

Claude Baumann, director of the Boarding School „Convict Episcopal de Luxembourg“, Europe

Ultimate ROBOLAB (UR) is an extension to the ROBOLAB software. This RCX programming environment is thought for anyone who wishes to go beyond the features of the standard RCX firmware, but still wants to profit from the incomparable graphical programming structure offered by ROBOLAB. Ultimate ROBOLAB is more than a concurrent product to other third party text-based languages like brickOS and pbForth that allow RCX programming at deep level. Compared to those powerful environments, Ultimate ROBOLAB is much easier to learn and to manipulate. The most important difference is that Ultimate ROBOLAB directly converts graphical program code to H8 native machine byte-code passing through a generously commented Assembly code. In fact UR creates standalone firmware files that are downloaded to the RCX.

This paper tries to introduce some outstanding UR functions through easy reproducible sample programs. In order to keep the introduction comprehensive for all levels of advanced users, the examples are divided into two main categories that are labeled by:

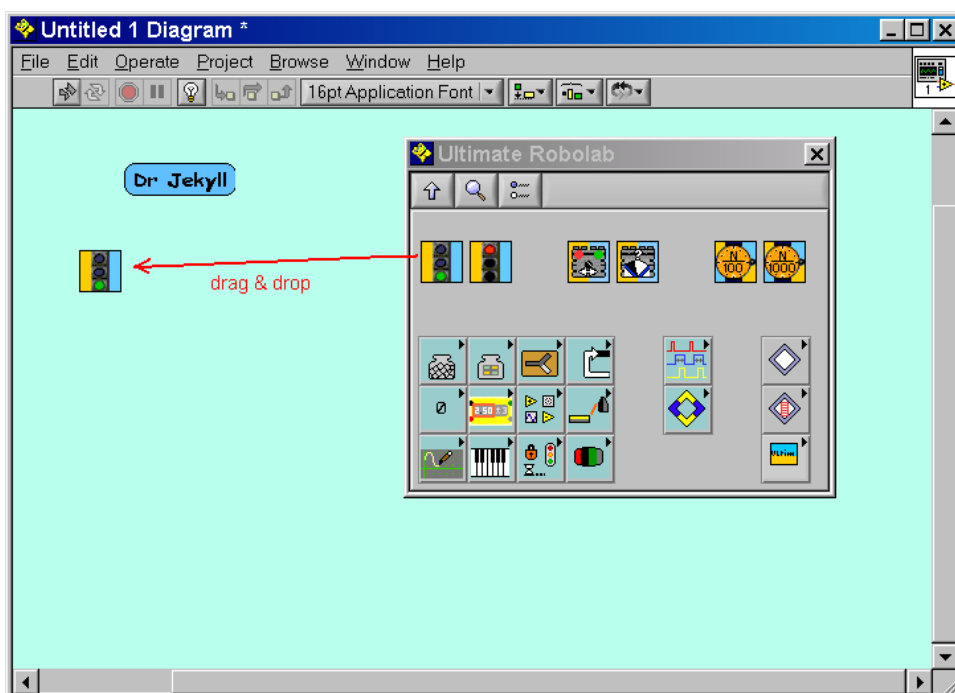
Dr Jekyll

Robert Louis Stevenson's strange scientist is a well-educated British gentleman that will stand here for any "gentle", easy understandable, graphical only source code. Nobody is frightened of Dr Jekyll.

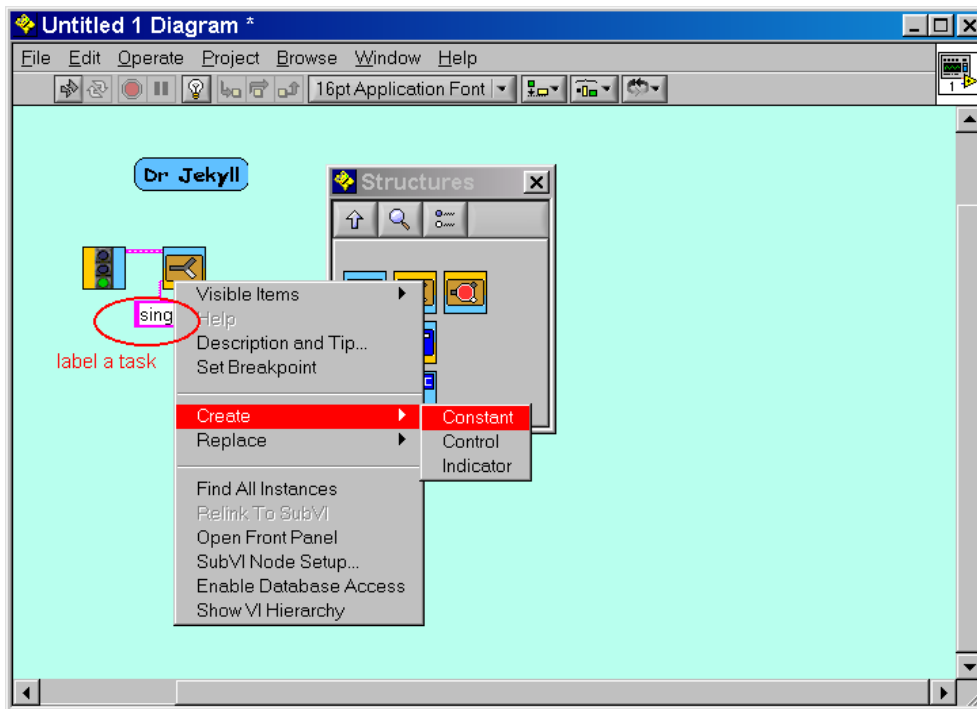
Mr Hyde

Dr Jekyll's terrific alter ego will mark any deep program part or description that should be reserved for courageous people that don't fear deep exploration and risks. Careful at that level !

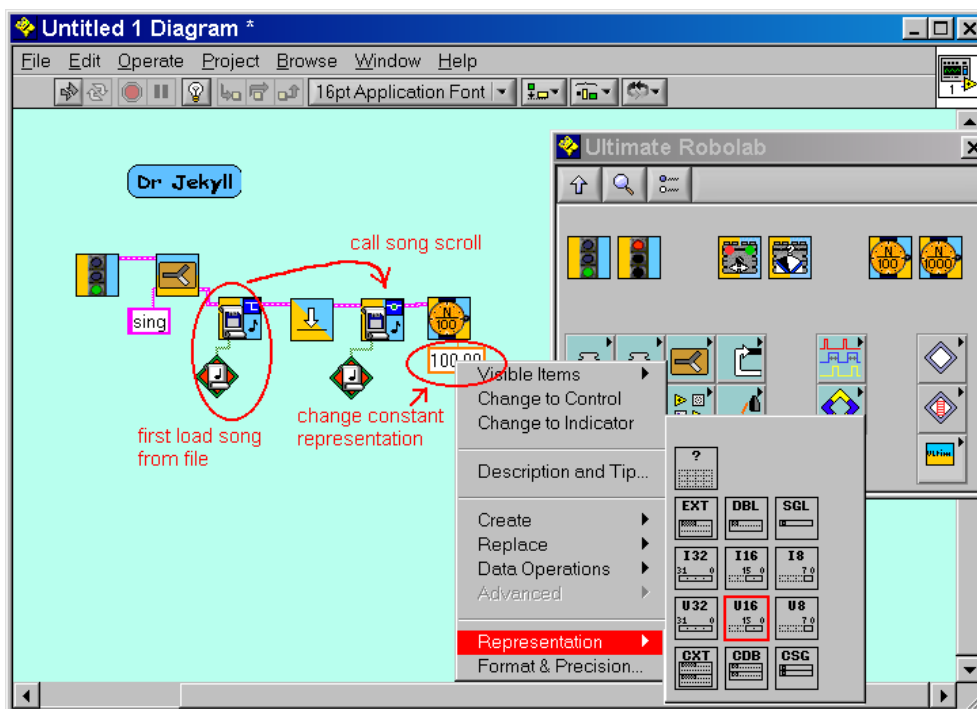
1. Flash a light, sing a song and say hello Dr Jekyll



Picture 1: Ultimate ROBOLAB program start with a special BEGIN icon.

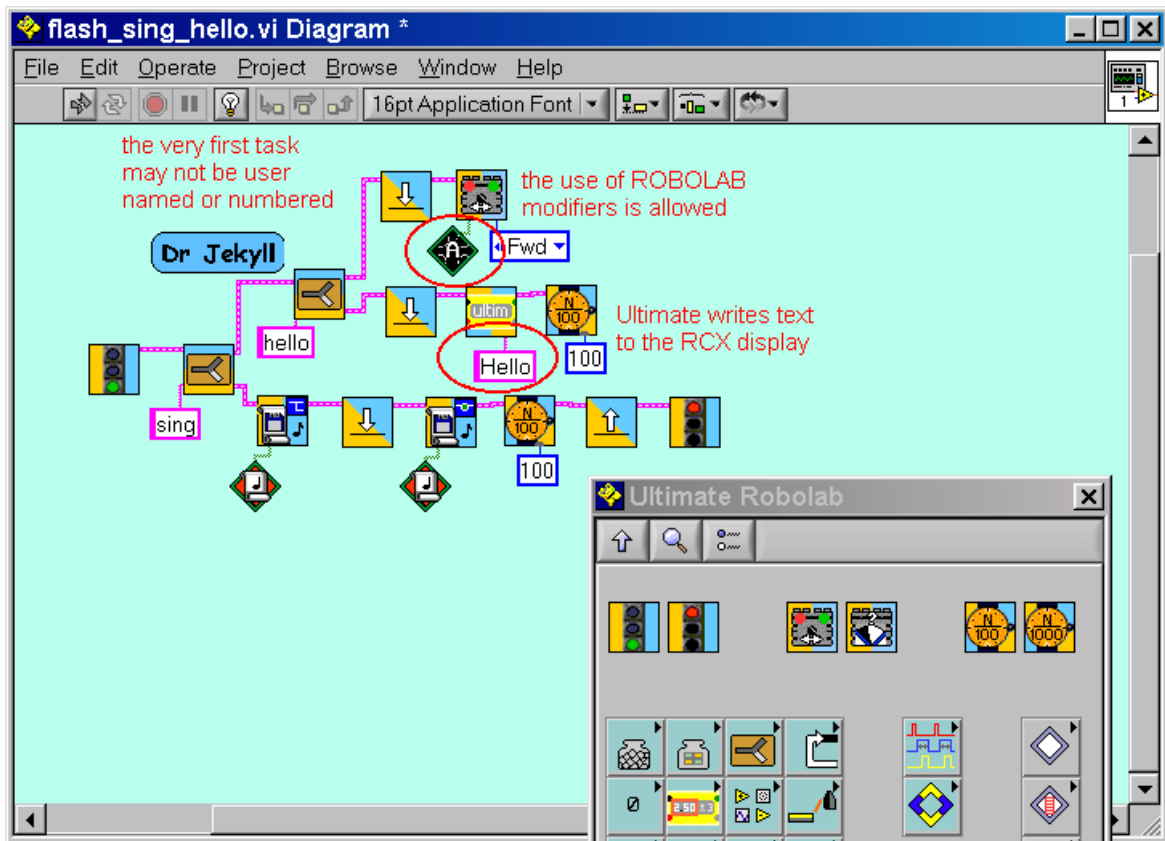


Picture 2: Ultimate ROBO LAB is built upon the LabVIEW programming environment. To get more detailed properties of any LabVIEW object, use the mouse right-click.

