



# ST Robotics

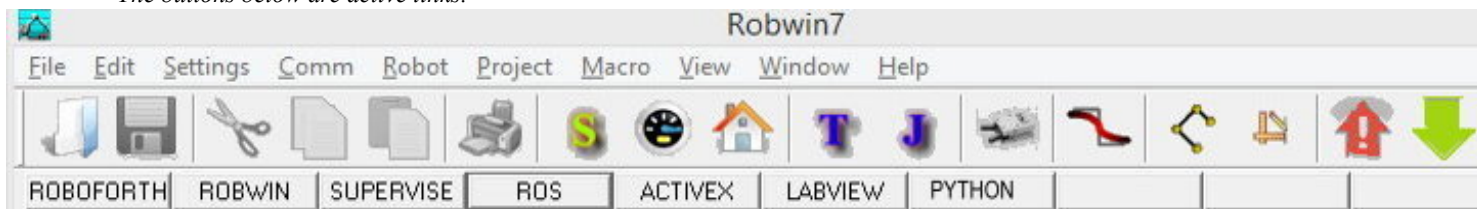
## Software



*"Almost anything in software can be implemented, sold, and even used given enough determination. There is nothing a mere scientist can say that will stand against the flood of a hundred million dollars. But there is one quality that cannot be purchased in this way - and that is reliability. The price of reliability is the pursuit of the utmost simplicity." - from 'The Emperor's Old Clothes' by C.A.R.Hoare*

- [ROBOFORTH](#) is the software that runs in the robot controller. It's a [fourth generation language](#). It uses simple natural language to describe what you want to do.
- It's interactive. You can type a command and get immediate movement from the robot, or action on the I/O such as valves or instruments.
- It is *not* graphical. Graphical programming looks good but is restricted to dialog boxes and buttons deemed important by the manufacturer/programmer who can only cater for known applications. But Forth is a complete programming language so whatever comes up there is always a way to do it.
- It's human - using English language words that are used like building blocks to define more powerful words. It is a completely different paradigm from syntax languages like C or Python.
- It has extensive I/O to control and interface with external devices, synchronized with robot motion if required together with structured programming of IF statements and loops.
- Easy positional programming (teaching) - named positions and paths, relative and absolute positions, Cartesian or joint control, palletizing, continuous path and a simple teachpad.
- [RobWin](#) is the computer GUI project management software under Windows that brings all the features of ST robots together on one screen in a 'tethered' programming environment.
- [LabView](#), [Matlab](#) - fully supported.
- [C](#), [Python etc.](#) You can access the system by setting up a simple protocol or with ST's ActiveX modules. Recommended practice is to program the robot's basic moves in RoboForth then call them up from your C or other supervisor.
- [ROS](#) Code for Robot Operating System.

The buttons below are active links.



## ROBOFORTH II

***The most extensive and powerful robot control system on the planet***

ROBOFORTH is a [4GL](#) robot control language designed to cover every eventuality in programming your robot, whether it be coating, assembly, laboratory handling, testing, or whatever. There are various means of acquiring spatial data and using it such as matrices, continuous path, object tracking, collision avoidance, plus numerous input-output features making it easy to interface with and control peripheral equipment at the same time as control the robot arm. Above all Roboforth is a real language i.e. has an extensive vocabulary of words, which, like a natural language, help you express your ideas to the system. As a language Roboforth is a means of communication between the human user and the robot arm: for the user to tell the robot what to do and how to do it; for the robot to tell the user what it is doing and what it knows.

Forth and Roboforth are organized as a linked list of words known as the dictionary. Together with each word

in the dictionary is its definition, which is usually in terms of simpler words lower down just like a regular dictionary. There are over 500 words in Roboforth available to the user but you typically only need a small vocabulary for any particular application. Programming in Forth and Roboforth consists of creating new words into the dictionary whose definitions are in terms of words already defined. New words may include data e.g. positional information about the robot. Values (or arguments) are passed between words on a stack.

Any given position of the arm may be specified as coordinates relative to a central position for the arm known as the HOME position. In ROBOFORTH these coordinates are in terms of the movement required by the motors of the arm to reach that position i.e. number of motor 'bits' counted from the HOME position. More usually they are expressed in millimeters in Cartesian X,Y,Z coordinates. To teach the robot you first move the arm to the required position using any of the means provided and secondly record the coordinates of that position. This record has a header which is another entry in the dictionary. When replayed the arm moves from one position to the next in turn (point-to-point programming). Motion through these points can be smooth and optionally at constant velocity. The gripper and other connected devices may also be operated or told when to operate in the program.

