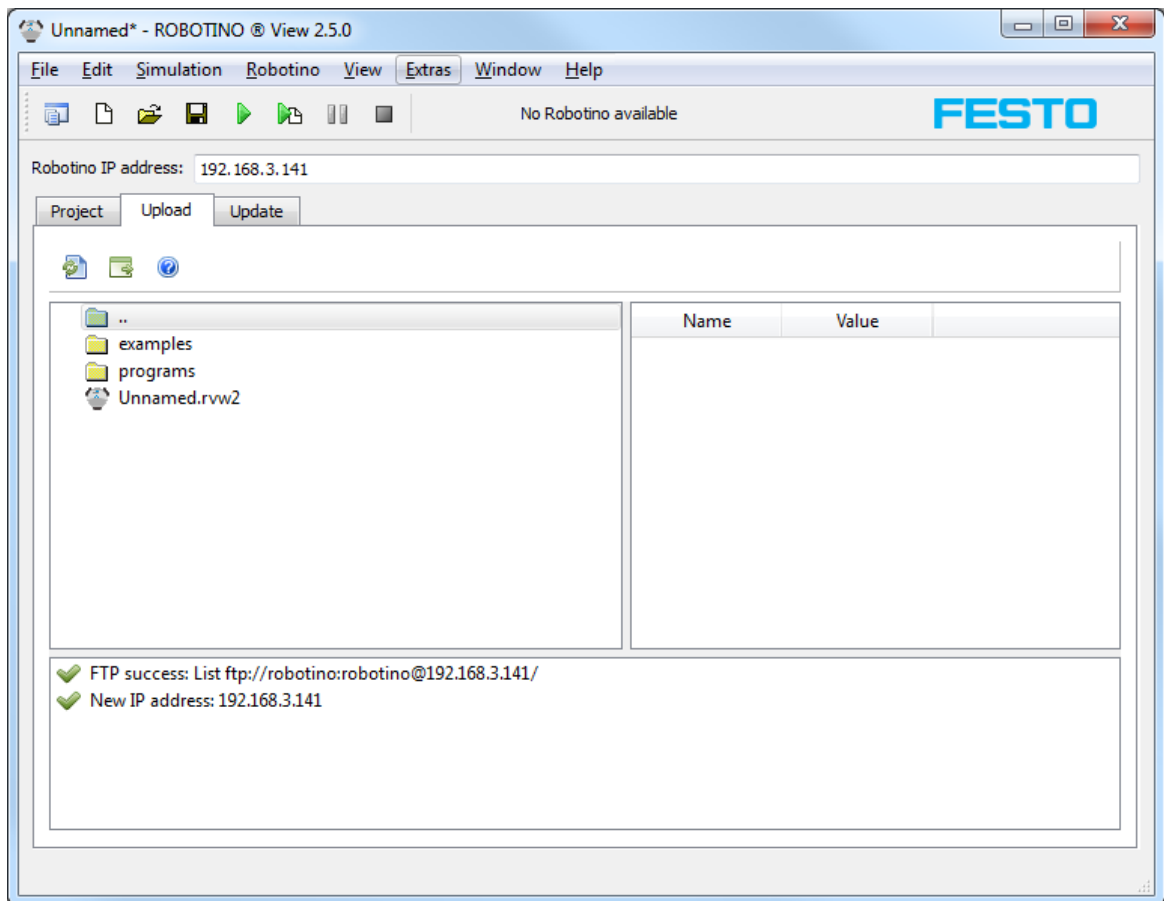
The button  is used to upload the current project into the currently viewed directory. Further details about uploading and executing projects can be found in the section [Upload and execute](#) ^[22].

3.12.1 Browse Robotino

Since version 2.0 of Robotino's flash card a FTP and a Telnet server are installed on the Ubuntu Linux system. FTP is used to display the files on Robotino and to upload projects.




After the first login the directory `/home/robotino` is displayed. In this case there are the subdirectories "examples" and "programs" and the Robotino View project "Unnamed" in the current directory. By clicking on one of the directories, the view is updated and the contents of the selected directory is shown. By clicking on a Robotino View project, the execution of this project on Robotino is started. See [Upload and execute](#) [22].

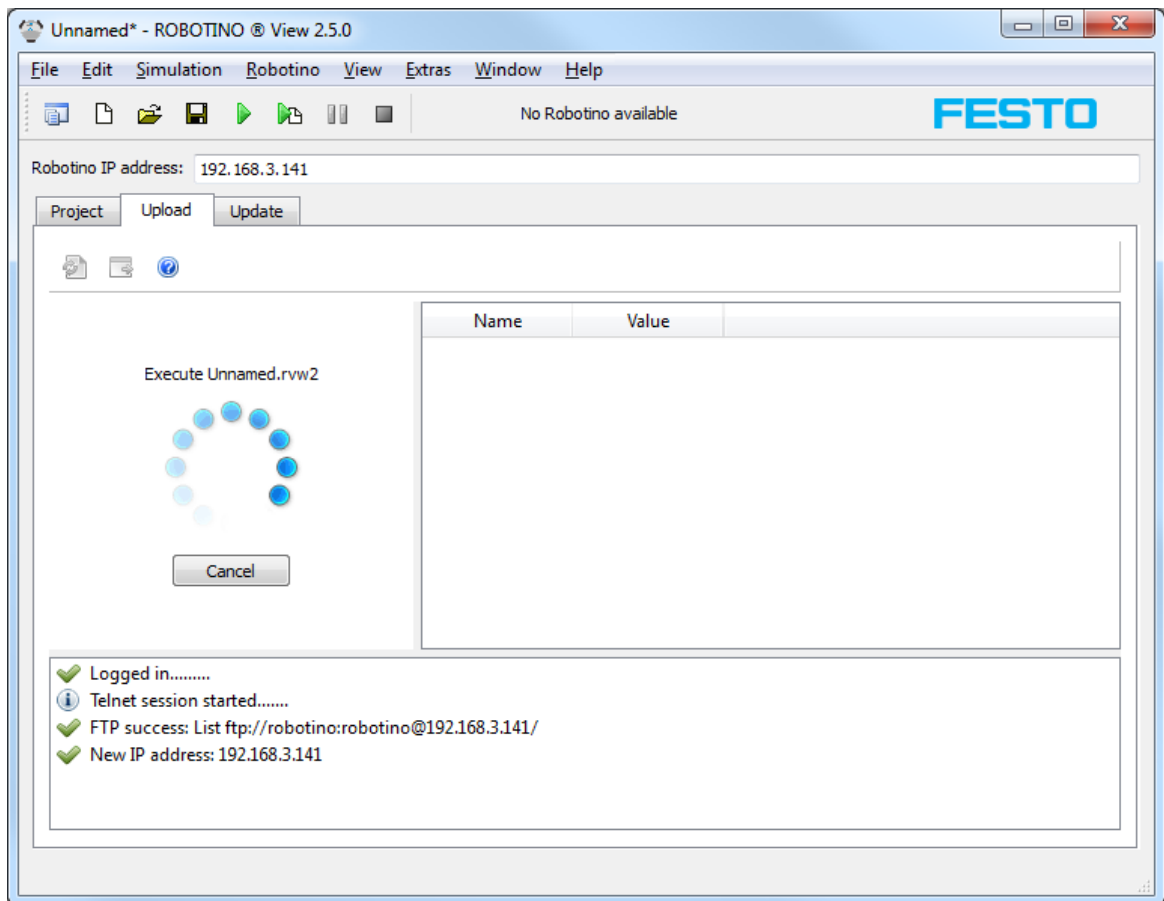


The FTP client integrated in Robotino View uses the user login "robotino" with password "robotino". Thus it is possible to log in e.g. with FileZilla and create subdirectories or remove uploaded projects.

3.12.2 Upload and execute

Before executing a project it is recommended to check if the Robotino View version installed on Robotino is the newest one. Package upgrade on Robotino is described in the section [Upgrade Robotino Packages](#) [23].

By clicking on a Robotino View project  in the directory view the execution of this project on Robotino with the Robotino View Interpreter is invoked. Before the execution of the project starts, the interpreter must be loaded. This process takes some seconds. The log window shows the current state.



After clicking on a Robotino View project a Telnet session is established with user login "robotino" and password "robotino". Immediately after the message "Loading project", the execution starts. The process can be canceled any time.

In the window next to the progress indicator the values of the global variables of the project executed on Robotino are displayed. The update speed can be configured in Extras ▶ Options... ▶ Upload & Execute ▶ Debug interval.


3.13 Upgrade Robotino packages

Since Robotino View version 2.4.0 and Robotino flash card version 2.0 it has been possible to upgrade the Linux packages installed on Robotino from Robotino View. This function is accessible via Robotino ▶ Software update.


The first time the upgrade dialog is called the first Robotino device's current IP address will be entered into the "Robotino IP address" input field. If there is no Robotino device in the current project, the input field remains empty.


Using Robotino® View

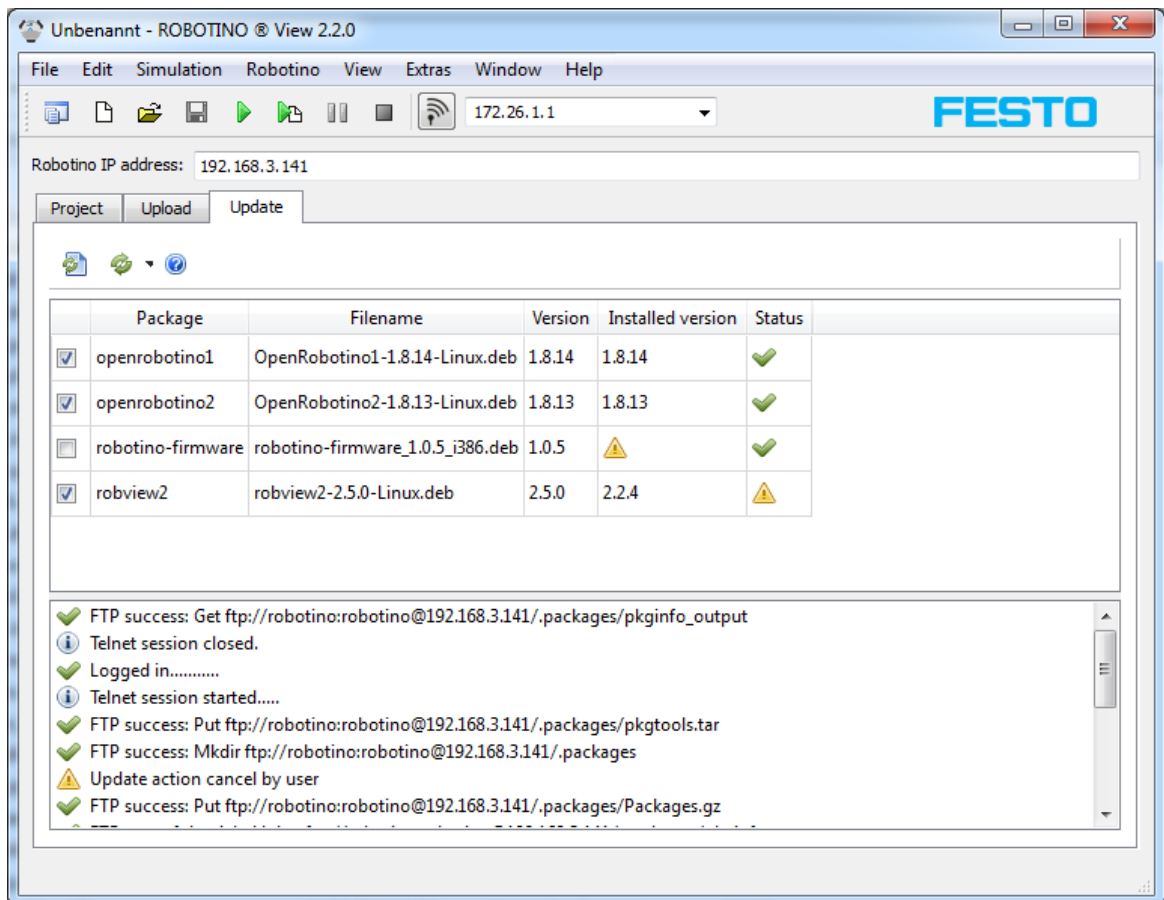
When the dialog is opened, the package information will be refreshed. During the refresh the whole application is locked. This action can be easily canceled, though.



A refresh can be forced via the symbol .

After a successful refresh the versions of local packages and packages installed on Robotino are displayed. The status symbols have the following meanings:


 No information is available or installed version is not up-to-date

 The package installed on Robotino is up-to-date





The package "robotino-firmware" is special. The upgrade routine checks if there is an EA09 IO board in Robotino. If an EA09 IO board is found, the version number will be retrieved directly from the IO board. If no EA09 board is present, the symbol  will be displayed instead of the version number. However, the package's status is  because the package "robotino-firmware" needn't be installed.

In the first column of the version view, packages can be added to or removed from the upgrade process. By default the packaged "openrobotino1", "openrobotino2" and "robview2" are designated for an upgrade.

In the screenshot above, the package "robview" installed on Robotino is not up-to-date. The local version is 2.5.0. On Robotino, the old version 2.2.4 is installed. Installation of new packaged is invoked with the symbol . The upgrade dialog shows that the action is performed. In the log windows the progress can be tracked. When the installation has been finished, the version view will be refreshed.

3.13.1 Robotino firmware installation

The package "robotino-firmware" is special. The upgrade routine checks if there is an EA09 IO board in Robotino. If an EA09 IO board is found, the version number will be retrieved directly from the IO board. If no EA09 board is present, the symbol  will be displayed instead of the version number. However, the package's status is  because the package "robotino-firmware" needn't be installed.


As the upgrade of Robotino's firmware by the package "robotino-firmware" is critical, this package won't be upgraded by default. Only if the exact reason for an upgrade is known, this packages should be added to the upgrade process. The installation of the firmware is described in the section Robotino firmware installation.

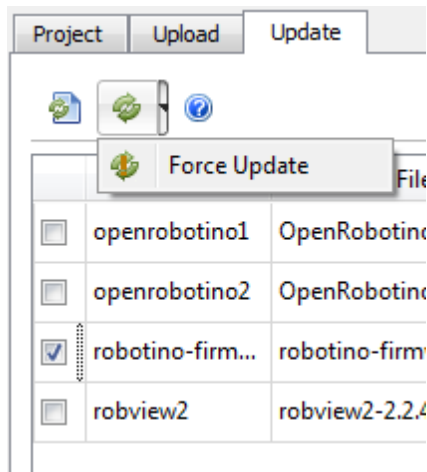
The firmware of the microcontroller (a NXP LPC 2378) on that IO board can be upgraded from Robotino's PC104. This process is critical. A failure of the firmware upgrade results in the following effects:

1. Robotino can no longer be turned off by pressing the On/Off button.
2. Pressing the On/Off button turns on Robotino. When the button is released, Robotino is turned off immediately.

concerning 1) By removing the command bridge, Robotino can be turned off

concerning 2) The On/Off button must be held until an other firmware upgrade was successful

To just upgrade the firmware (or repair it), only the "robotino-firmware" package should be selected. Then the installation can be forced via the button  "Force Update".



3.13.2 Interna

The upgrade process is based on a combination of Telnet, FTP and Linux commands concerning apt.

First the file pkgtools.tar from the directory install_folder/packages is copied into /home/robotino/packages. Via Telnet the file is unpacked. The script pkginfo.sh provides information about the installed packages.

The packages to be installed are copied via FTP from install_folder/packages to /home/robotino/packages. Additionally the file Packages.gz is copied. It contains package informations.

Initially, the script pkginstall.sh modifies /etc/apt/sources.list and enters the directory /home/robotino/packages as only package source. Then apt-get is used to install the packages.

pkgremove.sh forces removal of packages.

startOpenRobotino1.sh is invoked to restart the Robotino deamons.

4 Examples

4.1 Control programs

In this chapter a simple control program with alternative branches is realized.

4.1.1 Tutorial 2

This exercise shows how a control program with alternative branches is created. The complete program is located in the file `examples/sfc/tutorial2.rvw2`.

The complete control program looks as shown in figure 1.

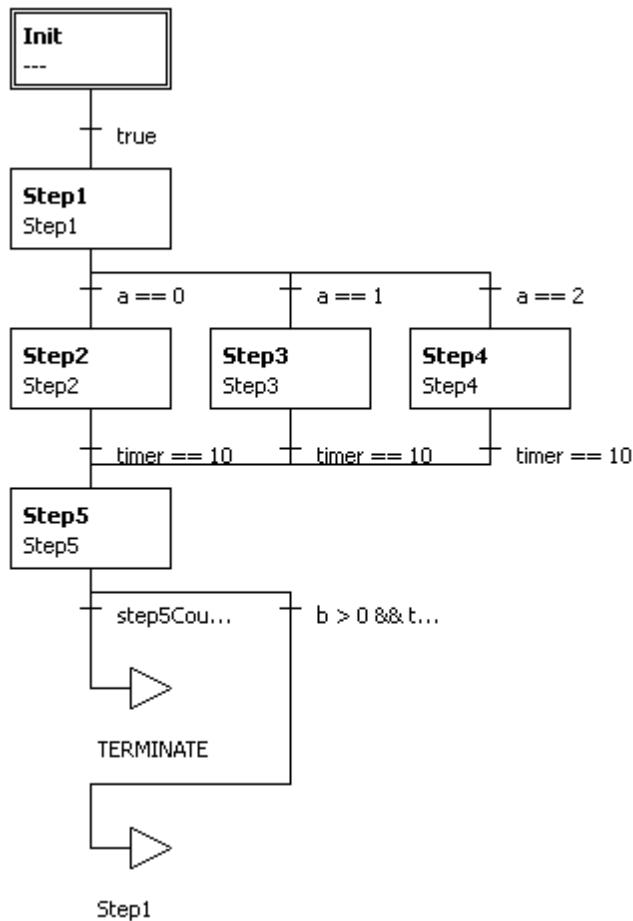



Fig. 1: the complete control program

In Step1 the value of `a` is changed. Thus in every cycle of the program one of the Steps Step2, Step3 and Step4 will be executed. Step5 compares the results produced by the previous steps. After the 6th execution of Step5 the program is stopped. Otherwise it continues with Step1.

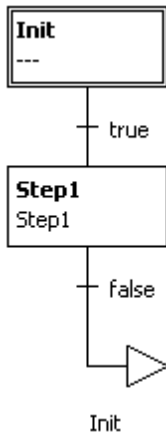
Create a new project

Create a new project by

- selecting File ► New
- pressing Ctrl + N
- selecting the symbol for creating a new project in the tool bar 

Examples

The main program contains the steps Init and Step1.



Create global Variables

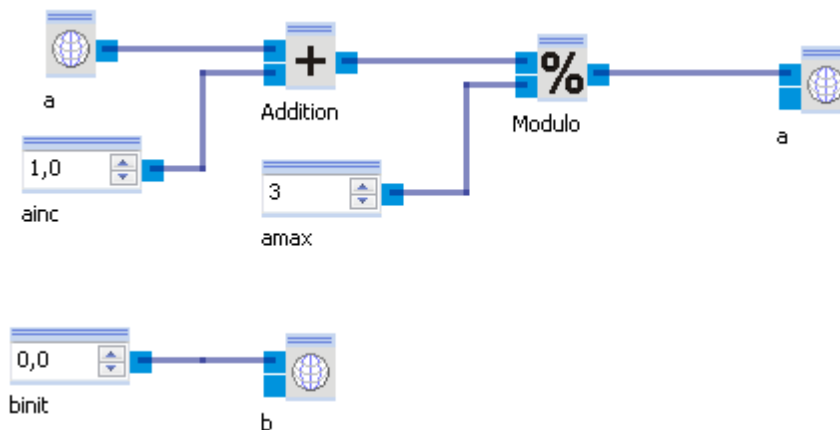
First, create the following [global variables](#)^[15]:

- timer
- a
- b
- step2count
- step3count
- step4count
- step5count

Assign the initial value -1 to "a". All other variables keep their initial value 0.

Program Step1

In this sub-program the global variable "a" is incremented by 1. To make sure that the value of "a" is always between 0 and 2, "a" will be calculated Modulo 3 and rewritten to "a". The value of "b" will just be set 0.

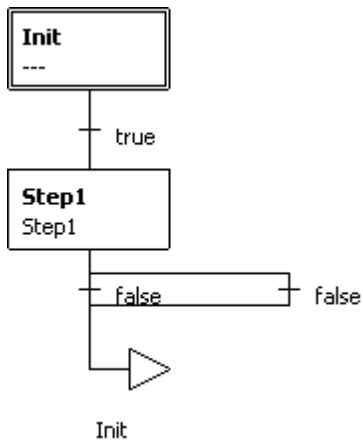



Create steps Step2, Step3 and Step4

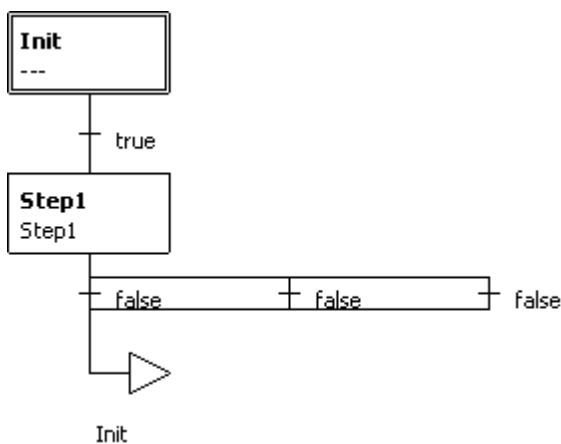
Now the steps will be created next to each other in alternative branches. To do that, select the transition condition below Step1.



You can see that the transition condition is selected by a dashed line round the condition.

Now click on the symbol to add an alternative branch on the right side .

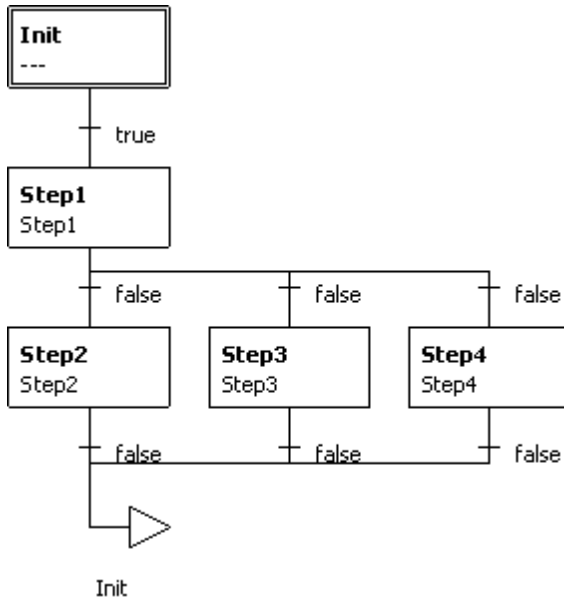


Expand the branching you have just created by selecting the transition condition on the right and selecting the "Alternative branch right" , symbol again.

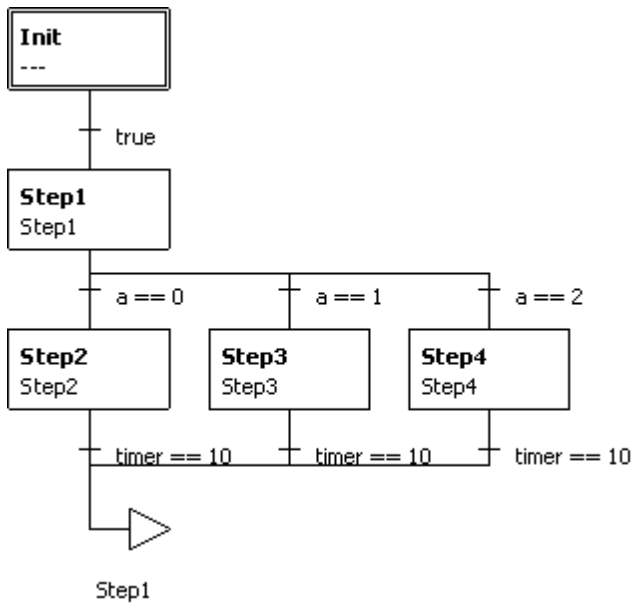


Now create three Steps in those three alternative branches and call them Step2, Step3 and Step4. To do that, select the entering condition of a branch and click on the "insert step after"  symbol. Then assign a sub-program of the same name to each step. To do that, double-click on the step and enter the name for the sub-program in the following dialog box. Alternatively, you can create a new sub-program with the tool bar button "Create new subprogram"  and assign it to the step.

Examples



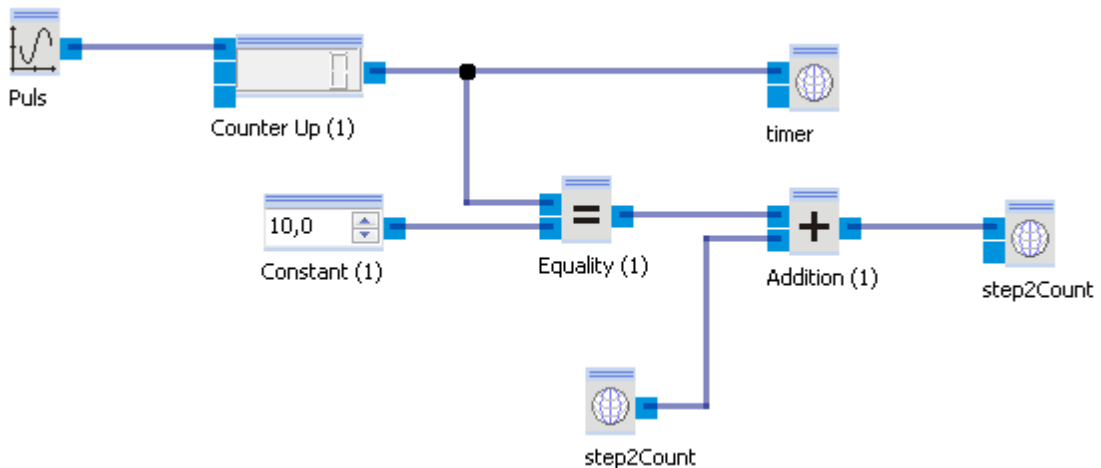
The entering and exit conditions of all three alternative branches are false at the moment. Change the entering conditions to `a == 0`, `a == 1` and `a == 2`. Use `timer == 10` as exit condition for all branches. Finally, change the final jump's destination from `Init` to `Step1`.



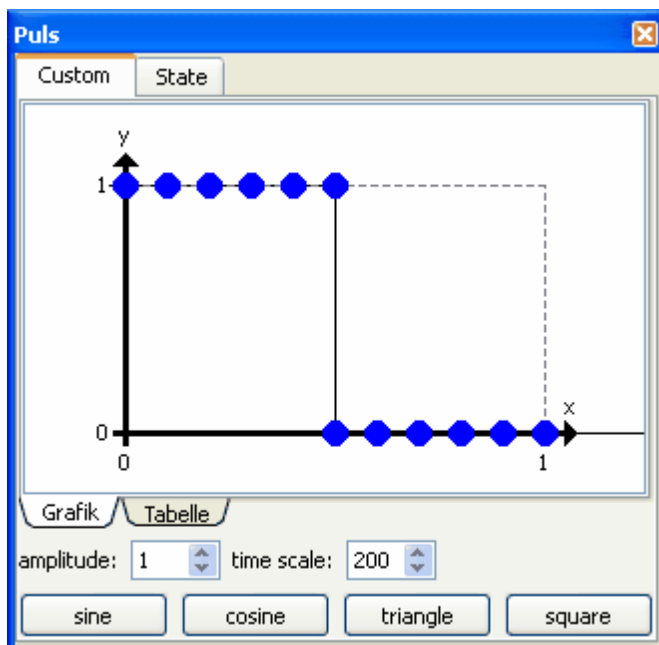
If you start the main program now, the program will hang in `Step2` because `"a"` is 0 during the first cycle and the global variable `"timer"` is not altered.

Program Step2, Step3 and Step4

The sub-programs assigned to the steps `Step2` to `Step4` are empty at the moment. The sub-program `Step2` is shown below.



Every 200ms the Arbitrary Waveform Generator creates a pulse of 100ms width and height 1. The settings for the Arbitrary Waveform Generator are shown below.




I.e. every 200ms there is a rising edge from 0 to 1. With every rising edge the counter increments its value by 1. After 2s the value will be 10. When the value of the counter is 10, the result of the comparison of the constant and the counter's value will be added to the current value of step2Count. As long as the comparison results in false, 0 is added. As soon as the comparison condition below the step in the main program is evaluated. When the global variable "timer" has the value 10, the sub-program will be left.

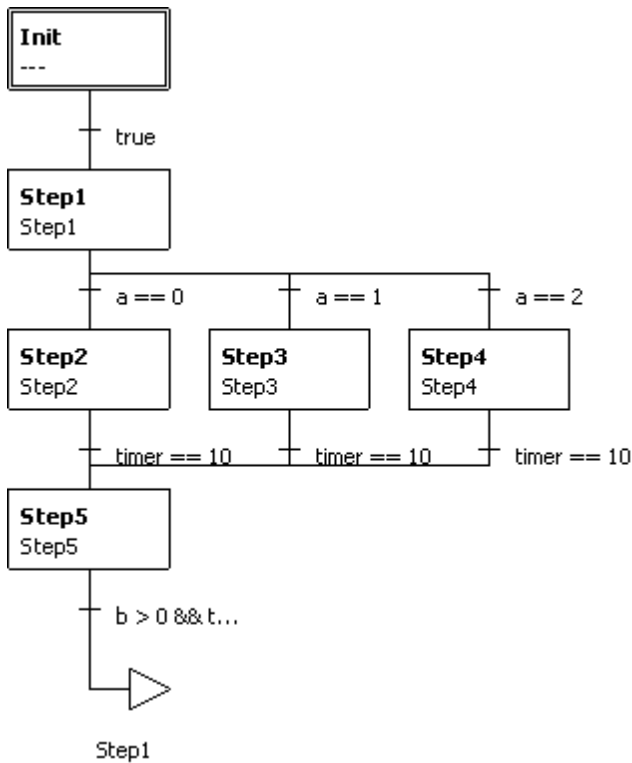
The sub-programs of Step3 and Step4 are built equivalently. Select all in Step2 (Ctrl+A) and copy it to Step3 and Step4. The only difference consists in the fact that step3count respectively step4count are read and written.

Once you start the main program, Step2, Step3 and Step4 will be executed cyclically for 2s each.

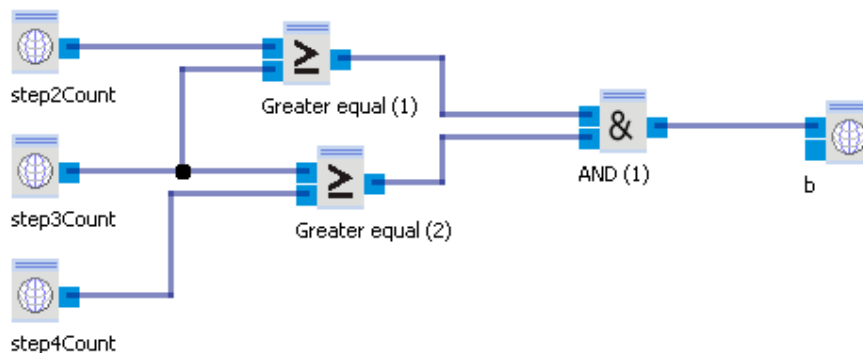
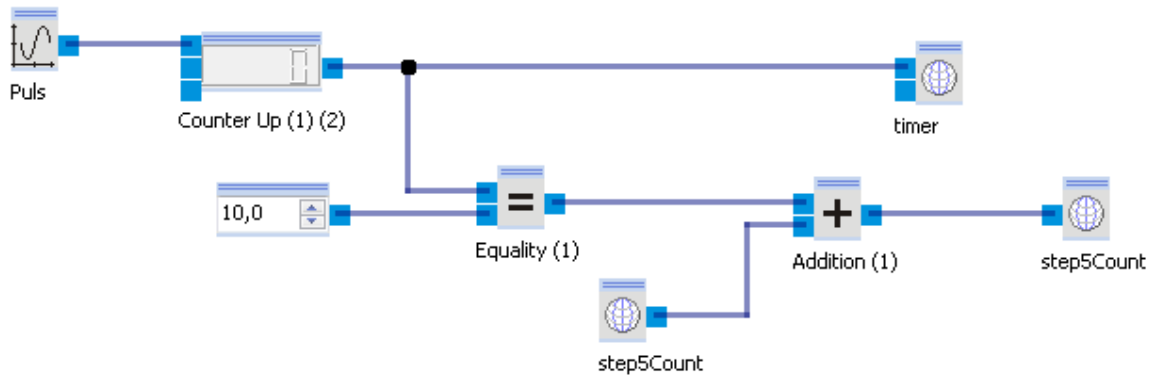
Examples

Create and program Step5

To add a new step after the alternative branching, select the final jump and click on the symbol to add a step before . Now create a sub-program named Step5 and assign it to Step5 just created. Change the transition condition below Step5 to `b > 0 && timer == 10`.





The sub-program Step5 is similar to Step2 to Step4. Copy Step2 to Step5 and change step2count to step5count. Beyond setting the global variables "timer" and "step5count" also a check is performed if the condition `step2count >= step3count >= step4count` is valid. If this is the case, the global variable "b" is set to 1. Otherwise "b" is 0. The condition must always be true when in a correct program execution because Step2, Step3 and Step4 are executed one after another because Step1 increments "a" by 1 in every cycle.



If the main program is started now, Step5 remains active for 2s if "b" is greater than 0.

Create program termination and jump to Step1

Now the program should be terminated when the value of "step5count" has reached 6. To achieve this, insert an alternative branch below Step5. Select the transition condition below Step5 and click on the symbol to insert an alternative branch on the left . Select the new branch's transition condition (at the moment it is false) and click on the symbol to create a new jump . Change the transition condition to `step5count == 6` and select TERMINATE as new jump destination.

The main program now looks as it was shown at the beginning.

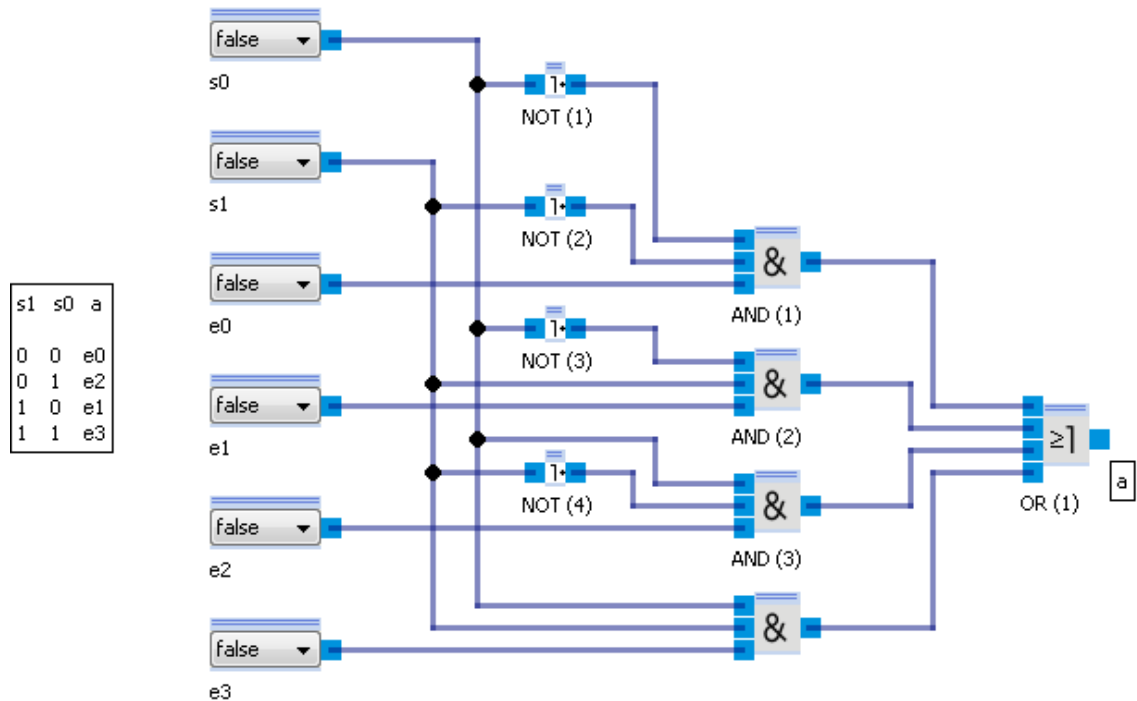
By the way, the alternative branch containing the jump to TERMINATE must be left of the branch with the condition `b>0 && timer == 10` because the initial conditions of alternative branches are evaluated from left to right. In the first 6 cycles the condition `step5count == 6` is not fulfilled. So the second branch's condition is evaluated.

One run of the main program lasts 24s now.

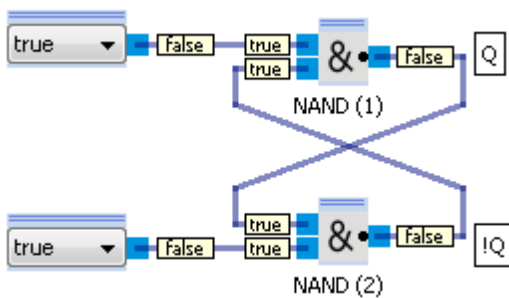
4.2 Logic

In this chapter well-known electrical circuits are realized with logical modules.

4.2.1 Multiplexer



4.2.2 FlipFlop



5 Function block library

The control programs created with Robotino® View consist of interlinked function blocks.

These are located in the [function block library](#)⁹⁾ and can be inserted into a sub-program via Drag&Drop.

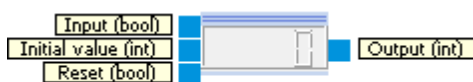
Function blocks are assigned to different categories. By clicking onto a category name with the left mouse button, the category folder is expanded. The following categories are available:

Name	Description
Logic ^[35]	Components as recognised from electronic logic modules
Mathematics ^[54]	Simple mathematical operations
Vector ^[69] analysis ^[69]	Analysis using two-dimensional vectors
Display ^[77]	Function blocks for visualization
Image processing ^[79]	Basic image processing functionalities
Generators ^[91]	Generation of signals
Filter ^[95]	Smoothing of signals
Navigation ^[96]	Driving mobile Robots
Input devices ^[116]	Function blocks for the interaction of the user with the control program
Data exchange ^[118]	Exchange data with external programs
My function blocks ^[171]	Tutorials for the development of own function blocks

5.1 Logic

The Logic category contains components as recognized from the electronic logic modules.

5.1.1 Counter up



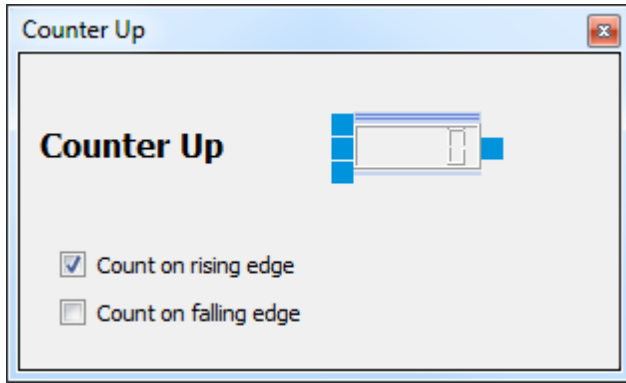
The counter counts the number of events at its Input connector

Inputs	Type	Default	Description
Input	bool	false	Counter input. Counter value is increased if the input changes from false to true and/or from true to false.
Initial value	int32	0	Counting starts with the value given here at sub-program start or if Reset is true.
Reset	bool	false	The counter is reset to its initial value if this input is true.

Function block library

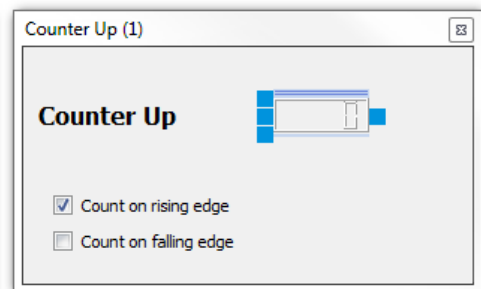
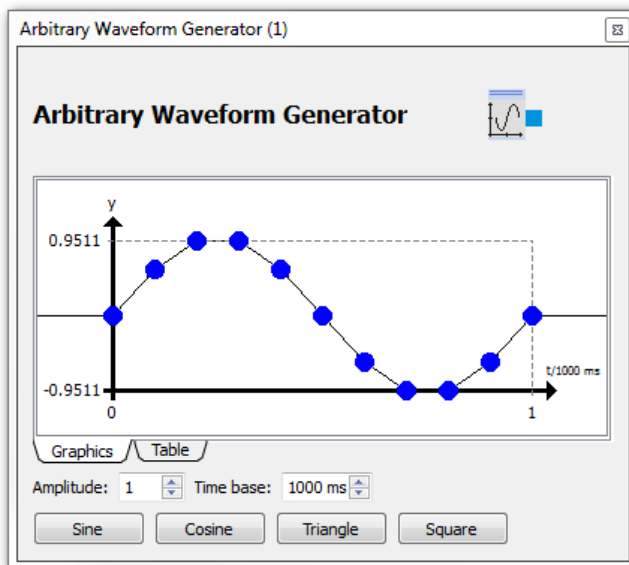
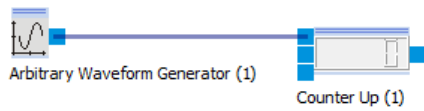
Outputs			
Output	int32		Counter value

5.1.1.1 Dialog



Count on rising edge	Increment the counter by 1 if the input at time t is false and at time t+1 true.
Count on falling edge	Increment the counter by 1 if the input at time t is true and at time t+1 false.