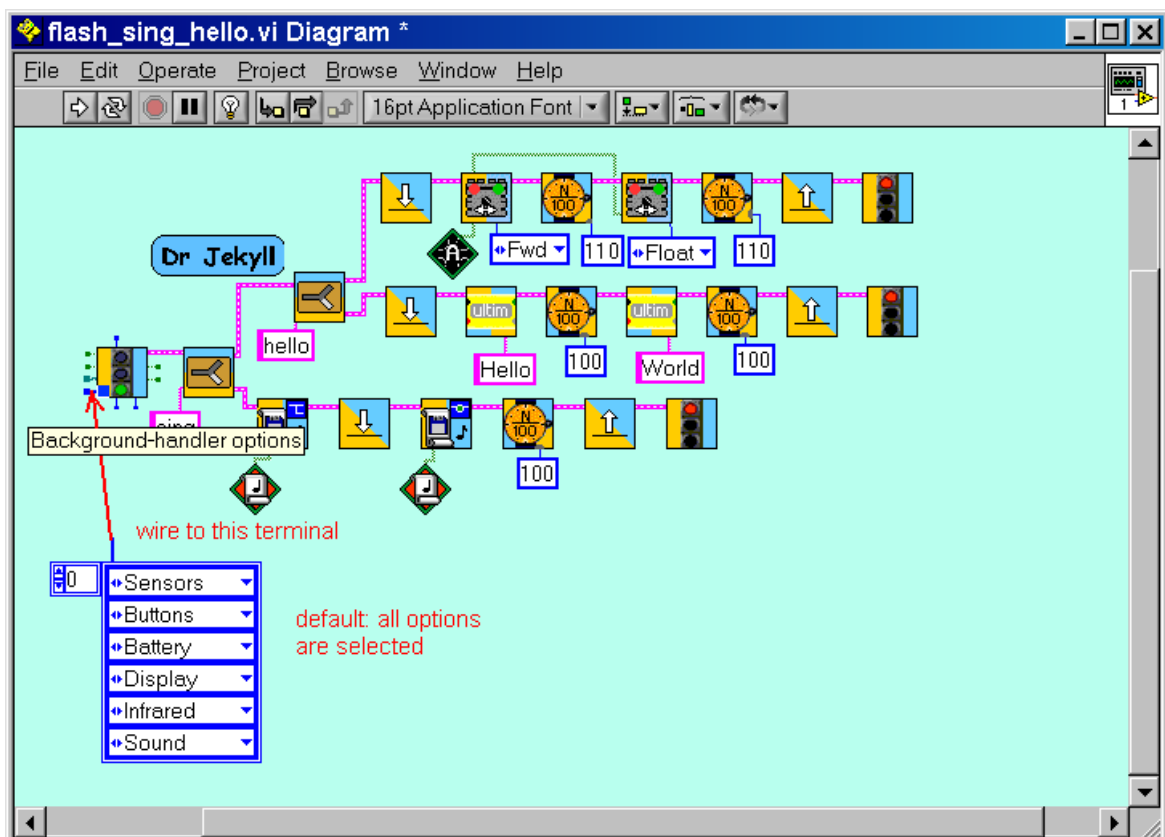


Picture 3: Often it is useful to change the representation of a LabVIEW constant. Note the different data types.

Ultimate ROBO LAB icons represent program steps that are similar to those known from standard ROBO LAB. But, of course, there also are many differences. Ultimate programs start with a special "BEGIN" sub.VI. Its function is to initialize both the cross-compiler program running on the PC and the RCX. This duality accompanies the user all through a program, but thanks to the ROBO LAB structure, it is always clear which device a certain function is linked to. One important difference between ROBO LAB and Ultimate ROBO LAB is the fact that in this advanced environment tasks, subroutines, functions, lands/jumps and containers may be either labeled by names or numbered dramatically improving the readability of programs.



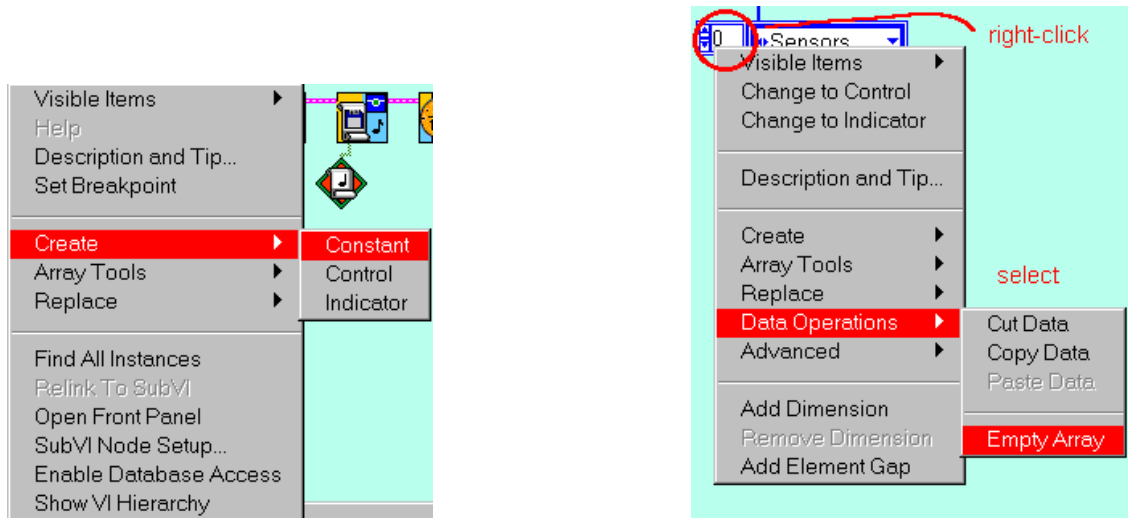
Picture 4: The RCX has a rather poor display. However UR allows writing text to it.



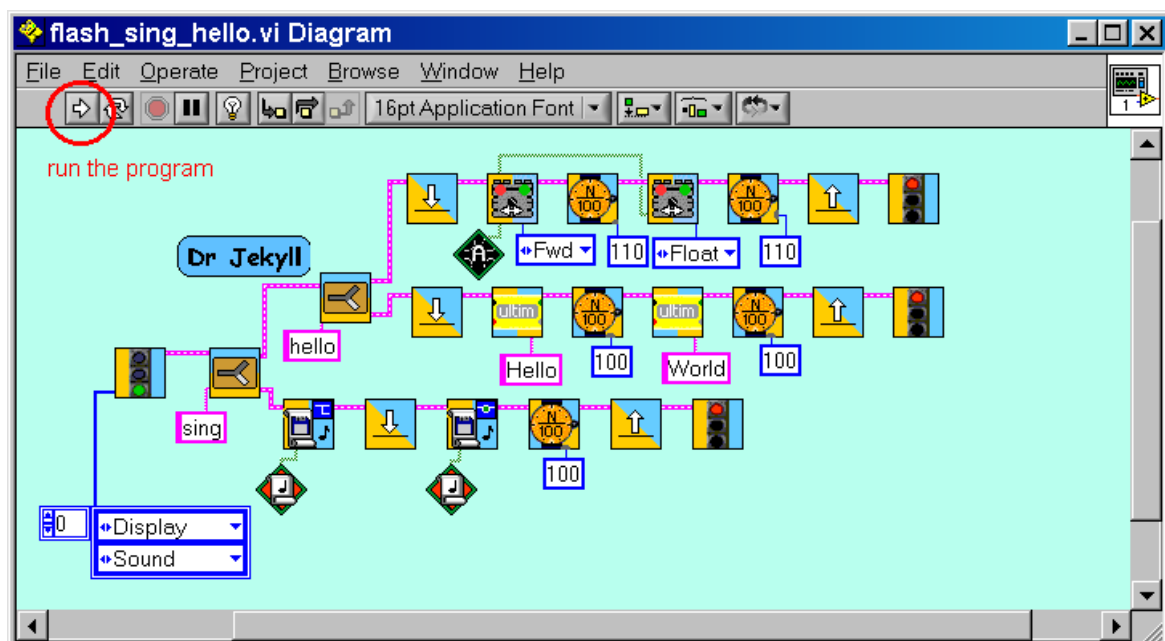
Picture 5: The user may optimize his code by unselecting the unneeded system functions that normally are run in the background.

With Ultimate ROBOLAB, the user can unselect various features that are not necessary for his current program. This shortens the program-code and thus reduces the download time and optimizes the main RCX-system task that is running in the background. If options are cleared, the cross-compiler automatically transfers the

related system-code from the background system task into the user tasks, wherever this is needed and possible. The option list is obtained by right-clicking on the respective terminal of the Begin icon and selecting *Create / Constant*. Note that disabling background functions may sometimes produce undesired side effects. For example: if buttons are disabled, the button-states are no longer checked by the system during the program execution. The consequence is that pressing the On/Off, Prgm or View-buttons has no effect, unless the user provides other code. BUT, in any case, pressing the Run-button stops the program execution, because a hardware interrupt is triggered. (Notice that UR also allows disabling this interrupt.)