Any activity may be tested in immediate mode or included in the definition of a new word of your own choosing. Error conditions can be re-programmed so, for example you can determine what the robot does and what happens to the associated equipment in the event of the stop button being pressed or in the event of a stall because of jammed product. These and other features make ROBOFORTH the most powerful robot control system in existence. Under development for 3 decades it is vast yet is still the easiest robot language to get started with and to accomplish any robot task whether simple or complex.

While **Roboforth** runs in the robot controller its counterpart, **RobWin** runs in the computer. RobWin is only used for programming and may be dropped once the robot is programmed.

RobWin provides a communications window through to the controller but also allows the user to quickly access many of the features of Roboforth by clicking buttons and by completing 'dialog boxes' and maintains disk files.

ST can arrange to do the programming for you but it is far better for you to learn the system yourself. It puts it under your control and you can revise it as you revise your process. ST ensures this with acclaimed manuals, 24/7 phone and email [support](#).

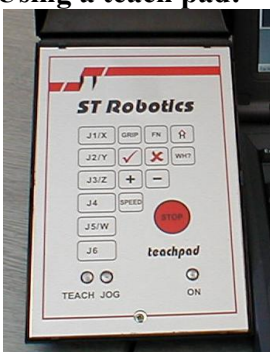
*"Tell me and I will forget, show me and I will remember, involve me and I will understand" - Aristotle*

## Moving the robot and teaching the robot are two separate operations.

### Moving the robot

may be achieved with any or a combination of four methods:

#### 1. Using a teach pad:



After clicking the teach button on screen simply select a joint or axis on the teach pad then press the + key to move in one direction or - in reverse; movement continues as long as the key is pressed. Low speeds are appropriate for accurate positioning and speed may be changed up or down on screen or from the teach pad.

#### 2. Commanding each joint:

Commands such as TELL SHOULDER 1000 MOVE which moves the shoulder axis 1000 counts. Other commands are: MOVETO (an absolute command), GRIP, UNGRIP, HOME, CARTESIAN, JOINT, ALIGN/NONALIGN and so on. There are about 250 similar commands and settings in Roboforth and a total vocabulary

of around 800 words.

#### 3. Positioning in Cartesian coordinates: (reverse kinematics)

Dialog boxes permit positioning in absolute or relative coordinates. Alternatively MOVE or MOVETO commands may be entered or included in a program. Wrist/hand angles may be specified or controlled. Tool

transformations depend on robot model and number of axes.

#### 4. Jog.

The teach pad may be used in 'Jog' mode in which the keys represent X, Y, Z, pitch and roll axes. Each press of the + or - key causes the robot to move a set increment in the selected direction. The increment may be reduced on screen or from the teach pad as the robot nears the target position.

#### 5. Android Bluetooth teach console

This also works in 'Jog' mode using a touch screen. It includes additional functions such as constant display of position, world and tool coordinate systems.



### Teaching the robot

may be achieved by choosing one of two 'entities':

#### 1. A Place:

This is a single named coordinate. It is self learning and self executing. It is created by the user with his/her given name using a dialog box or with a native command e.g. PLACE JIG.

To return the robot to this position later simply use the word JIG.

#### 2. A Route:

This is a list of spatial coordinates each of which is a row of numbers representing motors counts away from the zero (HOME) position or absolute Cartesian coordinates. The route is created by the user with his/her given name e.g. ROUTE I66. RobWin makes this simple with a dialog box that creates the entry in the controller as well as on disk. Coordinates are then added to the list by clicking 'insert position' or using the tick key on the teach pad. The robot moves from point to point with the command RUN, moving through them at optimum speed or at constant velocity. Associated commands provide editing and the ability to run parts of a route, or to retrace. A route is also used as a reference for discrete positions, for palletizing for example. Editing is achieved using dialog windows or with ROBOFORTH commands such as REPLACE which also permit self learning features, for example the robot can modify its own positions according to the programmed procedure. Functions such as gripper operation, delays, speed changes, spray/glue on/off etc. can be embedded in the route to take effect in the required sequence.