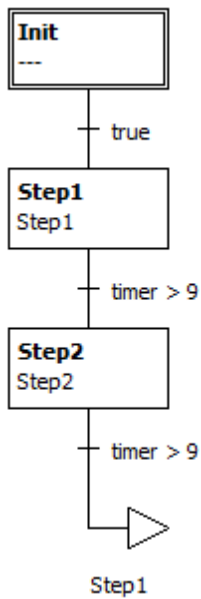
5.1.1.2 Example



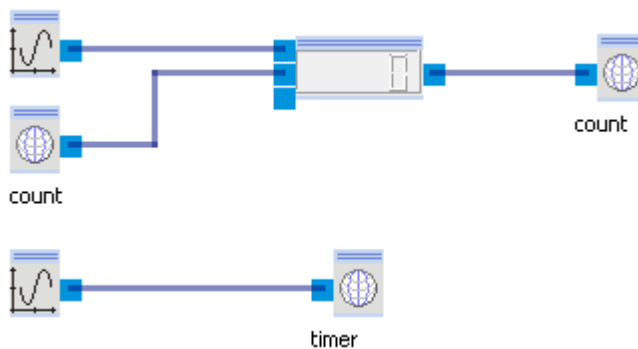
The "[Arbitrary Waveform Generator](#)^[92]" generates a sin waveform with amplitude 2 and frequency 1 Hz. The output of the generator is of type float. Values less equal 0 are converted to false. Value greater 0 are converted to true (see [type conversion](#)^[20]). The counter counts on rising edge, i.e. when the input changes from false to true. This happens exactly once per second at the beginning of the sine wave. The counter values represents therefore the time in seconds since sub-program start.

The following example shows how to use the initial value input to count over sub-program boundaries. The main program executes Step1 and Step2 sequentially. After Step2 is finished, we restart with Step1.

Hauptprogramm

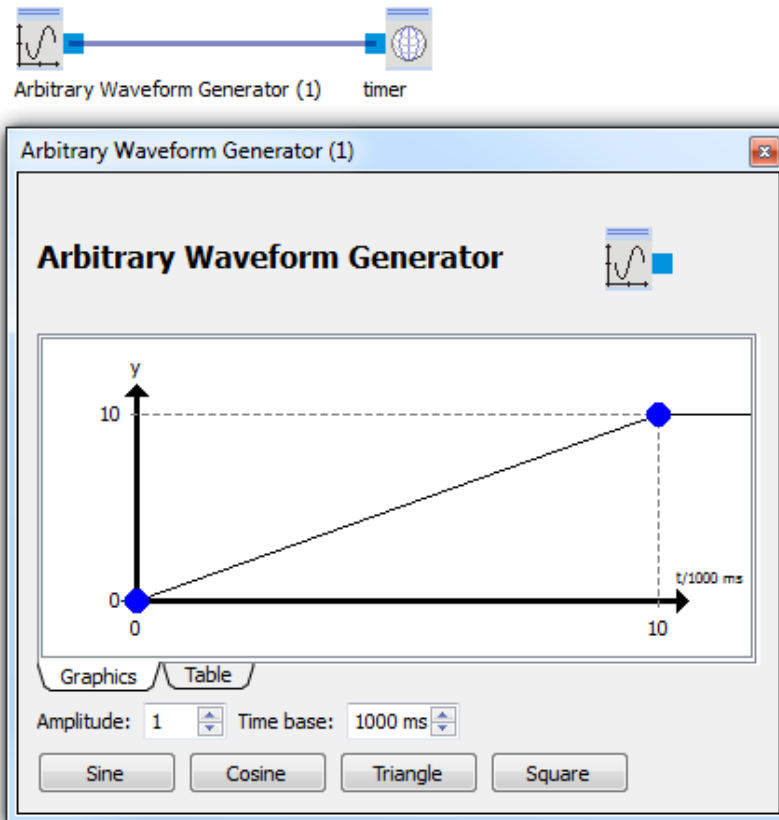


Step1



The Counter writes its result into the global variable "count". After restart of Step1 the global variable count is used as initial value for the Counter. Step1 is active until the second "[Arbitrary Waveform Generator](#)^[92]" generates a value greater 9. This happens after 10s.

Step2



Step2 is also 10s active.

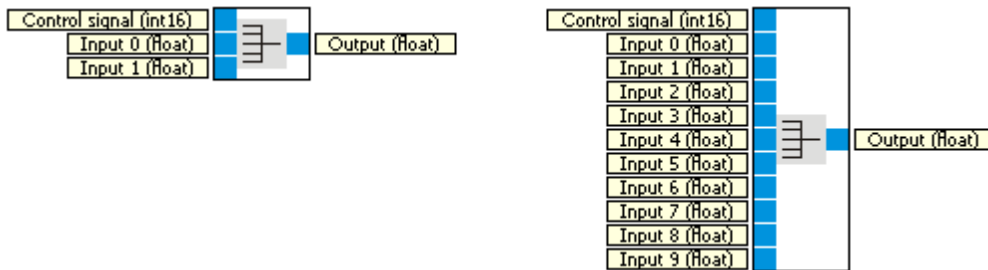
5.1.2 Counter down

Counter down is similar to [Counter up](#)^[35]. The only difference is that the counting value is decremented by 1 if a counting event occurs.

5.1.2.1 Dialog

See dialog of [Counter up](#)^[36].

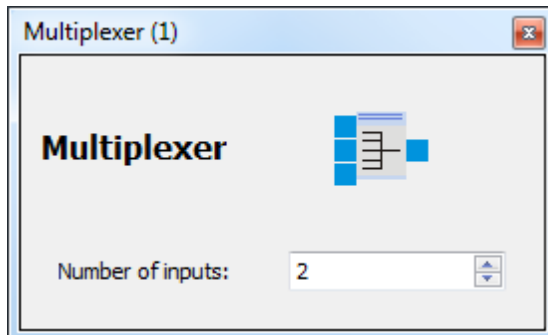
5.1.3 Multiplexer



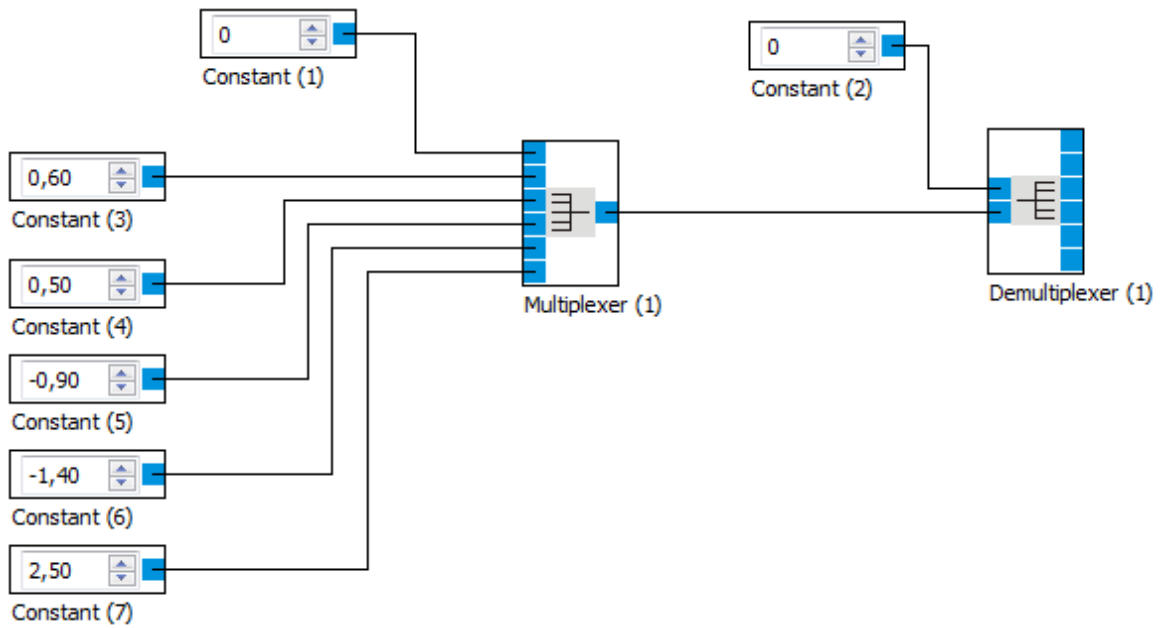
The Multiplexer connects its output to a selectable input.

Inputs	Type	Default	Description
Control signal	int	0	Determines the input that is connected to the output. If the control signal is less 0 or greater equal the number of inputs the output is 0.
Input 0	float	0	The value of input 0 is available at the output if the control signal is 0.
...			
Input 9	float	0	The value of input 9 is available at the output if the control signal is 9.
Outputs			
Output	float		The value of an input or 0 if the control signal is less 0 or greater equal the number of inputs.

5.1.3.1 Dialog

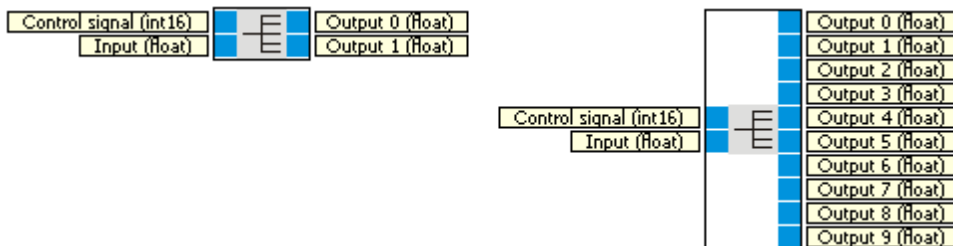


5.1.3.2 Example



see also Examples ▶ Logic ▶ [Multiplexer](#) ³⁴

5.1.4 Demultiplexer

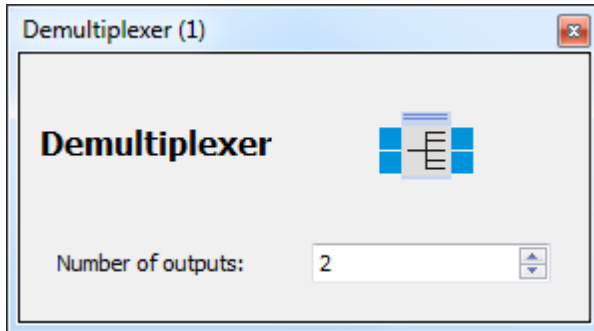


The demultiplexer connects one input to a selectable output.

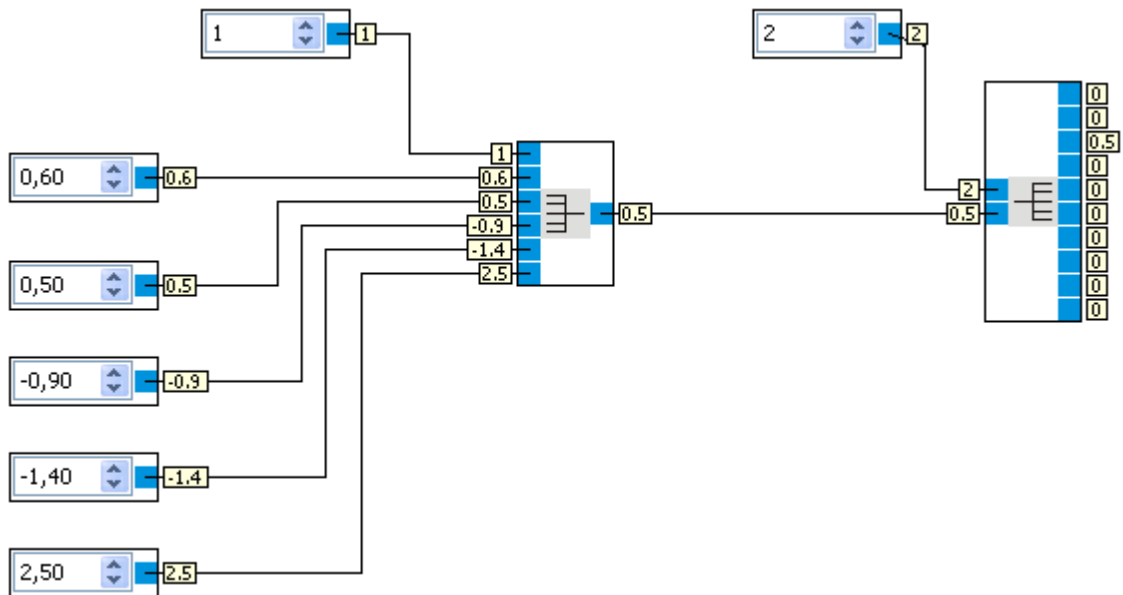
Inputs	Type	Default	Description
Control signal	int	0	Determines the output that is connected to the input. If the control signal is less 0 or greater equal the number of outputs all outputs are reset to 0.
Input	float	0	The value of an output if the control signal is greater equal 0 and less the number of outputs.
Outputs			
Output 0	float		Value of the input if the control signal is 0, otherwise 0.
...			

Output 9	float	Value of the input if the control signal is 9, otherwise 0.
----------	-------	---

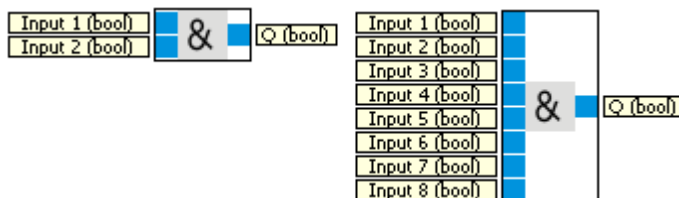
5.1.4.1 Dialog



5.1.4.2 Example



5.1.5 AND

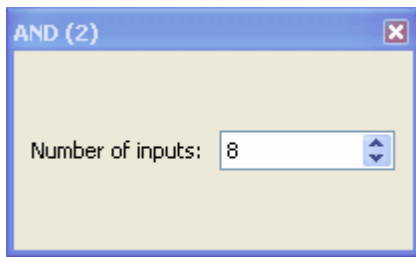


The Output of the AND is true only if all Inputs are true. See [type conversion](#)^[20] how numbers are converted to bool.

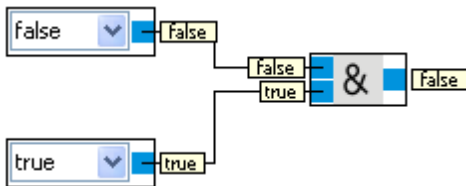
Inputs	Type	Default	Description
Input 1	bool	true	
...			
Input 8	bool	true	
Outputs			
Q	bool		see table below

Inputs								
1	2	3	4	5	6	7	8	Q
0	0	0	0	0	0	0	0	0
							1	0
						1		0
						1	1	0
					1			0
					1		1	0
					1	1		0
					1	1	1	0
				1				0
				1			1	0
				1		1		0
				1		1	1	0
				1	1			0
				1	1		1	0
				1	1	1		0
				1	1	1	1	0
			1					0
1	1	1	1	1	1	1	1	1

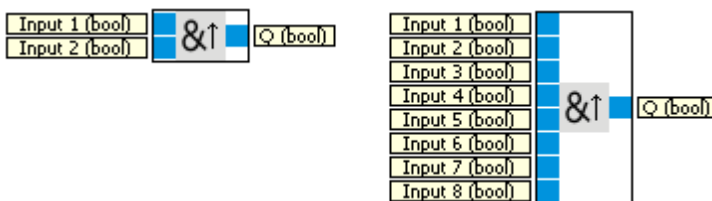
5.1.5.1 Dialog



5.1.5.2 Example



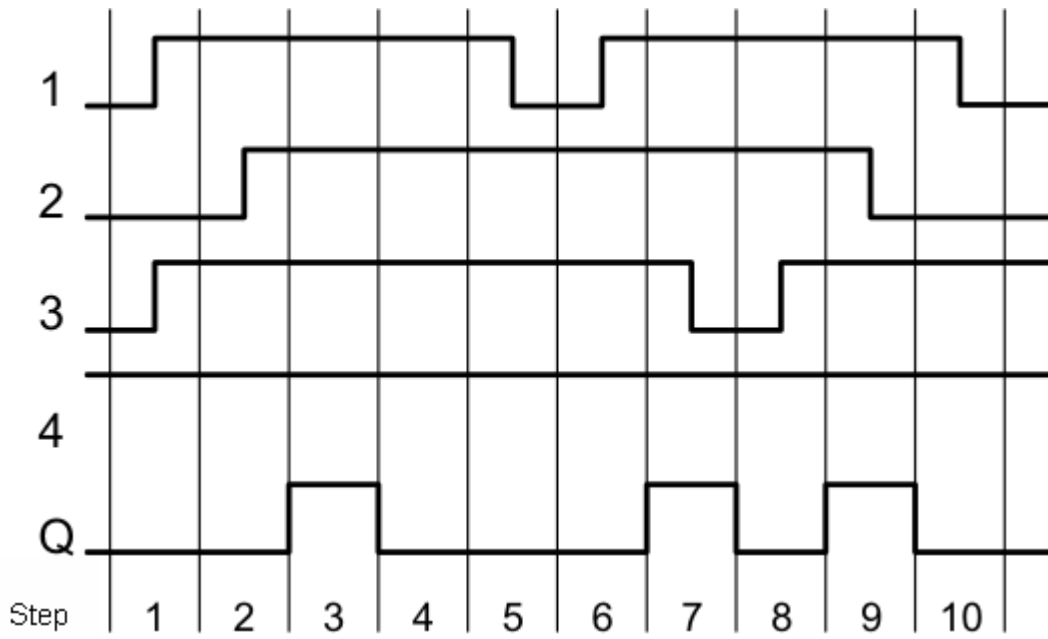
5.1.6 AND FL



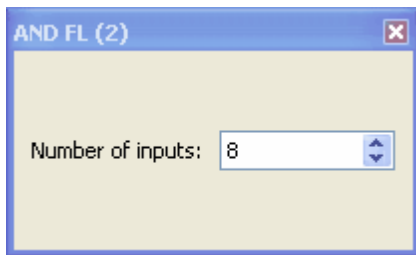
The output Q of the AND FL (with edge control) is only set to true if all inputs are true, and if at least one input was false during the previous cycle. See [type conversion](#) how numbers are converted to bool.

Inputs	Type	Default	Description
Input 1	bool	true	
...			
Input 8	bool	true	
Outputs			
Q	bool		see timing diagram

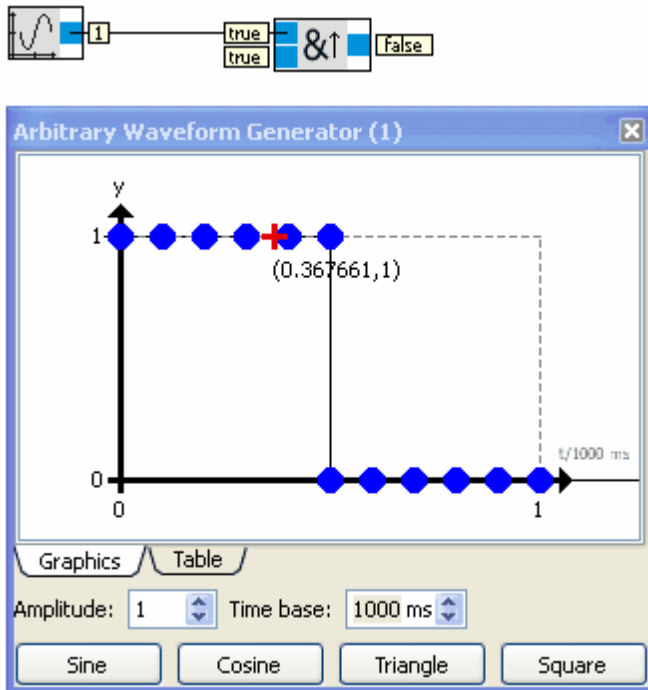
Timing diagram for the AND FL and four inputs.



5.1.6.1 Dialog

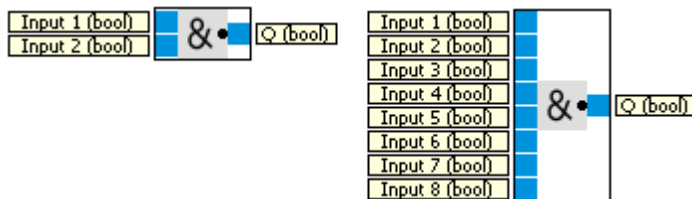


5.1.6.2 Example



When the output of the generator changes from 0 to 1 the output of the AND FL is true for one cycle.

5.1.7 NAND

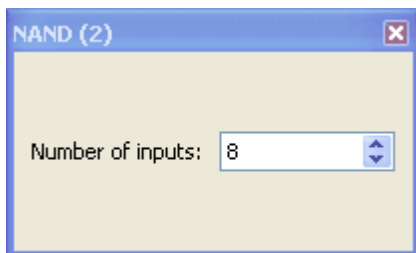


The Output of the NAND is false only if all Inputs are true. See [type conversion](#)^[20] how numbers are converted to bool.

Inputs	Type	Default	Description
Input 1	bool	true	
...			
Input 8	bool	true	
Outputs			
Q	bool		see table below

Inputs								
1	2	3	4	5	6	7	8	Q
0	0	0	0	0	0	0	0	1
							1	1
						1		1
						1	1	1
					1			1
					1		1	1
					1	1		1
					1	1	1	1
				1				1
				1			1	1
				1		1		1
				1		1	1	1
				1	1			1
				1	1		1	1
				1	1	1		1
				1	1	1	1	1
			1					1
1	1	1	1	1	1	1	1	0

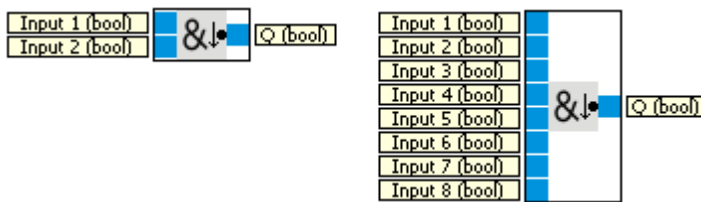
5.1.7.1 Dialog



5.1.7.2 Example

see Example ▶ Logic ▶ [FlipFlop](#) ³⁴

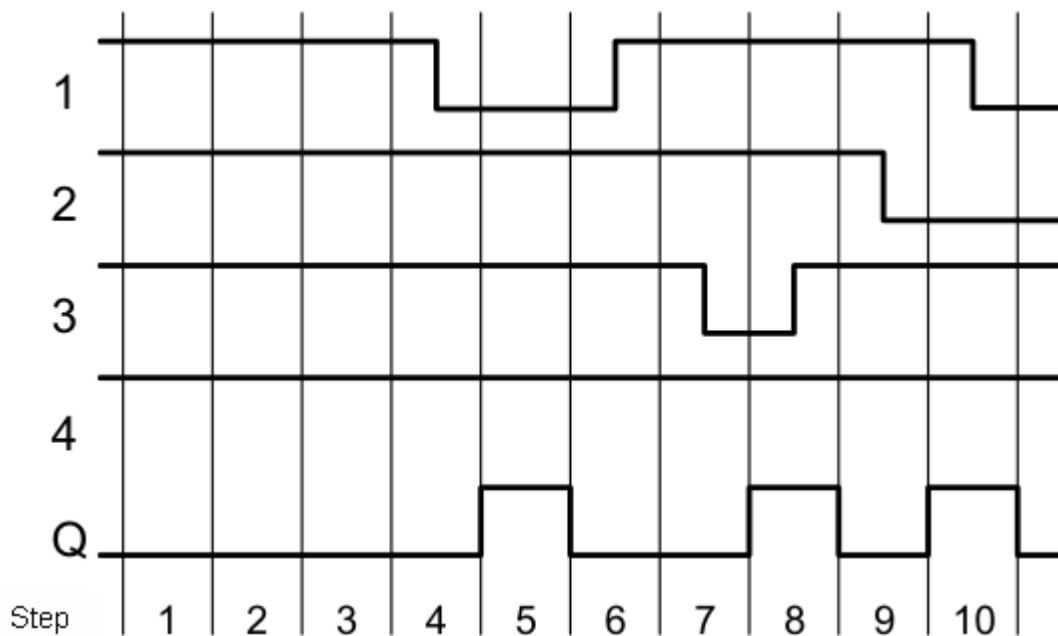
5.1.8 NAND_FL



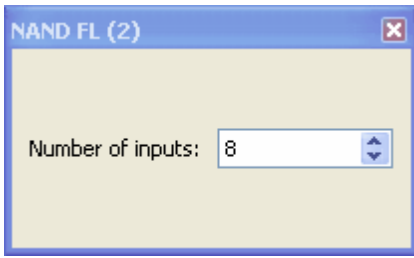
The output Q of the NAND with edge control is only set to true if at least one input is false, and if all inputs were true during the previous cycle. See [type conversion](#) ^[20] how numbers are converted to bool.

Inputs	Type	Default	Description
Input 1	bool	true	
...			
Input 8	bool	true	
Outputs			
Q	bool		see timing diagram

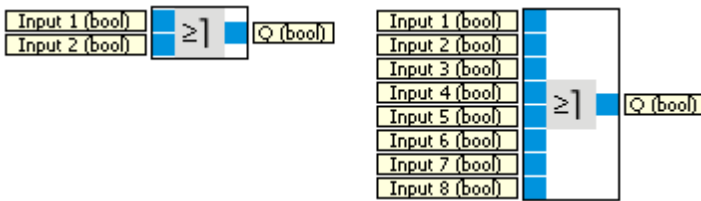
Timing diagram for the NAND with edge control and four inputs.



5.1.8.1 Dialog



5.1.9 OR



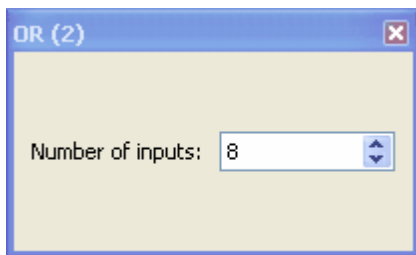
The Output of the OR is true only if at least one Input is true. See [type conversion](#)^[20] how numbers are converted to bool.

Inputs	Type	Default	Description
Input 1	bool	false	
...			
Input 8	bool	false	
Outputs			
Q	bool		see table below

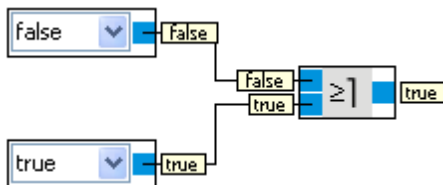
Inputs								
1	2	3	4	5	6	7	8	Q
0	0	0	0	0	0	0	0	0
							1	1
						1		1
						1	1	1
					1			1
					1		1	1
					1	1		1
					1	1	1	1
				1				1

				1			1	1
				1		1		1
				1		1	1	1
				1	1			1
				1	1		1	1
				1	1	1		1
				1	1	1	1	1
			1					1
1	1	1	1	1	1	1	1	1

5.1.9.1 Dialog



5.1.9.2 Example



5.1.10 XOR



The Output of the XOR is true if the inputs have different values. See [type conversion](#)^[20] how numbers are converted to bool.

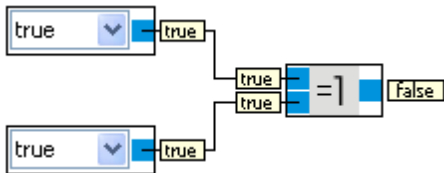
Inputs	Type	Default	Description
Input 1	bool	false	
Input 2	bool	false	

Function block library

Outputs			
Q	bool		see table below

Inputs		
1	2	Q
0	0	0
0	1	1
1	0	1
1	1	0

5.1.10.1 Example



5.1.11 NOT



The Output of the NOT is true if the input is false. See [type conversion](#) how numbers are converted to bool.

Inputs	Type	Default	Description
Input	bool	false	
Outputs			
Q	bool		see table below

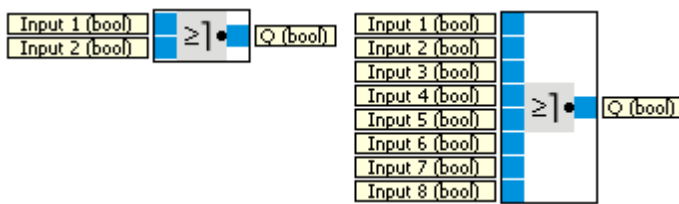
Inputs	
1	Q
0	1
1	0

5.1.11.1 Example



The example shows a specialness of the NOT function block. Input and output values are not shown next to its input or output connector. This has the advantage that the NOT takes only a very small amount of space and the data display does not overlap with data displayed by other function blocks.

5.1.12 NOR



The NOR's Output Q is true if all inputs are false. See [type conversion](#)^[20] how numbers are converted to bool.

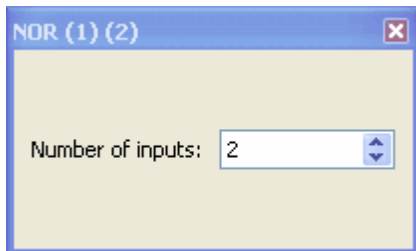
Inputs	Type	Default	Description
Input 1	bool	false	
...			
Input 8	bool	false	
Outputs			
Q	bool		see table below

Inputs								
1	2	3	4	5	6	7	8	Q
0	0	0	0	0	0	0	0	1
							1	0
						1		0
						1	1	0
					1			0
					1		1	0
					1	1		0

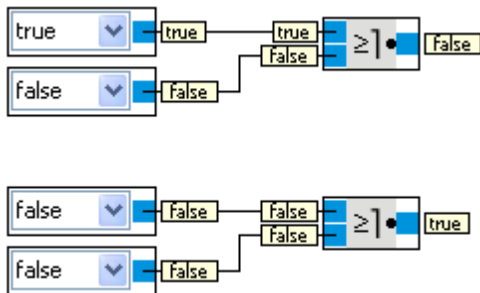
Function block library

					1	1	1	0
				1				0
				1			1	0
				1		1		0
				1		1	1	0
				1	1			0
				1	1		1	0
				1	1	1		0
				1	1	1	1	0
			1					0
1	1	1	1	1	1	1	1	0

5.1.12.1 Dialog



5.1.12.2 Example



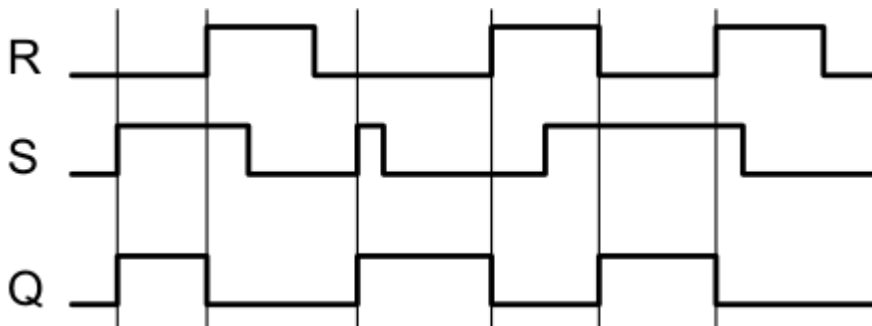
5.1.13 Latching relay



Output Q is set by input S. Input R resets output Q. See [type conversion](#)^[20] how numbers are converted to bool.

Inputs	Type	Default	Description
S	bool	false	If S is true Q becomes true.
R	bool	false	If R is true Q is reset to false. R overrules S.
Par	bool	false	Remanence: false: No remanence true: The current status is saved to remanent memory (independent of S or R).
Outputs			
Q	bool		Q is switched to true by S and remains true until R becomes true.

Timingdiagramm



5.1.14 Sample and hold element



If Sample is set false, the signal at Input can be kept at the current value. See [type conversion](#)^[20] how numbers are converted to bool.

Inputs	Type	Default	Description
Input	float	0	Input signal
Sample	bool	false	If true, Output will be connected to Input. If false, the current value will be frozen at Output.
Outputs			

Output	float	0	Last value of Input before Sample has been changed from true to false.
--------	-------	---	---

5.2 Mathematics

This category contains simple mathematical operations.

5.2.1 Arithmetic operations

5.2.1.1 Modulo



In mathematics, modular arithmetic (sometimes called clock arithmetic) is a system of arithmetic for integers, where numbers "wrap around" after they reach a certain value—the modulus. Modular arithmetic was introduced by Carl Friedrich Gauss in his book *Disquisitiones Arithmeticae*, published in 1801. (Source: http://en.wikipedia.org/wiki/Modular_arithmetic)

Inputs	Type	Default	Description
Dividend	int	0	
Divisor	int	1	
Outputs			
Remainder	int		Dividend mod Divisor

5.2.1.2 Division

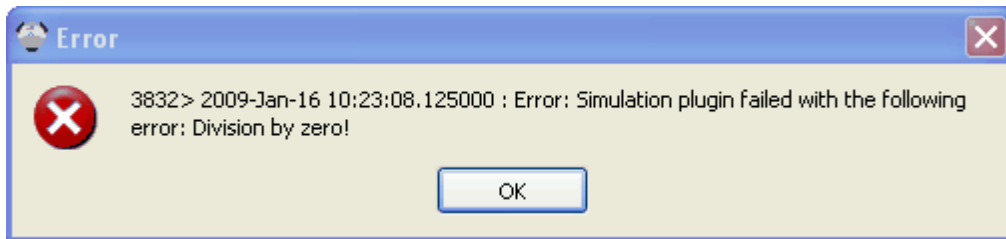


Calculates the quotient from dividend and divisor. See [http://en.wikipedia.org/wiki/Division_\(mathematics\)](http://en.wikipedia.org/wiki/Division_(mathematics)).

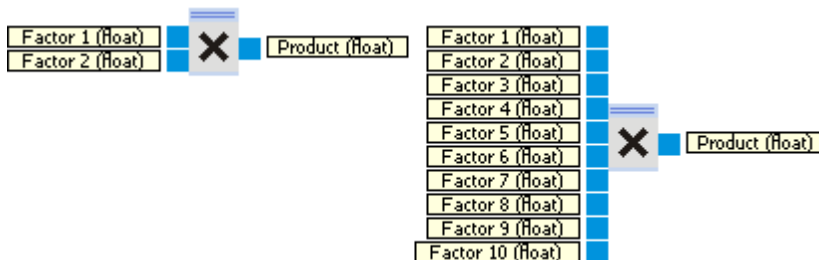
Inputs	Type	Default	Description
Dividend	float	0	
Divisor	float	1	
Outputs			

Outputs			
Quotient	float		Dividend divided by divisor.

If the dividend is unequal to 0 and the divisor equals 0 the simulation is stopped with the following error:



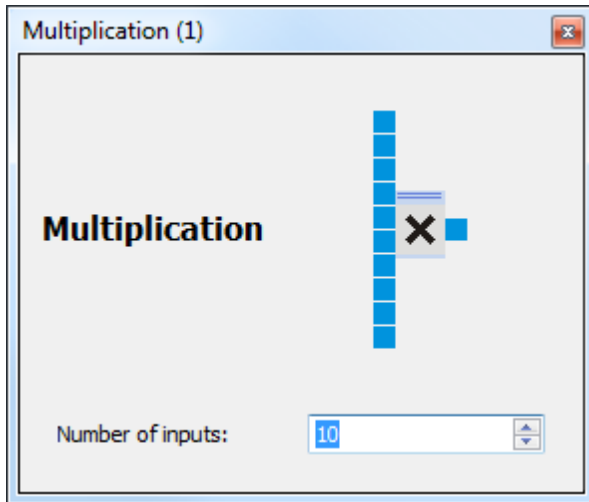
5.2.1.3 Multiplication



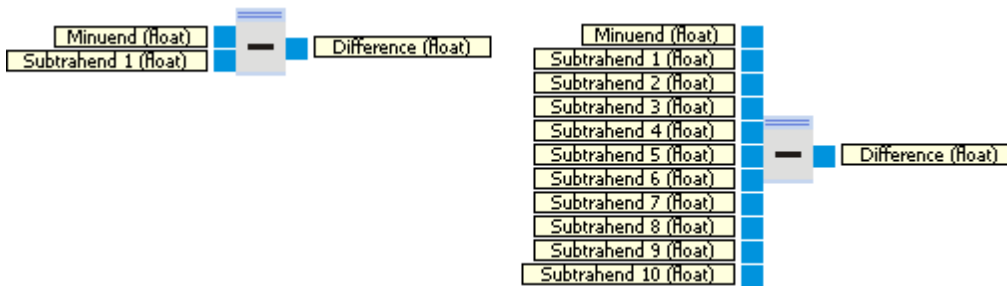
The Multiplication function block multiplies floating point numbers. See also <http://en.wikipedia.org/wiki/Multiplication>.

Inputs	Type	Default	Description
Factor 1	float	1	
...			
Factor 10	float	1	
Outputs			
Product	float		"Factor 1" * "Factor 2" * ... * "Factor 10"

5.2.1.3.1 Dialog



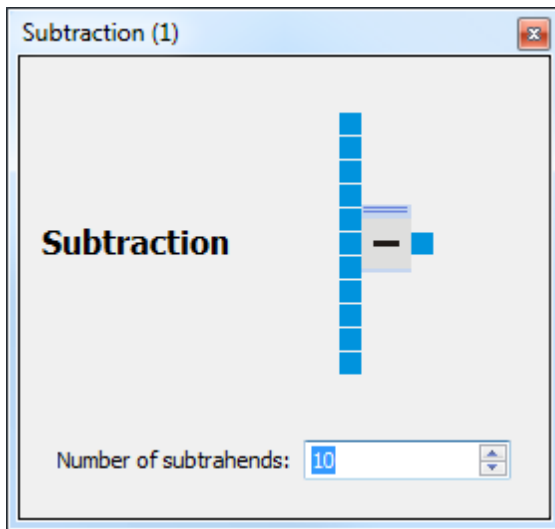
5.2.1.4 Subtraction



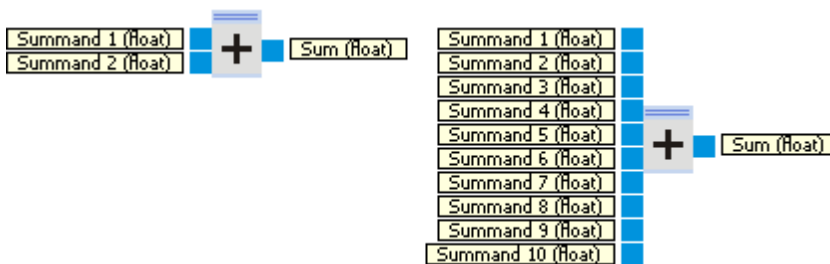
The Subtraction function block calculates the difference between the minuend and up to 10 subtrahends. See also <http://en.wikipedia.org/wiki/Subtraction>.

Inputs	Type	Default	Description
Minuend	float	0	
Subtrahend 1	float	0	
...			
Subtrahend 10	float	0	
Outputs			
Difference	float		Minuend - "Subtrahend 1" - "Subtrahend 2" - ... - "Subtrahend 10"

5.2.1.4.1 Dialog



5.2.1.5 Addition

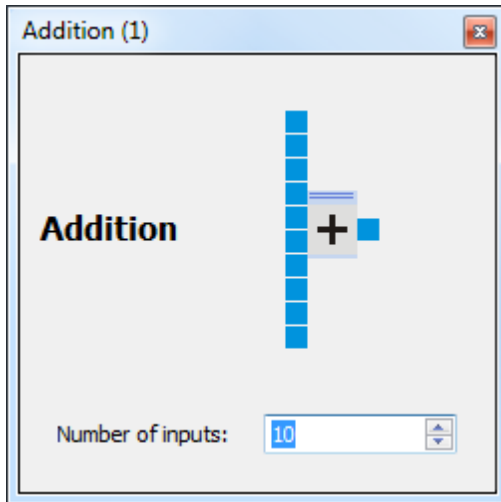


The Addition function block adds up to 10 summands. See also <http://en.wikipedia.org/wiki/Addition>.

Inputs	Type	Default	Description
Summand 1	float	0	
...			
Summand 10	float	0	
Outputs			
Sum	float		"Summand 1" + "Summand 2" + ... + "Summand 10"

Function block library

5.2.1.5.1 Dialog



5.2.2 Comparison Operations

5.2.2.1 Inequal



The output is true, if the absolute value of Input1 - Input2 is greater equal epsilon, with epsilon = 0.0000002384185792.