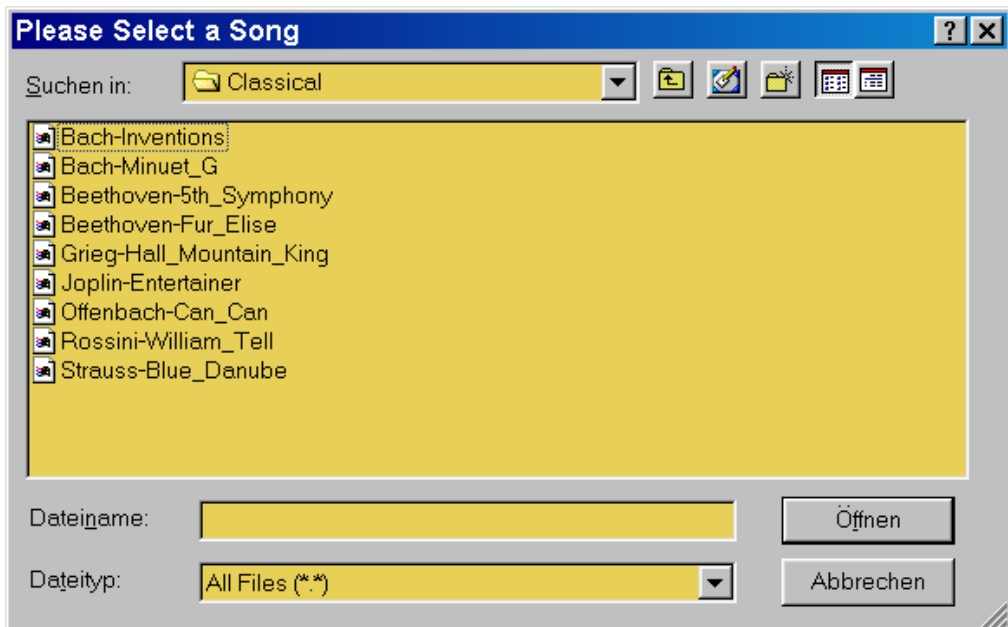


Pictures 6 & 7: To clear all the options, right-click into the list control and select *Data Operations / Empty Array*.



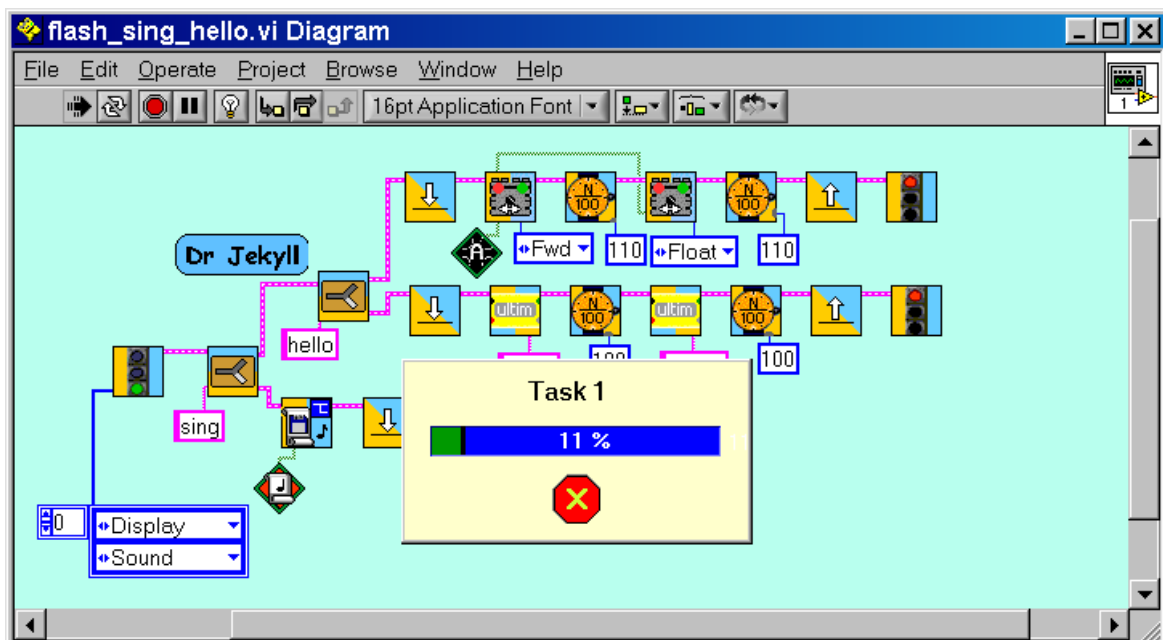
Picture 8: The compilation and downloading of UR programs are executed on the computer by clicking the run arrow.

As pointed out, the sample program first loads a song from a file to the red scroll. A dialog-window is opened inviting the user to choose a song from the usual ROBO LAB list.



Picture 9: UR supports standard ROBOLAB song scrolls.

The cross-compiler now is launched. Depending on the PC computing speed, the compiling process may take some time. Once this process has finished error-free, the downloading is started. One only task is downloaded, since Ultimate ROBOLAB merges the user program with system kernel routines to an individual firmware. The advantage of this approach is code-efficiency in terms of computing speed and memory economy. The disadvantage is a longer download time.



Picture 10: The UR download process is comparable to the standard ROBOLAB one.

Now, your RCX should write “Hello world” to the display, sing your selected song and flash a light that you connected to port A. The user may recognize that a system-task is working in the background through the little running man.

2. Save the program as a firmware in .srec format Mr Hyde

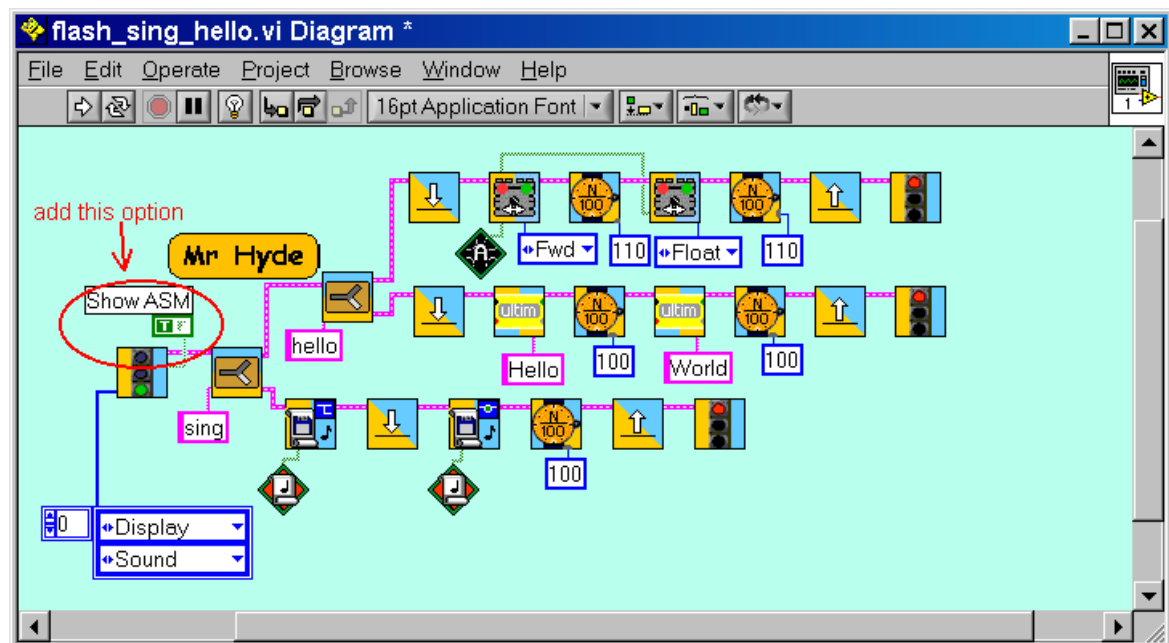
Usually a firmware is saved as a text. Standard ROBOLAB stores the original RCX firmware in the following folder : `..vi.lib\RCX1\Text` with the extension `.txt`. So, if you dare, you may open **firmware.txt** with a text-editor. (Please, be careful here. In any

case, save a clean copy somewhere, if ever you need to recover it!) The firmware should have the following aspect:

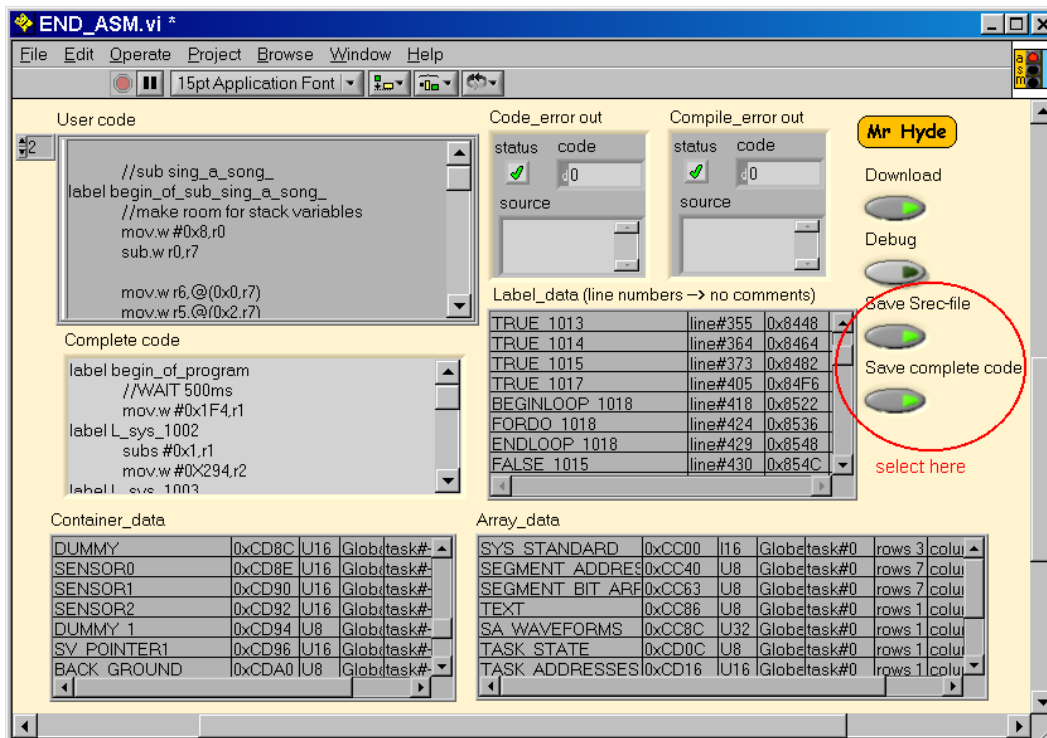
```
S00F00006669726D303333322E6C676F0A
S11380005E0082E4550254706DF06DF16DF26DF313
S11380106DF46DF56DF61B877901F1007903D700D6
S11380200D36AE7FB6E1440C0D160B01683D68EDCC
S11380300B0340EC7903D65D0D36AE00B6F0440870
S113804018EE68BE0B0340F0790683446B86C8C6FD
S1138050790686526B86C8C87906CD706B86C8CA05
S1138060790689246B86C8CC79068DD66B86C8CEF2
S11380707906C38C6B86C8D06A0CC8326B060000C4
S11380806B86FD9018EE6A8EC88B18CCAC004654F3
S11380907906C8546DF67906C85A5E003B9A0B8778
S11380A07906C8545E0082F07906C8545E008520C3
S11380B07906C8545E00CD20790.....
```

.....
These are H8-microcontroller firmware byte codes that are wrapped into a special code structure which is known as the Motorola S-record format (saved as ASCII-characters with file-extension .srec). Without exploring this here, only try to recognize the descriptors S0 and S1, each one initializing a record of a different type. A 2 digit hexadecimal number representing the number of bytes in the record and a 4-byte address follow them. Note that an RCX firmware always starts at RCX RAM-address 0x8000.