A **grid** (matrix) is also a list, organized in 2 dimensions and a **row** is one-dimensional. The number of rows and columns are specified in dialog boxes, the corners of the matrix learned and the system computes the rest of the positions and downloads them to the controller. This is useful accessing trays of items etc.

**Finally** all these learned and named entities are used in the procedure file to create new definitions or 'words' which determine how the positions are used, i.e. in what order, in what circumstances etc. For example a word might be defined using a PLACE named JIG and a matrix route named TRAY:

```
: GETPART
TRAY INTO
GRIP
UP
JIG
UNGRIP
WITHDRAW
;
```

Because Forth and Roboforth are stack oriented the position within the TRAY is given on the stack just before the command, for example

```
5 GETPART
```

will get a part from position 5 of the TRAY and put into the JIG

This may be typed as a command into the communications window, or sent from a supervisor, which computes the position number, or further compiled into an even higher level definition for example a loop that works through the whole tray.

Extensive **input-output** is also provided, which is easily programmed and integrated with robot motion, for example suppose a pump were connected to parallel port PA on bit 5. You could add a definition thus:

```
: PUMP PA 5 ;
```

You can then control the pump with simple PUMP ON and PUMP OFF. This could then be included in your higher level definition.

And maybe you want to only do this on a signal from a sensor connected to port PB bit 7:

```
: SENSOR PB 7 ;
: GETPART
SENSOR 1 WAIT
TRAY INTO
GRIP
UP
JIG
UNGRIP
WITHDRAW
PUMP ON
5000 MSECS
PUMP OFF
;
```

See how easy it is: [How-to videos](#)

---

## Advanced features

**Vactored execution:** allows post-programming Roboforth functions such as what happens on pressing the stop button.

**Programmable speed,** acceleration, rate of acceleration and emergency deceleration.

**Relative motion:** - Teaching a single motion path that can be repeated anywhere in the workspace. If you have the same motion required in a number of different places, for example to pull out sliding shelves from a shelf unit, START-HERE and END-THERE commands allow you to teach the robot one route that can be run from or to different starting or ending positions without having to teach a new path or route for each shelf. The same applies to lists of reference positions that can also be shifted around the workspace without reprogramming.

**Machine/instrument interface:** Robot motions maybe synchronized to or interlocked with inputs from/ outputs to other equipment.

**Constant speed:** program a path in equal length segments and set a fixed time for each segment.

**Curves/circles** at constant angular speed.

**Repeat** a route from the beginning without stopping, for example to keep going round a circular path.

**Elapsed and real time** delays and sync, Stopwatch function.

**Accurate real time** motions (for example like playing a piano).

**Object database** to track what objects are in what positions.

**Turnkey** operation, program restart

**Interrupts:** Real processor interrupts can also be programmed if required without any effect on robot motion. These are all programmed in Roboforth with the same ease as programming the rest of the system. An interrupt can be generated from an input or from an internal timer. For example you might want to take measurements through the analog interface every n milliseconds while the robot program is running and the robot is in motion to build a force/displacement profile for later upload to the PC. You can also embed machine code, that has a dictionary entry and can be included in other definitions.

**Sensors:** The dual processor arrangement permits one processor to control the robot while the other monitors a sensor or takes continuous measurements or to [search](#) for an event or value using the robot.

**Heuristics:** The robot can be programmed to learn it's own coordinates (self teaching), for example to modify a path you taught it so that it changes as a target position changes. A single path or route can have the same starting position but a variable ending position or vice versa, for example from different tray positions to the same instrument.

**Read back position** continuously or on demand from external program while robot is in motion.

**Straight lines** with pre-specified maximum deviation from straight.

**Recursion:** Because Roboforth is based on a real computer language your robot program can include calculations, branching, looping, [recursion](#) text, data and memory manipulation.

**Machine to machine** instructions for supervisors, Python, C++ etc.

**ROS** interface and code.

The above is just a glimpse of the immense power of this compact memory efficient system, containing hundreds of commands and unique extensions. It's a building block approach, which encourages programming interactively with the robot system, a true man-machine interface. RoboForth is faster, smaller and easier to use than any other robot programming system.

---

## RobWin7

**RobWin is a visual project management system for Windows.**

RobWin brings everything together in one screen - the robot controller and your files, the positional information and the procedures that use that information. Once programmed RobWin can be terminated and the new commands sent by other software or equipment or the controller can be set to auto-run with no computer. One window provides the user with direct communication with the robot controller, for entering commands, trying out small robot procedures before including them in the main program.