Inputs	Type	Default	Description
Input 1	float	0	
Input 2	float	0	
Outputs			
Output	bool		$\text{fabs}(\text{Input1} - \text{Input 2}) \geq \text{epsilon}$

5.2.2.2 Equal



The output is true, if the absolute value of Input1 - Input2 is less epsilon, with epsilon = 0.0000002384185792

Inputs	Type	Default	Description
Input 1	float	0	
Input 2	float	0	
Outputs			
Output	bool		$\text{fabs}(\text{Input1} - \text{Input2}) < \text{epsilon}$

5.2.2.3 Less equal



The Output is true, if Input1 is less equal Input2.

Inputs	Type	Default	Description
Input 1	float	0	
Input 2	float	0	
Outputs			
Output	bool		"Input1" less or equal "Input2"

5.2.2.4 Less



The Output is true, if Input1 is less Input2.

Inputs	Type	Default	Description
Input 1	float	0	
Input 2	float	0	
Outputs			
Output	bool		"Input1" less "Input2"

5.2.2.5 Greater equal



The Output is true, if Input1 is greater equal Input2.

Inputs	Type	Default	Description
Input 1	float	0	
Input 2	float	0	
Outputs			
Output	bool		"Input1" greater or equal "Input2"

5.2.2.6 Greater



The Output is true, if Input1 is greater Input2.

Inputs	Type	Default	Description
Input 1	float	0	
Input 2	float	0	
Outputs			
Output	bool		"Input1" greater "Input2"

5.2.3 Functions

5.2.3.1 Absolute Value



Gives the absolute value of Input.

Inputs	Type	Default	Description

Input	float	0	
Outputs			
Output	float		abs(Input)

5.2.3.2 Transfer Function



With the transfer function, it is possible to realize any mapping of the input x to the output y.

Inputs	Type	Default	Description
x	float	0	
Outputs			
x	float		see Dialog [61]

5.2.3.2.1 Dialog

With the dialog of the Transfer function function block it is possible to define interpolation points for the mapping $y(x)$. The default interpolation points are

$$p_0 = (x_0, y_0) = (0, 0)$$

$$p_1 = (x_1, y_1) = (10, 10)$$

These points define the following mapping $y(x)$

$$y = y_0 \text{ if } x \leq x_0$$

$$y = x \text{ if } x > x_0 \text{ and } x \leq x_1$$

$$y = y_1 \text{ if } x > x_1$$

Boundaries

$p_0 = (x_0, y_0)$ is the first interpolation point

$p_n = (x_n, y_n)$ is the last interpolation point

If $x < x_0$: $y = y_0$

If $x > x_n$: $y = y_n$

Mapping

If we have a list of interpolation points p_0, p_1, \dots, p_n the mapping $y(x)$ is given by:

$$y = y_0 \text{ if } x \leq x_0$$

$$y = (y_1 - y_0) / (x_1 - x_0) * (x - x_0) + y_0 \text{ if } x > x_0 \text{ and } x \leq x_1$$

$$y = (y_2 - y_1) / (x_2 - x_1) * (x - x_1) + y_1 \text{ if } x > x_1 \text{ and } x \leq x_2$$

...

$$y = y_n \text{ if } x > x_n$$

Move points

Interpolation points can be moved, added and removed. To move an interpolation points you can use the Graphics-View and move the points with the mouse pointer. In the Table-View the x, y values of the interpolation points can be edited. The x value of an interpolation point can never be smaller than the x value of the earlier interpolation point and never be greater than the x value of the following interpolation point.

Adding points

In the Graphics-View you can add a new point anywhere by using the context menu available by clicking with the right mouse button.

Insert point
Import from clipboard
Export to clipboard
Help

In the Table-View the context menu is available by clicking with the right mouse button into a row.

Insert point before
Insert point after
Remove point
Import from clipboard
Export to clipboard
Help

You can choose to insert the new point before or after the current row.

Delete points

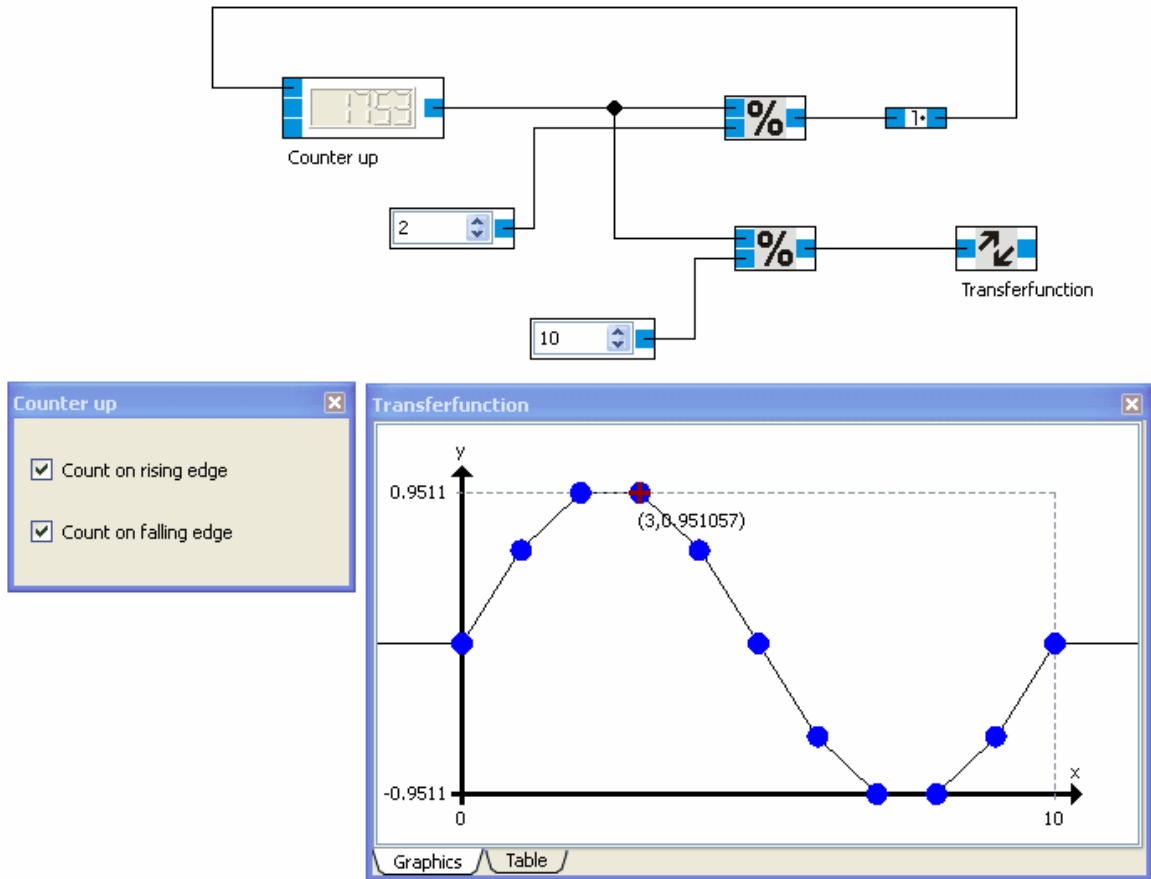
Points are deleted in both the Graphics-View and the Table-View via the context menu after right mouse click onto a point or row. If there is only a single interpolation point left, the function for deleting this point is deactivated.

Import/Export of interpolation points

The clipboard can be used to import and export the list of interpolation points. By this data can be exchanged with programs like Excel or Matlab. The function for Import/Export is available via the context menu in both the Graphics and Table-View.

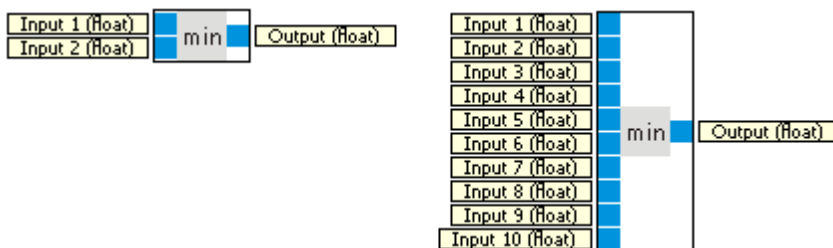
Function block library

5.2.3.2.2 Example



The Counter up is incremented every simulation step. The counting value is restricted to the range [0,10]. The Transferfunction defines a sine wave with 10 interpolation points.

5.2.3.3 Minimum



The value of the output is the minimal value from all inputs.

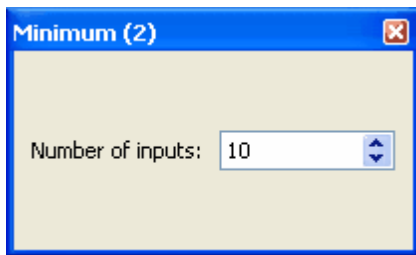
Inputs	Type	Default	Description
		t	

Input 1	float	1e+037	
...			
Input 10	float	1e+037	
Outputs			
Output	float		min("Input 1", "Input 2", ..., "Input 10")

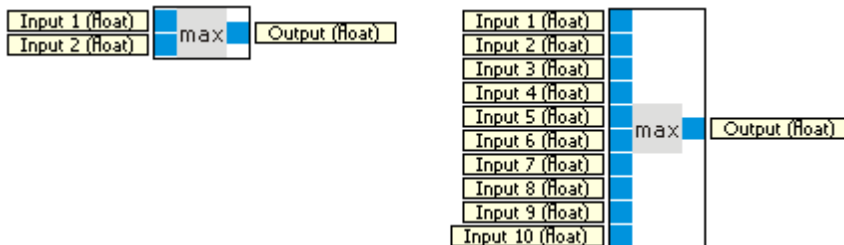
1e+037 = 10 pow 37

largest possible floating point number

5.2.3.3.1 Dialog



5.2.3.4 Maximum



The value of the output is the maximal value from all inputs.

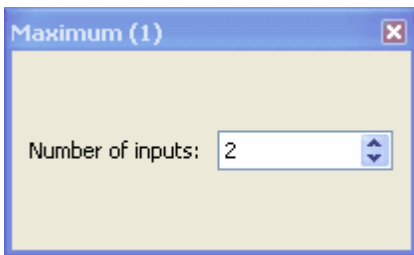
Inputs	Type	Default	Description
Input 1	float	-1e+037	
...			

Function block library

Input 10	float	-1e+037	
Outputs			
Output	float		max("Input 1", "Input 2", ... , "Input 10")

-1e+037 = - (10 pow 37)
 smallest possible floating point number

5.2.3.4.1 Dialog



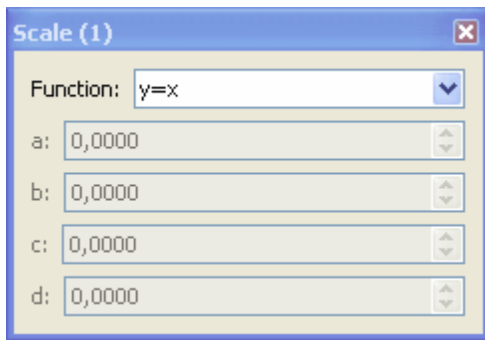
5.2.3.5 Scale



Easy scaling of values.

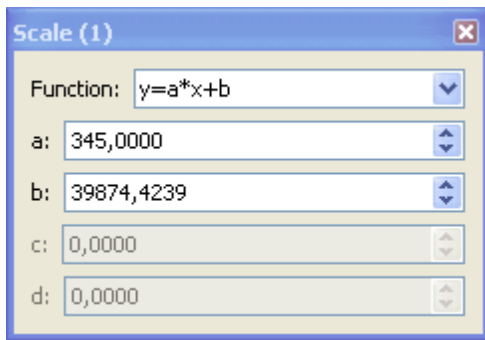
Inputs	Type	Default	Description
x	float	0	
Outputs			
y	float		see Dialog ⁶⁷

5.2.3.5.1 Dialog



Choose a function from the combo box. The default function is the identity mapping.

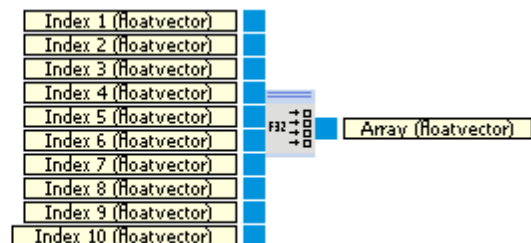
Depending on the function selected the parameters are editable or not. If you choose the function $y=a*x+b$, you can edit the parameters a and b.



The mapping here is $y(x) = 345 * x - 39874,4239$

5.2.4 Arrays

5.2.4.1 Float array composer

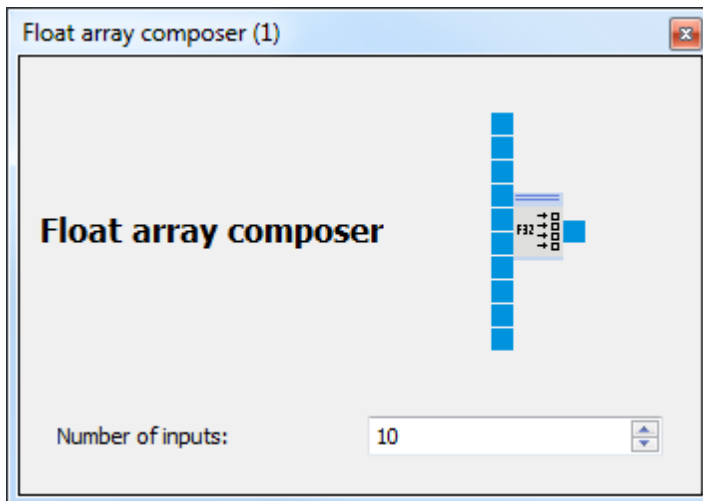


Function block library

The Float array composer creates a float array from up to 10 float values or arrays. For the type conversion from float values to float arrays see [type conversion](#) [20].

Inputs	Type	Default	Description
Index 1	float array	empty array	
...			
Index 10	float array	empty array	
Outputs			
Array	float array	empty array	(Index 1, ..., Index 10)

5.2.4.1.1 Dialog



5.2.4.2 Float array decomposer



The Float array decomposer extracts a sub array from a float array.

Inputs	Type	Default	Description

Array	float array	empty array	The array decompose.
Start index	int	1	The value at position Start index of the array will be the first value of the resulting array.
Length	int	1	The resulting array consists of Length values beginning at the value at position Start index of the input array.
Outputs			
Sub array	float array	empty array	(Array [Start index], ..., Array [Start index + Length - 1])

5.2.4.3 Float array index access



The index access module allows access to the single values of a float array.

Inputs	Type	Default	Description
Array	float array	empty array	Array to be accessed.
Index	int	1	Index of the value to be accessed.
Outputs			
Value	float	0	Value at position Index .

5.3 Vector analysis

This category contains the basic vector analysis methods for two-dimensional vectors.

5.3.1 Vector operations

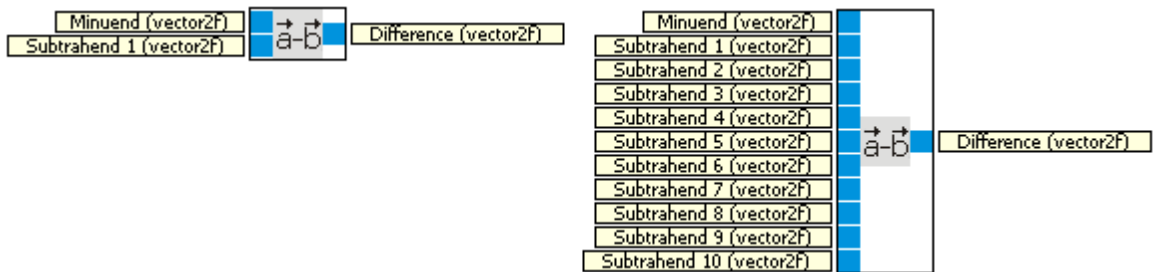
5.3.1.1 Dot product



Gives the scalar product (or dot product) of two vectors. See also http://en.wikipedia.org/wiki/Scalar_product.

Inputs	Type	Default	Description
Vector 1	vector2f	(0, 0)	
Vector 2	vector2f	(0, 0)	
Outputs			
Product	float		

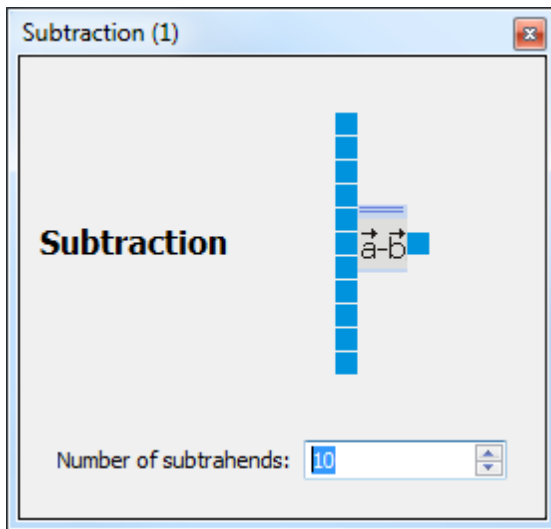
5.3.1.2 Subtraction



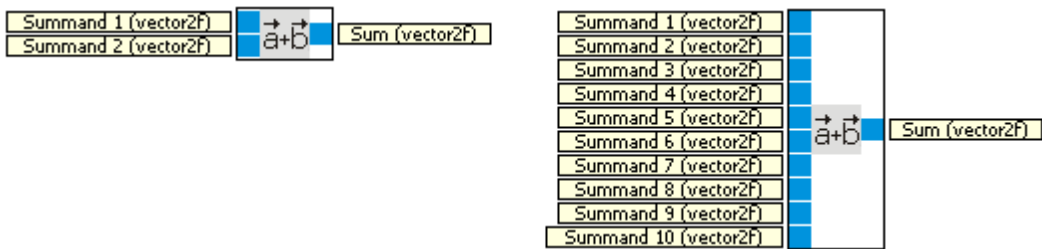
The Subtraction function block calculates the difference between the minuend and up to 10 subtrahends. See also http://en.wikipedia.org/wiki/Vector_addition#Vector_addition_and_subtraction.

Inputs	Type	Default	Description
Minuend	vector2f	(0, 0)	
Subtrahend 1	vector2f	(0, 0)	
...			
Subtrahend 10	vector2f	(0, 0)	
Outputs			
Difference	vector2f		Minuend - "Subtrahend 1" - "Subtrahend 2" - ... - "Subtrahend 10"

5.3.1.2.1 Dialog



5.3.1.3 Addition

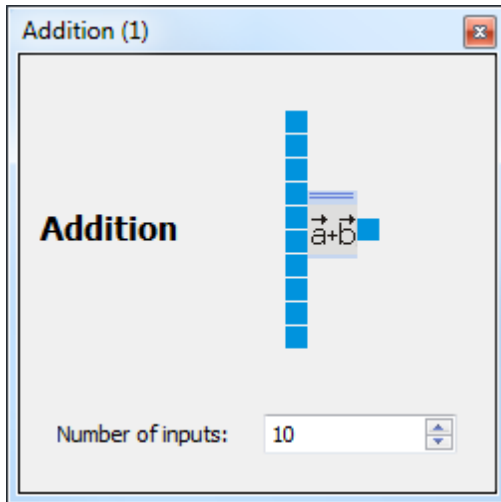


The Vector-Addition function block adds up to 10 summands. See also http://en.wikipedia.org/wiki/Vector_addition#Vector_addition_and_subtraction.

Inputs	Type	Default	Description
Summand 1	vector2f	(0, 0)	
...			
Summand 10	vector2f	(0, 0)	
Outputs			
Sum	vector2f		"Summand 1" + "Summand 2" + ... + "Summand 10"

Function block library

5.3.1.3.1 Dialog



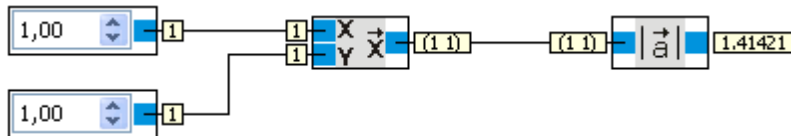
5.3.1.4 Norm



Calculates the Euclidean norm of the input vector. See also http://en.wikipedia.org/wiki/Vector_norm.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Outputs			
Norm	float		

5.3.1.4.1 Example



The norm of vector (1, 1) is square root of 1+1 = 1.41421....

5.3.2 Element operations

5.3.2.1 Division



Per element division of Vector by Divisor.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Divisor	float	1	
Outputs			
Result	vector2f		Vector = (x0, x1) Result = (x0 / Divisor, x1 / Divisor)

5.3.2.2 Subtraction



Per element subtraction of Minuend from Vector.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Minuend	float	0	
Outputs			
Result	vector2f		Vector = (x0, x1) Result = (x0 - Minuend, x1 - Minuend)

5.3.2.3 Addition



Per element addition of Summand to Vector.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Summand	float	0	
Outputs			
Result	vector2f		Vector = (x0, x1) Result = (Summand + x0, Summand + x1)

5.3.2.4 Multiplication



Per element multiplication of vector by factor.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Factor	float	1	
Outputs			
Result	vector2f		Vector = (x0, x1) Result = (Factor * x0, Factor * x1)

5.3.3 Transformations

5.3.3.1 Vector to Polar



Split up Vector into its polar components. See also http://en.wikipedia.org/wiki/Polar_coordinate_system.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Outputs			
Length	float		The length (norm) of Vector
Phi	float		The angle in degrees between Vector and the x-axis.

5.3.3.2 Vector to Cartesian



Split up Vector into its cartesian components. See also http://en.wikipedia.org/wiki/Cartesian_coordinate_system.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Outputs			
x	float		x component of Vector
y	float		y component of Vector

5.3.3.3 Polar to Vector



Create a vector from its length and orientation.

Inputs	Type	Default	Description
Length	float	0	Length (norm) of the Vector.
Phi	float	0	Angle in degrees between Vector and the x-axis.
Outputs			
Vector	vector2f		Vector with length Length and orientation Phi .

5.3.3.4 Cartesian to Vector



Create a vector from its cartesian components.

Inputs	Type	Default	Description
x	float	0	x component.
y	float	0	y component.
Outputs			
Vector	vector2f		Vector (x , y).

5.3.3.5 Rotate

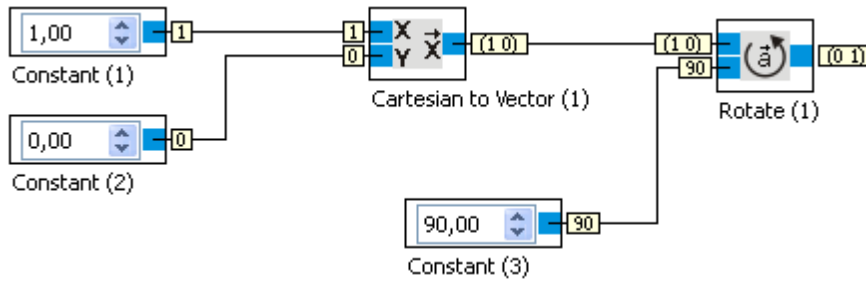


Rotates the vector by the specified value in degrees.

Inputs	Type	Default	Description
Vector	vector2f	(0, 0)	
Phi	float	0	Rotation angle
Outputs			

Result	vecto r2f	Vector rotated by Phi.
--------	--------------	-------------------------------

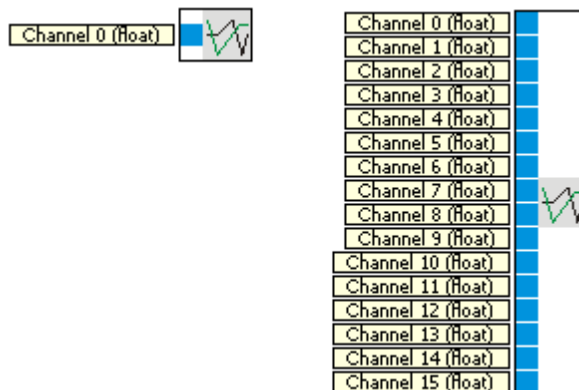
5.3.3.5.1 Example



5.4 Display

This category contains the function blocks for the visualization of data.

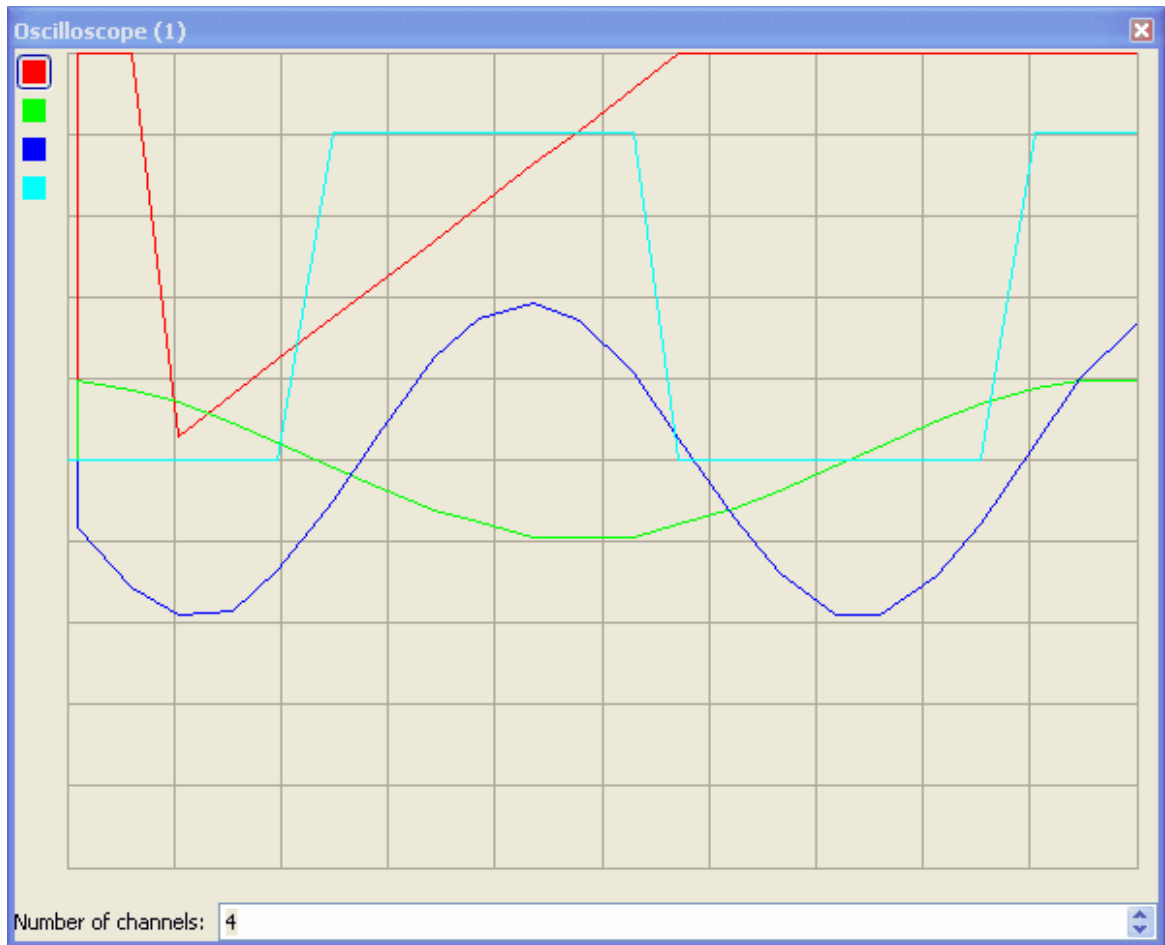
5.4.1 Oscilloscope



The Oscilloscope is used to visualize up to 16 channels.

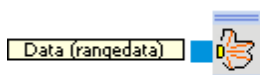
Inputs	Type	Default	Description
Channel 0	float	0	
Channel 1	float	0	
...			
Channel 16	float	0	

5.4.1.1 Dialog



The dialog visualizes the signals on the channels. For every channel, settings can be changed, like amplification, e.g. It is also possible to deactivate single channels.

5.4.2 Laser range finder data display

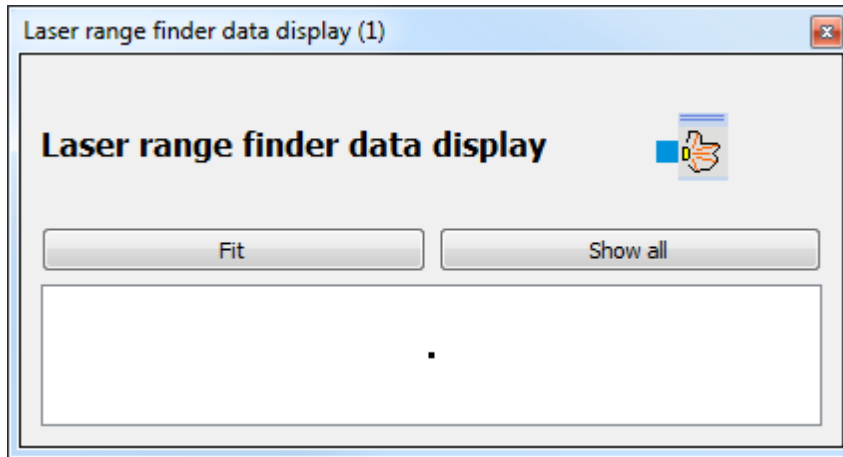


The Laser range finder data displays the data from a optional laser-scanner

Input	Typ	Description
-------	-----	-------------

Data	laser range data	
------	------------------	--

5.4.2.1 Dialog



5.5 Image processing

This category contains function blocks for image processing.

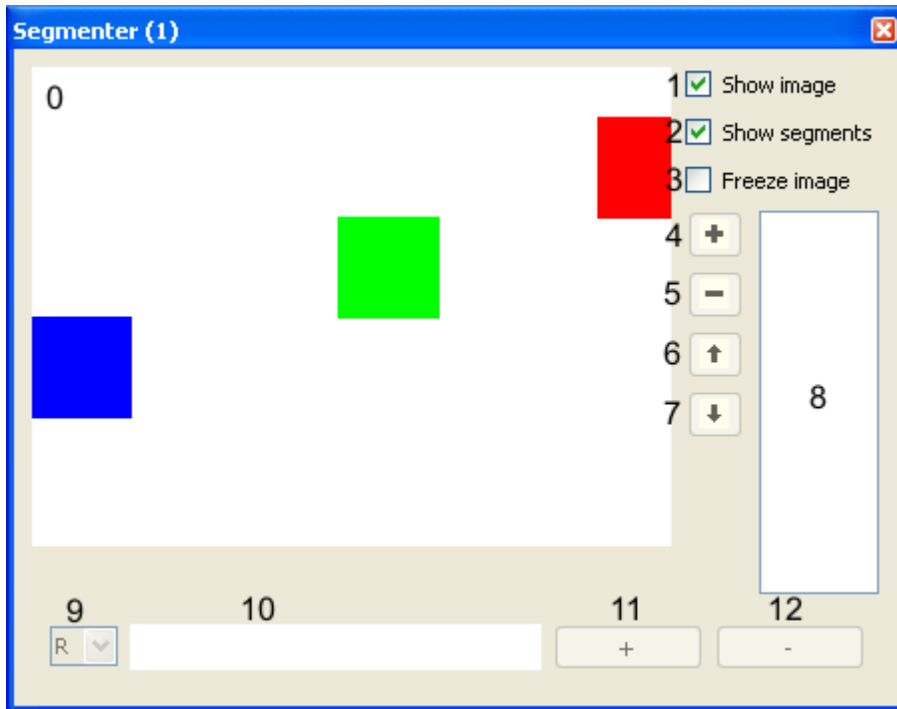
5.5.1 Segmenter



The Segmenter function block partitions the input image in multiple segments. The output image contains a list of segments found.

Inputs	Type	Default	Description
Input	image		Input image
Outputs			
Output	image		Output image augmented with the list of segments found

5.5.1.1 Dialog



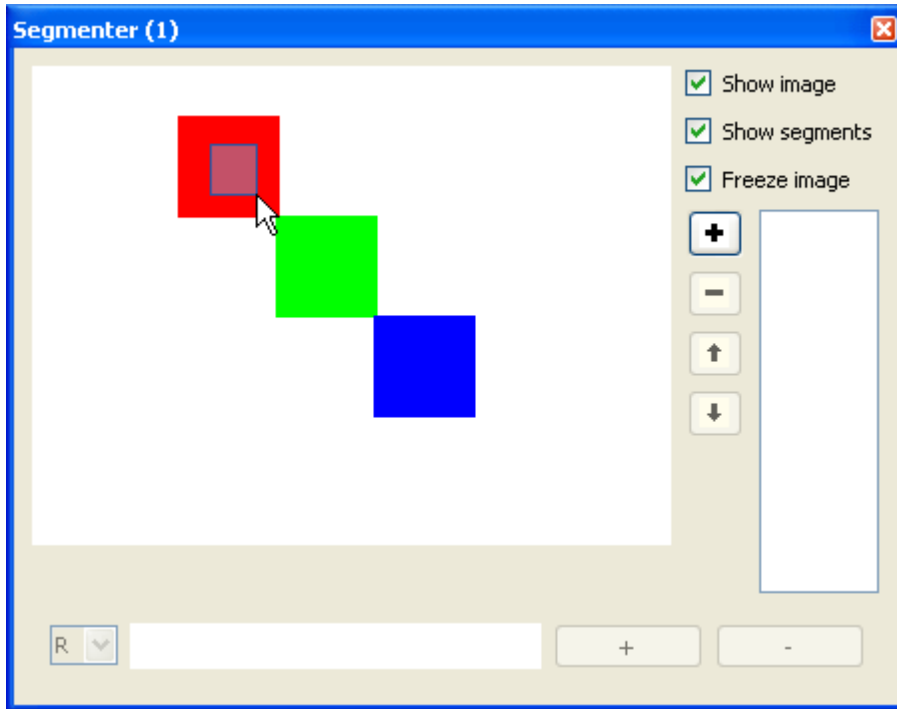
Button Description

/

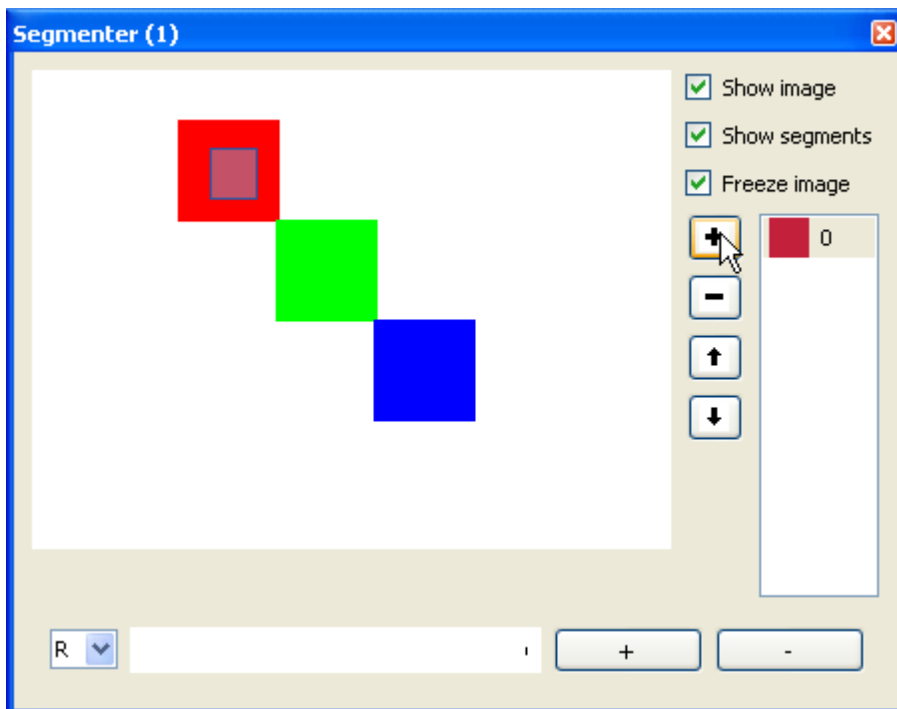
Display

- 0 Display of the input image and of segments found
- 1 When checked the input image is displayed
- 2 When checked found segments are display
- 3 When checked the current input image is hold
- 4 Add the current selection in the input image to the list of segments
- 5 Delete a segment
- 6 Move segment up
- 7 Move segment down
- 8 List of segments
- 9 Selector of the color channel for segment optimization
- 10 Display of values within the selected channel of the currently active segment
- 11 Close gaps within the values of the selected channel
- 12 Thin out values of the selected channel

To recognize the red square as a single segment, mark a region within the red square with the mouse.

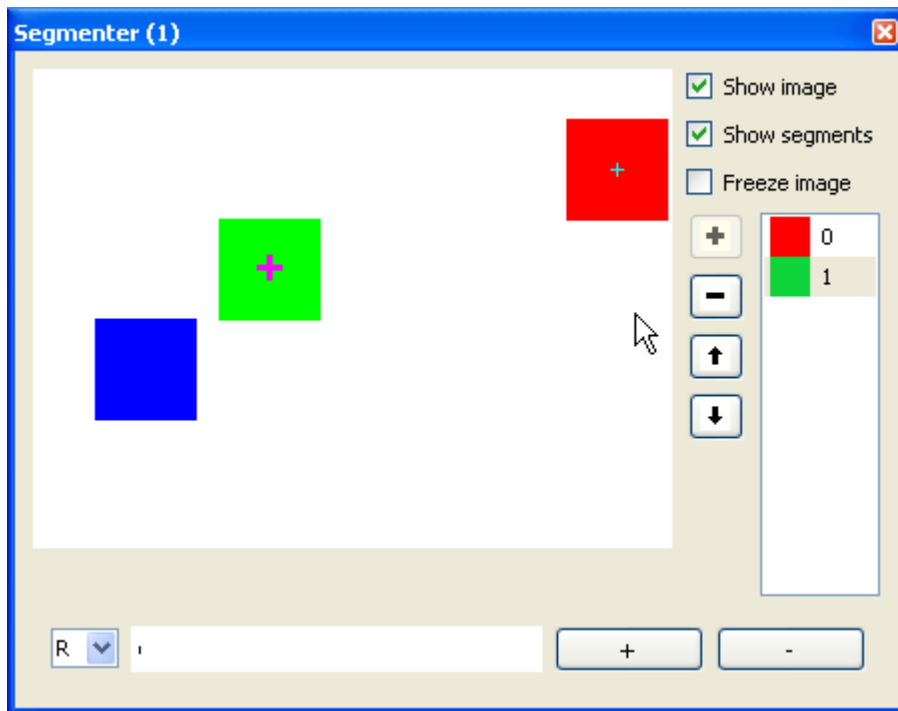


Click onto the + (button 4) to add your selection to the list of segments.



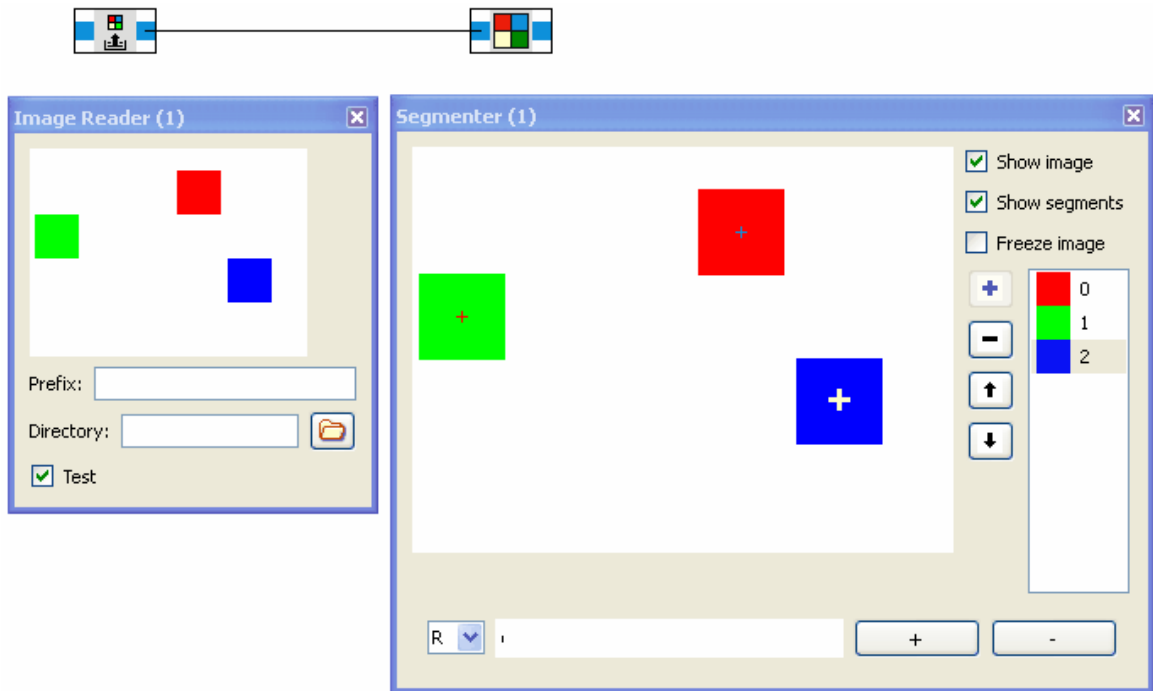
Function block library

The center of gravity of the segment is displayed with a cross. When the image is changing (deactivate the Freeze image checkbox) the center of gravity moves with the red square. Now repeat the procedure to add the green square to the list of segments.