Ultimate ROBOLAB allows you to save your program in the S-record format. To do this, the “Show ASM” option (**AS**sembly **M**nemonic) must be set to TRUE in the Begin sub.vi. If you now run the program, *End_asm.vi*, a special front panel is opened, offering a deep insight into the cross-compiler activity.



Picture 11: To get more information about the compiled object files change the Boolean option.



Picture 12: Aspect of a compiled program with additional information about variable data, label addresses, etc.. Further options allow saving the object file

Saving the program in the S-record format gives the user the opportunity to download it with any firmware downloader program and share the program with people that aren't working with ROBOLAB. Selecting the option "Save complete code" is very useful when debugging programs.

3. Scaling numbers with the RCX Dr Jekyll

One of the most important qualities of Ultimate ROBOLAB is the fact that it supports many different data types. This enormously extends the possibilities in using the RCX as a real computer. In many robot-projects, it is necessary to calculate proportionalities or other mathematical functions.



The standard RCX firmware only supports 2-bytes signed integer data-type. This often forces the programmer to use complex tricks - even for simple calculations - in order to be able to do the required operations.

For example: you must convert rotations that are measured by the LEGO rotation sensor to traveled distances, let's say:

$$d [cm] = -3.247 \cdot r [rot] \quad (1)$$

Doing this scaling operation with integer numbers only is possible, if the scaling factor is transformed to a rational number. The most trivial number would be $-3247 / 1000$.

(1) can be rewritten to:

$$d = -\frac{3247}{1000} \cdot r$$

or

$$d = -3247 \cdot r \div 1000 \quad (2)$$

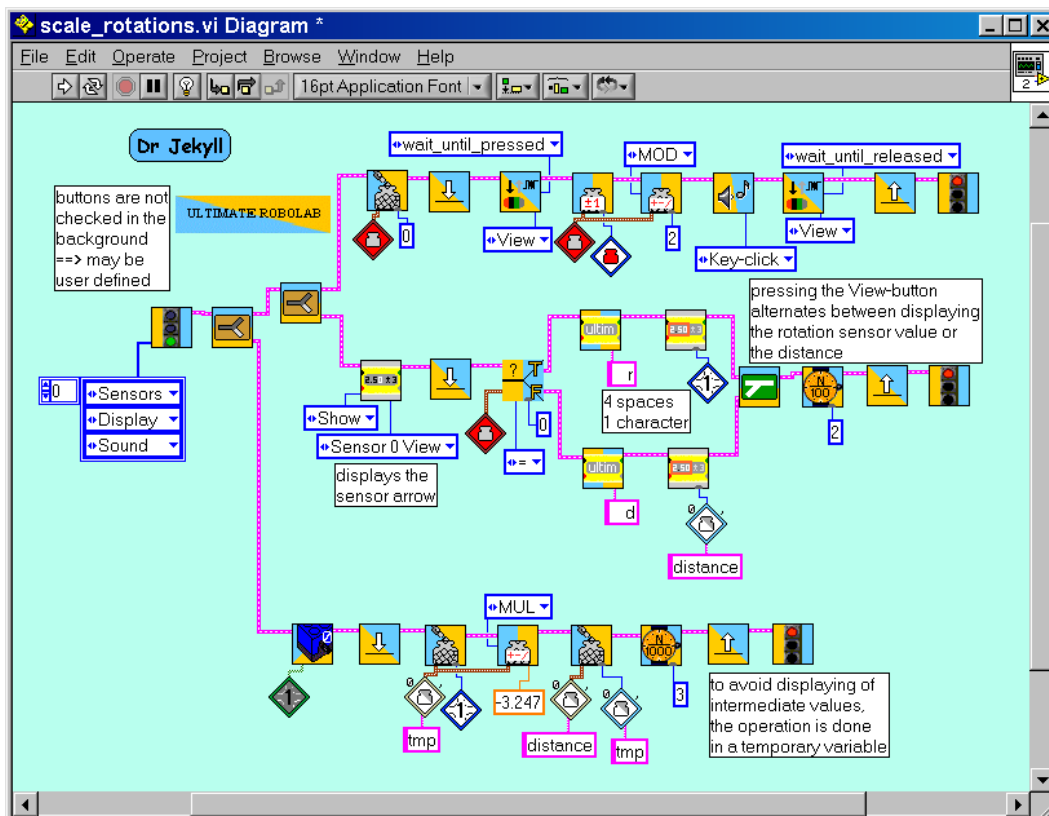
The division must of course be considered as an integer division. This looks very clear, but, what are the real limits of equation (2)?

The integer number d must be localized in the interval $[-32768, 32767]$. Thus r may not exceed $-32768 \text{ DIV } 3247 = -10$ or $32767 \text{ DIV } 3247 = +10$, otherwise there will be an erroneous data overflow. To solve this problem it is necessary to first determine the absolute limits of r and deduce the best estimation for the scaling factor. Suppose that the robot won't ever do a longer travel than ± 90 rotations. In this special case, the scaling numerator must stay within the limits $[-32768 \text{ DIV } 90 = -364; +364]$.

Now the job is to find the best quotient that fulfils the condition. However this is a rather complicated task. (Only think how long it took for mathematicians to find the better approximation for $\pi \approx 355 / 113$ than the usual $22 / 7$!) There exist various mathematical tools that allow the conversion from decimal to rational. If we choose for example $276 \text{ DIV } 85$, the precision is $6E-5$. The equation may then be expressed as:

$$d [\text{cm}] = -276 \cdot r \div 85 \quad (3)$$

😊 With Ultimate ROBO LAB this scaling operation is much easier to implement. The following program returns the scaled value as the result of a floating-point multiplication. Of course, there also is a price to pay for the facility, since floating-point operations need longer computation time than integer operations. But, in most of the cases this doesn't play too an important role, because robot-reactions generally are slow compared to the computing performances of the microprocessor that controls the robot.



Picture 13: A useful program to follow the conversion from raw rotation pulses to distance.

The program is composed of three parallel tasks. The upper task controls the View-button. Anytime the button is pressed, the red container is incremented. The following modulo function makes sure, the container is reset to zero, if the value exceeds 1. Thus the task may only produce two different values and thus states 0 and 1. The task runs a powerful button test sub.vi which in fact is a real button-“debouncer”.

TIP: When using buttons, keys or switches in electronics, engineers always have to deal with a problem known from every day's life. You certainly have observed that switching on or off the lights in your living room can produce disturbances in radio receiving. The action of establishing contact between two conducting media generates electric sparks combined with the emission of radio waves. This phenomenon appears even in microelectronics, where only weak currents and voltages are engaged. When a micro-controller has to monitor a button-state, every pressing or releasing passes a transition phase during which the states very quickly oscillate between high and low. A typical duration of this transition phase -known as **bouncing**- is 0.5msec, largely depending on the physical characteristics of the switching device.



Picture 14: Typical bouncing pattern

In microelectronics what causes most of the trouble is the fact that, whenever a precise action is desired, an interrupt for instance, it could be triggered a few times instead of only one time. When programming the RCX with the LEGO touch-sensors that you connect to one of the three input-ports, the danger of bouncing is very small, because of the rather slow reaction of the sensor system and an astute programming of the basic ROM-functions. Nevertheless you could have problems, especially if you select the high-speed sensor sampling modes that will be described later in this book (see RCX sensor ports chapter). But, in the case of the 4 RCX buttons (Run-View-Prgm-On/Off), you will necessarily have to provide a debouncing mechanism. Since you won't be able to change the RCX hardware -there exist various hardware debouncing tricks- you must realize it by software means.

One way to do a very simple and efficient debouncing is to read the button state a certain number of times at a rhythm of 250Hz - 1kHz. Each time the button returns a non-desired state, either pressed or released, the program enters a very fast loop to wait until the correct state is established once again. During this loop, the debounce-counter is reset, since the wrong state proves that the desired stable state has not been reached. When the debounce-counter reaches the verified number of correct states, we are certain that the state is stable.

The second task controls the display. If the rotation-sensor values are displayed, the last display-digit shows the character *r*. If the distance is returned, the display writes the letter *d*. The final task computes the conversion of the rotation-sensor values. Because the result is read asynchronously in the display-task, it is important to do the scaling in a temporary variable that is only transferred to the floating-point container labeled “*distance*” after the calculation to avoid that the RCX could display intermediate values.

3. Calling mathematical functions Mr Hyde

Single precision numbers used with UR are 4 bytes long. Since the RCX micro-controller isn't able to do 4-bytes operations in one protected cycle, it is possible that during multitasking processes, there rarely might appear erroneous intermediate values, even if temporary variables are being used. Please refer to the document “**Robust robot programming with Ultimate ROBOLAB**” to learn more about access conflicts under the conditions of multitasking.

TIP: the H8 processor is an 8-bit processor that has hardware-implemented 16-bit move instructions. Since there is no 32-bit transfer operation, problems could appear with multitasking. Each 32-bit variable is composed of two independent words HI and LO.