## Features

- Project Management - maintains all your programming both in the controller and on disk. As locations are learned and edited the data is built up both in the computer and in the controller. RobWin maintains the data files and the header files invisibly and provides a text editing window for the procedure file. Matrix generation is made easy and learned locations can be renamed, edited and so on all on-screen. In addition it can communicate and upload/download data. A typical screen is shown below.

- Single click commands.

RobWin generates Roboforth commands, which it sends down to the controller and they appear in the communications window along with your own.

- Single click learning.
- Integration of teach pad.
- Matrix generation (palletizing)

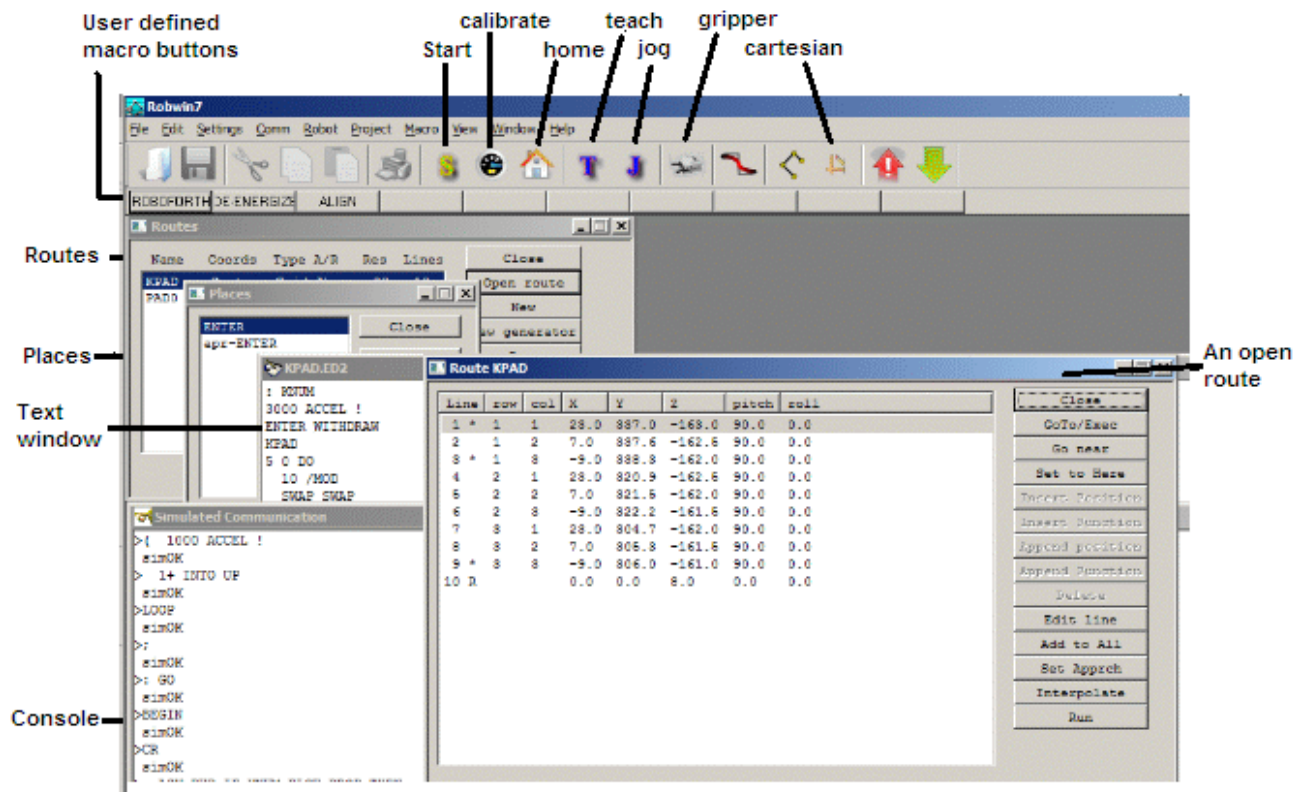
It's simple - just create the matrix with how many rows and columns you want. Then teach the system 3 corners of the matrix and press "interpolate". RobWin does the rest.

- Complex path generation.

Using the curve generator you can piece together paths comprising straight lines and curves of specified angle and radius.