Now there are two segments within the list of segments. The currently selected segment is marked with a bold cross.

5.5.1.2 Example



The [image reader](#)^[118] operates in test mode and generates a sequence of test images. The segmenter searches for connected regions within the input image that fit the colors in the list of segments. The center of gravity of the segments found is shown.

5.5.2 Segment Extractor

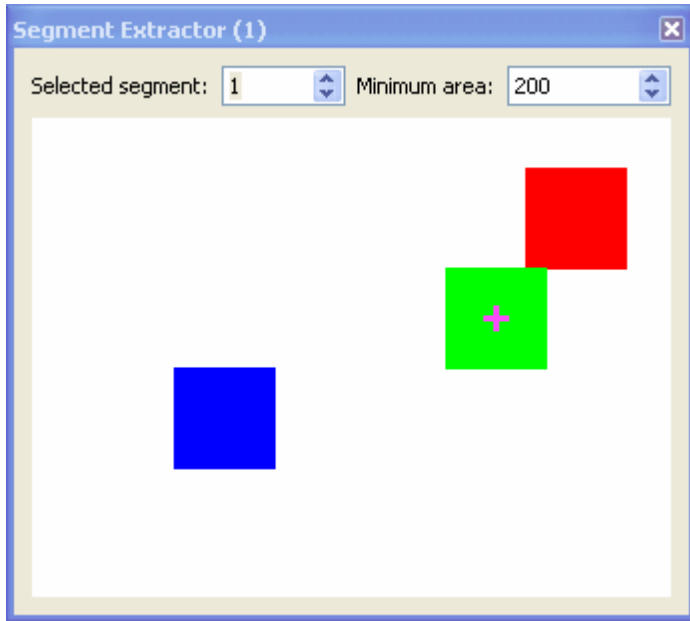


Get the position and size of a segment from an image augmented with a list of segments by the [Segmenter](#)^[79] function block.

Inputs	Type	Default	Description
Input	image		Augmented image
Selected segment	int	0	Number of the segment information is queried from.
Minimum area	int	200	The Segment found output will become true only if the segment contains at least number of pixel given here.

Outputs			
x	int		x-coordinate of the center of gravity of the segment found. If no segment is found x=0.
y	int		y-coordinate of the center of gravity of the segment found. If no segment is found y=0.
Area	int		Number of pixel within the segment. If no segment is found Area = 0.
Segment found	bool		True if the segment is found. False otherwise.

5.5.2.1 Dialog

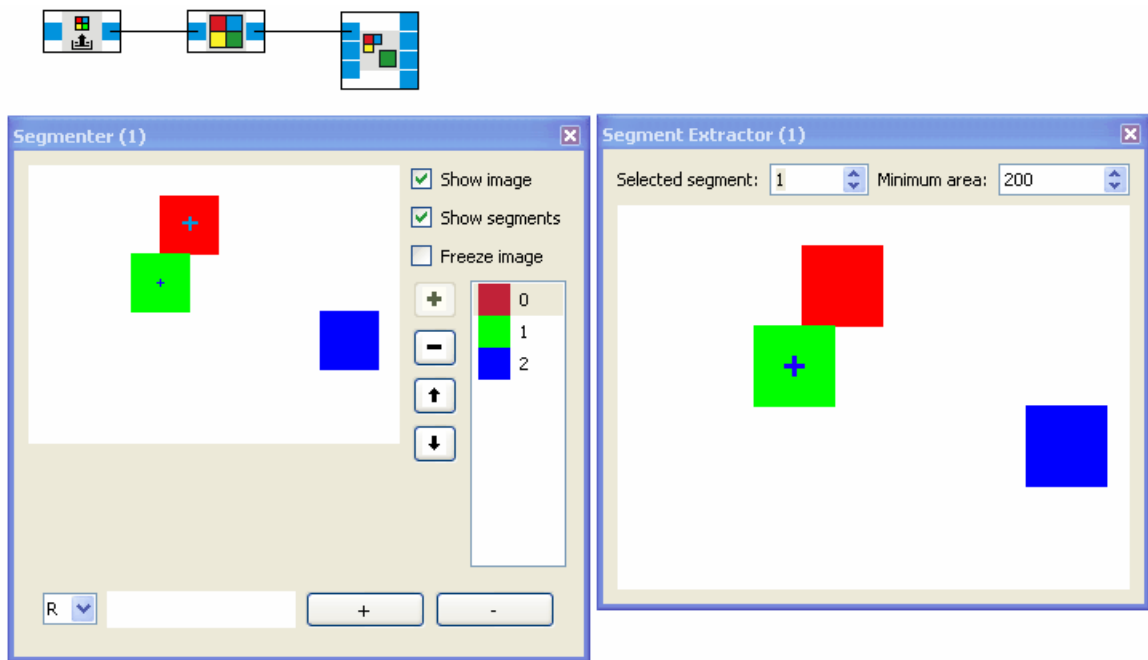


Selected segment The spinbox is editable if the input connector "Selected segment" is not connected. The segment number to search for.

Minimum area The spinbox is editable if the input connector "Minimum area" is not connected. The minimum number of pixel the segment must contain.

Shows the segments within the input image. The selected segment is marked by a cross (if the segment is found).

5.5.2.2 Example



The image reader creates a test sequence with three colored squares. The segmenter searches the image for red, green and blue regions. The segment extractor looks for the segment with number 1 (the green segment) and marks its center of gravity with a cross.

5.5.3 Line Detector

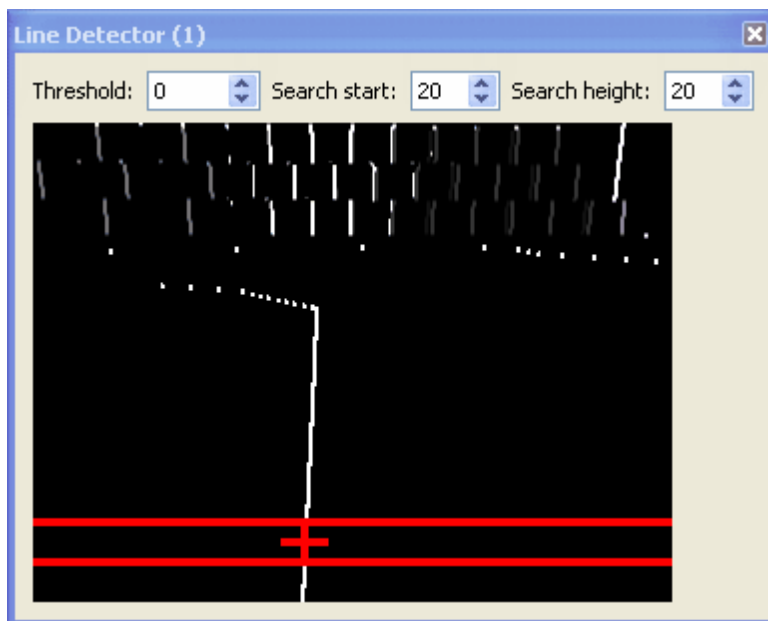


Searches for a line in the input image.

Inputs	Type	Default	Description
Input	image		Input image
Threshold	int	0	The threshold is defines the sensitivity of the line detection algorithm to discontinuities within the image. To cancel noises choose a higher threshold. Range: [0, 255]
Search start	int	20	The algorithm starts searching for a line starting at "Search start" from the bottom.

Search height	int	20	The image is searched from the bottom up for the detection of edges. The limit value defines the number of lines the image is searched starting at the bottom plus "Search start" in order to detected a segment in the form of a line.
Outputs			
x	int		x-position of the line located at the bottom edge of the image
Line found	bool		True if a line is found. False otherwise.

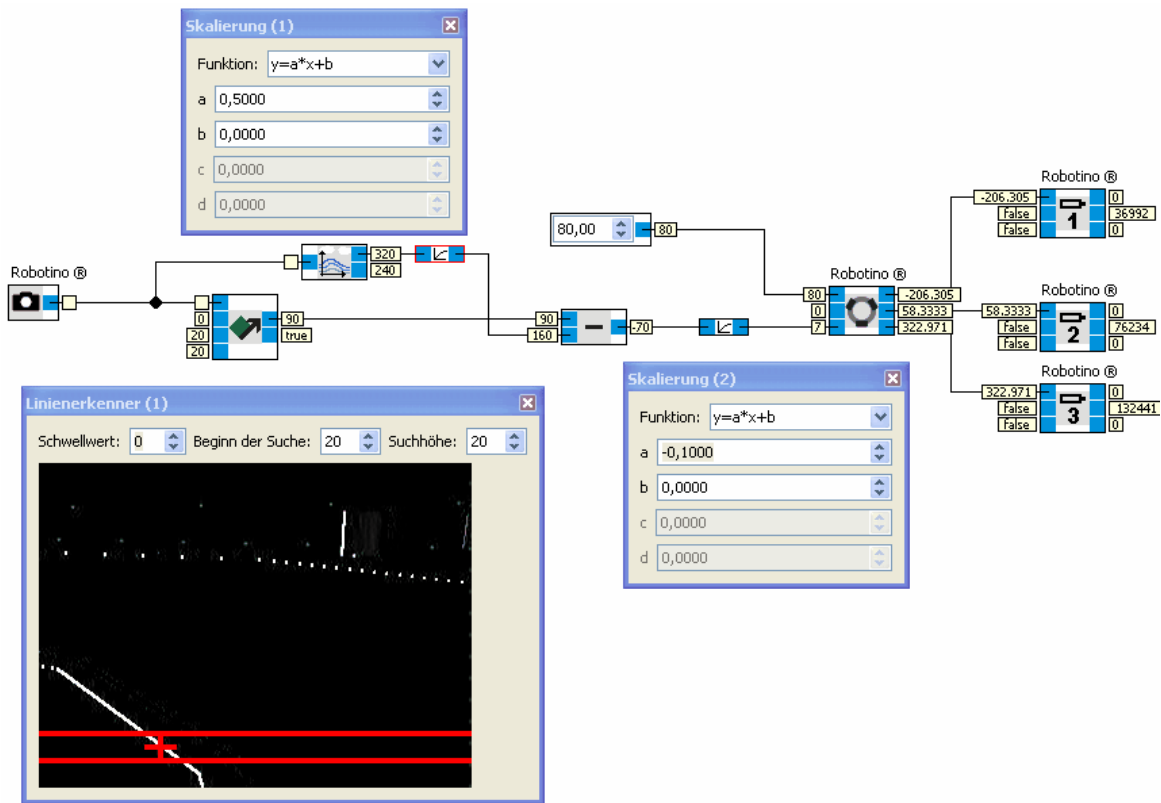
5.5.3.1 Dialog



The area which is search for a line is marked by the horizontal red lines. The bottom line marks the "Search start". The top line marks "Search start" + "Search height".

The red + marks the dark to light edge of the line seen from left to right.

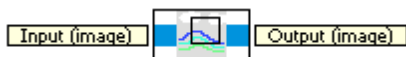
5.5.3.2 Example



The image from Robotino's camera (here from the Robotino Simulator) is used as input for the line detector. We use the [Image Information](#)^[89] function block to map the x position of the line from the range [0, image width] to [-image width/2, image width/2] which is in our case [-160,160]. The [scale](#)^[66] function block is used to switch the sign and to scale the output of the [subtraction](#)^[56] function block.

The value can be used directly to rotate Robotino so that Robotino turn right if the line is to its right and turns left if the line is to its left. With a constant forward velocity Robotino follows the line.

5.5.4 ROI



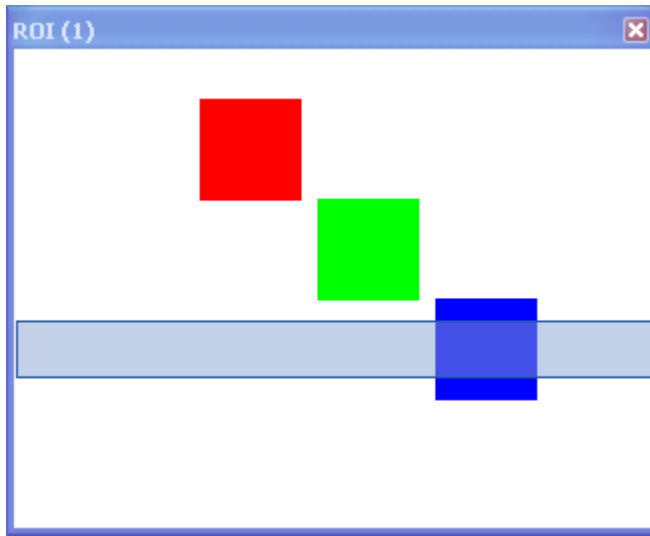
Select an interesting region inside the input image (Region Of Interest, ROI).

Inputs	Type	Default	Description
		t	

Function block library

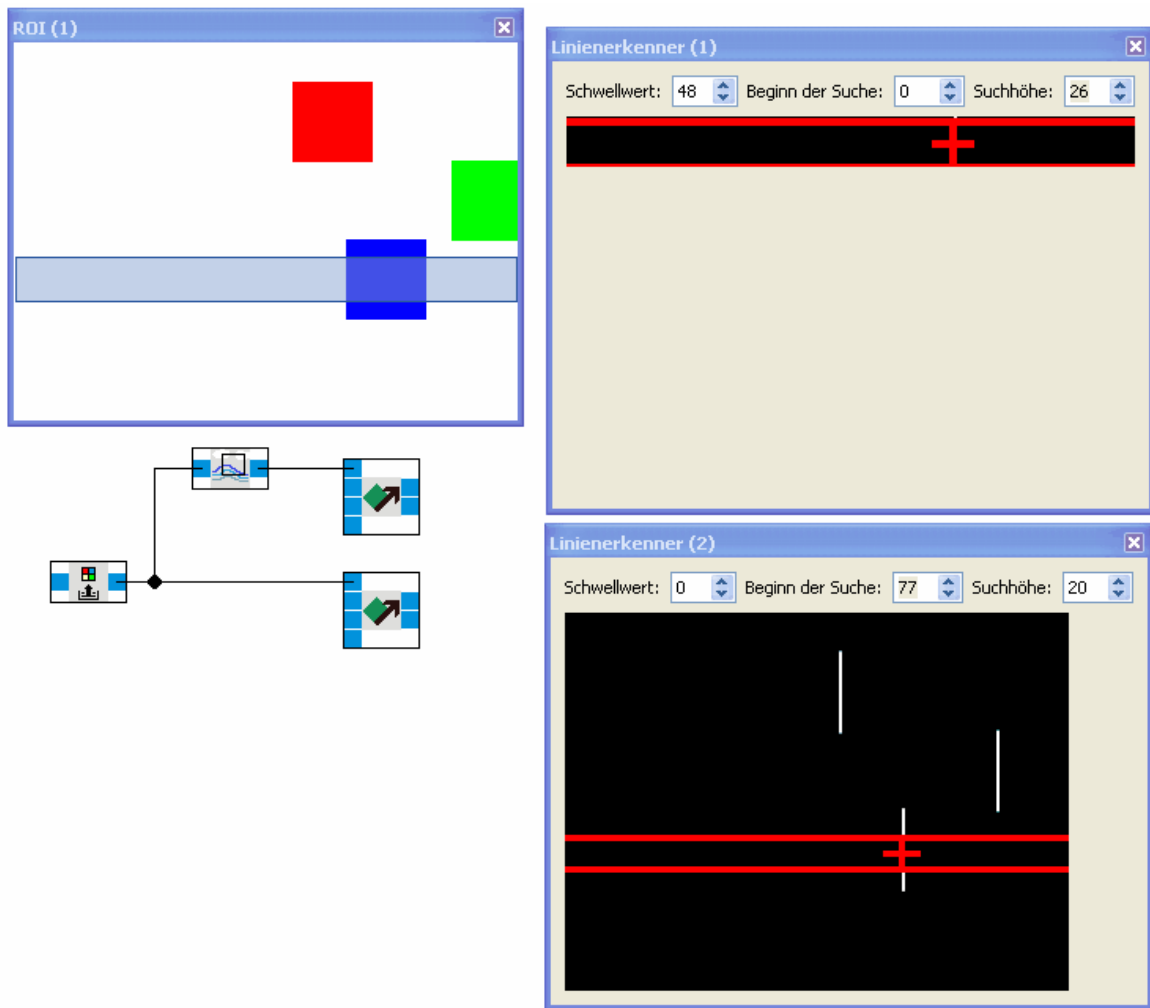
Input	image		Input image
Outputs			
Output	image		The output image is augmented with the ROI information. Later image processing takes place inside the ROI only.

5.5.4.1 Dialog



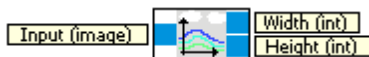
The region of interest can be marked with the mouse.

5.5.4.2 Example



The Image reader generates a test sequence of images. The bottom Line Detector uses the whole image while the upper Line Detector searches the ROI only.

5.5.5 Image Information

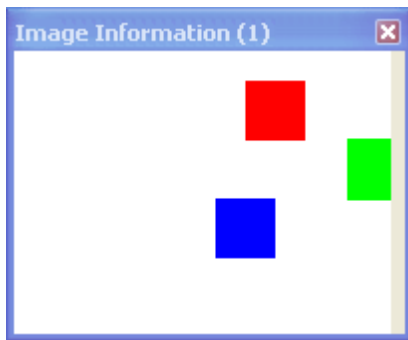


Get the width and height of the input image.

Inputs	Type	Default	Description
Input	image		Input image

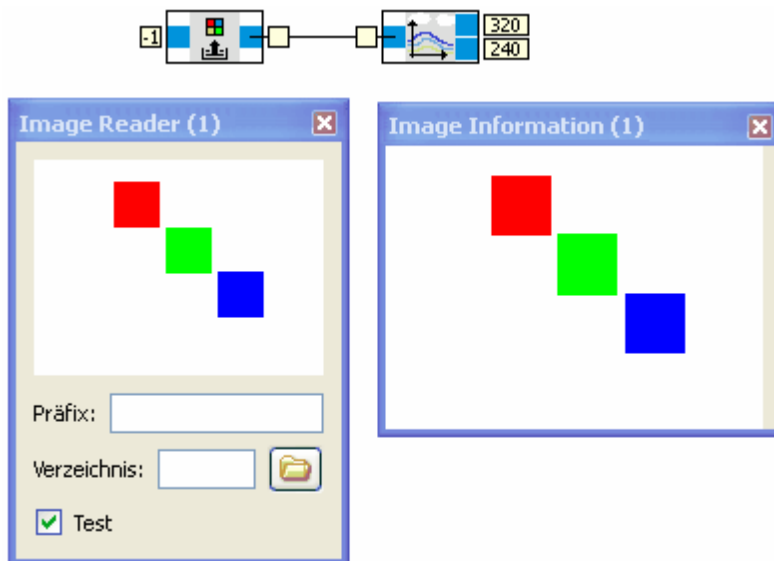
Outputs			
Breite	int		Image width in pixel.
Höhe	int		Image height in pixel.

5.5.5.1 Dialog



The dialog shows the input image.

5.5.5.2 Example



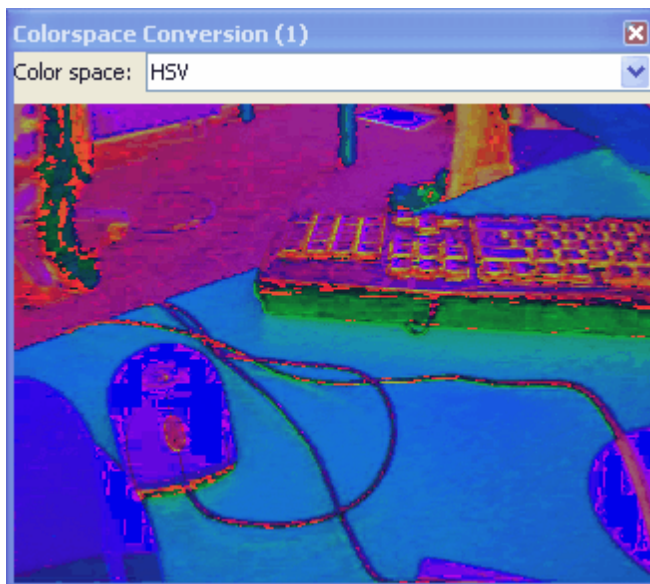
The images of the test sequence generated by the image reader have a resolution of 320 x 240 Pixel.

5.5.6 Colourspace conversion



Inputs	Type	Default	Description
Input	image		Input image
Outputs			
Output	image		Converted image

5.5.6.1 Dialog

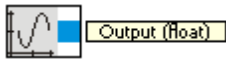


In the colourspace conversion dialog the target color space can be selected.

5.6 Generators

This category contains numerous function blocks to create signals.

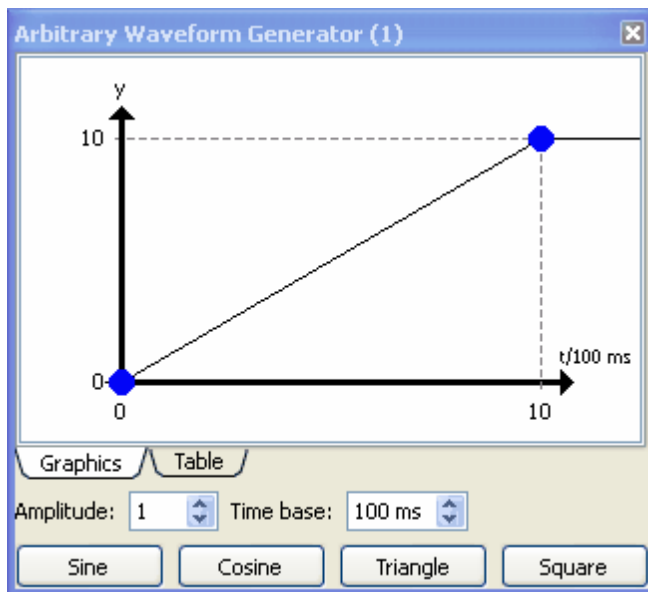
5.6.1 Arbitrary waveform generator



Generation of adjustable waveforms. See also http://en.wikipedia.org/wiki/Arbitrary_waveform_generator.

Inputs	Type	Default	Description
		t	
Outputs			
Output	float		The generated signal.

5.6.1.1 Dialog



The upper part of the dialog is similar to the dialog known from the [Transfer Function](#) ^[61].

In addition the Arbitrary waveform generator has the following parameters and buttons:

Amplitude The output of the generator is multiplied by this value.

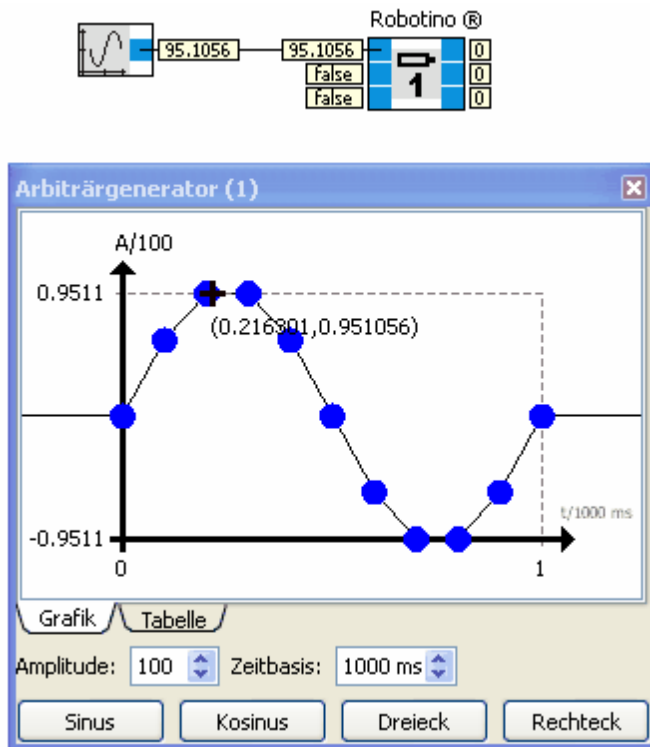
Time base The unit of the x-axis. In the current example with a time base of 100ms the value 10 is reached after 1s.

Sine Generates interpolation point approximating a sine wave.

Cosine Generates interpolation point approximating a cosine wave.

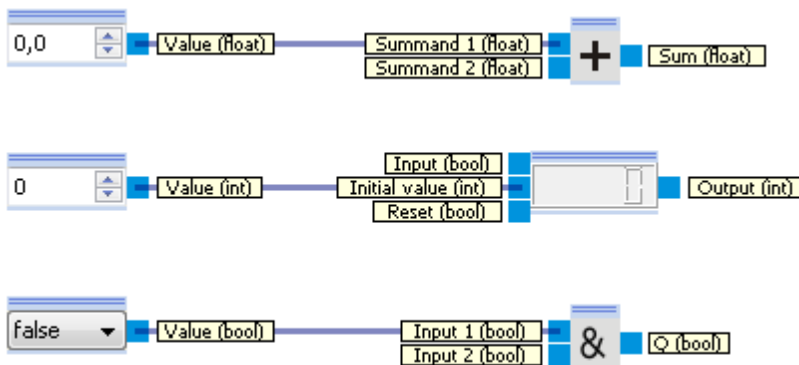
- Triangle Generates interpolation point approximating a triangle wave.
- Square Generates interpolation point approximating a square puls wave.

5.6.1.2 Example



Robotino's Motor 1 rotates driven by a sine waveform.

5.6.2 Constant



Generation of a constant value. The type of the constant and also the graphical display changes with the type of the connected input connector.

The input of the value can be performed directly within the program.

Inputs	Type	Default	Description
Outputs			
Value	float, int, bool	0 false	/ The value displayed.

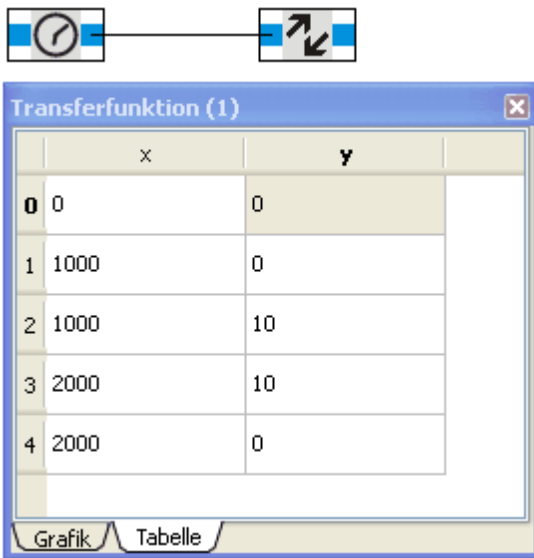
5.6.3 Timer



Measures the time in milliseconds since program start. If reset is true, the measurement is restarted.

Inputs	Type	Default	Description
Reset	bool	false	If true, the measurement is restarted.
Outputs			
Time	float		Time in milliseconds since program start or since Reset changes from true to false.

5.6.3.1 Example



Timer and [Transfer Function](#) ⁶¹⁾ generate a puls of amplitude 10 1s after program start.

5.6.4 Random generator



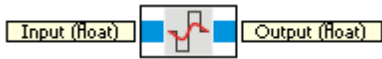
The random generator creates random numbers within a specific range.

Inputs	Type	Default	Description
Maximum	float	1	Upper bound of the range.
Minimum	float	0	Lower bound of the range.
Outputs			
Value	float	0	Random number between Minimum and Maximum .

5.7 Filter

This category contains function blocks for filtering and smoothing of signals.

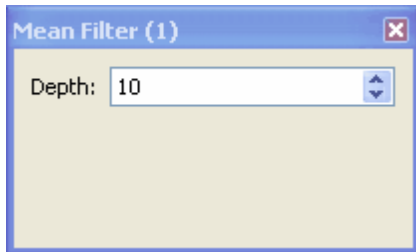
5.7.1 Mean filter



Calculates the mean of the input value for up to 1000 steps.

Inputs	Type	Default	Description
Input	float	0	Input signal
Outputs			
Output	float		Mean value

5.7.1.1 Dialog

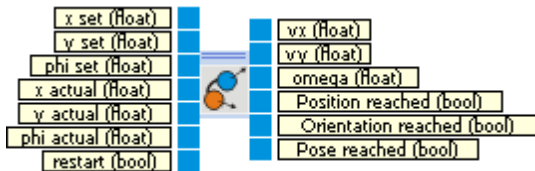


Depth is the number of previous time steps which are taken to calculate the mean value.

5.8 Navigation

This category comprises function blocks used for navigation.

5.8.1 Position Driver



The position driver is used to drive Robotino to a given position.

The position driver generates velocity and angular velocity set values to drive Robotino from the actual to the set position.

Inputs	Type	Unit	Description
x set	float	mm	x coordinate of the set position in the global coordinate system.
y set	float	mm	y coordinate of the set position in the global coordinate system.
phi set	float	deg	phi angle of the set position in the global coordinate system.
x actual	float	mm	x coordinate of the actual position in the global coordinate system.
y actual	float	mm	y coordinate of the actual position in the global coordinate system.
phi actual	float	deg	phi angle of the actual position in the global coordinate system.
restart	bool		Restart movement
Outputs			
vx	float	mm/s	set velocity in x direction in Robotino's local coordinate system
vy	float	mm/s	set velocity in y direction in Robotino's local coordinate system
omega	float	deg/s	set angular velocity.
Position reached	bool		Is true if $v_x=v_y=0$, i.e. the set position is reached.
Orientation reached	bool		Is true if $\omega=0$, i.e. the set orientation is reached.
Pose reached	bool		Is true if both position and orientation are reached.

See [Movements](#)^[100]

5.8.1.1 Dialog

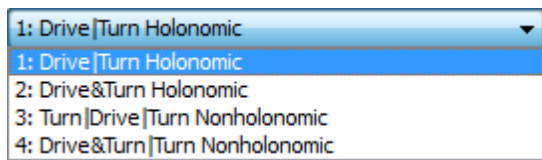
The dialog box is titled "Position Driver (1)". It features a logo with a blue 'H' and a gear. Below the logo, there are two graphs. The first graph plots velocity $v/\text{mm/s}$ on the y-axis (ranging from 0 to 300) against distance d/mm on the x-axis (ranging from 0 to 100). The second graph plots angular velocity $\omega/\text{deg/s}$ on the y-axis (ranging from 0 to 50) against distance d/deg on the x-axis (ranging from 0 to 20). Both graphs show a linear ramp starting from the origin. Below each graph are "Graphics" and "Table" tabs. At the bottom of the dialog, there is a dropdown menu showing "1: Drive|Turn Holonomic", two input fields for "Velocity ramp: 1000ms" and "Angular velocity ramp: 1000ms", and a label "Active movement: 1: Drive|Turn Holonomic".

The dialog is split into three parts.

The upper part reflects the mapping from distance to the target position d (in mm) to the driving velocity v (in mm/s).

The middle part reflects the mapping from angular distance to the target orientation d (in 1°) to angular velocity ω (in $1^\circ/s$). The angular distance is in the range $[0^\circ, 180^\circ]$. Clockwise and counter clockwise rotations are treated similar. Rotation will be performed clockwise or counter clockwise so that the angular distance is minimal.

With the ComboBox



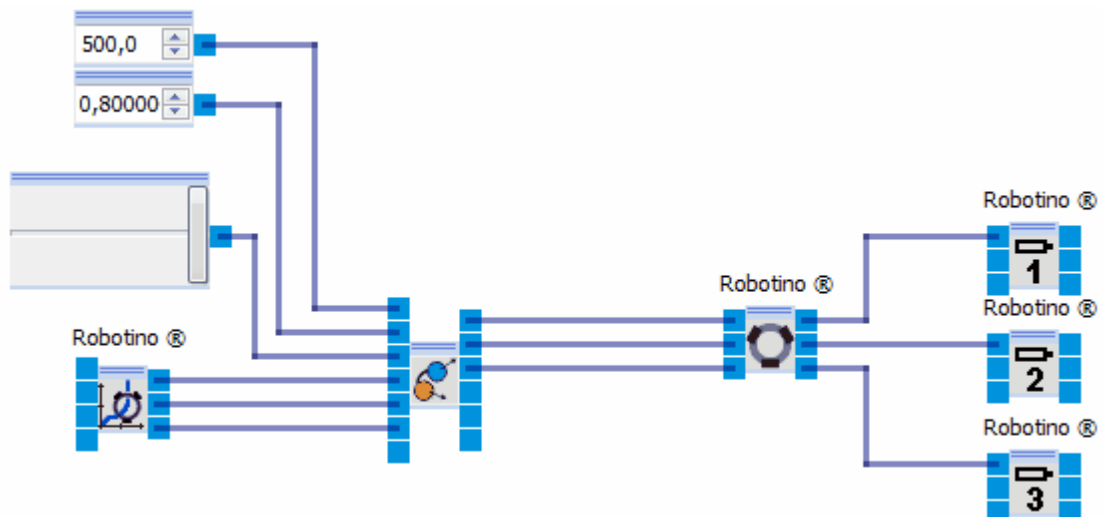
the kind of movement can be selected (see [Movements](#)^[100]). The selected movement becomes the active movement

1. at program start.
2. when the input "restart" is set true.

The velocity ramp is the time in milliseconds after which 100% of the desired velocity is reached. This avoids an abrupt jump of velocity at the beginning of the movement.

The angular velocity ramp is the time in milliseconds after which 100% of the desired angular velocity is reached. This results in a damping of the movement when a new rotation begins.

5.8.1.2 Example



5.8.1.3 Movements

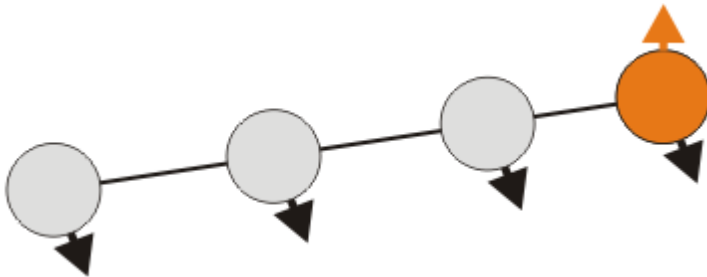
Four different kinds of movements are possible. Two of them are applicable for holonomic and nonholonomic vehicles each. As Robotino has a holonomic drive - all three degrees of freedom can be altered independently - Robotino can perform all four kinds of movement. For nonholonomic movements the output v_y is 0.

Movements start when the program starts or when the input "restart" becomes true. Effectively, in the 2nd case the movement begins when the input "restart" is reset to false.

Movement 1 - drive, turn - (holonomic)

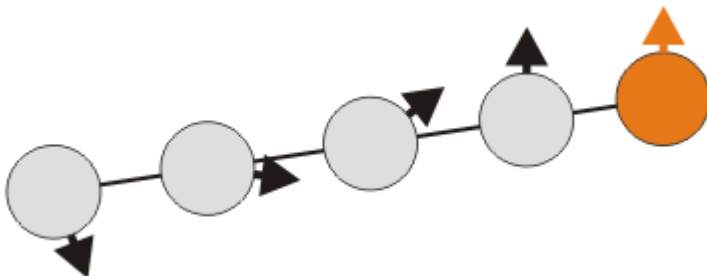
Step 1: drive to the target position keeping the orientation at the initial position

Step 2: after reaching the target position turn until the target orientation is reached



Movement 2 - drive & turn - (holonomic)

Step 1: drive and simultaneously turn to the target orientation

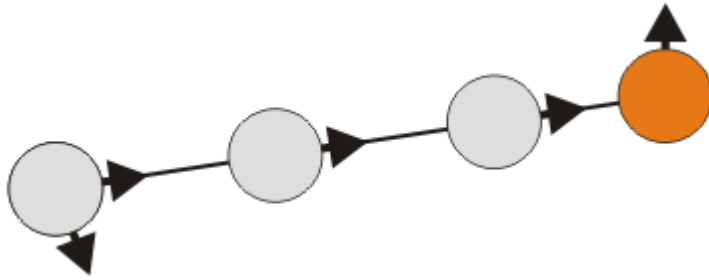


Movement 3 - turn, drive, turn - (nonholonomic)

Step 1: turn to the driving direction

Step 2: drive to the target position

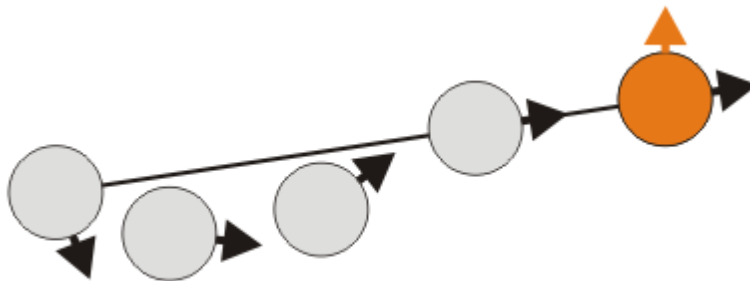
Step 3: after reaching the target position turn to the target orientation



Movement 4 - drive & turn, turn - (nonholonomic)

Step 1: Drive and turn in to driving direction

Step 2: after reaching the target position turn until the target orientation is reached



5.8.2 Constant pose



In the input box the pose is specified. Coordinates are separated by spaces characters.

Input	Resulting pose
x y phi	(x, y, phi)
x y	(x, y, invalid)
x	invalid Pose
	invalid Pose

The orientation phi is specified in degrees.

Example:

10.5 20 120

results in $x=10.5$ $y=20$ and orientation= 120°

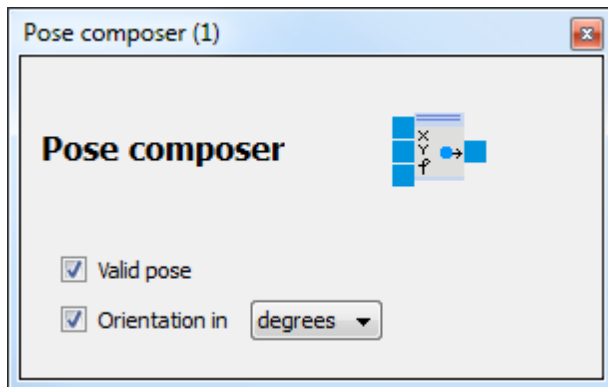
Inputs	Type	Default	Description
Outputs			
Pose	pose	invalid pose	The constant pose's value. The value for the orientation is displayed in radians at the output.

5.8.3 Pose composer



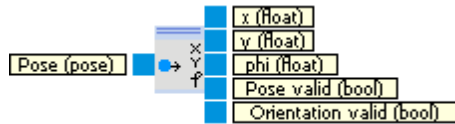
Inputs	Typ	Unit	Default	Description
x	float		0	The pose's x component.
y	float		0	The pose's y component.
phi	float	degrees	0	The pose's orientation in degrees. The unit can be switched to radians in the Dialog ^[102] .
Outputs				
Pose	pose		(0, 0, 0)	The pose (x, y, phi) composed from the single values. The value for the orientation is displayed in radians at the output.

5.8.3.1 Dialog



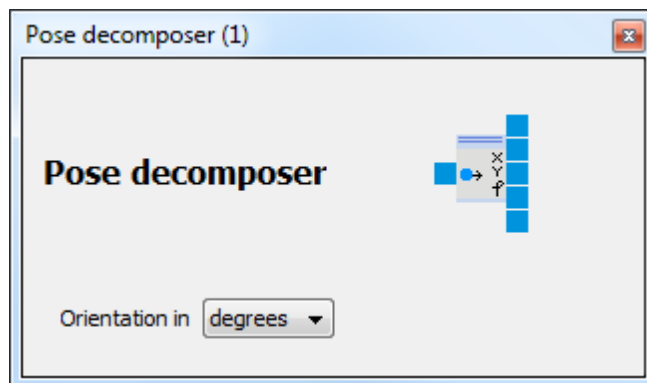
Valid pose	Specifies if the pose is valid. Invalid poses will be ignored in paths.
Orientation	Specifies if the orientation is valid and its unit (degrees or radians).

5.8.4 Pose decomposer



Inputs	Type	Unit	Default	Description
Pose	pose		(0, 0, 0)	Pose to decompose.
Outputs				
x	float		0	The pose's x component.
y	float		0	The pose's y component.
phi	float	Grad	0	The pose's orientation in degrees. The unit can be switched to radians in the Dialog ^[102] .
Pose valid	bool		false	true if the pose is valid.
Orientation valid	bool		false	true if the orientation stored in the pose is valid.