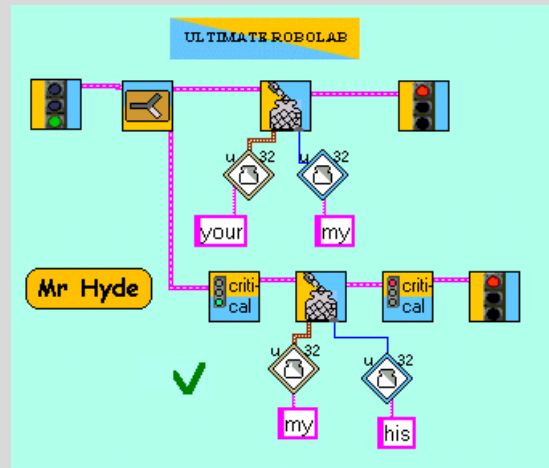
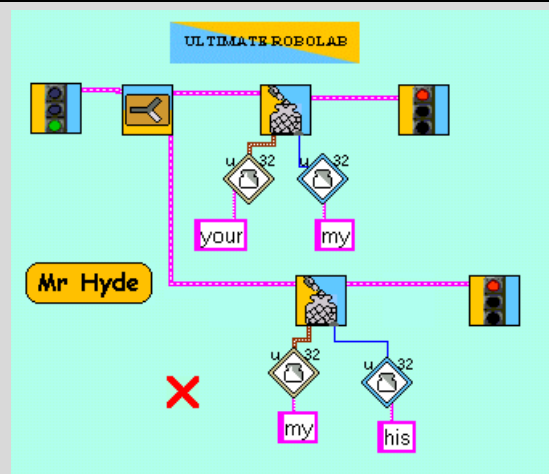
Imagine you are writing to a global floating point container in one task, and you are reading the same container in another task:

- Let's say, at time t_1 **my** = 65535
- at t_2 the writing task starts setting **my** to the value of **his**, that we suppose to be 65536 -> **his (HI) = 1** is transferred to **my (HI)**

==> **my** = 65536 + 65535 = 131071 !!! for a short moment

- imagine that at time t_3 the reading task becomes active and sets **your (HI)** to **my (HI)** ==> the old low byte value of your is still there, let's say it was 127, so **your** = 65536 + 127 = **65663 for an instant !!! wrong**
- at t_4 the reading task continues with the second move: **your (LO) = my (LO)** ==> **your** = 131071 !!! wrong

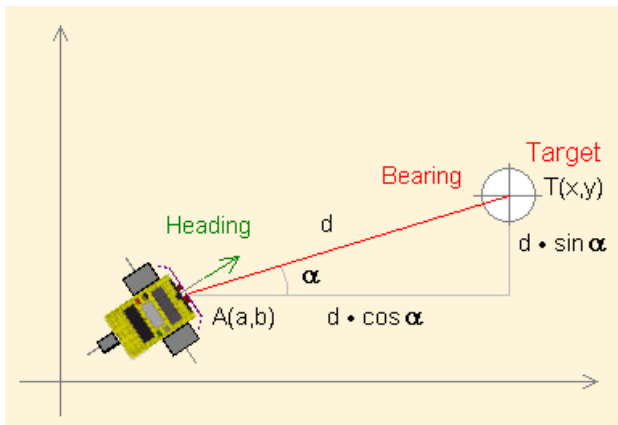
This can be one of the best-known sources for serious program-bugs. Hence, it is a good idea to specify as **CRITICAL** the writing to a global variable that is read in another task. This ensures that the writing task will not be quit during the move-phase.



Picture 15&16: Unsafe and safe data access.

Ultimate ROBOLAB has a certain number of remarkable facilities that distinguish this software from any other RCX programming tool. For instance there are fast trigonometric and transcendental functions that will be exposed here. "Fast" must be considered in relationship to the 16MHz clocked H8/300 CPU that forms the heart of the RCX micro-controller. The RCX needs only 26ms to calculate a sine or a cosine at 1E-6 precision. Besides the speed, a further particularity of the implemented algorithm is that the sine and the cosine are computed at the same time. This characteristic is somehow predestinated for mobile robot projects, where way-integrations must be operated.

Imagine that your robot ought to find its way from the known actual position **A** to a target location **T**. The robot also should know its orientation in the plane (heading). The goal is to have the robot compute the correct bearing α and the distance d in order deduce the motor actions that are necessary to reach the target point.



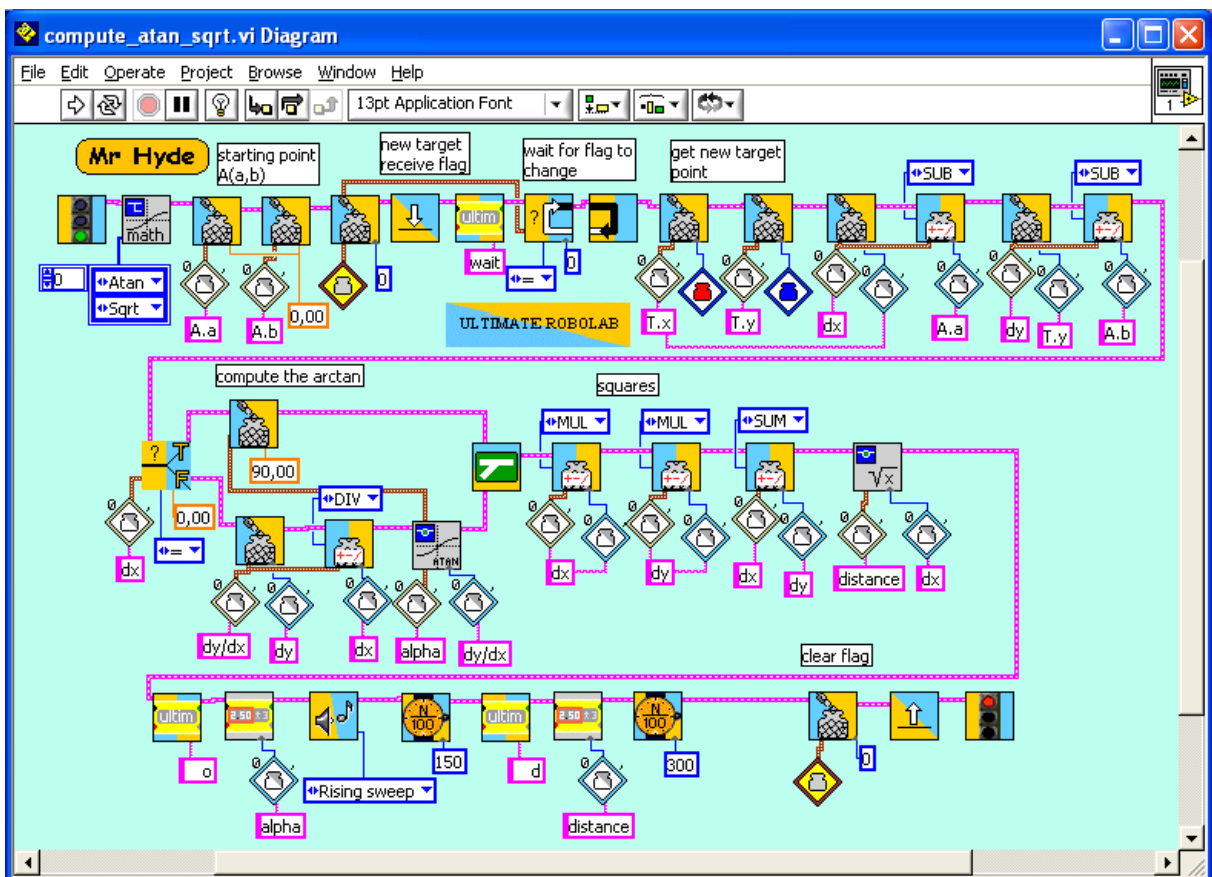
Picture 17: A mobile robot used to teach trigonometry!

From trigonometry we have:

$$\alpha = \arctan\left(\frac{y-b}{x-a}\right)$$

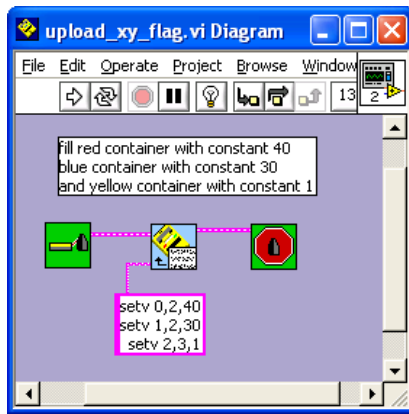
$$d = \sqrt{(x-a)^2 + (y-b)^2}$$

With Ultimate ROBO LAB it is very easy to yield these values:



Picture 18: Calculating the course of a mobile robot is a great occasion to learn about the square root and trig-functions.

If you run this program, the display will show the word “wait” until you change the Red, Blue and Yellow containers using the LASM **setv** command through standard ROBO LAB direct mode. Note that the Yellow container is used as a flag to tell the RCX that a new value set has been sent via the infrared channel. (Note that the RCX program doesn’t make any distinction of the quadrant.)

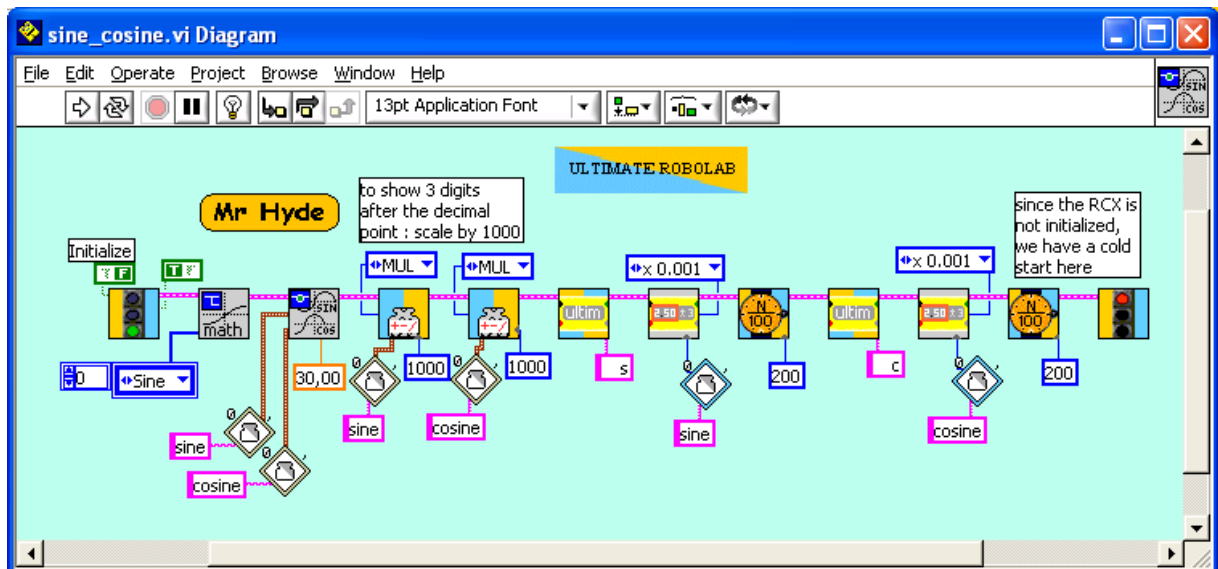


Picture 19: This standard ROBOLAB direct mode program must be used in combination with the program from picture 18.