**Typical RobWin7 screen:**





## Functions available

Pull-down menus from left to right:

File: open ed2 text file, download text file to controller, upload/download binary, print.

Edit: usual cut and paste etc. for the text files.

Settings: Load/save a settings file joint names, Cartesian axes, configure system.

Comm: Change baud rates also change fonts. Save what you wrote to a text file. Verify what you wrote to check for any duplicate words.

Robot: display or change position of robot in joint or Cartesian coordinates, relative or absolute moves.

Choose modes, for example smooth mode, continuous path mode.

Project: open/save project files.

Macro: you can write/edit macro commands, which are strings sent to the controller. Ten wide buttons are available for this. In the example above you can see three buttons programmed with the commands ROBOFORTH DE-ENERGIZE and ALIGN.

View: changes which tool bars show.

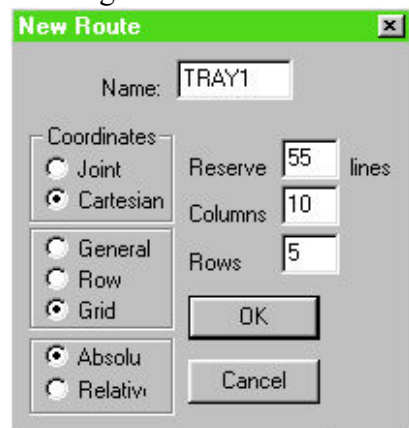
Window: usual Windows feature.

**Buttons, left to right**

Open file, save file, cut, copy, paste, START, CALIBRATE, HOME, TEACH mode - invokes the teach pad, JOG mode - invokes the teach pad in Cartesian mode, GRIP/UNGRIP (toggles), SMOOTH mode, JOINT mode, CARTESIAN mode, upload data from controller, download text to controller.

## A typical open route dialog box

showing creation of a new matrix TRAY1 in Cartesian mode, 10 columns by 5 rows.



**New Route**

Name:

Coordinates:  
☐ Joint  
☒ Cartesian

Reserve  lines

Columns

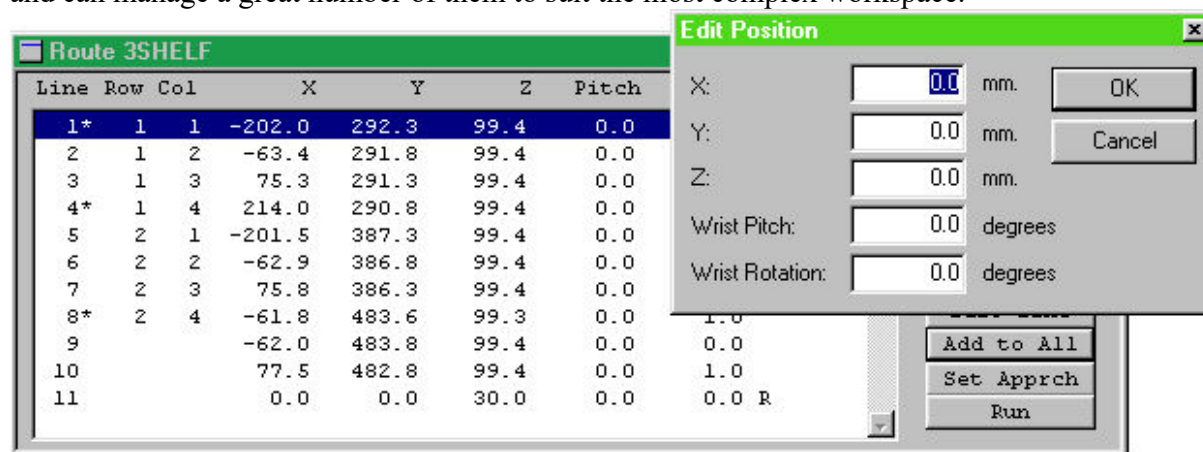
Rows

☐ General  
☐ Row  
☒ Grid

☒ Absolu  
☐ Relativ

## A typical route editing

Here a Cartesian matrix, "route" 3SHELF is being edited using add-to-all, which can shift the entire matrix in any direction. This matrix has 10 positions (could just as easily be several hundred) comprising a 4 by 2 matrix plus two extra positions plus a relative position (marked R). The system adds the relative position to the other 10 to provide another 10 displaced from the first (in this case by 30mm in the Z direction), for example as approach or via positions. This economizes on memory space. Only three corners of a matrix need be taught (marked with asterisks) then the interpolate button is pressed and all the in-between positions are computed. RobWin and Roboforth are designed to permit creation and editing of very large matrices with ease and can manage a great number of them to suit the most complex workspace.