5.8.4.1 Dialog

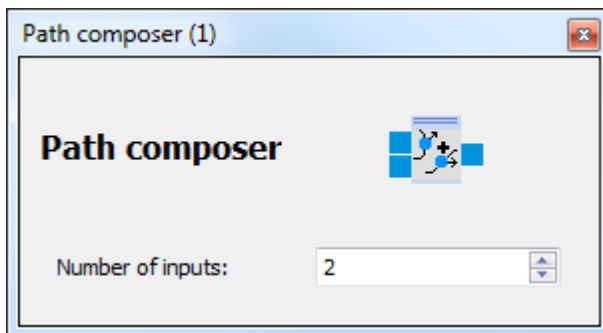


5.8.5 Path composer



Inputs	Type	Default	Description
Path 1	path	empty path	The first sub-path. A single Pose will also be accepted because pose is convertible into path. See Type conversion ^[20] .
...			
Path 20	path	empty path	The last sub-path. A single Pose will also be accepted because pose is convertible into path. See Type conversion ^[20] .
Outputs			
Path	path	empty path	The path composed from the sub-paths Path 1 + ... + Path 20

5.8.5.1 Dialog



5.8.6 Path decomposer



Cuts a subpath out of a path. A path consists of a list of poses.

Index	Pose
1	p1
2	p2
...	

N	pN
---	----

The inputs **Start** and **Length** specify the initial pose and the length of the decomposed path. **Start** must be in $[1;N]$. If **Start** < 1 , value 1 is used internally. If **Start** $>$ the length of the path, the result will be an empty path. **Length** must be in $[0;N-\mathbf{Start}+1]$. If **Length** ≤ 0 the result will be an empty path. If **Length** $> N-\mathbf{Start}+1$, the result will be the subpath starting at index **Start**.

Examples:

Path = p1, p2, p3, p4, p5, p6, p7, p8, p9, p10

Start = 3

Length = 5

Subpath = p3, p4, p5, p6, p7

Path = p1, p2, p3, p4, p5, p6, p7, p8, p9, p10

Start = 0

Length = 1

Subpath = p1

Path = p1, p2, p3, p4, p5, p6, p7, p8, p9, p10

Start = 11

Length = 1

Subpath = empty path

Path = p1, p2, p3, p4, p5, p6, p7, p8, p9, p10

Start = 1

Length = 0

Subpath = empty path

Path = p1, p2, p3, p4, p5, p6, p7, p8, p9, p10

Start = 2

Length = 20

Subpath = p2, p3, p4, p5, p6, p7, p8, p9, p10

Inputs	Type	Default	Description
Path	path	empty path	The path to decompose.
Start	int	1	The pose at index Start of the path becomes the first pose of the decomposed path.

Length	int	1	The decomposed path consists of Length poses starting at the pose at index Start of the path to decompose.
Outputs			
Subpath	path	empty path	The resulting path begins with the pose at index Start and consists of Length poses.

5.8.7 Path driver

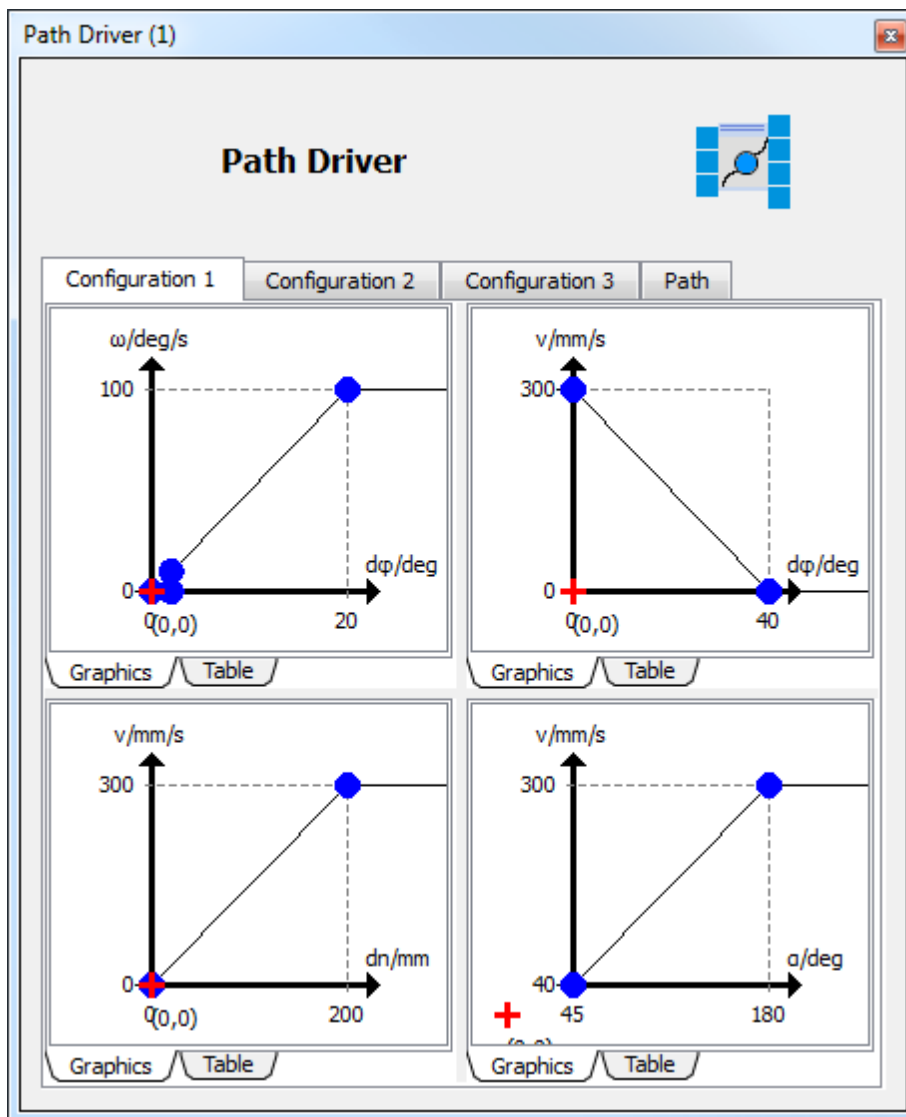


With the path driver it is possible to drive along paths.

From the path and the actual pose the velocity and the angular velocity are calculated so that Robotino drives straightforward along the path's single poses.

Inputs	Type	Unit	Default	Description
Path	path		empty path	The path to drive.
Actual pose	pose		(0, 0, 0)	The actual pose determined by odometry or SLAM.
Restart	bool		false	Restart the movement.
Outputs				
Velocity	float	mm/s		Forward velocity.
Angular velocity	float	deg/s		Angular velocity.
Position reached	bool			True, if the path is empty. Otherwise true, when the virtual point is located on the last path segment and $v(d) = 0$.
Next waypoint	pose			The next target way point.

5.8.7.1 Configuration dialog 1

**Top left**

Correlation between angular velocity and angular error $d\phi$.

Top right

Correlation between forward velocity and angular error $d\phi$.

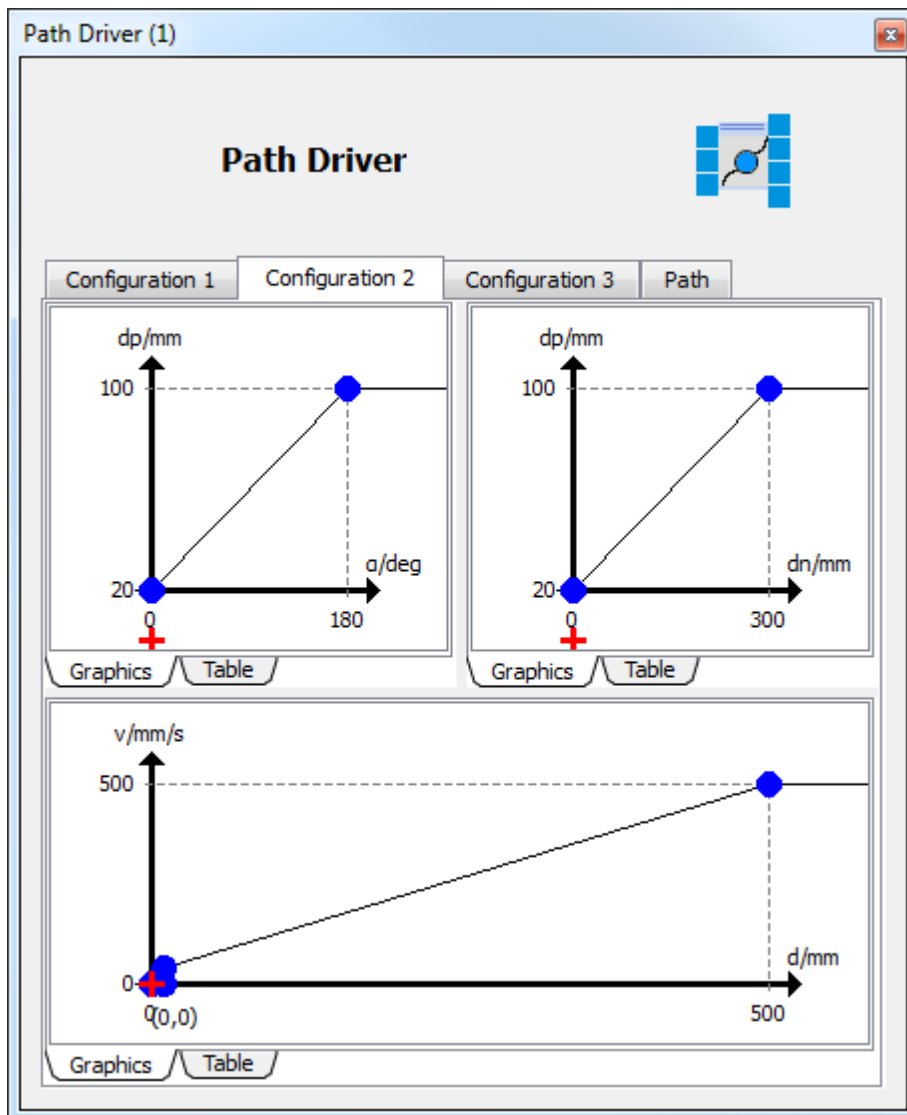
Bottom left

Correlation between forward velocity and distance to the next way point.

Bottom right

Correlation between forward velocity and angle to the next path segment.

5.8.7.2 Configuration dialog 2



Top left

Correlation between the robot's distance to the virtual way point and the angle to the next path segment.

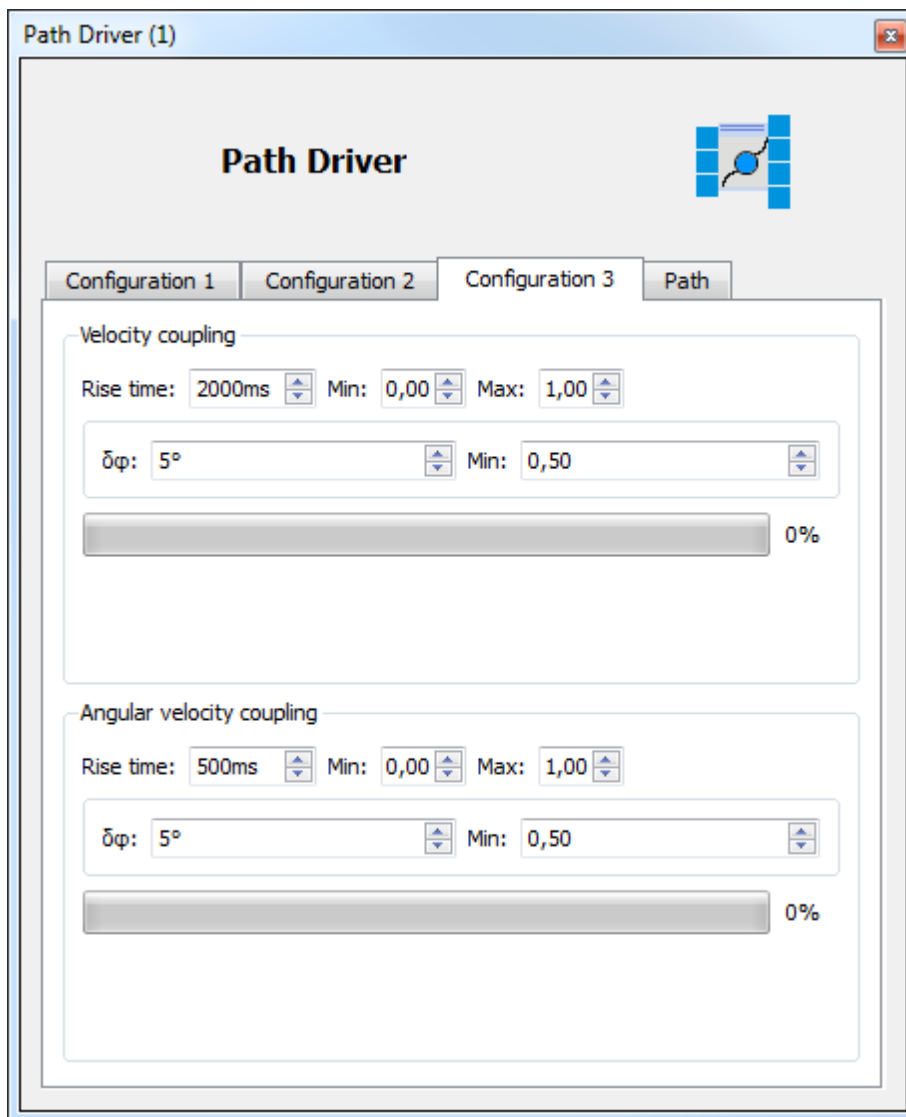
Top right

Correlation between the robot's distance to the virtual way point and the distance to the next way point.

Bottom

Correlation between forward velocity and distance to the end of the path.

5.8.7.3 Configuration dialog 3



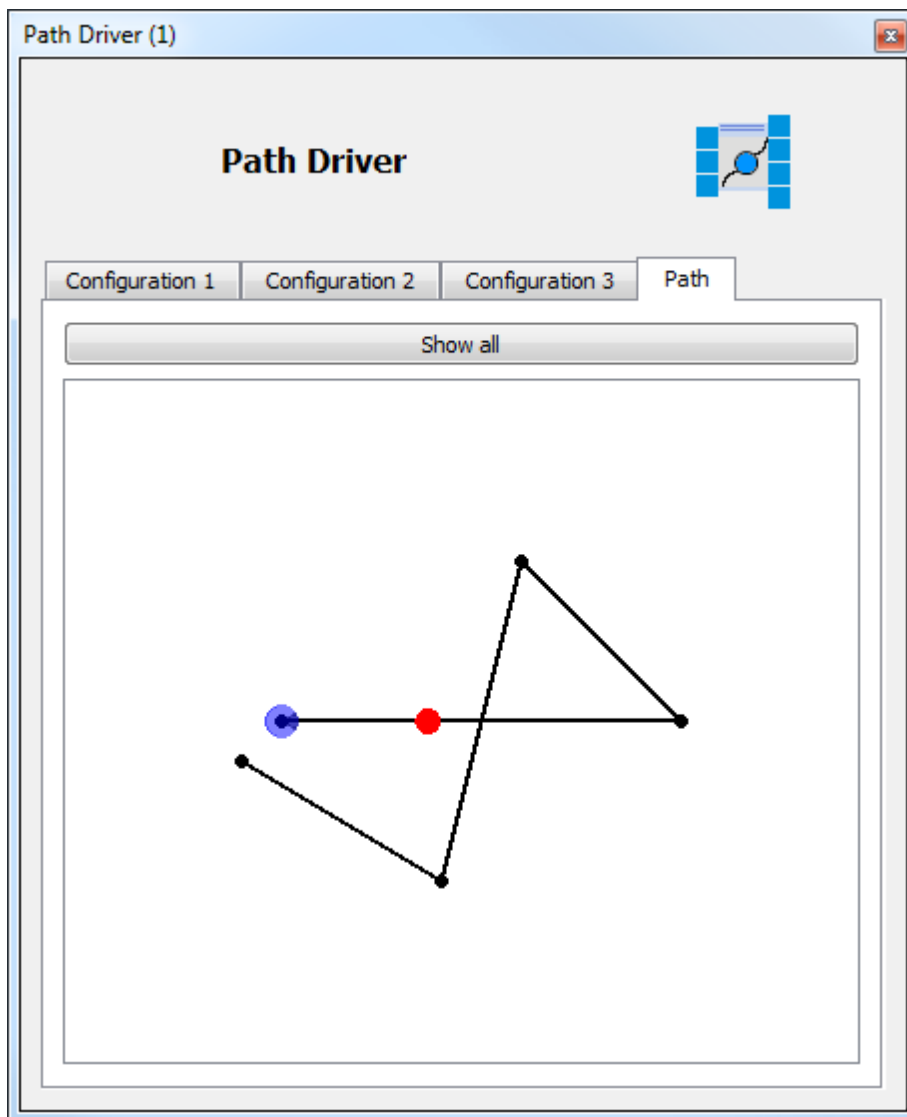
Top

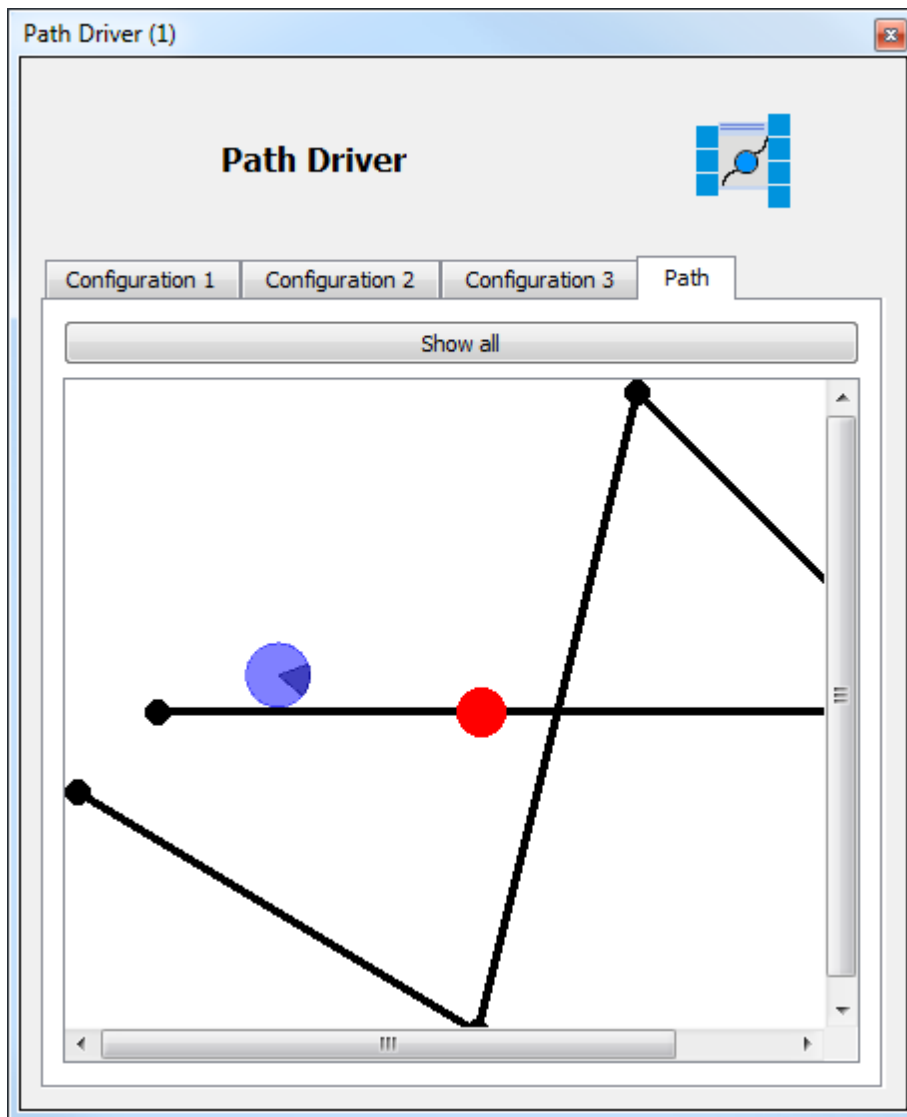
Adaption of the coupling factor between the velocity calculated due to the configuration in dialogs 1 and 2 and the real velocity.

Unten

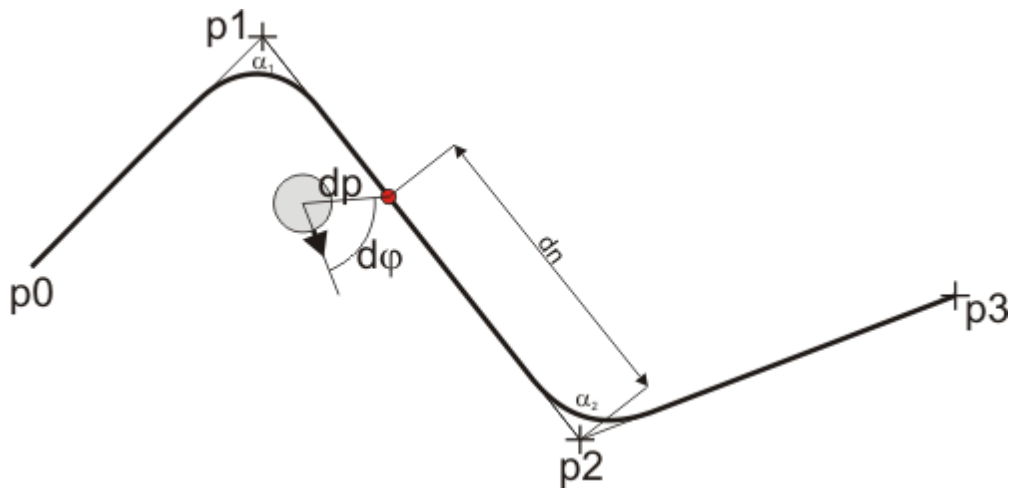
Adaption of the coupling factor between the angular velocity calculated due to the configuration in dialogs 1 and 2 and the real angular velocity.

5.8.7.4 Path view





5.8.7.5 Strategy



The path driver function block creates a path which first connects the way points straight-lined.

The robot is driven with a virtual way point (painted as a red dot in the figure above). Given the robot's current position, the virtual way point will be placed on the path that the distance between robot and virtual way point is d_p (distance virtual point). The virtual way point can only move along the path towards the path's end, i.e. if the robot moves away from the virtual way point, it remains unchanged. Due to the regulation on the virtual point, the path will be smoothed. The greater d_p is, the greater is the smoothing.

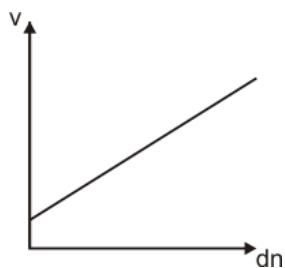
Angular velocity parameterization

The angular velocity $\omega(d\phi)$ is specified via the function block dialog dependent from the angular error $d\phi$. $d\phi$ is the angle between the robot's current orientation and the line from the robots center to the virtual way point.

Velocity parameterization

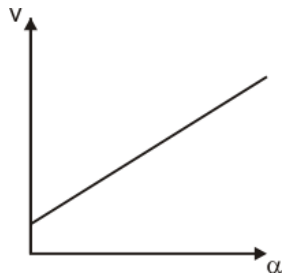
The velocity is also specified dependent from $d\phi$ and named $v(d\phi)$. So it is possible to slow down the movement if the robot is no longer oriented properly.

To be able to reduce velocity if the path has a bend, the velocity is also specified as a function $v(d_n)$ of the distance between the virtual point and the next way point. A typical curve of $v(d_n)$ is



I.e. the velocity shall decline if the robot gets closer to the way point.

But we want to slow down the robot depending on the angle α_n . α_n is the angle between the current and the next path segment. If $\alpha_n = 180^\circ$ (i.e. the path leads straight-forward through the way point) the velocity is not to be reduced. If α_n approaches 0° (a very strong bend) the robot must be slowed down strongly. Therefore the function $v(\alpha_n)$ is needed. A typical curve of $v(\alpha_n)$ looks like this



I.e. the smaller α_n is the smaller is the velocity.

These three velocity profiles $v(d_n)$, $v(d_n)$ und $v(\alpha)$ are used to calculate the overall velocity $V(d_n, \alpha)$:

$$V_p(d_n, \alpha) = \min(v(d_n), \max(v(\alpha_n), v(\alpha)))$$

Driving to the last way point

To slow down when the end of the path is reached, the velocity depending on the remaining distance to be driven is specified and called $v(d)$. The target is supposed to be reached when the velocity as a function of the remaining distance to be driven is zero.

The unsmoothed velocity results in:

$$V(d, d_n, \alpha) = \min(v(d), V_p(d_n, \alpha))$$

Smoothing of velocity and angular velocity

There are two other parameters available to smooth the movement.

The velocity coupling is the time in milliseconds that is needed for the coupling v_{CC} between the calculated velocity $V_p(d_n, \alpha)$ and the real velocity to reach the value 1.

The angular velocity coupling is the time in milliseconds that is needed for the Coupling ω_{CC} between the calculated angular velocity $\omega(d_n)$ and the real velocity ω to reach the value 1.

$$dv = v_{CC} * (V_{p_t} - V_{p_{t-1}})$$

$$\text{velocity} = V_{p_{t-1}} + dv$$

$$d\omega = \omega_{CC} * (\omega(d_n)_t - \omega(d_n)_{t-1})$$

$$\text{velocity} = \omega(d_n)_{t-1} + d\omega$$

The subscript t means the value at time t . $t-1$ means the value one time step before t .

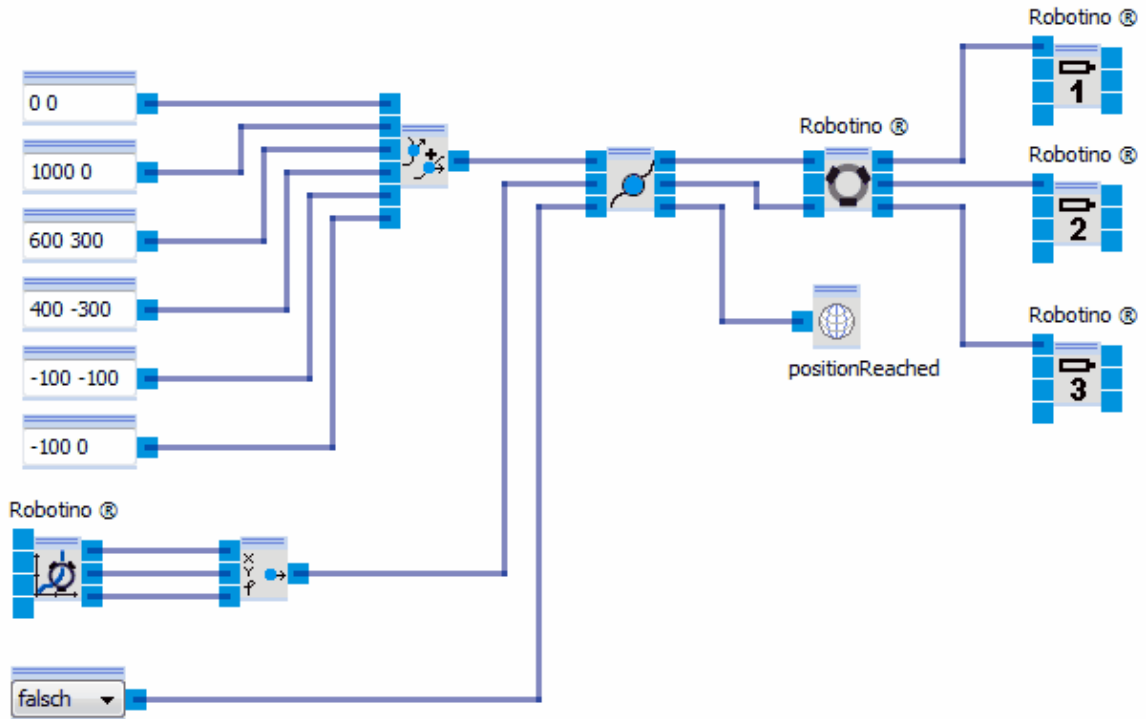
Function block library

At restart vCC is initialized with 0 and increases to 1 within the time specified by the velocity coupling.

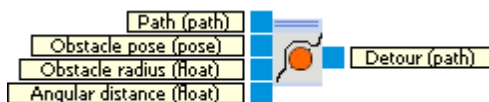
At restart omegaCC is initialized with 0 and increases to 1 within the time specified by the angular velocity coupling.

If the virtual point jumps to a new path segment, vCC and omegaCC will be reset to 0.

5.8.7.6 Example



5.8.8 Obstacle avoidance

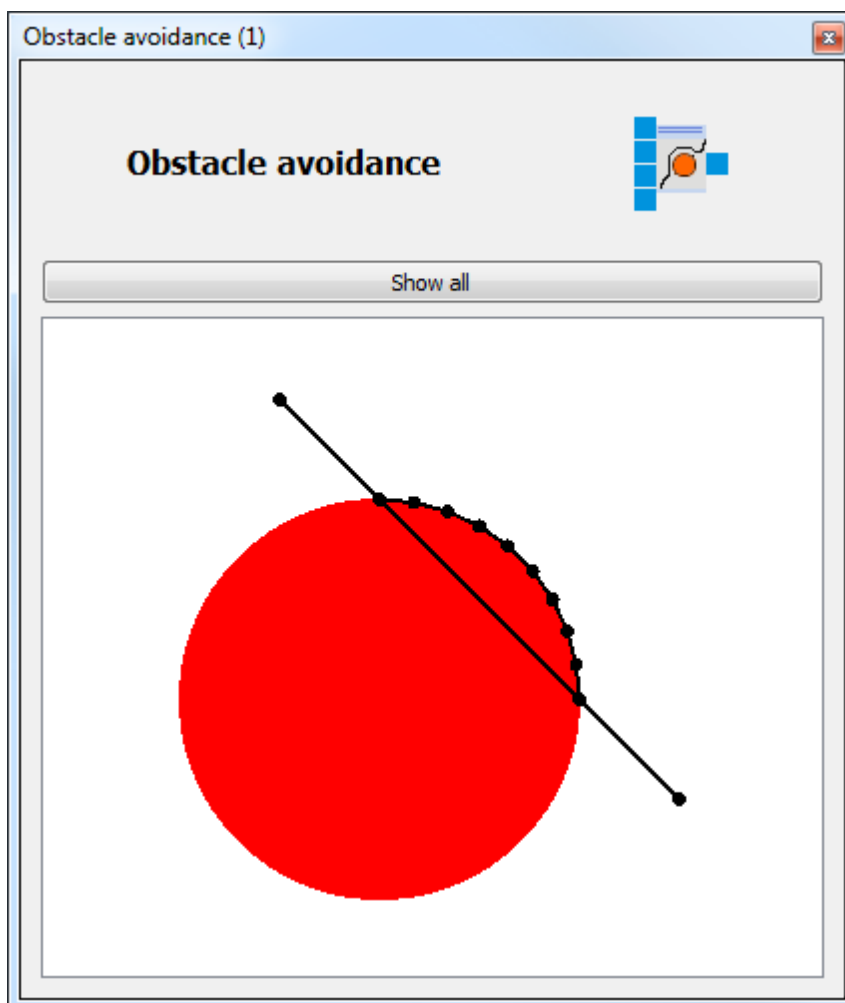


The module Obstacle avoidance calculates a detour for a path round a circular obstacle.

Inputs	Type	Unit	Default	Description
Path	path		empty path	The path to be driven.

Obstacle pose	pose		(0, 0, 0)	The position of the circular obstacle.
Obstacle radius	float	mm	100	The radius of the circular obstacle.
Angular distance	float	Grad	10	The maximum angular distance between two points of the detour round the obstacle.
Outputs				
Detour	path		empty path	Detour round the obstacle.

5.8.8.1 Dialog



The dialog shows the original path, the obstacle and the detour.

5.9 Input Devices

This category supplies the function blocks for realizing the interaction with the user.

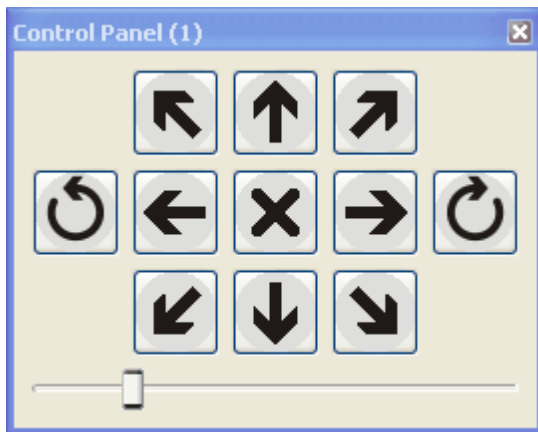
5.9.1 Control Panel



A control panel usable with the mouse.

Outputs	Type	Description
vx	float	Velocity in x-direction
vy	float	Velocity in y-direction
omega	float	Angular velocity.

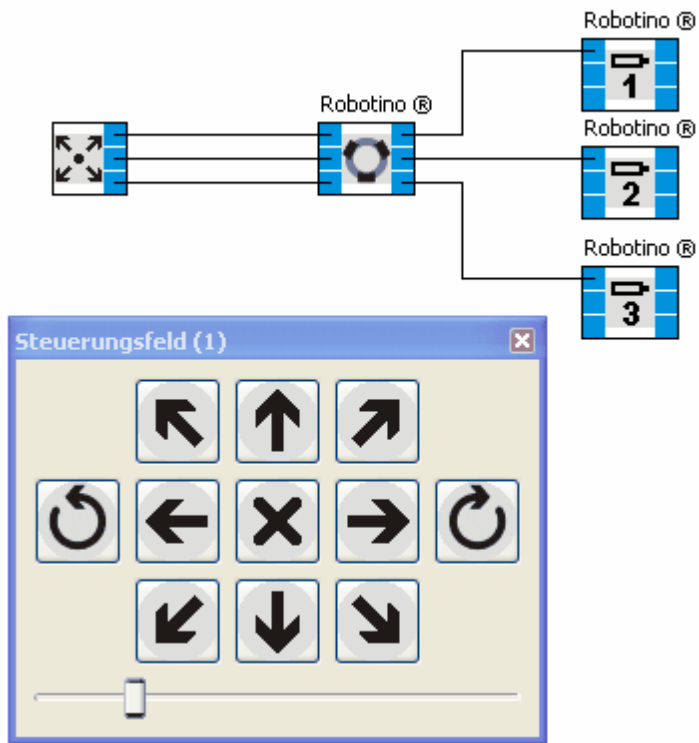
5.9.1.1 Dialog



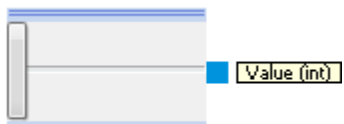
The control panel can be used as follows:

- By clicking one of the buttons the robot system is moved into the arrow's direction.
- By clicking one of the two circular arrows a rotation into the corresponding direction is performed.
- By clicking the button in the middle the movement is stopped.
- The movement's velocity is adjusted via the slider.

5.9.1.2 Example

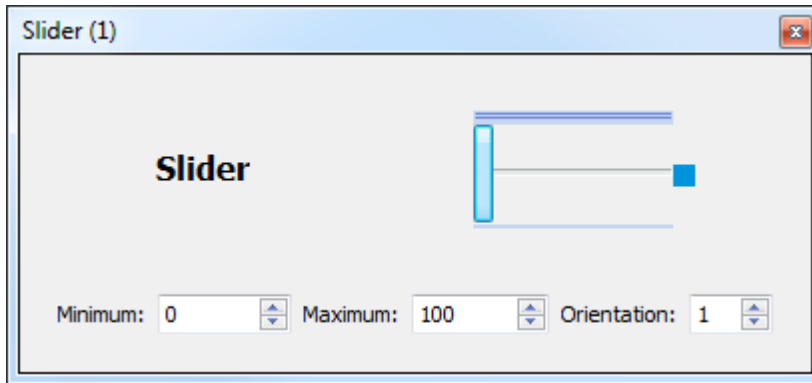


5.9.2 Slider



The slider creates any integer value within a specified range.

5.9.2.1 Dialog



In the dialog the slider's range and orientation (1 = horizontal, 0 = vertical) can be adjusted.

5.10 Data exchange

This category contains function blocks for data exchange within Robotino® View or with external applications.

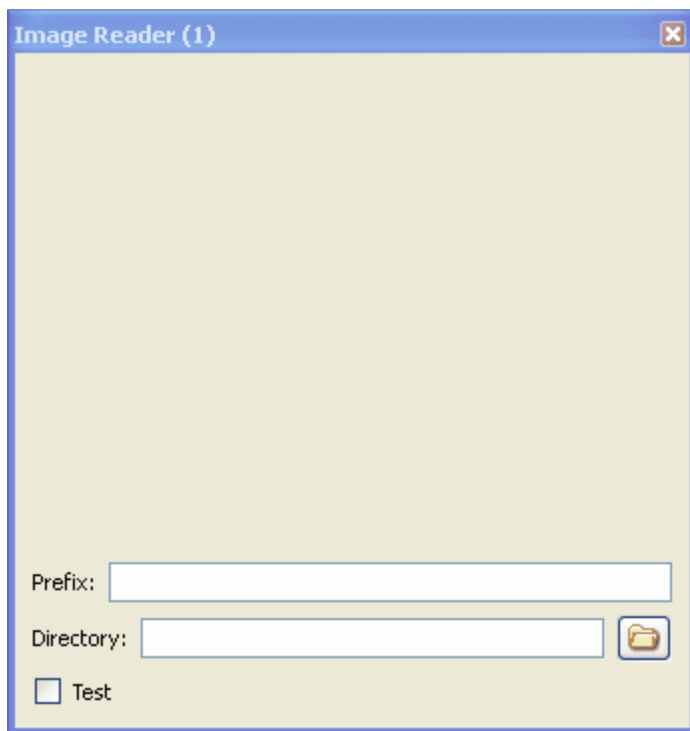
5.10.1 Image Reader



The image reader reads single JPEG images from a picture sequence from the file system. Path and prefix can be specified in the [dialog](#)^[119].

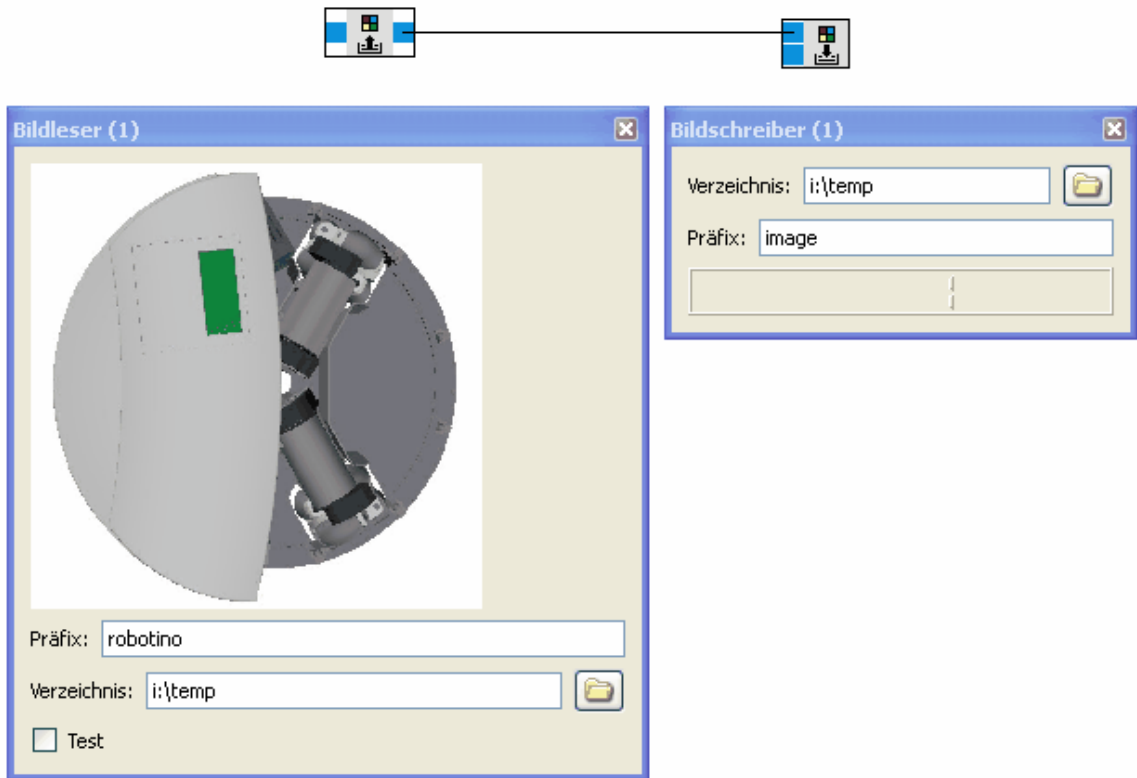
Inputs	Type	Default	Description
Number	int16	-1	Number of the desired image. If Number = -1, the image number is automatically increased by 1 in every step, starting with 0.
Outputs			
Output	image		JPEG image from file "<Path>/<Prefix><Number>.jpg" or "<Path>/<Prefix>_<Number>.jpg". If the file does not exist, the number will be prepended leading zeros up to a total length of 4 until the image file is found.

5.10.1.1 Dialog

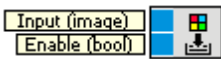


In the dialog it is possible to specify the path and prefix of the picture sequence that is to be read.