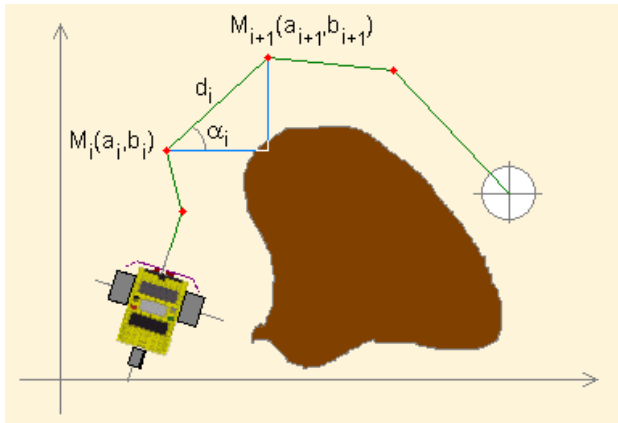
It should not be too a difficult task to program the mobile robot to drive to the target point. In any case, designing and programming such a robot is a great exercise to learn about trigonometry fundamentals and the art of navigation.

Now imagine that your robot is meeting obstacles on its way. The normal track can no longer be maintained and the navigation tasks entirely change. First, the robot must surround the obstacle, second it must compute the off-course from information that it collects with its sensors or positioning devices and finally, it must recalculate the new course.

The robot must be able to estimate its location and its orientation. It's a good practice to use a compass and a odometer for this purpose. We suppose that the user perfectly is able to design such a mobile robot-base. Now, to compute its actual position from the information is quite uncomplicated with Ultimate ROBOLAB.



Picture 20: This powerful UR function computes both the sine and the cosine at the same time. Note that for this example the RCX is not initialized, which means that it is operated at the lowest possible level.



Picture 21: If the robot must surround an obstacle, it should iteratively compute its actual position from the former known locations.

The present navigation process is called “way-integration”, since each new estimation of the actual location is the result of an iterative summation:

M_{i+1} :

$$a_{i+1} = a_i + d_i \cdot \cos \alpha_i$$

$$b_{i+1} = b_i + d_i \cdot \sin \alpha_i$$

Most iterative processes run the risk to accumulate considerable errors. In any case they must be maintained as small as possible. The robot should therefore be able to get most precise information about its orientation and the traveled distance. The UR algorithm to yield the trig values is advantageous because of the synchronous computation of the sine and the cosine.

5. Driving RC-servo motors with the RCX Dr Jekyll

People often insist that “Mindstormers” should keep “within the system”, respecting building rules and design conventions. Absolutely purists never would tolerate squeezing, gluing, not speaking of importing non-LEGO pieces. They look at the LEGO system as if it represented a table of atomic elements, each one invariable and indivisible. Less radical users however consider it as a simple, freely changeable material. It is comprehensible that the LEGO Company rather allies to the first category and does not support cutting or other forms of "bad" use of their unique pieces.

In practice, it is usually recommended that during basic Mindstorms training courses, instructors first impose the pure-LEGO-design-rule in order to help the participants to familiarize themselves with the system and get top-skilled in manipulating the endless number of parts. Limiting the allowed pieces to a known sub-set such as a precise LEGO box even intensifies this discipline, forcing people to try out all kind of combinations from a voluntary restricted set. Doing so they learn to prune dead-ends and even to anticipate valuable solutions, increasing the speed of problem solving for future challenges. The limitation of a single set often provides even experienced LEGO users with an added challenge in engineering with limited material.

At deeper levels however, especially when switching to high ceiling robot projects, designers will encounter more and more problems to connect their robot to the rest of the world. Since every LEGO robot is a real object that is likely to interact with its environment, the designers will always have to deal with various difficulties to interface it with real world. As long as they stay inside of the LEGO system, things

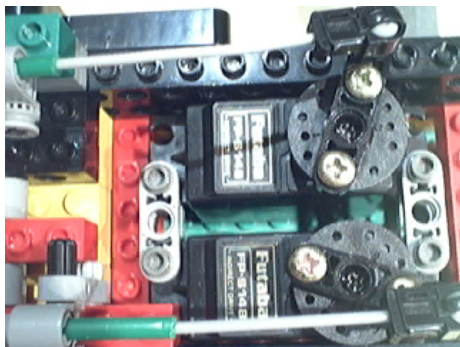
appear rather clear and predictable, not only thanks to the high precision of each LEGO brick, but when establishing the contact to real environment, imprecision and ambiguity sully the clean universe. When challenges become really complex and difficult, people necessarily discover the limits of a closed system. That's the moment to move off-roads and to start cutting, gluing, squeezing, and misusing LEGO pieces.

Nevertheless, some conventions should be respected when interfacing LEGO objects:

- as long as possible stick to the pure-LEGO-design-rule
- only allow LEGO-"violations" or non-LEGO pieces, if the designers agree that there is no other solution
- report the decision and the design-steps, so that others may reproduce them
- especially in a school environment also report the costs linked to the non-LEGO operation

All of this particularly concerns the RCX. Even at high level, the programmable brick should be kept untouched. There is one truth here, which says that any changed or damaged LEGO piece is lost to its owner for other future flexible use, because the piece is somehow alienated to the rest of the LEGO system. The more you allow altering, the more you reduce the number of combinations of your set. It is therefore a good exercise of creativity, for example, to find a solid and reliable solution for the simple problem of wiring non-LEGO electronics to the RCX without allowing original LEGO wires to be cut, or to develop a way to fix a 9V battery to a LEGO robot (which might be necessary to power an advanced sensor), or to attach the sensor's PCboard to the robot-base... Interfacing the RCX requires a good knowledge of the brick and the object to which it is supposed to be linked. It is a matter of ingenuity how well Mindstormers will succeed in developing valuable connections.

One of the rare weaknesses of the LEGO technic system is the absence of precise servomotors. It is difficult to imitate such a device with LEGO-motors, even if astute gearings are added. One idea is to try to integrate popular RC-servos, known from modeling, into the robot-structure. This generates the problem of joining different module systems. One has to improvise, adjusting the connection to the current situation. There exist numerous models of RC-servos.



Picture 22: RC-servo motors may easily be included into LEGO designs.

Ultimate ROBO LAB has a complete built-in RC-servo control kernel that keeps working all the normal Ultimate capabilities. The user may control up to three RC-servos. This was one of the hardest development challenge to generate the required

RC-control signals with enough stability to guarantee a resolution below the servo dead-band (normally 3 - 5°) and to keep the RCX fully working for the known Ultimate flexibility.

First let's have a decent look to what has been done so far with RC-servos. Ralph Hempel, the creator of pbForth has, along with Andreas Peter, developed various add-on circuits that allow the RCX to be used with two servomotors. Hempel has provided special program commands in pbForth to drive RC-servos through the RCX, namely SERVO_INIT and SERVO_SET. (also see: Dave BAUM, Michael GASPERI, Ralph HEMPEL, Luis VILLA *Extreme Mindstorms, An Advanced Guide to LEGO Mindstorms*, p.313.)

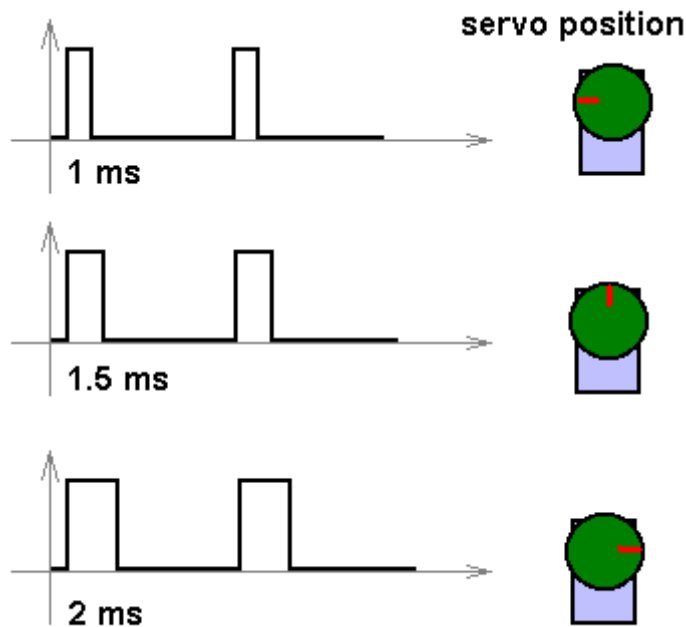
The author of this document has published an article in the leading European electronics journal *Elektor* (July/August 2004 issue) : *IR Servo Motor Interface for RCX*, where he presents an interface circuit, which can control up to three servos of the type used in radio-controlled models via the IR-interface of the RCX brick.

The challenge for the present software solution evidently was to avoid as much as possible building and soldering additional electronics-circuits. The result is a package of Ultimate ROBOLAB sub.vis that gives the user the possibility to very easily use those servomotors.