Line	Row	Col	X	Y	Z	Pitch
1*	1	1	-202.0	292.3	99.4	0.0
2	1	2	-63.4	291.8	99.4	0.0
3	1	3	75.3	291.3	99.4	0.0
4*	1	4	214.0	290.8	99.4	0.0
5	2	1	-201.5	387.3	99.4	0.0
6	2	2	-62.9	386.8	99.4	0.0
7	2	3	75.8	386.3	99.4	0.0
8*	2	4	-61.8	483.6	99.3	0.0
9			-62.0	483.8	99.4	0.0
10			77.5	482.8	99.4	0.0
11			0.0	0.0	30.0	0.0

**Edit Position**

X:  mm.

Y:  mm.

Z:  mm.

Wrist Pitch:  degrees

Wrist Rotation:  degrees

## Programming techniques:

Routes and places may be created using RobWin. These are entered in the dictionary in the robot controller and the positional data goes into a data area that holds all the coordinates up to a maximum of 4000 coordinates. The data area is maintained simultaneously in the computer. The system creates a hidden text file for route and place names (header file). The procedure continues in a second text file that is edited in its own window and downloaded to the controller after each edit. The data in routes and places may be edited as the program is developed. Sometimes waypoints need to be added.

## Supervisors

A supervisor is software that supervises the process running in the controller, perhaps collecting data and usually controls other equipment besides the robot, supervising the entire process. Once RobWin has been used to create spatial data, define procedures, device handling strategies etc. these procedures, known as

words, may be sent to the controller via serial link, for example using PRINT commands in BASIC or via RobX ST's ActiveX interface. Proprietary supervisors may be used for example [Overlord](#) from Process Analysis Automation (PAA) which has been used successfully to manage and schedule large systems comprising diverse instruments serviced by ST robots. Using RobX you can also write your own supervisor in VB or C. A protocol is given for any other language e.g. Java or Python. Another good approach is to use [LabVIEW](#) from National Instruments using RobX. We also support Matlab and ROS.

## ActiveX

If you are writing your own supervisor you can set up your own RS232 handler (or USB/RS232 converter) or can use ST's ActiveX interface. Commands or 'methods' are of the form:

```
short OpenComm(short Port, long BaudRate);
```

Opens the communications port. The return value is 1 if successful, 0 for failure. Use the method GetCommErrorString() to get a description of the problem.

```
short SendString(LPCTSTR String);
```

Sends the string to the controller. The string should end with a "Carriage Return" character (ASCII 0D hex). Typical strings will be definitions you have programmed into the controller using Roboforth for example GETPLATE. The return value is 1 if the string was successfully sent, 0 if a communication error occurred. Use the method GetCommErrorString() to get a description of the problem. The controller must be in the "ready" state (as reported by GetStatus) before invoking this method. After sending the string invoke GetStatus() until the "ready" state is received.

```
short GetStatus();
```

Gets the controller status as follows:

- 1: Communication error
- 0: Waiting (still working on the last command)
- 1: Ready, but not OK (some robot error)
- 2: Ready, received (and completed) OK

After receiving status 1 or 2, GetResponse gets the last line sent by the controller. If status is -1, use GetCommErrorString() to get a description of the problem.

```
CString GetResponse();
```

This gets the controller response. If status was 2 the response string indicates what went wrong, for example you told the robot to go to a position it could not reach, or to put an object in a place already occupied.

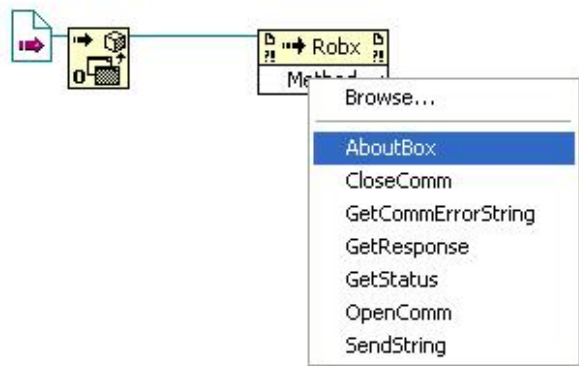
Other methods include about box, closing the comms port, getting error codes etc.