5.10.1.2 Example



5.10.2 Image Writer

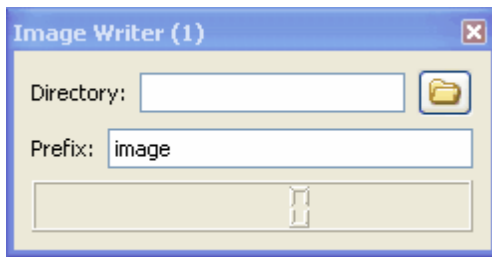


The image writer writes a sequence of JPEG images to the file system. Path and prefix can be specified in the [dialog](#)^[12]. The image's number is increased by 1 in every step, starting with 0.

Every single image is saved under "<Path><Prefix><Number>.jpg". The number has at least 4 digits, including leading zeros.

Inputs	Type	Default	Description
Input	image		Next image of the sequence.
Enable	bool	true	The image writer is active.

5.10.2.1 Dialog



5.10.2.2 Example

See example [image reader](#)^[120].

5.11 Variables

Global variables are a kind of special. For all global variables there are function blocks for reading and writing available in every sub-program. These function blocks do always show the variable's name and can't be renamed.

Global variables can be added, removed and renamed in the [variable manager \(main program view\)](#)^[15]. Furthermore, they can be assigned initial values.

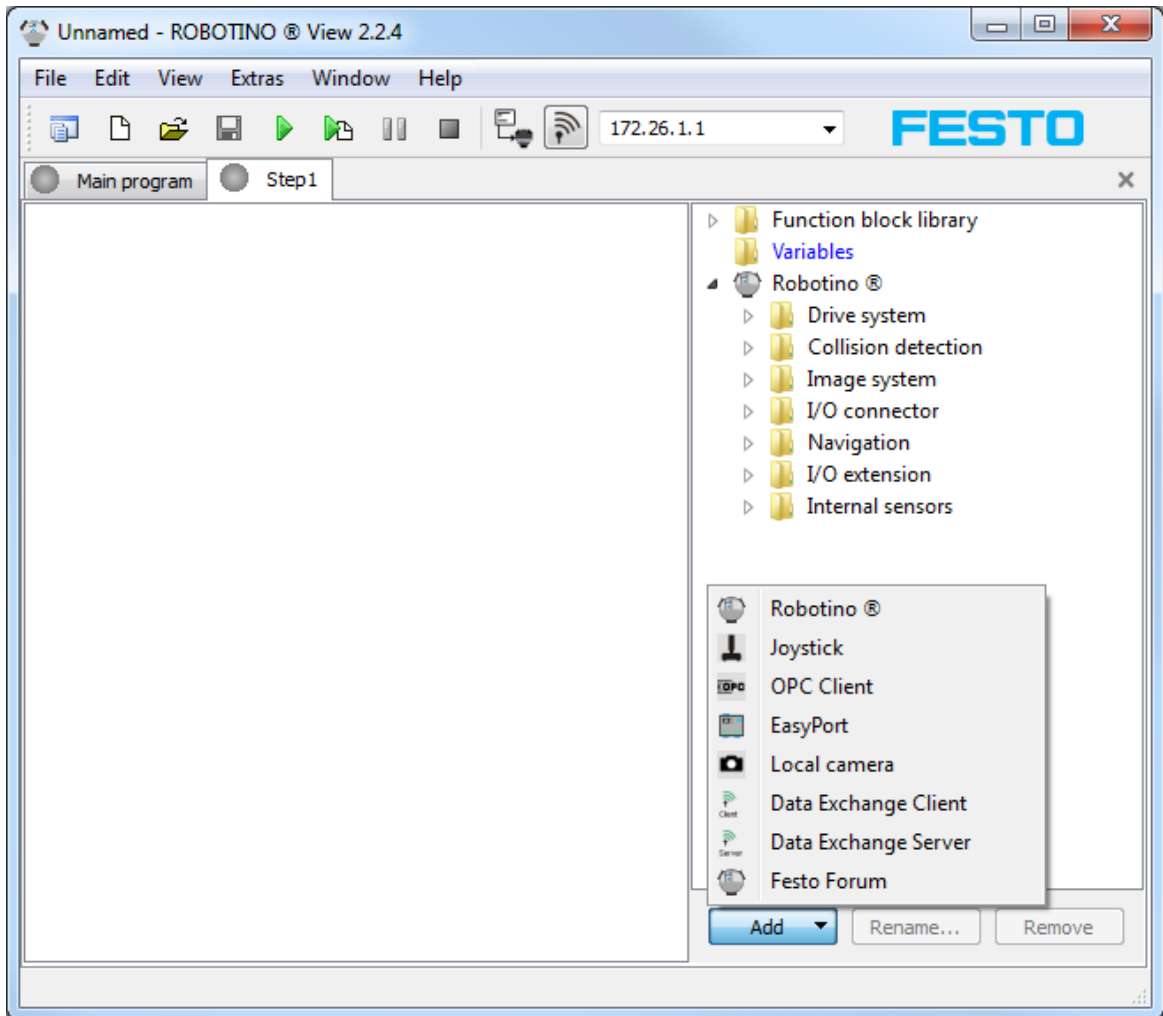
Adding, removing and renaming global variables is also possible in the function block library by right-clicking on the device "Variables" and selecting "Add" or right-clicking on the variable's reader or writer and selecting "Remove" or "Rename".

6 Devices

Devices establish the connection between Robotino View and the outside world. The device "Robotino" can communicate with real Robotino or a simulated one. The device "Joystick" can read the positions of the axes of a joystick attached to the computer.

6.1 Add and edit

When creating a new project the device "Robotino" is automatically added. To add more devices you have to change the current view to a subprogram.



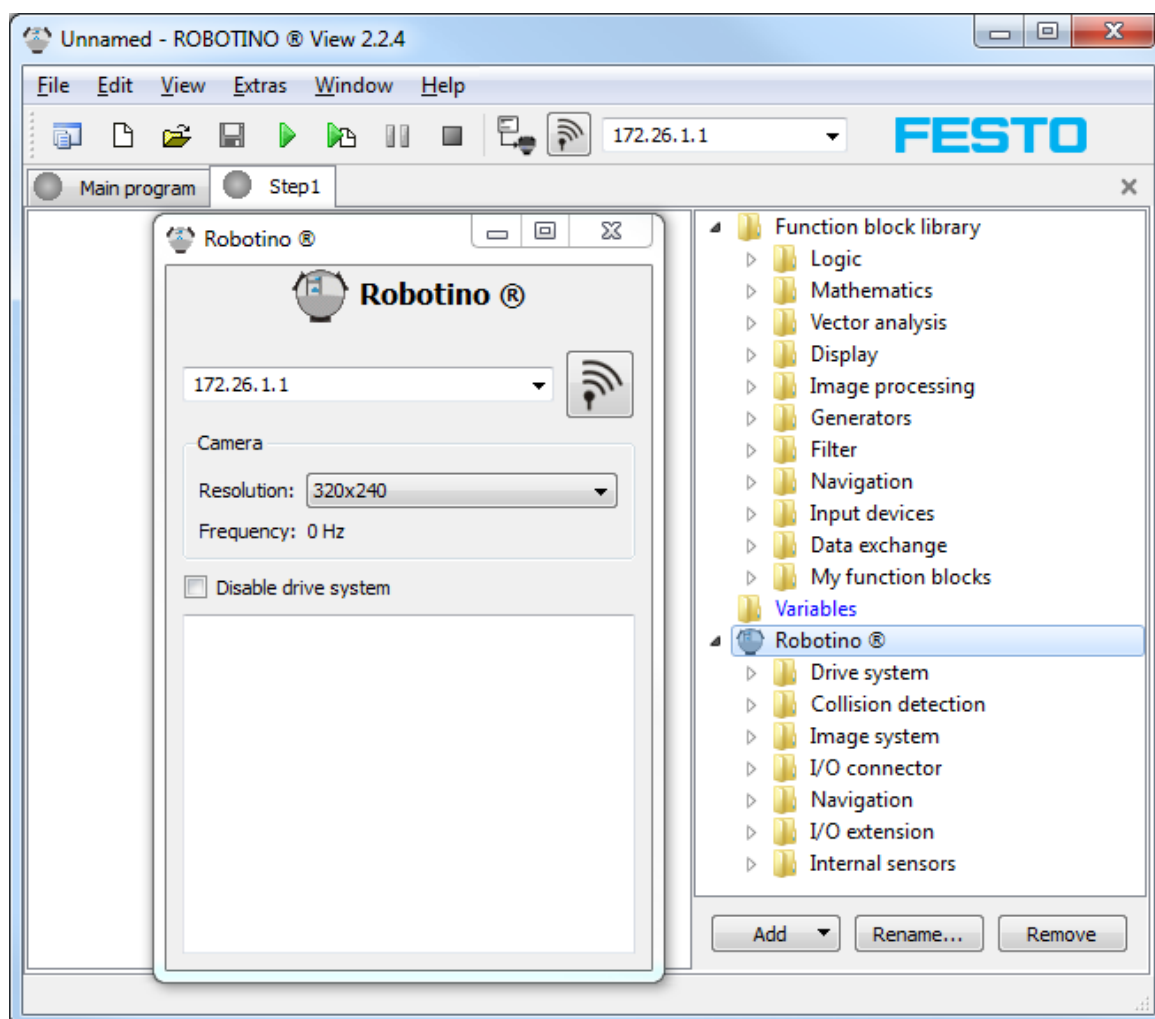
Below the function block library devices can be added using the "Add" button. The device chosen will appear underneath the device "Robotino" in the function block library.

New devices get a unique name. This name can be changed using the "Rename" button, if the device has been selected in the function block library.

The "Remove" button is used to remove devices from the current project. This function is available only if no function blocks of the devices are used within the project.

6.2 Show dialogs

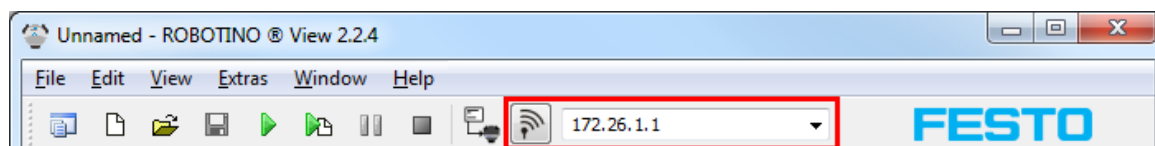
Every device has a configuration dialog. This dialog is opened by double clicking the device in the function block library



6.3 Robotino

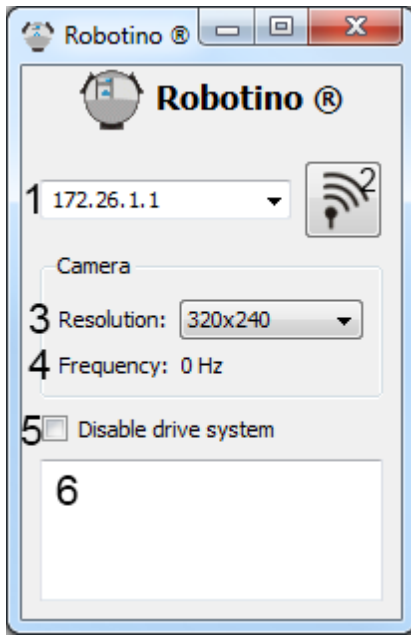
The Robotino Device provides access to sensors and actors of the Robotino® robot system.

6.3.1 Toolbar



You can find the IP address input field and connect button within the main [toolbar](#)¹¹. The IP address input and connect button refer to the first Robotino device in the list of devices in the device manager. The function of the IP address input field and the connect button is identical to the one in the device dialog.

6.3.2 Dialog



The dialog of the Robotino device will be shown after double-clicking on the Robotino device.

1	IP address input	Robotino's default IP address is 172.26.1.1. If you want to connect to Robotino Sim (running on the same computer as Robotino View) the IP address is 127.0.0.1:8080. 8080 is the port number, at which the Robotino server listens to incoming connections. If more than one Robotino is simulated, the port number can be higher.
2	Connect button	By clicking on this button a connection to Robotino will be established or closed.
3	Resolution	The requested resolution of images taken by Robotino's camera.
4	Frequency	Frequency of image updates
5	Disable drive system	If checked, Robotino's motors are deactivated
6	Message window	Display of various message in text form.

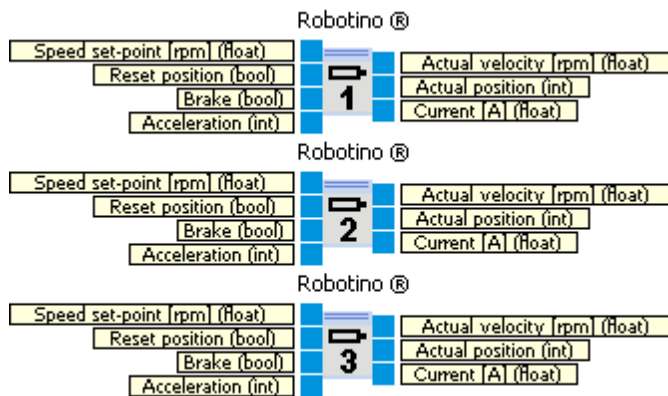
6.3.3 Function blocks

The function blocks allow the usage of the Robotino device in a subprogram.

6.3.3.1 Drive system

This folder contains function blocks to control Robotino's drive system.

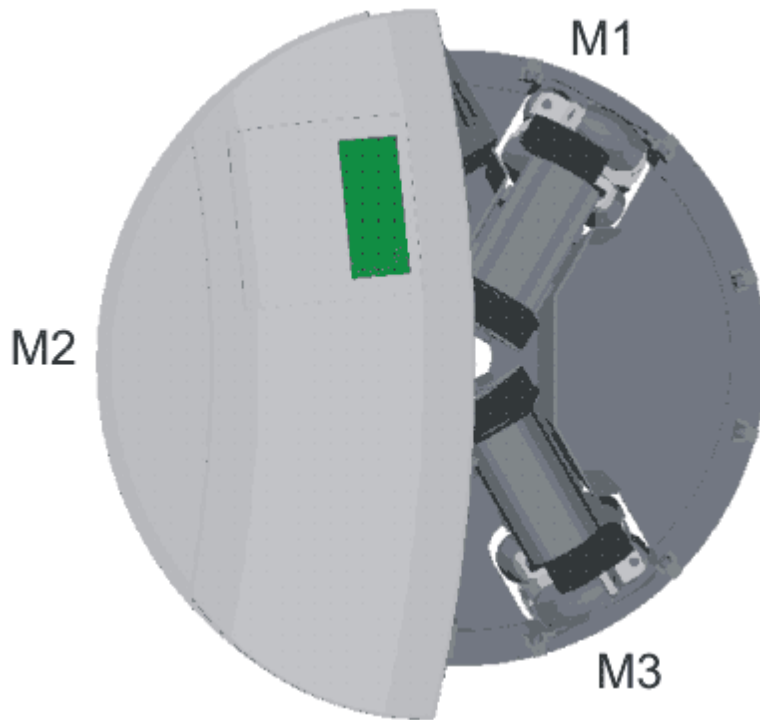
6.3.3.1.1 Motor



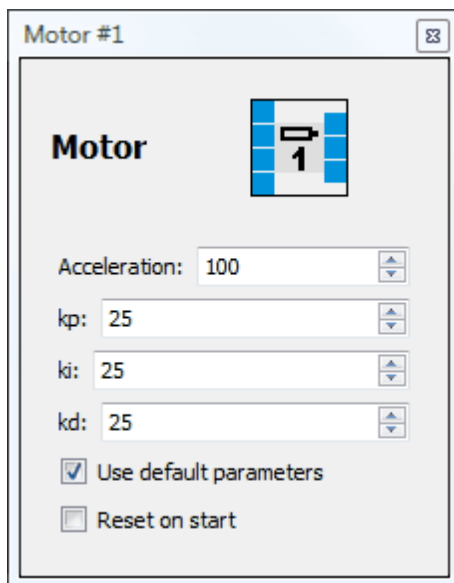
Access to one Robotino's motors. The motor number is displayed.

Inputs	Type	Unit	Default	Description
Speed set-point	float	rpm	0	The speed set-point of the motor control in rounds per minute. Please note that there is a 16:1 gear between motor and Robotino's wheel.
Reset position	bool		false	If true the tick counter of the motor's encoder is reset to 0.
Brake	bool		false	If true the motor is stopped.
Acceleration	int		100	Coupling of speed set-point at the input and the speed set-point really transmitted (see ▶ Dialog 126)
Outputs				
Actual velocity	float	rpm		The actual velocity of the motor.
Actual position	int			The number of ticks counted since power up of Robotino or since "Reset position" had been true and the false. The ticks are generated by the motor's encoder which generates 2000 ticks per round.
Current	float	A		The current measured at the motor's H-bridge.

Devices



6.3.3.1.1 Dialog



Parameter	Description
-----------	-------------

Acceleration	Acceleration/Deceleration factor. With the maximum value 100 speed set-points are given directly to the motor's controller. With smaller values differences between speed set-points are flattened over time. This can be used to generate smooth motions of Robotino.
kp	Proportional term of the motor's PID controller
ki	Integral term of the motor's PID controller
kd	Differential term of the motor's PID controller
Use default parameters	Use the values for kp, ki and kd implemented in Robotinos firmware. These default values are also used if you set kp=ki=kd=255.
Reset on start	Initialize the Actual position with 0 at program start

Velocity control of each motor is performed by a PID controller

$$u(t) = K_p \left(e(t) + \frac{1}{T_N} \int_0^t e(t') dt' \right) + K_d \dot{e}(t)$$

The parameters are:

$$K_p$$

$$K_i = 1/T_n$$

$$K_d$$

From the values set in the dialog the controller parameters are calculated as:

$$K_p = kp / 2$$

$$K_i = ki / 1024$$

$$K_d = kd / 2$$

Default values are

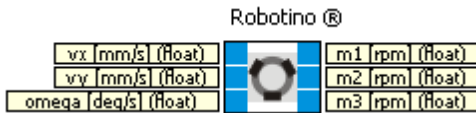
$$kp = 25$$

$$ki = 25$$

$$kd = 25$$

Devices

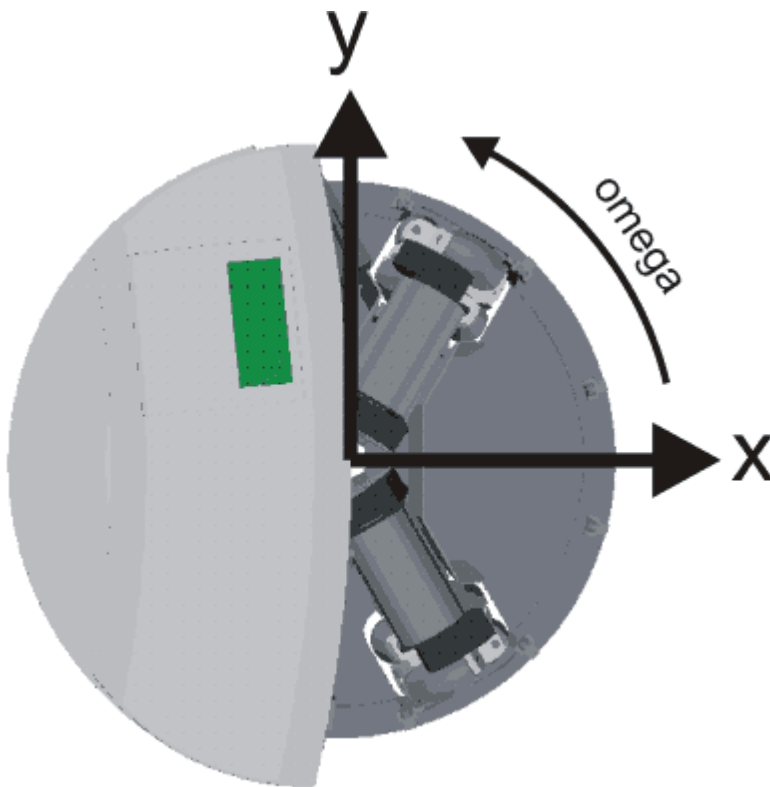
6.3.3.1.2 Omnidrive



Calculates the speed set-points of motor 1, 2 and 3 according to set-velocities vx, vy and omega.

Inputs	Type	Unit	Default	Description
vx	float	mm/s	0	Set-velocity in x-direction in Robotino's local coordinate system.
vy zurücksetzen	float	mm/s	0	Set-velocity in y-direction in Robotino's local coordinate system.
omega	float	deg/s	0	Set-rotational velocity.
Outputs				
m1	float	rpm		Speed set-point motor 1
m2	float	rpm		Speed set-point motor 2
m3	float	rpm		Speed set-point motor 2

The function block "Omnidrive (inverse)" calculates vx, vy and omega from the motors' rotation speeds.



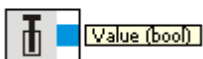
The image shows Robotino's local coordinate system. A positive rotational velocity ω generates a counter-clockwise rotation when looking from top onto Robotino.

6.3.3.2 Collision detection

Here you can find function blocks referring to sensors for detecting obstacles.

6.3.3.2.1 Bumper

Robotino ®

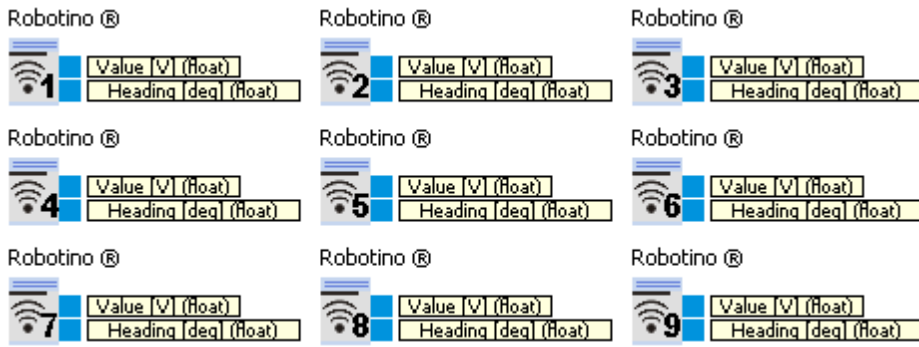


A tactile sensor is integrated into the bumper. If contacted, the sensor supplies an output signal.

Inputs	Type	Default	Description
Outputs			
Value	bool		True if there is a contact.

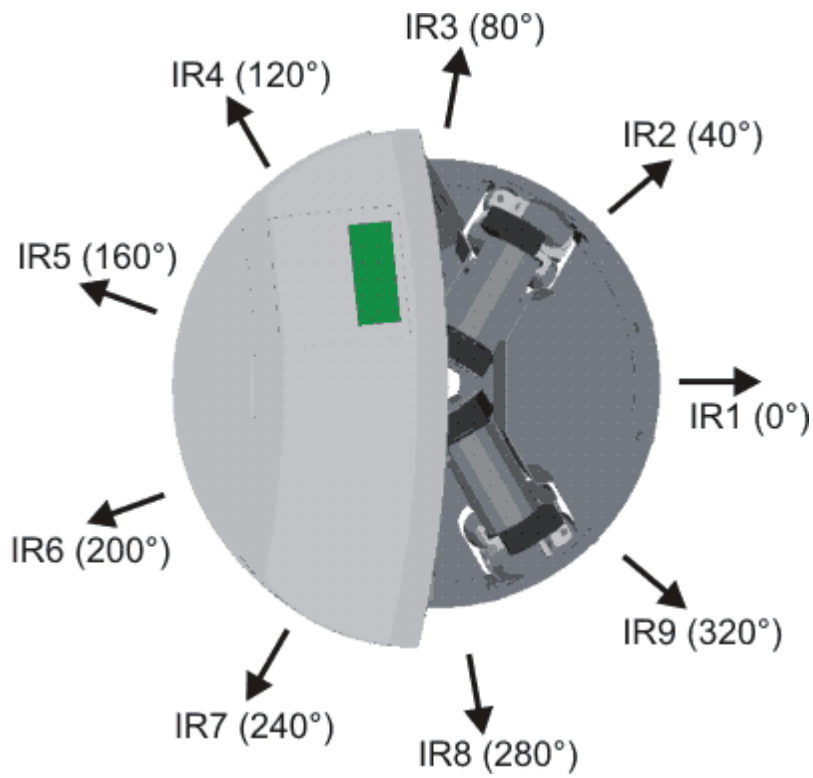
Devices

6.3.3.2.2 Distance sensors



The sensor reading of a distance sensor.

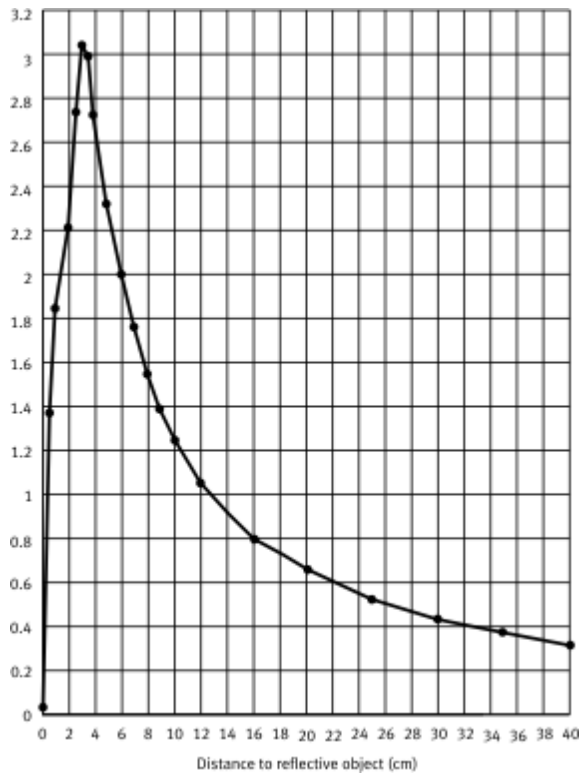
Inputs	Type	Unit	Default	Description
Outputs				
Value	float	Volt		Analog reading of the distance sensor in V. The scaling and conversion of a distance value must be effected by the user.
Heading	float	Degree		The heading of the sensor in Robotino's local coordinate system (see image below). The heading is calculated from the sensor number as $\text{Heading} = 40^\circ \times (\text{Number} - 1)$



6.3.3.2.2 Example

The data sheet of the distance sensor (its a Sharp GP2D120) shows the mapping between distance to an object in cm and the sensor's analog output signal in Volt.

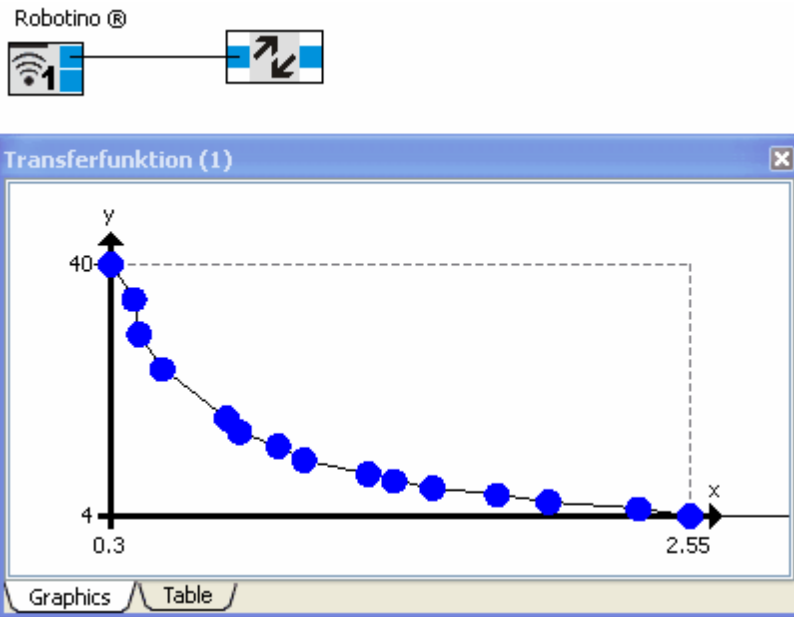
Devices



With this mapping it is easy to configure a [transfer function](#)^[61] so that the analog voltage is mapped to distance to object in cm. Please notice that this mapping is the inverse of the mapping shown above. This means that we have to skip distances smaller 4cm. Distance smaller 4cm can not be distinguished from distances larger 4cm, because the analog voltage output of the sensor is the same.

Furthermore the analog digital converter measures voltages up to 2,55V only.

The mapping from analog voltage to distance is also influenced by the material of the detected object. Overall it is best practice to measure the mapping by yourself.



The values of the [transfer function](#) ^[61] are given below. You can Copy&Paste these values into your own [transfer function](#) ^[61].

0.3	40
0.39	35
0.41	30
0.5	25
0.75	18
0.8	16
0.95	14
1.05	12
1.3	10
1.4	9
1.55	8
1.8	7
2	6
2.35	5
2.55	4

6.3.3.3 Image system

This folder contains function blocks to use Robotino's camera.

Devices

6.3.3.3.1 Camera

Robotino ®

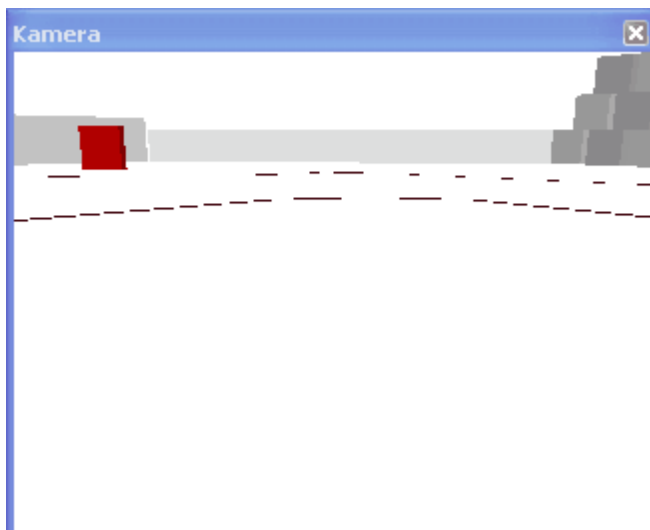


The live image of Robotino's camera.

Inputs	Type	Default	Description
Outputs			
Image	image		The image from Robotino' camera

To set the image resolution you can use Robotino's [device dialog](#)^[124].

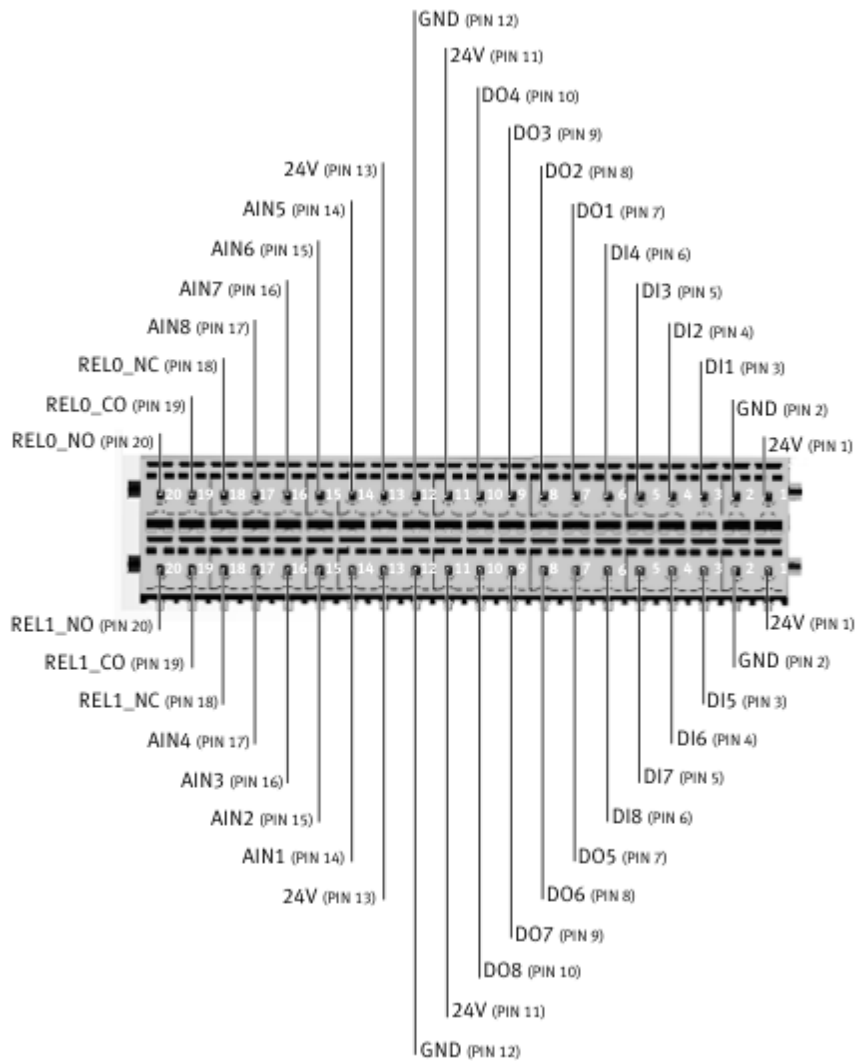
6.3.3.3.1 Dialog



Shows the latest image. To adjust the image resolution see [device dialog](#)^[124].

6.3.3.4 I/O connector

Here you can find function blocks to access Robotino's I/O connector.



6.3.3.4.1 Relay



Switch relay 1 and 2.

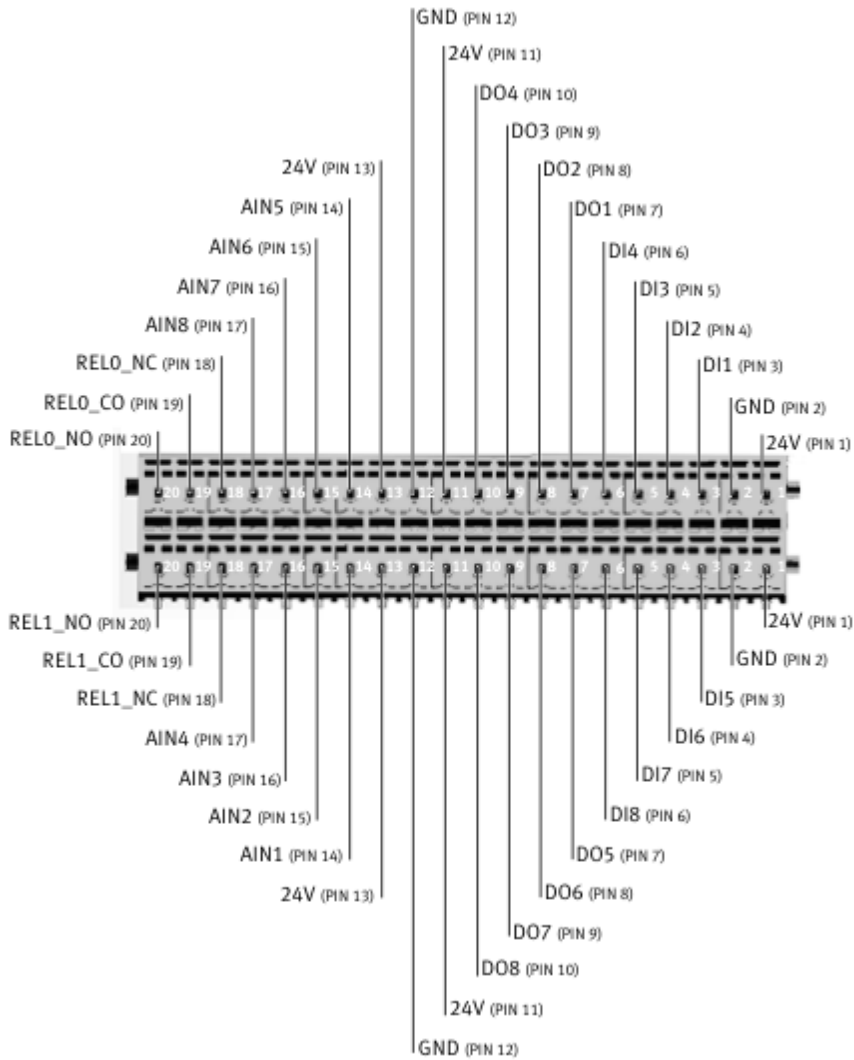
Inputs	Type	Default	Description
--------	------	---------	-------------

Devices

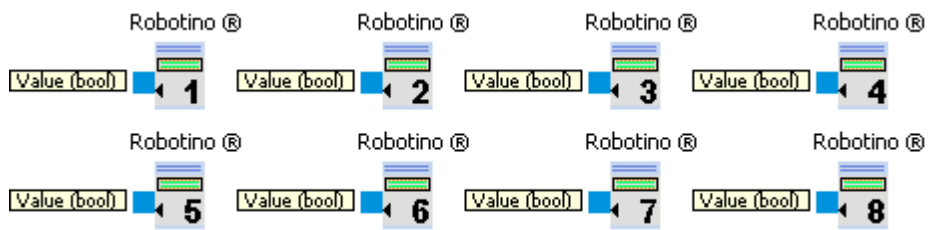
Value	bool	false	If false the relay is switched off.
-------	------	-------	-------------------------------------

The connectors for relay 1 are REL1_NO, REL1_CO and REL1_NC.

The connectors for relay 2 are REL2_NO, REL2_CO and REL2_NC.



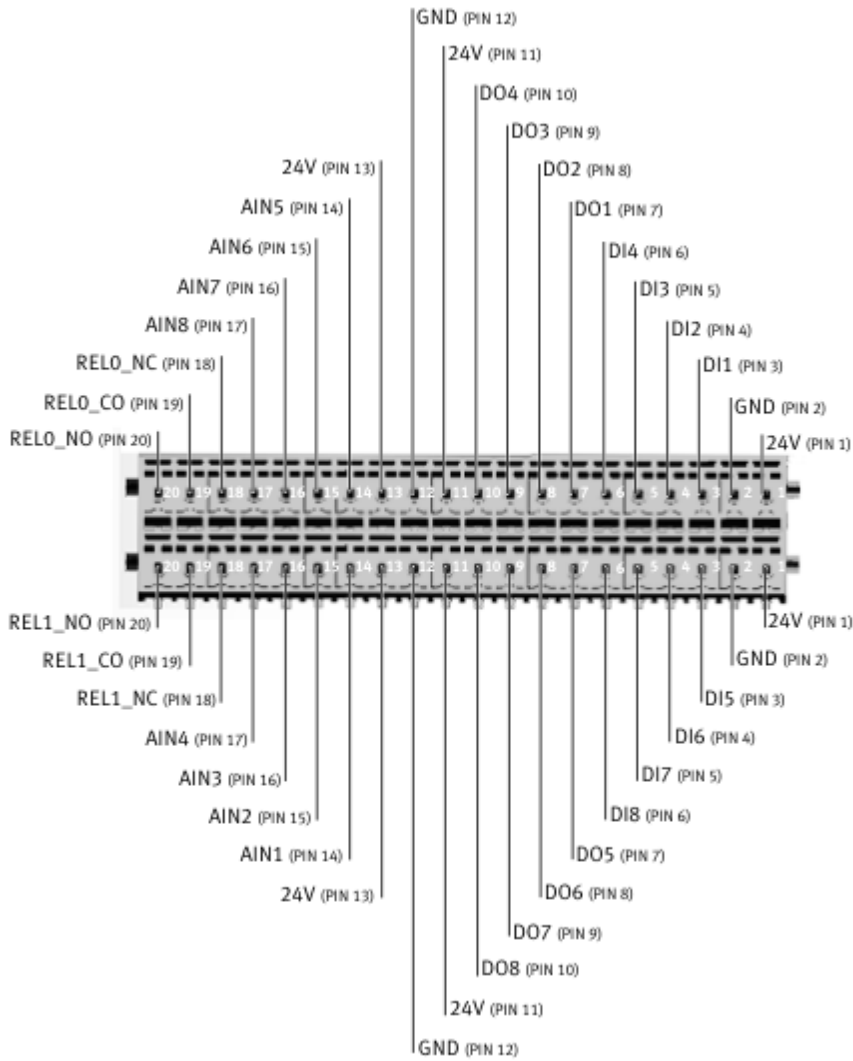
6.3.3.4.2 Digital output



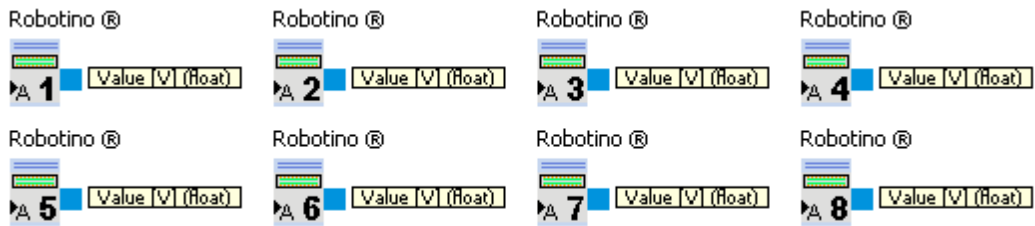
Set a digital output.

Inputs	Type	Default	Description
Value	bool	false	If true the output at Robotino's I/O connector is +10V. Otherwise the output is 0V.

The connector for digital output x is DOx with x in [1;8].