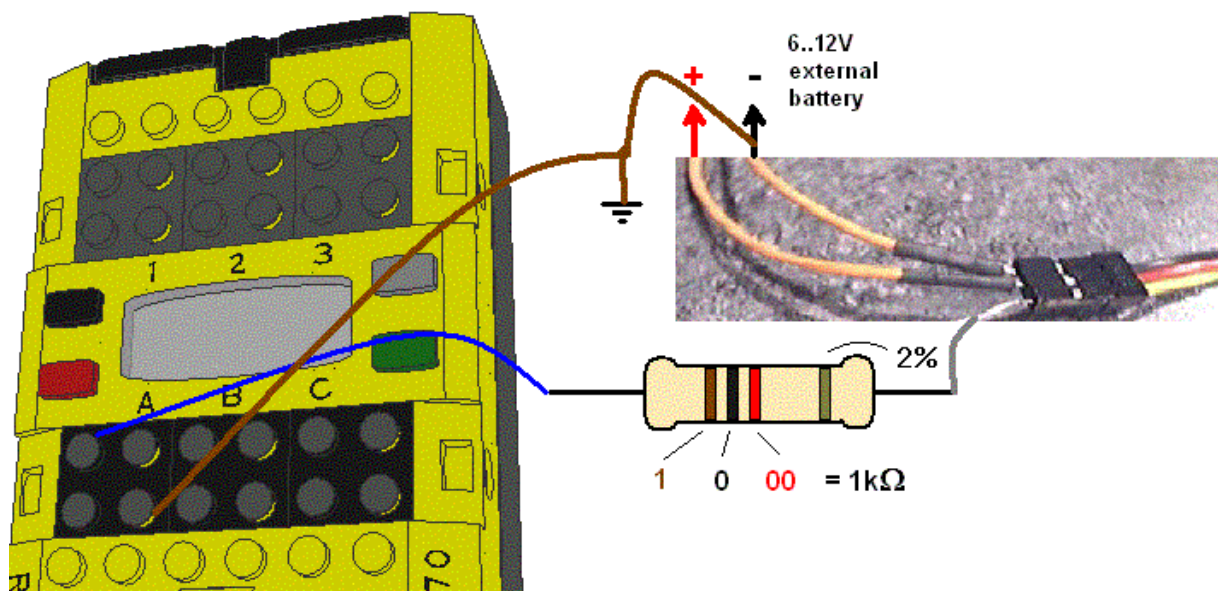
Picture 23: RC-servo motors are controlled with a short 30-50Hz pulse. The information about the servo-position is transported by the pulse width.

The oscilloscope shows the generated signal. The test-program changes the servo-position from 10 to 100% in steps of 10. Meanwhile the background-system keeps the running man in motion, sound is active, so do sensors, battery survey and so on. A further user task controls the blinking of the lamp at RCX output port B to prove that the RCX system is working correctly.

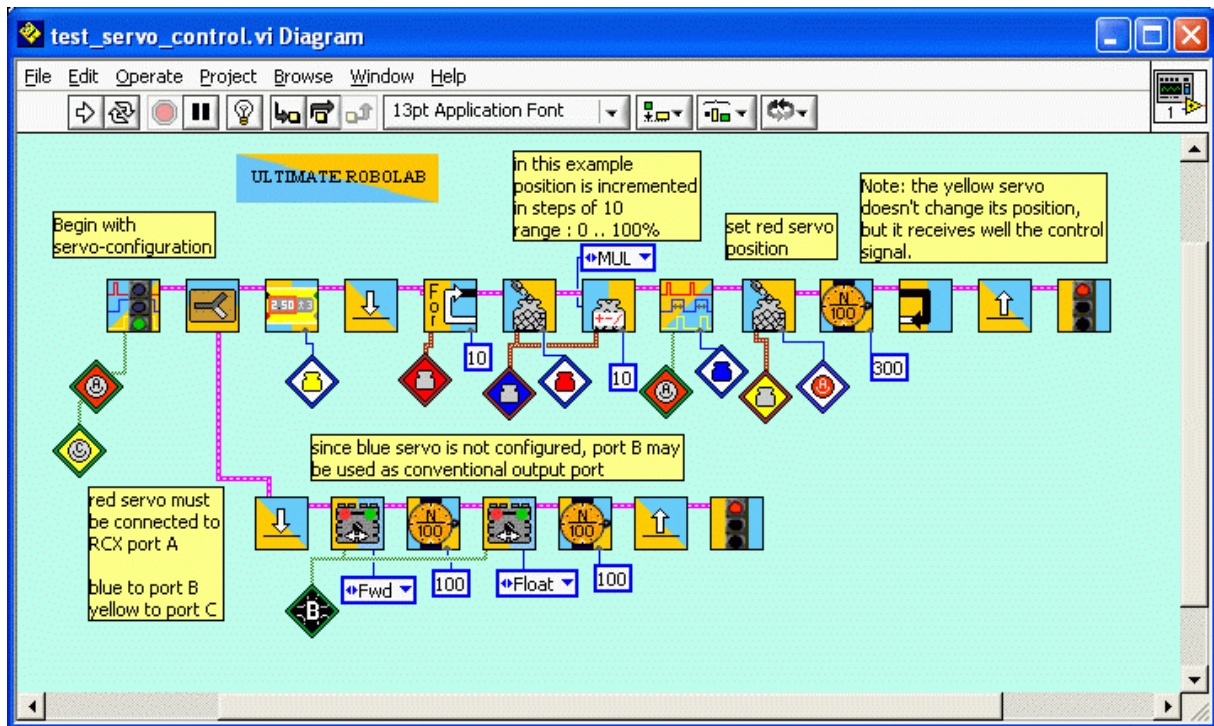


Picture 24: The pulse width controls the position of the servomotor-shaft.

With the servo-library the user can individually configure the RCX output to generate a 50 Hz signal with pulse width from 1..2ms with a resolution of 100 steps. In order to protect the RCX against any damage, RC-servomotors should be connected to it as shown in picture 25. In any case the servo must have an external power-supply and a common ground with the RCX output port. Experienced people only should do the wiring. It could be a good idea to ask an electronics specialist to prepare a diode-protected connection that can safely be used by anybody.



Picture 25: **Mr Hyde** Correctly connecting the RC-servomotor to the RCX is the only bulk in using the servo control kernel that requires more attention.



Picture 26: Ultimate ROBO LAB allows easiest access to the servomotors.

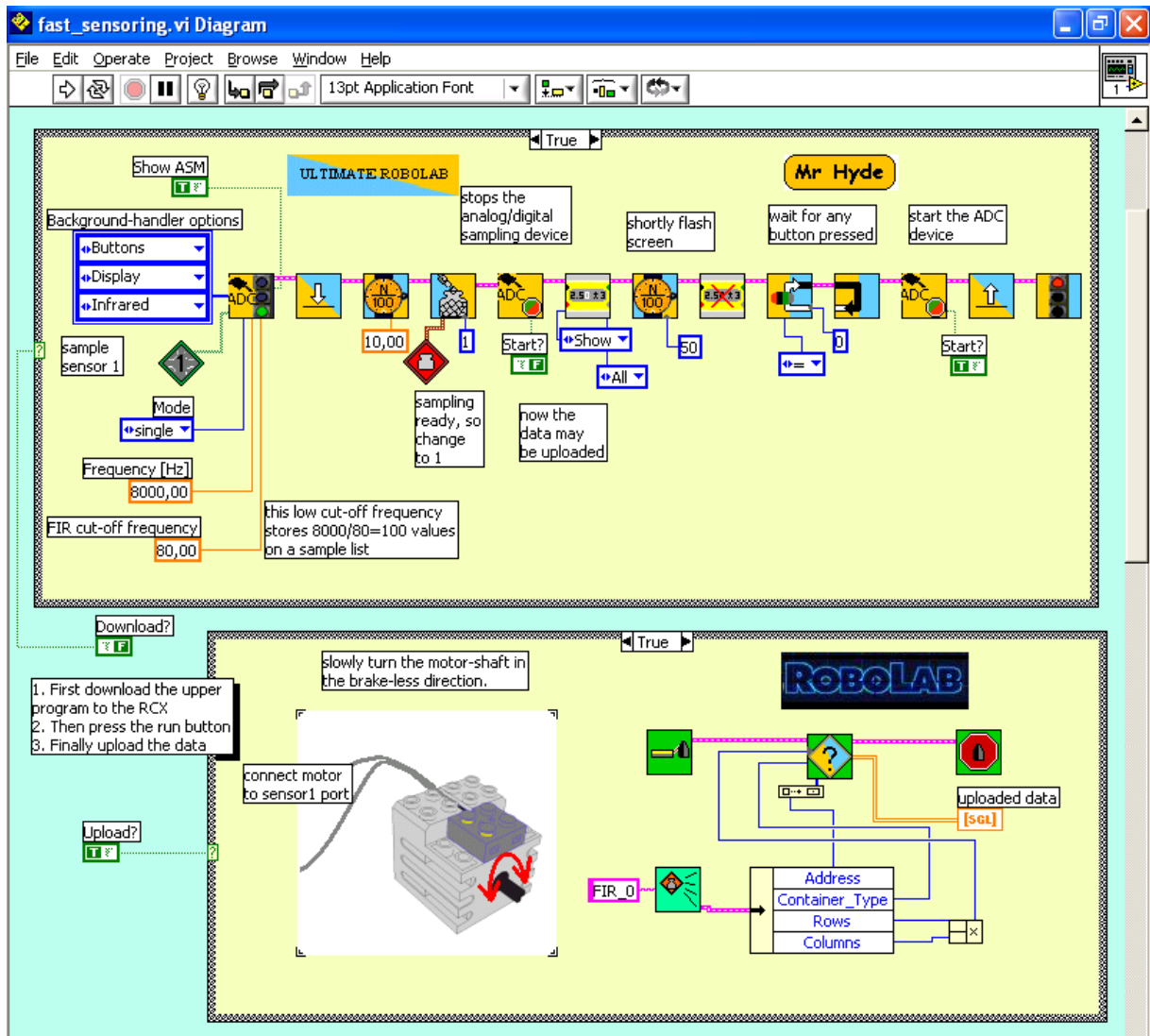
Note that each single servo can be calibrated by changing the zero-point (set_new_intercept) or the sensitivity (set_new_slope).

6. Very fast passive sensor sampling with the RCX Mr Hyde

Normal RCX sensor sampling is done at 322Hz. The reason for this rather slow rate must be seen in the astute “powered” or active sensors that need to be powered during 3ms and read during 0.1ms. This timing is maintained with passive sensors. However, sensing can be increased in speed to significantly higher rates. With Ultimate ROBO LAB active sensor sampling is possible up to about 3kHz whereas passive sensor reading may be done at rates up to 55kHz.

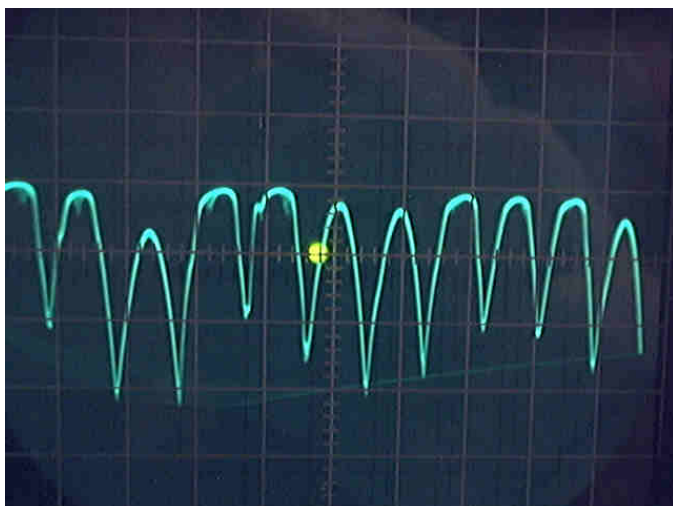
Fast timing operations are rather difficult to implement into a limited device like the RCX. But the LEGO choice to use the H8 micro-controller in the RCX reveals itself as a lucky one regarded the architecture and the embedded periphery of the controller.