---

## LabVIEW

Controlling an ST robot from LabVIEW also involves using ActiveX. The ActiveX 'server' is easily accessed from LabVIEW. - see [How Can I Access an ActiveX Server in LabVIEW?](#).

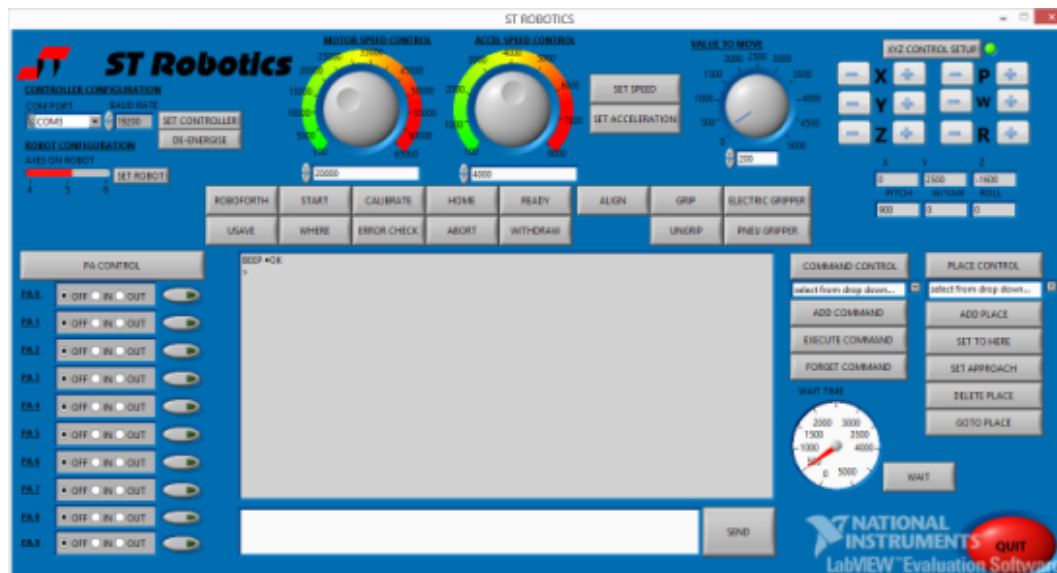




In addition a whole set of LabVIEW VIs written for us by National Instruments is included with the ST software disk.

See this [example](#) from another ST user.

## LabView based dashboard



## Matlab

Matlab examples and starter .m files are provided.

## Python

<https://pypi.org/project/r12/>

<https://github.com/OlinRoboticsAndBioinspiration/st/blob/master/st.py>

## Linux for R12

<https://github.com/adamheins/r12>

[ROS](#) ...for [R12](#) ...for [R17](#)

[A simple communications protocol from any language](#)

---

## Manuals

Software manuals are provided on disk only; hard copy is no longer used. Manuals are also online at <http://strobotics.com/manuals/manuals.htm>

---

## Forth, RoboForth and syntactic computer languages

[Fourth Generation Languages](#) (4GLs) tend to be list oriented where as third generation languages (3GLs) are syntax oriented. Examples of 4GLs are: Logo, Lisp (as used by Autodesk ACAD) and Prolog. Postscript is another example that uses postfix notation like Forth. Such languages are known as [concatenative](#). Examples of 3GLs are: BASIC, COBOL, C, Java, Python

C was created around the same time as Forth but has developed considerably more and is the preferred language for most computer applications whereas Forth is ideal for embedded systems. It puts you in direct contact with the hardware without working through an operating system.

C programmers often dislike Forth but soon appreciate the simplicity. In the words of Yoda: "You must unlearn what you have learned"

### Forth:

[Sun Microsystems](#) uses Forth for their OpenBoot implementation

The [Rosetta probe and Phlae lander](#) are programmed in Forth as are many spacecraft..

\* For more information visit [www.forth.com/forth/index.html](http://www.forth.com/forth/index.html) though ST's Forth was written and compiled by David Sands, and is not a Forth Inc. product.

This Forth and RoboForth II are copyright David N Sands, 1981 to present

\* [The Tower of Hanoi problem solved in Forth and with a robot](#)

\* [A simulated analog computer in Forth generating a sinewave without a sin function](#)