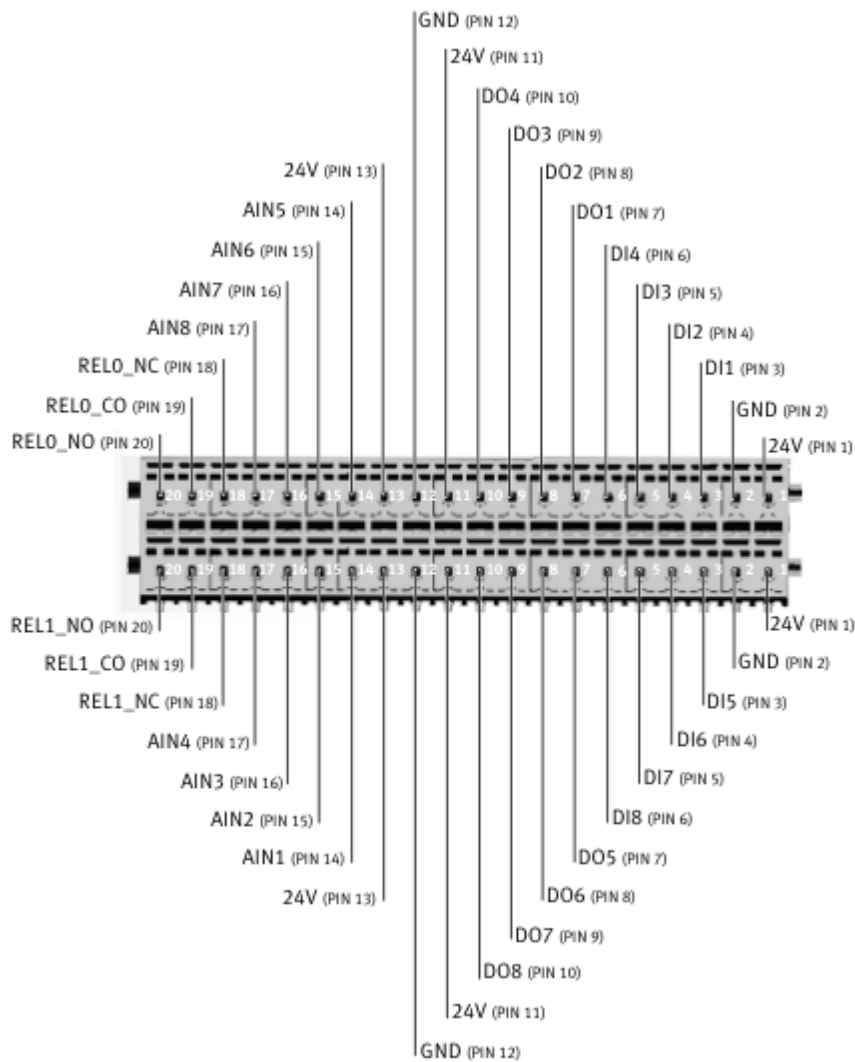
6.3.3.4.3 Analog input



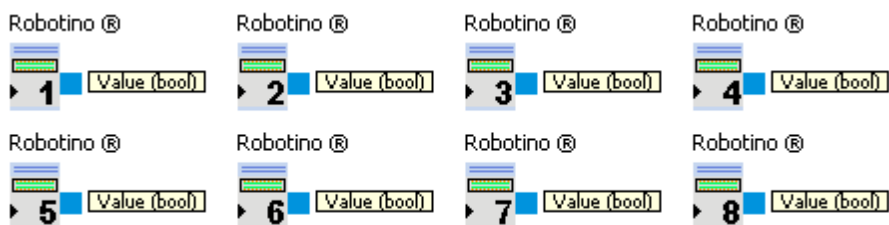
Reads the value of an analog input.

Outputs	Type	Unit	Description
Value	float	Volt	The measured voltage. Range [0;10].

The connector for analog input x is AINx with x in [1;8].



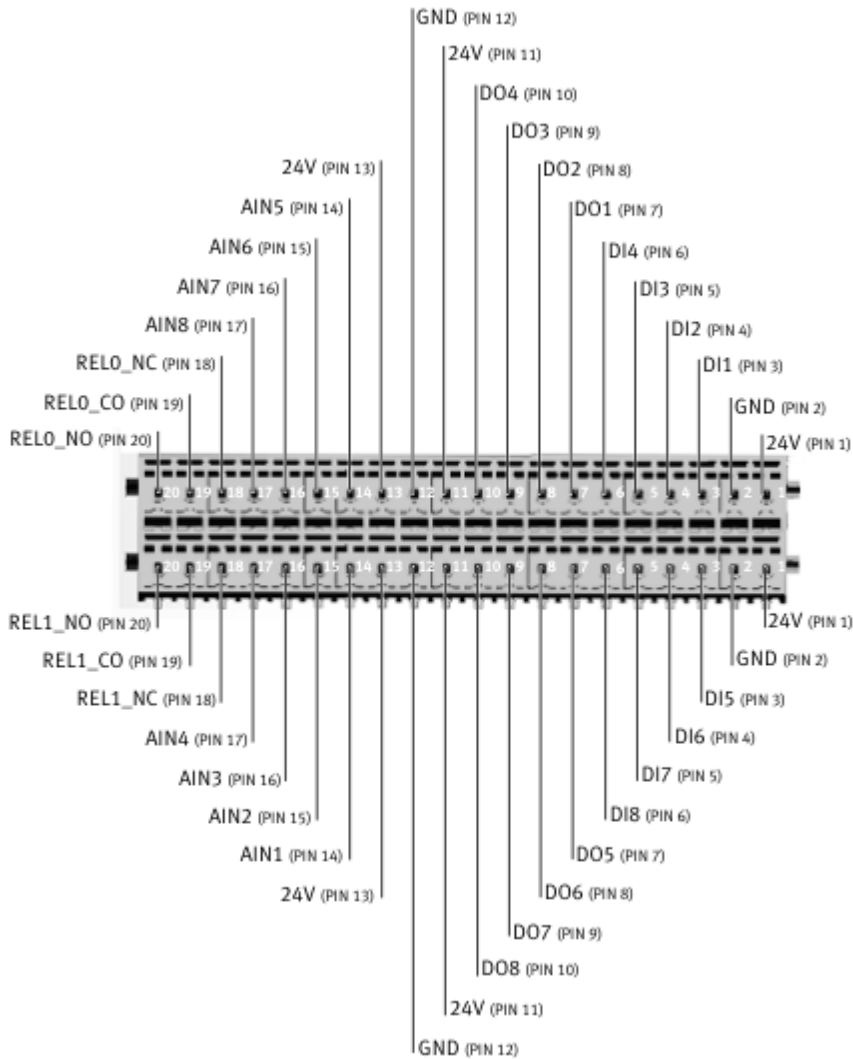
6.3.3.4.4 Digital input



Reads the value of a digital input.

Outputs	Type	Description
Value	bool	The value at Robotino's I/O connector. Voltages less 5.75V are mapped to false. Values greater 8.6V are mapped to true. If the voltage at the connector is between 5.75V and 8.6V the value remains unchanged.

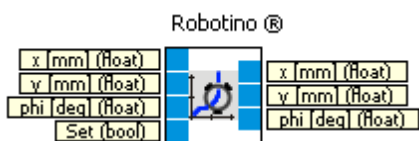
The connector for digital input x is DIx with x in[1;8].



6.3.3.5 Navigation

This folder contains function blocks for the location of Robotino.

6.3.3.5.1 Odometry



For this functionality a 1GB Compact-Flash memory card for Robotino (V 1.7 or higher) is needed.

(No functionality with 256MB memory cards, Version <=1.6)

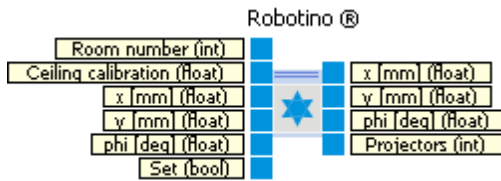
Odometry is the use of data from the movement of actuators to estimate change in position over time. See <http://en.wikipedia.org/wiki/Odometry>.

The rotation of wheels is measured with the highest time resolution possible. In every time step the distance driven by the vehicle is calculated from the wheels rotational speed. These very small distances from the single time steps are integrated over time. This leads to the actual position relative to the starting position. This method yields good local performance. On long distances or under adverse conditions (wheels slip because of dust on the floor, drift because of preferential direction of the carpet) this method leads to very large errors. On this account odometry is always combine with other methods to compensate for the described errors.

Inputs	Type	Unit	Default	Description
x	float	mm	0	The new x-position. Odometry is reset to the new position if "Set" is true.
y	float	mm	0	The new y-position. Odometry is reset to the new position if "Set" is true.
phi	float	Degree	0	The new orientation. Odometry is reset to the new position if "Set" is true.
Set	bool		false	If true, the odometry is set to the values from inputs x, y, and phi. To reset the odometry to (0,0,0) you only need to set this input true for one time step. The other inputs do not need to be connected, because the default values are 0.
Outputs				
x	float	mm		The current x-position from the odometry in global coordinates.
y	float	mm		The current y-position from the odometry in global coordinates.
phi	float	Degree		The current orientation from the odometry in global coordinates.

Devices

6.3.3.5.2 North Star



North Star ® is a sensor which determines Robotino®'s absolute position with the help of projectors.

Inputs	Type	Unit	Default	Description
Room number	int		1	The room number in which Robotino is currently located. Rooms are enumerated starting with 1.
Ceiling calibration	float	mm	1	Distance between detector and ceiling. If the ceiling height is 3m the distance between detector and ceiling is about 2800mm.
x	float	mm	0	x-position of the origin set by the input "Set".
y	float	mm	0	y-position of the origin set by the input "Set".
phi	float	Degree	0	Orientation of the origin set by the input "Set".
Set	bool		false	If true, the current pose (x,y,phi) is used as origin.
Outputs				
x	float	mm		The current x-position in global coordinates.
y	float	mm		The current y-position in global coordinates.
phi	float	Degree		The current orientation in global coordinates.
Projektoren	int			Number of visible projectors.

Room ID	Projector type	Switch-Setting	Frequency [Hz]
0	ProjectorKit NS1	Spot 1 SW2 = 0 Spot 2 SW2 = 8	1040 2350
1	ProjectorKit NS1	Spot 1 SW2 = 1 Spot 2 SW2 = 9	1150 2450
2	ProjectorKit NS1	Spot 1 SW2 = 2 Spot 2 SW2 = A	1250 2560
<hr/>			
3	Projector NS2-50Hz	1	Spot A 3025 Spot B 3925
4	Projector NS2-50Hz	2	Spot A 3125 Spot B 4025
5	Projector NS2-50Hz	3	Spot A 3225 Spot B 4125



ProjectorKit NS1

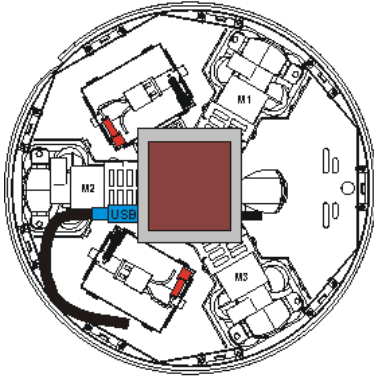


Projector NS2

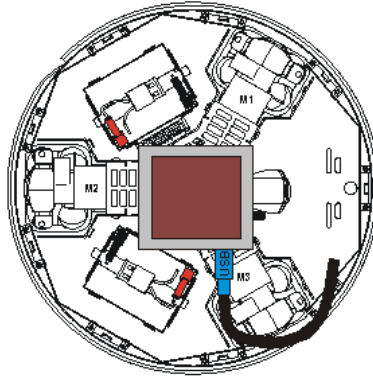
The Northstar detector can be attached to Robotino in different ways. Depending on the configuration the file `/etc/robotino/robotino.xml` on robotino must be adapted with a text editor. The value for the orientation must be set according to the figure below.

```
<NorthStar>
  <!--The orientation of the northstar sensor. See www.openrobotino.org-->
  <Orientation value="1" />
</NorthStar>
```

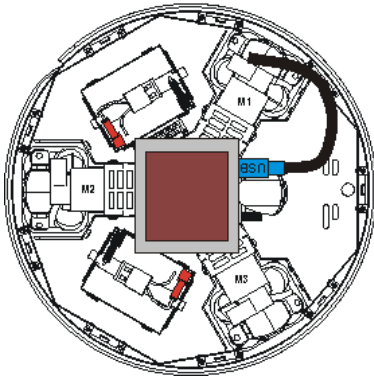
0



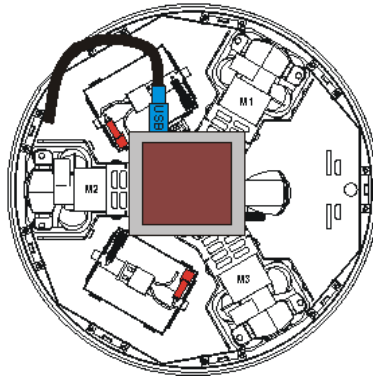
1



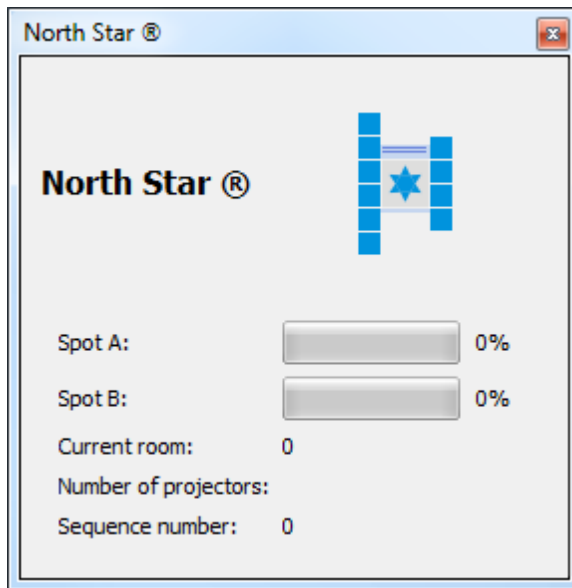
2



3



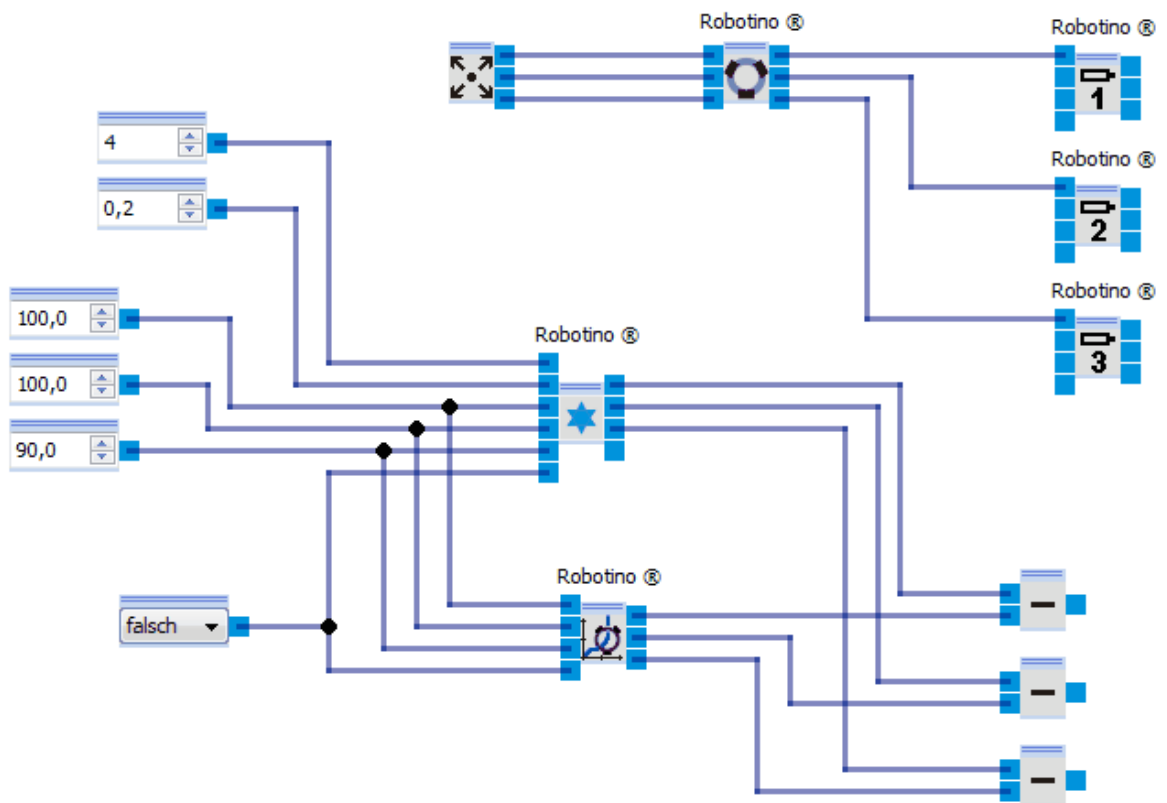
6.3.3.5.2 Dialog



Spot A	Intensity of the first light spot emitted by the projector.
Spot B	Intensity of the second light spot emitted by the projector.
Current room	The room number detected by the North Star sensor.
Number of projectors	The number of visible projectors.
Sequence number	The sequence number is incremented by one each time the North Star sensor provides new readings.

Devices

6.3.3.5.2 Example



The control panel is used to move Robotino.

Northstar shall detect the projector that belongs to room 4. The new NorthStar projector must be set to room 1.



Via the boolean constant (true/false) the coordinate systems of Northstar and the odometry can be transformed so that "current Northstar pose" == "current odometry pose" == (100,100,90).

By subtracting the components of Northstar and odometry, you can see the error occurring between odometry and Northstar.

6.3.3.6 I/O extension

This folder contains function blocks to Robotino's other hardware interfaces.

6.3.3.6.1 Encoder input



This function block reads values from Robotino's encoder input. The encoder input interprets signals from a digital motor encoder (A,B-channel gray-code). Rising and falling edges are considered. This leads to a quad effective resolution.

Example: The motor encoder of Robotino' s drive motors has a resolution of 500 ticks. Effectively 2000 ticks are counted.

Inputs	Type	Unit	Default	Description
Reset position	bool		0	If true the actual position is reset to 0.
Outputs				
Actual velocity	int	ticks/s		The measured velocity.
Actual position	int	ticks		The sum of all ticks measured since the start of robotino or since "Reset position" = true.

6.3.3.6.2 Power output



This function block assigns set point values to Robotino's power output (former Motor 4). The power output can only be used if the sub-program doesn't use the gripper.

The output is instantiated by a H-bridge, which can deliver up to 5A continuous current. The H-bridge is driven by a high frequency PWM signal and one bit for direction. The sign of the set point given by the input corresponds to the direction bit. The absolute value of the set point influences the PWM signal. A set point of 0 does not generate any PWM signal, i.e. no current is delivered by the H-bridge. A set point of 50 leads to a high-low-ratio of the PWM signal of 50%. A set point of 100 generates a constant high, i.e. the H-Bridge delivers maximum current.

Inputs	Type	Unit	Default	Description
Set-point	int		0	Sets direction bit and PWM signal. Range -100 to 100. Values less -100 are interpreted as -100. Values greater 100 are interpreted as 100.
Outputs				
Current	float	A		Der durch die H-Brücke fließende Strom.

The current delivered by the power output is limited by default. To change or disable this limitation edit /etc/robotino/robotino.xml on Robotino. The new values are assigned after 2 seconds.

6.3.3.6.3 Gripper



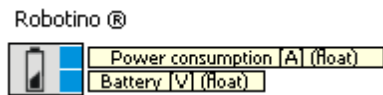
Use this module with a Festo Robotino Gripper. The Gripper can only be used if the current sub-program does not contain a [power output](#)^[148].

Inputs	Type	Default	Description
Open	bool	false	If true the gripper is opened.
Outputs			
Opened	bool		True if the gripper reached its opened position
Closed	bool		True if the gripper reached its closed position.

The Gripper must be connected to port X15 at the PCB behind the battery: brown cable (+) to the left, blue cable (-) to the right.

6.3.3.7 Internal sensors

6.3.3.7.1 Power management

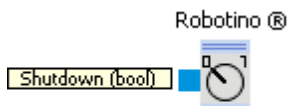


The power management module of Robotino.

Inputs	Type	Unit	Default	Description
Outputs				
Power consumption	float	A		The current drawn from Robotino's batteries.
Battery	float	Volt		Battery voltage.

Devices

6.3.3.7.2 Shutdown



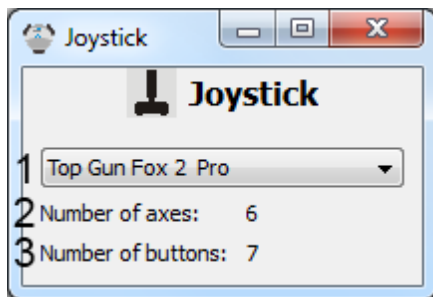
Shut down and switch off Robotino.

Inputs	Type	Default	Description
Shutdown	bool	false	If true, Robotino shuts down and is turned off.

6.4 Joystick

The device "Joystick" allows access to a locally attached joystick.

6.4.1 Dialog

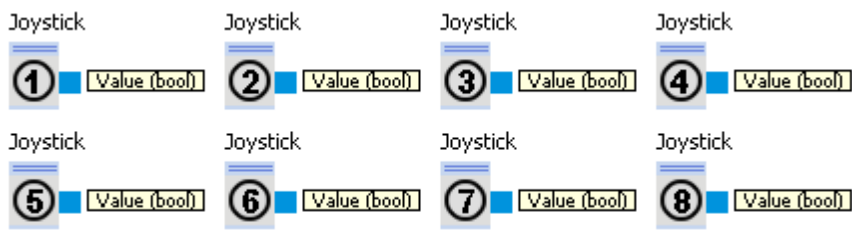


1	List of available joysticks	The combo box contains an entry for every joystick available at this computer. This list is updated whenever a new joystick is attached or detached from this computer. By selecting a joystick its buttons and axes become available through the corresponding function blocks.
2	Number of axes	The number of axes of the selected joystick.
3	Number of buttons	The number of buttons of the selected joystick.

6.4.2 Function blocks

Function blocks to read button and axis states.

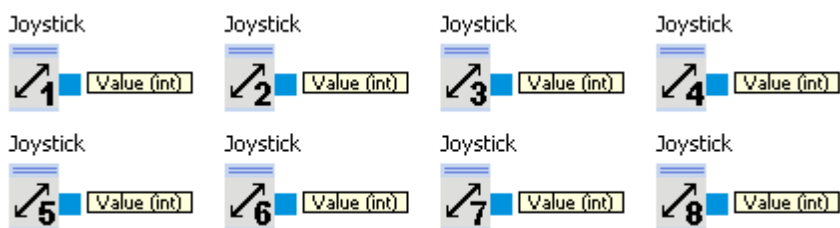
6.4.2.1 Button



Reads the state of a joystick's button.

Outputs	Type	Description
Value	bool	True if the button is pressed, false otherwise.

6.4.2.2 Axis



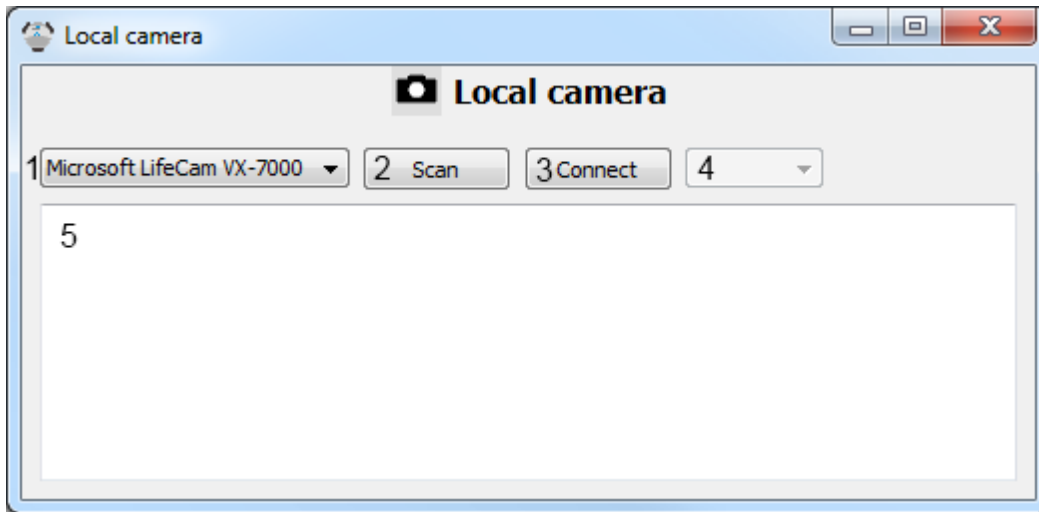
Read the position of a joystick's axis.

Outputs	Type	Description
Value	int	Range -1000 to 1000.

6.5 Local camera

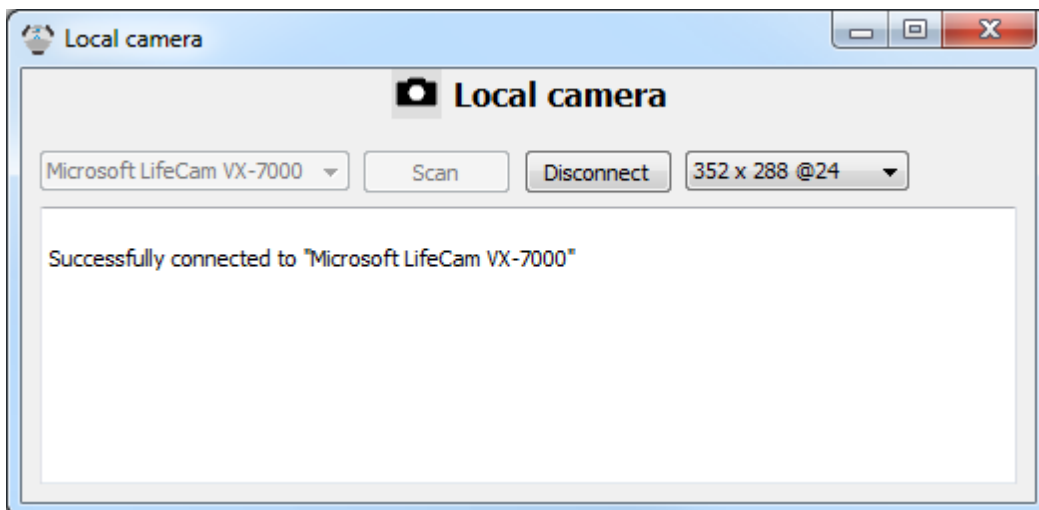
The device "Local camera" allows access to a camera that is attached to the computer (e.g. a webcam).

6.5.1 Dialog



1	List of available cameras	All cameras attached to the system are shown here. The list is updated when a new camera is attached to or removed from the computer.
2	Scan	Update the list of available cameras.
3	Connect/Disconnect	Establish/Close a connection to the selected camera.
4	Resolution/Color depth	Choose the resolution and color depth here. All resolutions supported by the camera are available.
5	Message window	Display of various message in text form.

After selecting a camera a connection must be established. Then it is possible to set the resolution.



6.5.2 Function blocks

The function blocks allow to use the device "Local camera" in a sub-program.

6.5.2.1 Camera

Local camera

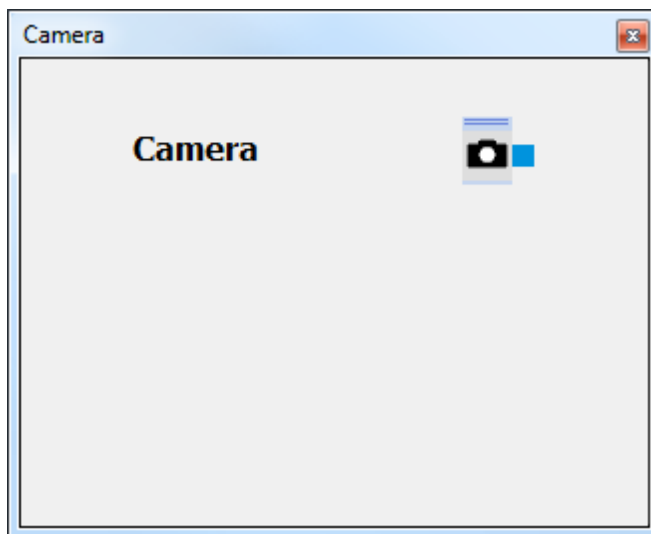


Live image of the local camera.

Inputs	Type	Default	Description
Outputs			
Image	image		Live image

Color depth and resolution can be selected in the [device dialog](#)^[152].

6.5.2.1.1 Dialog



The camera dialog shows the current image. To adjust the image resolution see [device dialog](#)^[152].

6.6 OPC Client

OPC is a standardised interface between different software applications and drivers of different hardware modules (e.g. PLC).

Multiple OPC-Clients can connect to one OPC-Server.

A (special) OPC-Server will often be provided by the common PLC manufacturers.

In the below sample a Festo EasyPort will be connected to RobotinoView via the free Festo EzOPC-Server.

The EzOPC-Server allocates the in-/outputs of up to 4 EasyPorts using so called "Groups" and "Tags":

- Device1 shows as group "EasyPort1 "
- Output 1 therefore shows as Tag "EasyPort1.OutputPort1 "

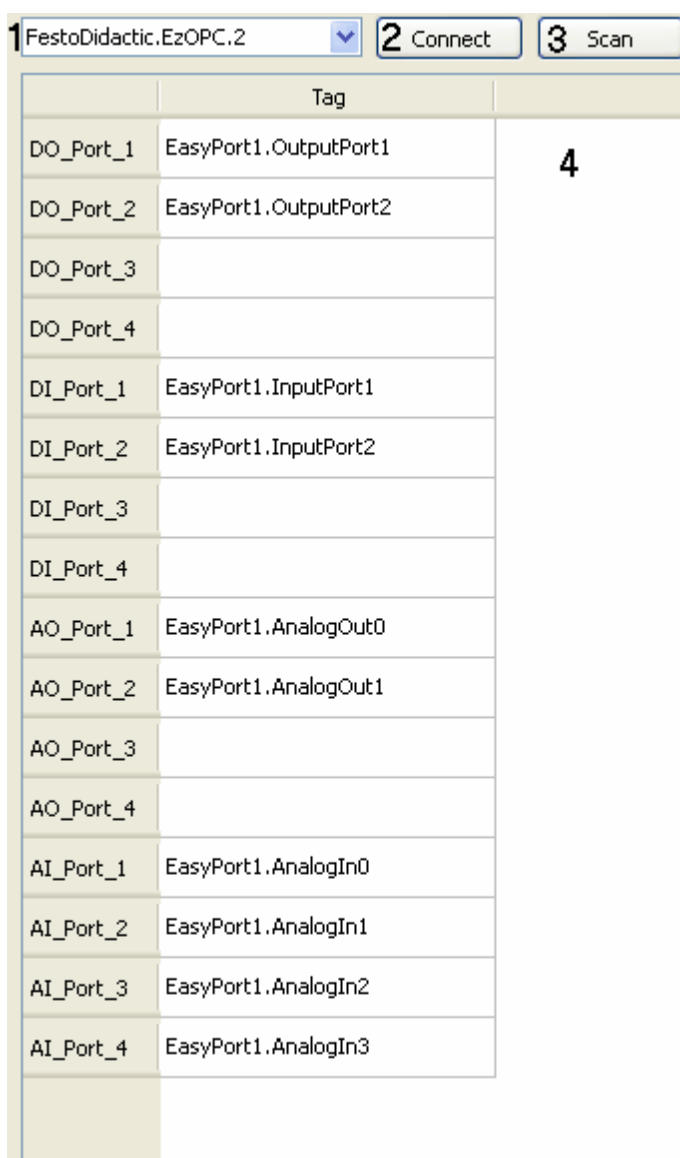
Please follow these steps:

1. Install the EzOPC-Server.
2. Start the EzOPC Server and choose "Process Simulation..." and "PLC via EasyPort".
3. Start RobotinoView.
4. Add the "OPC Client" device.
5. From the device dialog's context menu select Predefined settings ▶ "Festo EzOPC EasyPort". The default values for the EasyPort will be loaded.
6. If needed select "FestoDidactic.EzOPC.1".
7. Start the connection.
8. Use the OPC Client function blocks to access the OPC-data of the EasyPort's in-/outputs.

Hint: if you would like to use a PLC of a different manufacturer you need a OPC server or OPC client of this manufacturer. Use an OPC client to see which tags are available on your PC's OPC server.

Downloads and additional information can be found at <http://www.opcconnect.com/>

6.6.1 Dialog



1	Select OPC server	The combobox lists all available local OPC servers.
2	Connect	Establish a connection to the selected OPC server.
3	Scan	Update the list a OPC servers.
4	Mapping	This table defines the mapping from function blocks to OPC "Tags".

Row	function block
DO_Port_1	digital Output 1
DO_Port_2	digital Output 2
DO_Port_3	digital Output 3

Devices

DO_Port_4	digital Output 4
DI_Port_1	digital Input 1
DI_Port_2	digital Input 2
DI_Port_3	digital Input 3
DI_Port_4	digital Input 4
AO_Port_1	analog Output 1
AO_Port_2	analog Output 2
AO_Port_3	analog Output 3
AO_Port_4	analog Output 4
AI_Port_1	analog Input 1
AI_Port_2	analog Input 2
AI_Port_3	analog Input 3
AI_Port_4	analog Input 4

The context menu provides the following functionality:

Predefined settings ▶	
Load	
Save	
<hr/>	
Help	

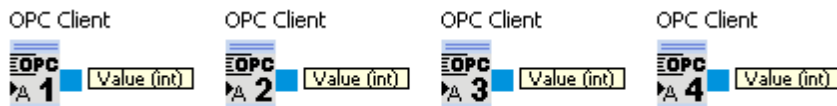
Predefined settings ▶ Festo EzOPC VirtualPLC	Load a mapping suitable for VirtualPLC
Predefined settings ▶ Festo EzOPC EasyPort	Load a mapping suitable for EasyPort
Load	Load a mapping from file.
Save	Save the current mapping to a file.
Help	Show this help page.

6.6.2 Function blocks

Function blocks allow use of the "OPC Client" device in a sub-program.

6.6.2.1 Inputs

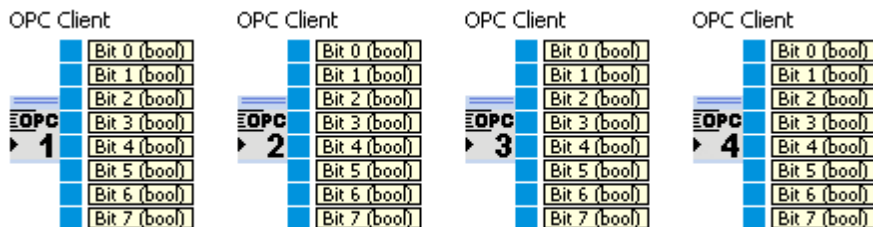
6.6.2.1.1 Analog input



Reading of the "Tag" mapped to AI_Port_x with x in [1;4]

Outputs	Type	Description
Value	int	Range 0 to 65535

6.6.2.1.2 Digital input



Read bit values of the "Tag" mapped to DI_Port_x with x in [1;4]

Outputs	Type	Description
Bit 0	bool	True if bit 0 is set.
...		
Bit 7	bool	True if bit 7 is set.

6.6.2.2 Outputs

6.6.2.2.1 Analog output

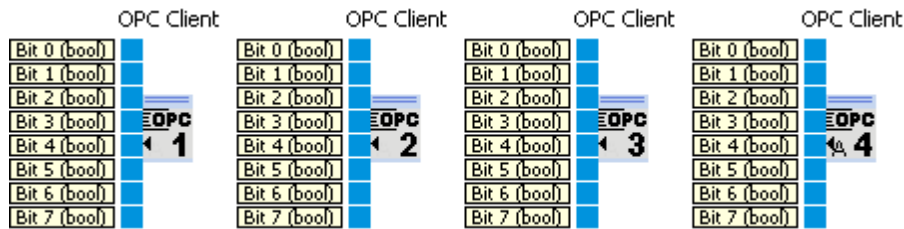


Devices

Write Value to the "Tag" mapped to AO_Port_x with x in [1;4]

Inputs	Type	Description
Value	int	Range 0 to 65535

6.6.2.2.2 Digital output



Set bits of "Tag" mapped to DO_Port_x with x in [1;4]

Inputs	Type	Description
Bit 0	bool	If true the value send to the OPC server is increased by $2^0 = 1$.
...		
Bit 7	bool	If true the value send to the OPC server is increased by $2^7 = 128$.

6.7 Data exchange

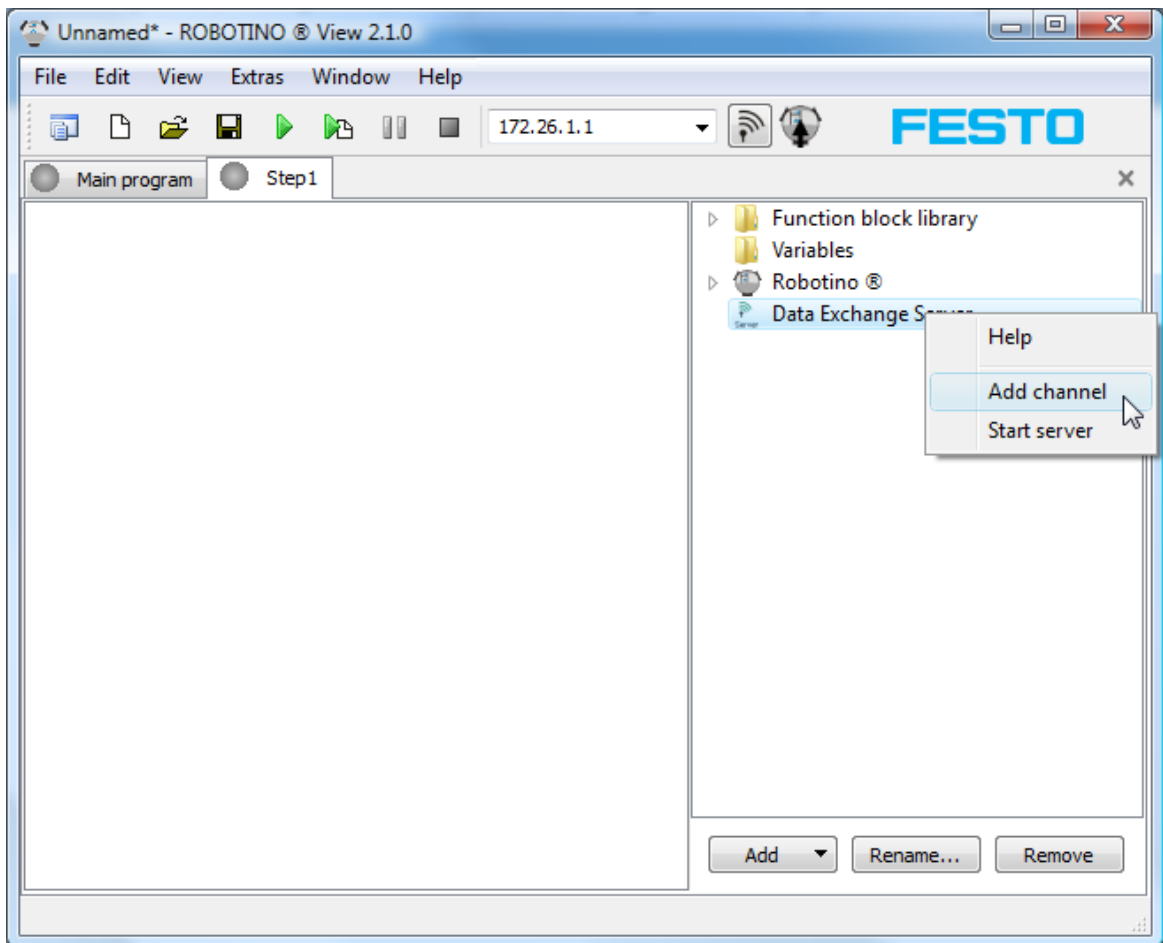
Devices from this category are used to exchange data between different Robotino View instances over a network.

6.7.1 Server

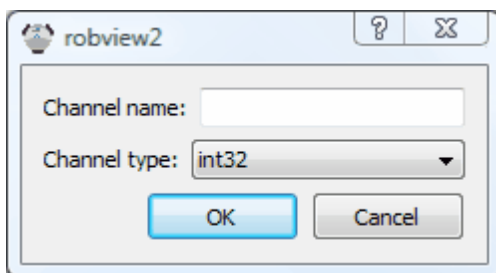
The data exchange server can be used by an arbitrary number of clients to exchange data over an arbitrary number of communication channels. The communication channels are create at server side. The communication channels created are broadcasted to all clients. The clients can choose over which channels they are going to exchange data with the server.

Server and clients are have equal rights when exchanging data. When a client writes data into a communication channel the data is transfer to the server and from there to all other clients. If more than one participant is writing to the same channel it is unpredictable which datum the communication channel contains in the end.