In this introduction to Ultimate ROBO LAB we will explore the easiest way to operate passive sensor sampling at frequencies from 100Hz to 20kHz. A special Begin sub.vi reconfigures the RCX hardware timer0 that normally is used with sound. Instead of controlling the RCX speaker, a timer0 interrupt starts the Analog Digital Converter (ADC) at the desired rate. The RCX stores the sensor-value into an array that is also serves as the base of a special Finite Response Filter (FIR). This digital filter may be used for anti-aliasing or averaging. The normal sensor value that is read through the *Value of Sensor* icon is the filtered value, whereas the FIR array contains the raw high speed sampled values. The program in picture 27 is a good example to show what can be done with this new feature. The program is composed of two sections. First the simple and nearly self-explanatory UR program and second the standard ROBO LAB Direct mode upload function using the LASM *pollm* opcode that has been integrated into the midst sub.vi. The lower green icon is an additional UR function that helps extracting the address and the size of the data-array. (*deep loch* function palette !)

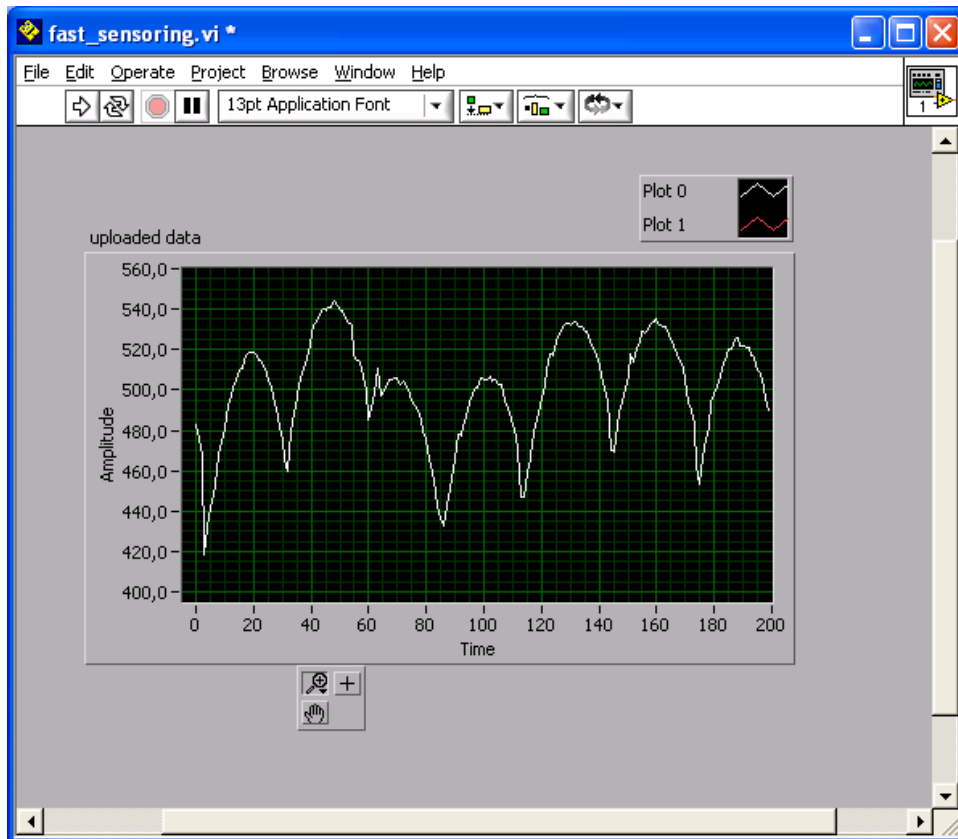


Picture 27: Ultimate ROBOLAB allows very high speed sampling. The data upload is operated with standard ROBOLAB Direct Mode from the data-address (FIR_0) using the LASM *pollm* command which is supported by Ultimate ROBOLAB

To test the program we suggest to connect a LEGO motor to sensor port 1 and to slowly turn the motor-shaft in the brakeless direction. The motor will serve as a DC generator producing a typical waveform that normally would only be made visible with an oscilloscope.



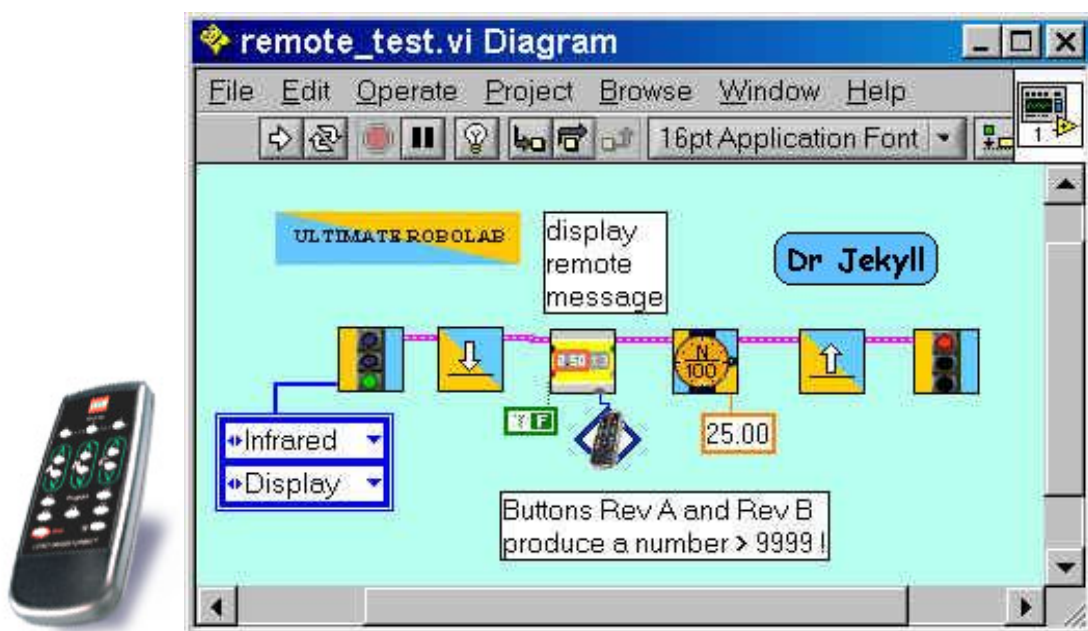
Picture 28: Oscilloscope-shot of the LEGO motor generated wave.



Picture 29: The uploaded data forms an interesting wave that is generated by the motor coils during shaft-rotations.

7. Using the LEGO remote control Dr Jekyll

Ultimate ROBOLAB gives the user very easy access to the LEGO remote control. A special modifier may be called in any UR program that has configured the infrared channel.



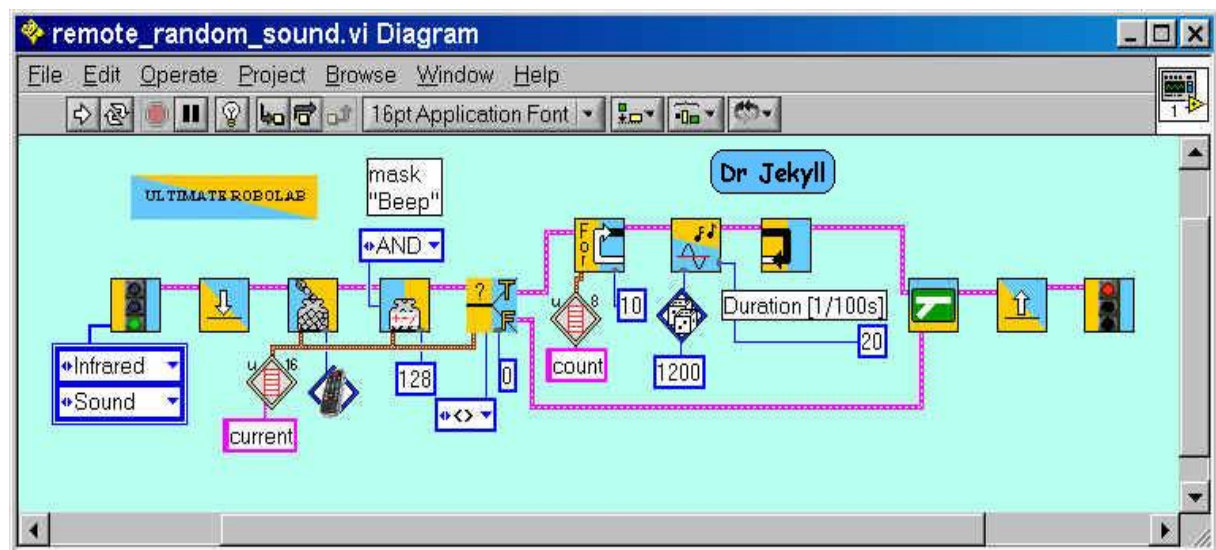
Picture 30 & 31: Using the LEGO remote control may be a simple but efficient way to intervene into a robot-program with more buttons than the poor RCX “key-board” made of only 4 buttons.

Bit (i)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value (2^i)	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Function	Rev B	Rev A	Fwd C	Fwd B	Fwd A	Message 3	Message 2	Message 1	Beep	STOP	P5	P4	P3	P2	P1	Rev C

Picture 32: The LEGO remote control sends different values to the RCX depending on the button that currently is pressed. The values form a mathematical base which means in this particular case that any number from 0 to $2^{(15-1)}$ can be represented by one precise button combination.

UR is able to accept button-combinations. For example, pressing *Message 1* and *Message 3* buttons at the same time will return the value $256+1024=1280$ to the RCX. The base-elements may be extracted by masking it through logical AND / OR functions.

Here a sample program to have the RCX beep when the corresponding remote control button is pressed:



Picture 33: To make sure to produce a button reaction to any pressing of the “Beep” button it is necessary to mask the remote-control value.