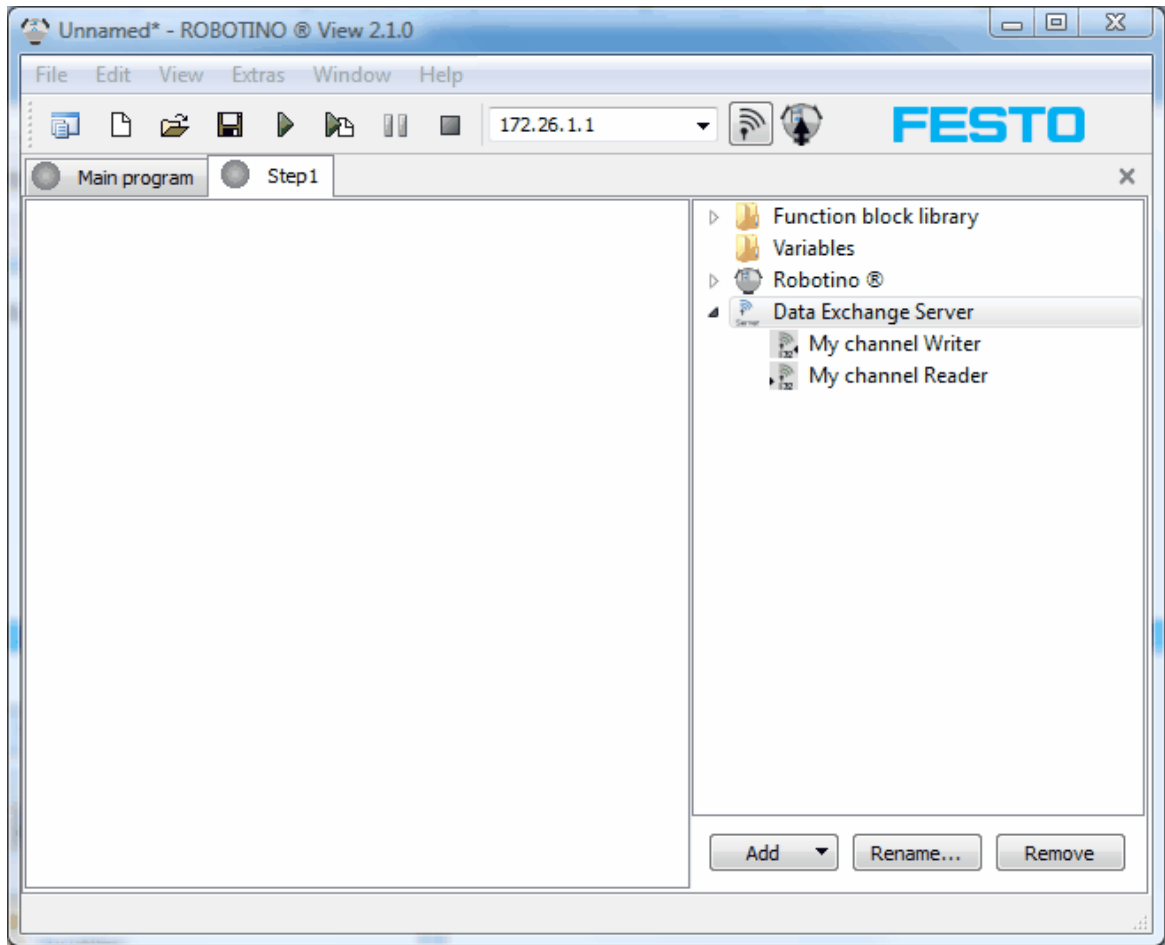
After adding the data exchange server device to the function block library the communication channels can be added via the servers context menu.



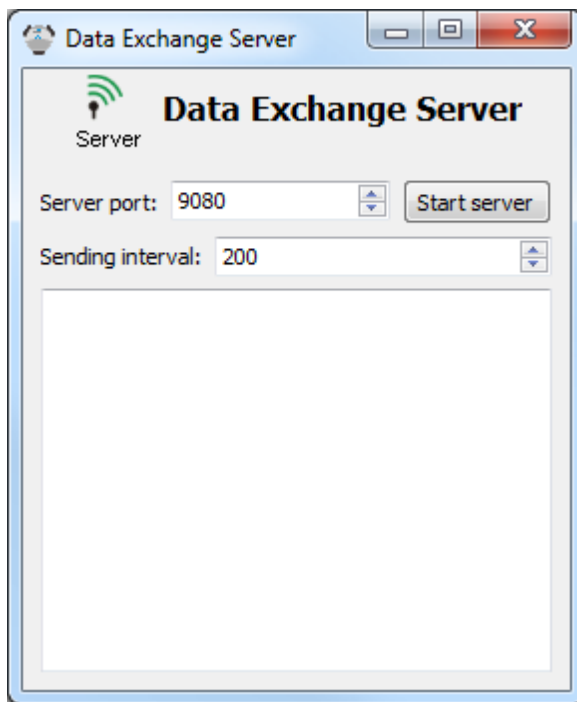
A dialog is displayed to enter the name and the type of the new channel.



The channel's name must be unique and must contain ASCII characters excluding the "/" character only. By pressing Ok the channel is created. In the function block library two new function blocks appear named "channel name Writer" and "channel name Reader". These function blocks are used to write to or read from a communication channel.



6.7.1.1 Dialog



The data exchange server's dialog is opened by double clicking onto the device symbol in the function block library.

Server port is the TCP port the server is listening for incoming connections.

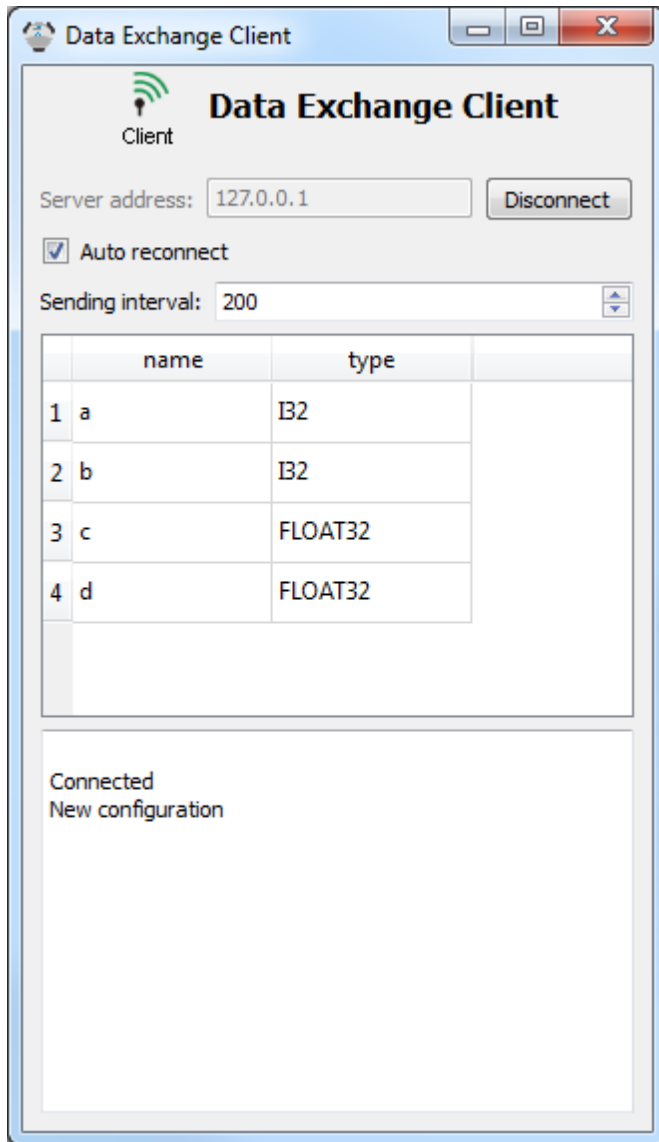
Sending interval is the time interval after a transmission that must elapse before the next transmission is permitted.

By "Start server" the server starts listening. From now on clients can connect to the server.

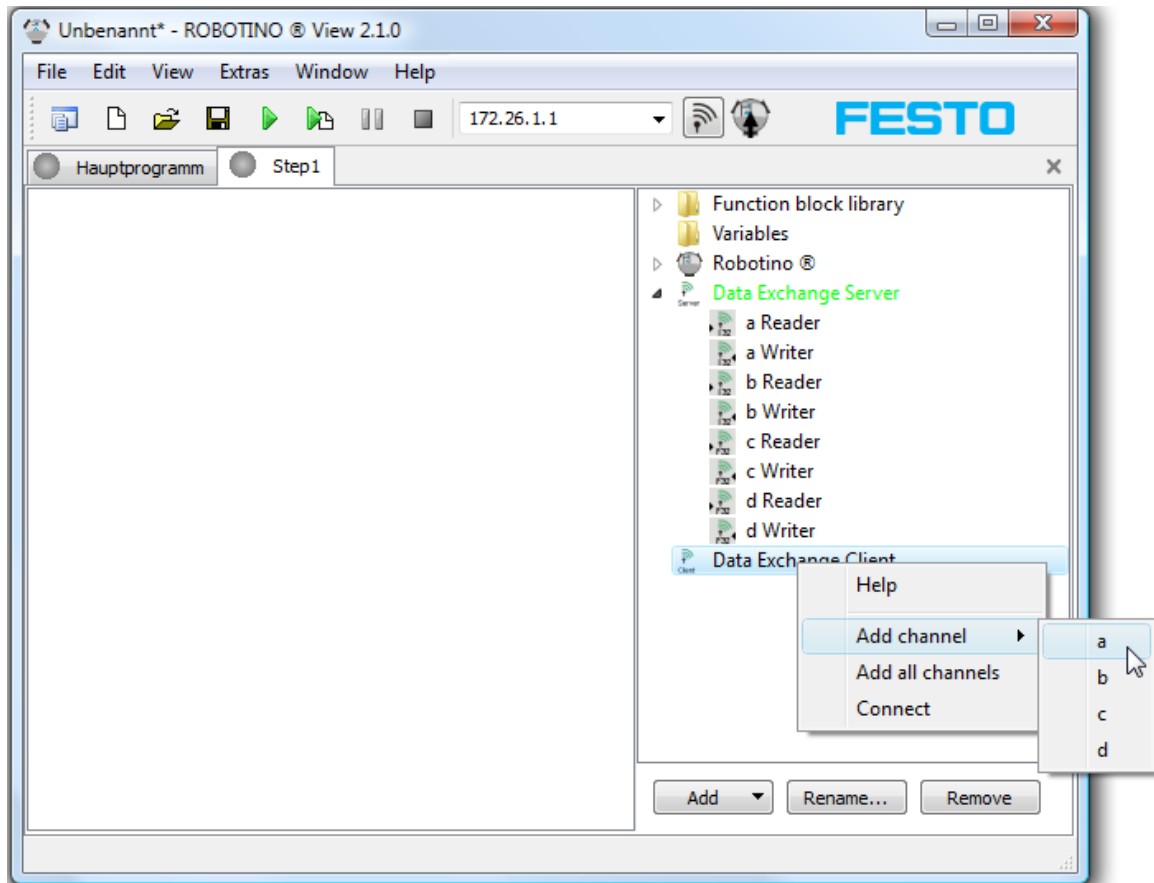
6.7.2 Client

The data exchange client connects to a [data exchange server](#)^[158]. Afterwards data can be exchange with the server using the server's communication channels.

After the client successfully connected to the [data exchange server](#)^[158] the list of communication channels is available.

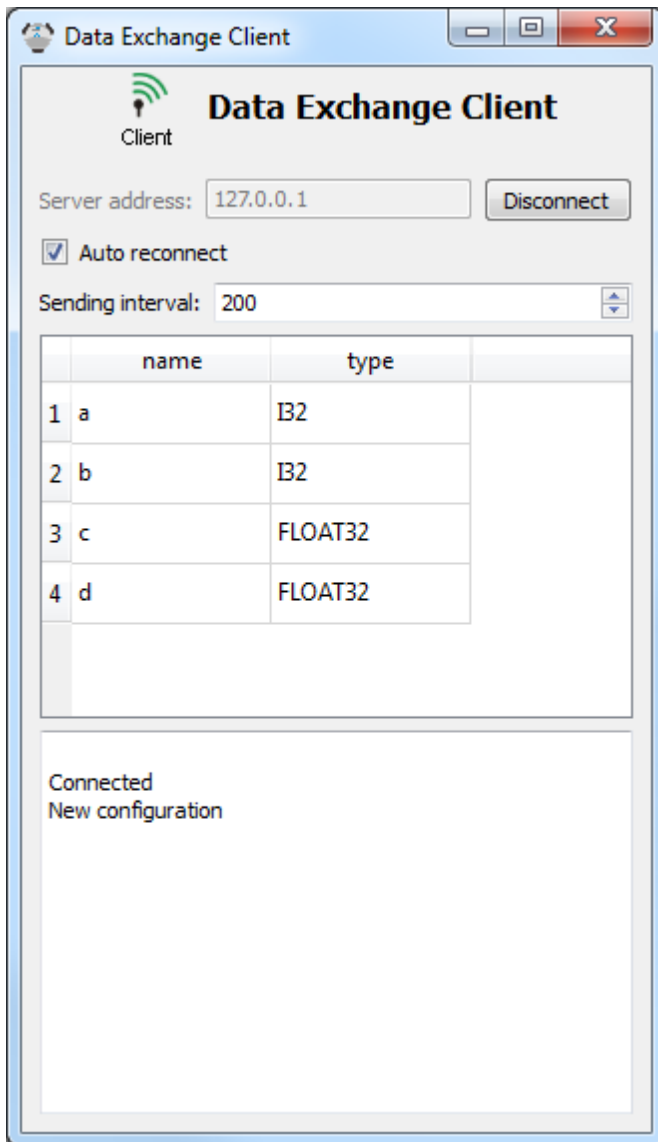


At server side the communication channels a,b of type I32 (integer with 32 bit) and c,d of type FLOAT32 (floating point with 32 bit) had been created. These channels can now be added to the client in the function block library.



As with the [data exchange server](#)^[158] the function block library shows two function blocks after adding a channel to the client. Via the client's context menu channels can be added one by one or all at once. Using the "Connect" entry from the context menu the connection to the server can be established without using the client's dialog.

6.7.2.1 Dialog



Server address is the IP address of the server the client wants to connect to. If only the IP address is given the connection is established using the server's default port 9080. If the server is listening on a different port the port number can be specified after the IP address separated by a ":".

If the server is listening on the local host at port 8000, the clients server address should be 127.0.0.1:8000.

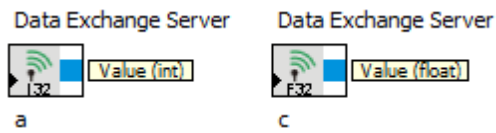
If "Auto reconnect" is active the clients tries to establish a new connection after the current connection goes down.

Sending interval is the time intervall after a transmission that must elapse before the next transmission is permitted.

6.7.3 Function blocks

The function blocks are used to exchange data with the devices.

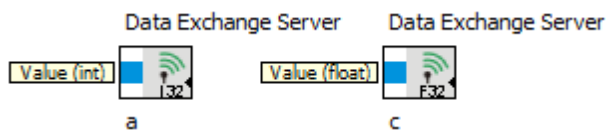
6.7.3.1 Reader



The Reader reads data from a communication channel.

Outputs	Type	Description
Value	int, float, float array, laser range data	The value of the communication channel.

6.7.3.2 Writer



The Writer writes data into a communication channel.

Inputs	Type	Default	Description
Value	int, float, float array, laser range data	0	The value is send to the server and the broadcasted to all clients.

6.8 UDP data exchange

With the UDP data exchange device data can be exchanged between Robotino View and external applications via UDP.

6.8.1 Protocol

Specification of the data structure

Byte	Function
0	Message ID

1-2	Number of Bytes of the whole message N. Type is UINT16 ^[167]
3	Checksum (to be initialized with 0 when the package is generated, see Checksum ^[166])
N-1	Message's last byte

6.8.1.1 Checksum

If the message is shorter than 100 byte, the sum s0 will be calculated from the whole package's single bytes. If the message contains 100 bytes or more, s0 will be calculated from the message's first and the last 50 bytes.

In both cases the checksum byte must be initialized with 0. The checksum is calculated to

checksum = 0xff - s0

```
unsigned char checksum( const unsigned char* payload, unsigned int payloadLength ) const
{
    unsigned char s0 = 0;

    if( payloadLength < 100 )
    {
        for( int i = 0; i < payloadLength; ++i )
        {
            s0 += payload[i];
        }
    }
    else
    {
        for( int i = 0; i < 50; ++i )
        {
            s0 += payload[i];
        }
        for( int i = payloadLength-1; i >= payloadLength - 50; --i )
        {
            s0 += payload[i];
        }
    }

    return ( 0xFF - s0 );
}
```

To check if the package has been transmitted correctly, the whole message's single bytes will be accumulated to the byte sum s1 if the message is shorter than 100 byte. If it contains 100 bytes or more, s1 is calculated from the message's first and last 50 bytes.

The package is correct if

s1 = 0xFF

6.8.1.2 Data types

Type	Width in bytes	Description
UINT16	2	Byte0: low Byte1: high On a little endian system a UINT16 data value can be copied directly into the payload. Example: <pre>//encoding uint16 value = 9873; char payload[2]; uint16* p = reinterpret_cast<uint16*>(payload); *p = value; //decoding value = *(reinterpret_cast<const uint16*>(payload));</pre>
INT32	4	Byte0: low Byte3: high On a little endian system a INT32 data value can be copied directly into the payload. Example: <pre>//encoding int32 value = -3459873; char payload[4]; int32* p = reinterpret_cast<int32*>(payload); *p = value; //decoding value = *(reinterpret_cast<const int32*>(payload));</pre>
UINT32	4	Byte0: low Byte3: high On a little endian system a UINT32 data value can be copied directly into the payload. Example: <pre>//encoding uint32 value = 3459873; char payload[4]; uint32* p = reinterpret_cast<uint32*>(payload); *p = value; //decoding value = *(reinterpret_cast<const uint32*>(payload));</pre>

6.8.1.3 Message 0

Byte	Function
0	0
1	36
2	0

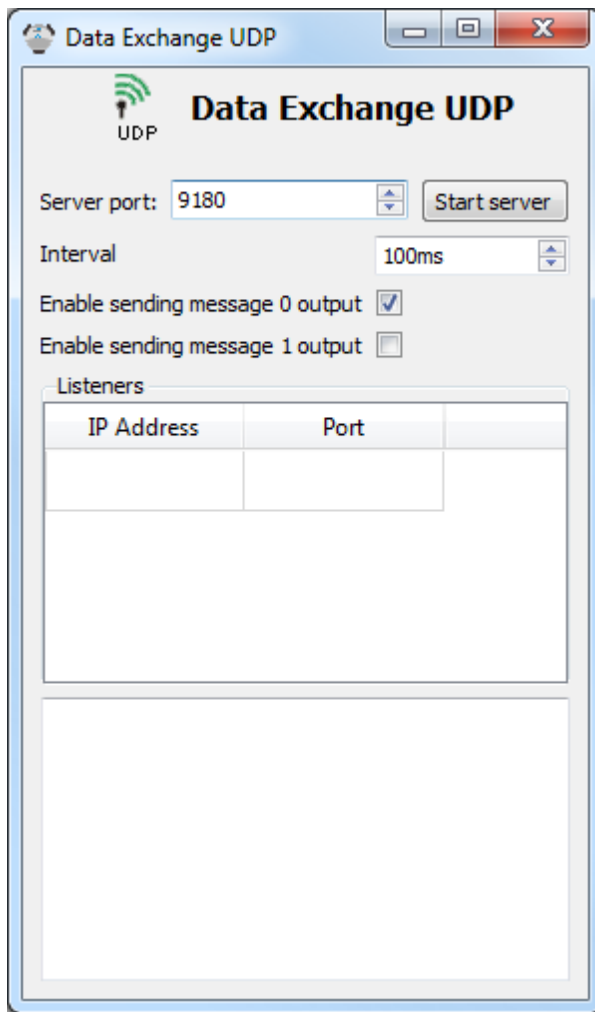
Devices

3	Checksum ¹⁶⁶
4-7	INT0 of type INT32 ¹⁶⁷
8-11	INT1 of type INT32 ¹⁶⁷
12-15	INT2 of type INT32 ¹⁶⁷
16-19	INT3 of type INT32 ¹⁶⁷
20-23	INT4 of type INT32 ¹⁶⁷
24-27	INT5 of type INT32 ¹⁶⁷
28-31	INT6 of type INT32 ¹⁶⁷
32-35	INT7 of type INT32 ¹⁶⁷

6.8.1.4 Message 1

Byte	Function
0	1
1	36
2	0
3	Checksum ¹⁶⁶
4-7	INT0 of type INT32 ¹⁶⁷
8-11	INT1 of type INT32 ¹⁶⁷
12-15	INT2 of type INT32 ¹⁶⁷
16-19	INT3 of type INT32 ¹⁶⁷
20-23	INT4 of type INT32 ¹⁶⁷
24-27	INT5 of type INT32 ¹⁶⁷
28-31	INT6 of type INT32 ¹⁶⁷
32-35	INT7 of type INT32 ¹⁶⁷

6.8.2 Dialog



The dialog of the UDP data exchange device can be opened by double-clicking on the device entry in the function block library.

In the dialog both sending and receiving UDP datagrams can be configured:

With "Server port" the UDP port number at which the server listens for datagrams and from which datagrams are sent is configured.

With "Start server" the server is started. Once the server has been started, UDP data packages are received and interpreted and sent.

"Interval" is the time interval after a transmission that must elapse before the next transmission is permitted.

For each message (message 0 or message 1) sending can be turned on and off individually.

IP addresses and ports of data receivers can be entered into the "Listeners" table. If no port is specified, port 9180 will be used by default.

6.8.3 Function blocks

The function blocks are used to exchange data with the devices.

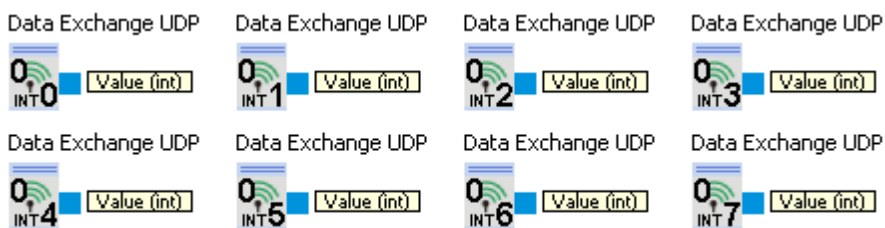
6.8.3.1 Message 0

The function blocks in category Message 0 allow sending and receiving data.

6.8.3.1.1 Input

The inputs of message 0 provide received values.

6.8.3.1.1 Reader



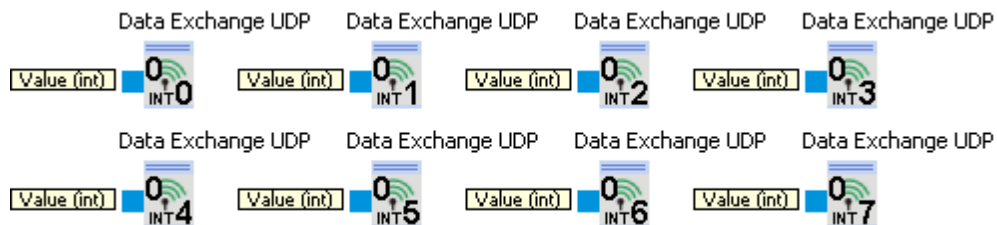
The reader reads data and outputs received data. There is a reader for each of INT0 to INT7.

Outputs	Type	Description
Value	int	The received value

6.8.3.1.2 Output

The outputs are used to send values.

6.8.3.1.2 Writer



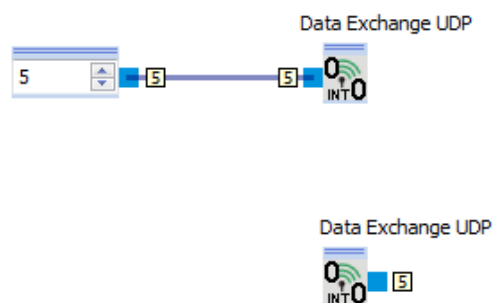
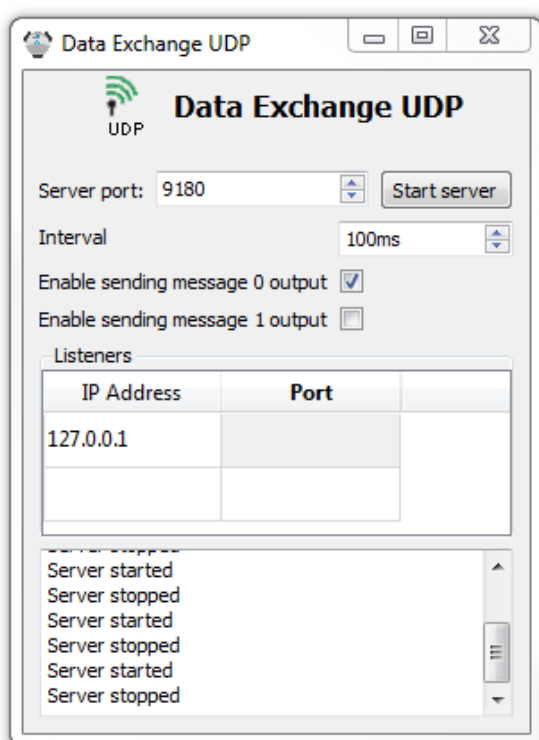
The writer takes the data to send and passes them to the device to send it to the receivers via UDP. There is a writer for each of INTO to INT7.

Inputs	Type	Description
Value	int	The value to send

6.8.3.2 Message 1

Message 1 is identical to [Message 0](#)¹⁷⁰.

6.8.4 Example



7 Programming

To compile function blocks and devices the Robotino® View 2 API is necessary.

7.1 My function blocks

You can find the following examples in

%ProgramFiles%\Festo\RobotinoView2\units\robview\MyFunctionsBlocks

or respectively

%ProgramFiles(x86)%\Festo\RobotinoView2\units\robview\MyFunctionsBlocks

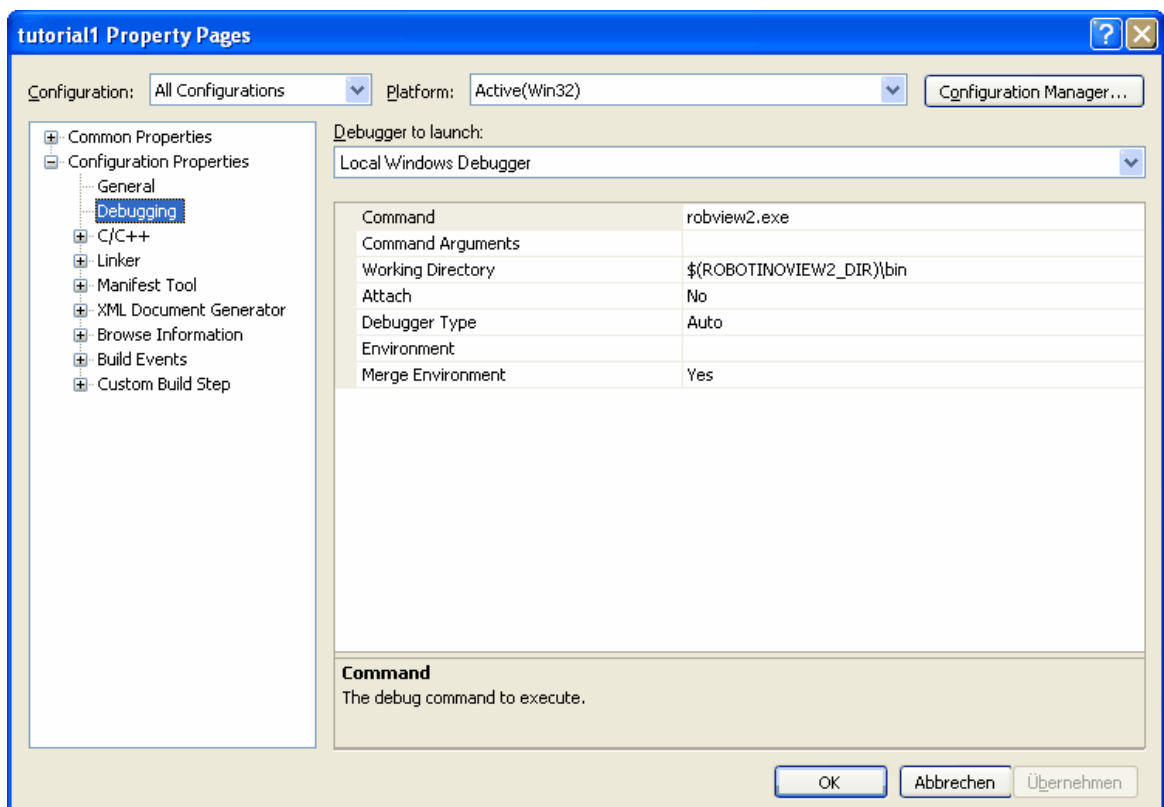
on 64 bit systems. The environment variable %ProgramFiles% stores the path to the installed application. Normally this is "C:\Program Files".

Before opening Visual Studio Solution tutorialx.sln you should run the script

RUN THIS FIRST THEN START VS.cmd

from the current tutorial folder. The script generates user specific settings, that can not be stored in the sln file and enable debugging of function blocks.

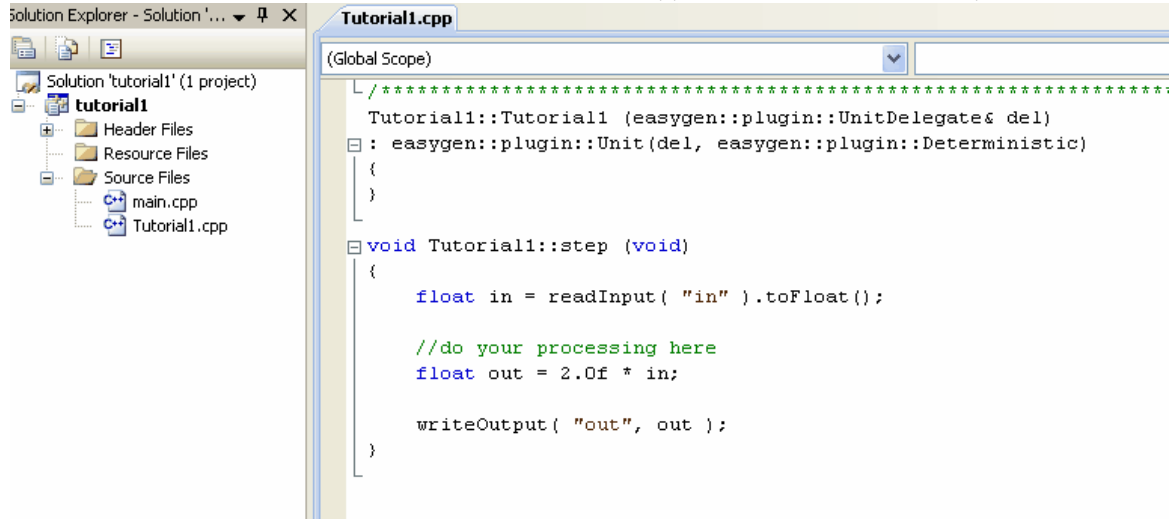
For the debugging to work, Robotino View 2 must be specified as executable with the correct working directory in the project settings as shown below. This settings will be set correctly automatically if you have executed "RUN THIS FIRST THEN START VS.cmd" before as described above.



7.1.1 Tutorial 1

Folder: tutorial1.unit

This tutorial explains how to build a function block with one input and one output connector. The relevant code can be found in Tutorial1.cpp in the step() function.



```

Solution Explorer - Solution '...' X
Tutorial1.cpp
(Global Scope)
/*****
Tutorial1: Tutorial1 (easygen::plugin::UnitDelegate& del)
: easygen::plugin::Unit(del, easygen::plugin::Deterministic)
{
}

void Tutorial1::step (void)
{
    float in = readInput( "in" ).toFloat();

    //do your processing here
    float out = 2.0f * in;

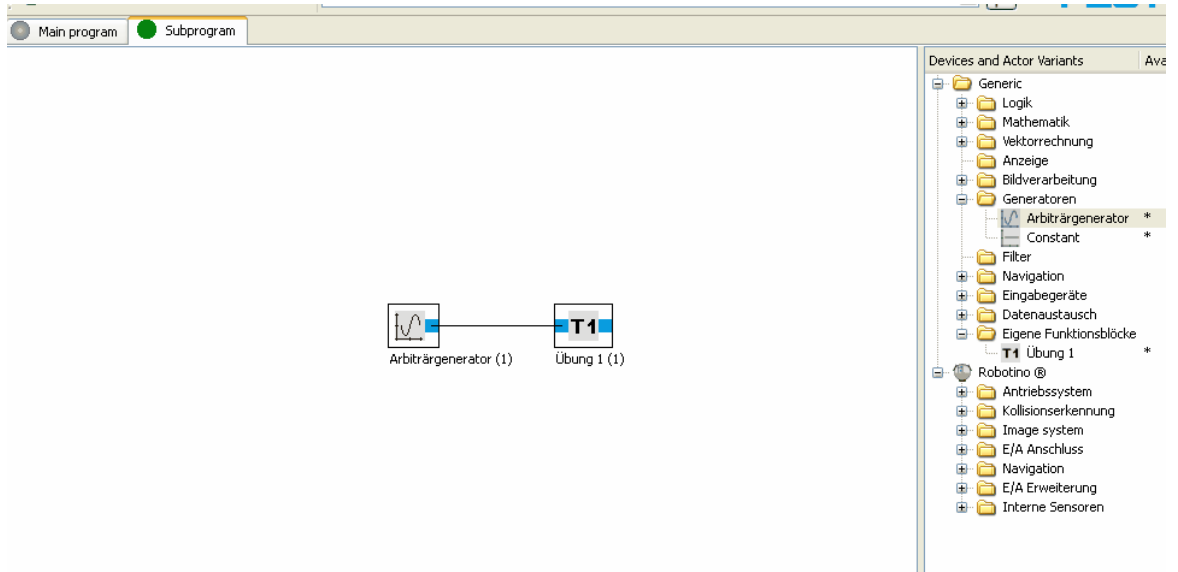
    writeOutput( "out", out );
}
*****/

```

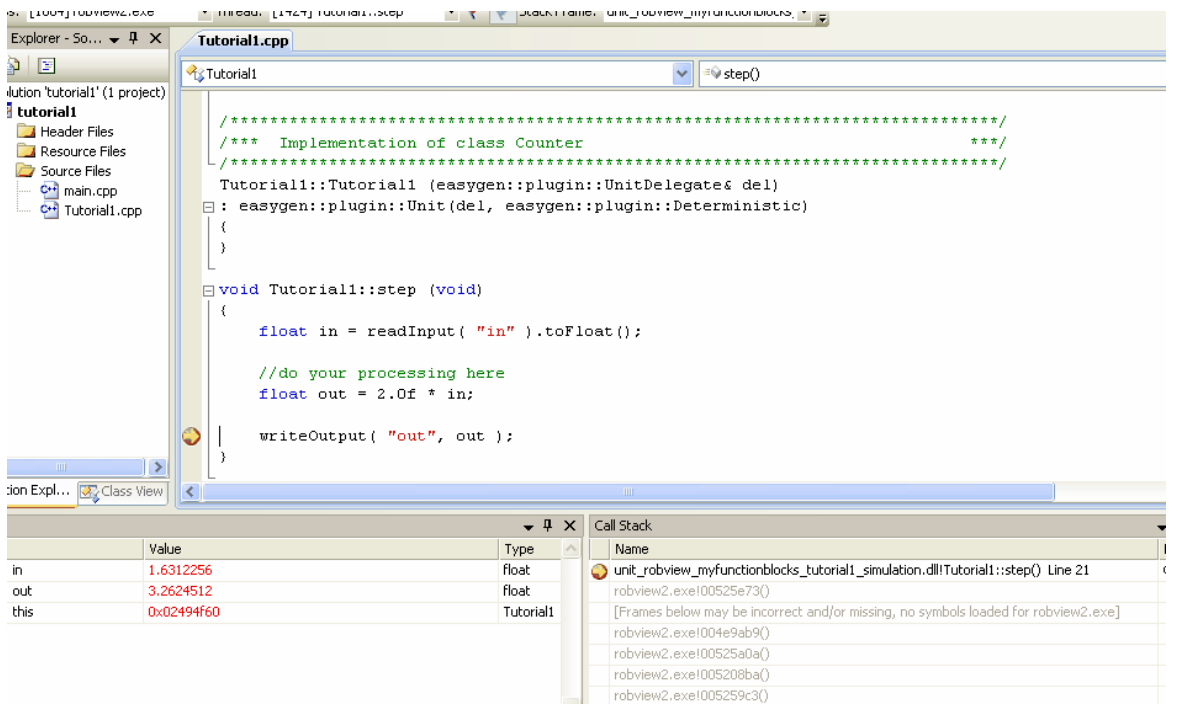
The input value "in" is multiplied by 2 and then written to the output. Do whatever you like here. To see what happens in your code start the debugger by pressing the F5 key. The function block is compiled and linked and Robotino View 2 is started. Please ignore the dialog that Robotino View 2 does not contain debugging information. As you do not want to debug Robotino View 2, but only your function block, this message is irrelevant.

Create a sub-program in Robotino View 2 containing the Tutorial 1 function block from My function blocks. Start the simulation of the sub-program.

Programming



Place a break point in your step() method.



- A -

ABS 60
 absolute value 60
 add devices 121
 addition 57, 71, 74
 analog input 138
 AND 41, 43, 45, 47, 48, 49, 50, 51, 52, 53
 AND FL 43
 arrays 67

- B -

bumper 129

- C -

C++ 173
 camera 134, 151
 cartesian 75, 76
 changes 8
 client 154, 158, 165
 color space 91
 compare 58, 59, 60
 connect to Robotino 18
 constant 93, 101
 control panel 116
 Cosine wave 92
 counter 35, 38
 create function block in C++ 173

- D -

data exchange 118, 154, 158, 165
 devices 121, 123, 150
 devision 73
 digital input 139
 digital output 137
 display 77
 distance 130
 division 54

- E -

encoder input 147
 equal 58, 59, 60
 example 26, 34, 173

- F -

filter 96

firmware 25

FlipFlop 34, 52

function 60, 61, 64, 65, 66

function block 14, 34, 35, 38, 39, 40, 41, 43, 45,
 47, 48, 49, 50, 51, 52, 53

function block connection 14

function blocks 124

- G -

generator 91, 92, 93, 94, 95

getting started 13

global variables 15, 121

greater 60

gripper 149

- I -

image information 89

image processing 79, 83, 85, 87, 89, 91

input 116, 117, 129, 130, 138, 139, 147, 149, 150

install 8

- J -

joystick 150

- K -

keyboard shortcuts 19

- L -

language 9

latching relay 34, 52

length 72

less 59

line detector 85

load program 14

- M -

math function 54, 55, 56, 57, 58, 59, 60, 61, 64,
 65, 66, 67

maximum 65

mean filter 96

minimum 64, 65

Modulo 54

motor 125, 128

multiplexer 39, 40

multiplication 55, 74

- N -

NAND 45
 NAND FL 47
 navigation 96, 101, 102, 103, 104, 106, 114, 140, 141, 142
 new project 13
 NOR 51
 norm 72
 North Star 142
 NOT 41, 43, 45, 47, 48, 49, 50, 51, 52, 53

- O -

obstacle avoidance 114
 odometry 141
 omnidrive 128
 OPC 154, 158
 operating sytem 25
 OR 41, 43, 45, 47, 48, 49, 50, 51, 52, 53
 oscilloscope 77
 output 135, 137, 148, 149

- P -

path 104, 106, 114
 path driver 106
 polar 75, 76
 pose 101, 102, 103
 position 96
 position driver 96
 power management 149
 power output 148, 149

- R -

random 95
 reagon of interest 87
 relay 135
 robotino 123, 125, 128, 129, 130, 134, 135, 137, 138, 139, 147, 148, 149, 151
 ROI 87
 rotate 76
 RS 52
 RS-FlipFlop 34, 52

- S -

Sample and Hold 53
 scalar 69, 74

scale function 66
 scope 77
 segment 83
 segment extractor 83
 segmenter 79
 sensor 130, 134, 151
 server 154, 158, 165
 shortcuts 19
 Sine wave 92
 slider 117
 Square wave 92
 sub program 16
 subtraction 70, 73
 subtraction 56

- T -

terminology 13
 timer 94
 transfer function 61
 Triangle wave 92
 tutorial 26, 34, 173
 type conversion 20

- U -

UDP 165
 uninstall 8
 updates 8, 20, 25
 upload project 20

- V -

variable 101
 vector 69, 70, 71, 72, 73, 74, 75, 76

- W -

waveform generator 92
 workspace 9

- X -

XOR 41, 43, 45, 47, 48, 49, 50, 51, 52, 53