ROBOTICS SELF-LEARN TUTORIAL

Introduction

Aim of the tutorial is to provide a comprehensive introduction to the programming and use of ST robot arms using ROBOFORTH II. This will include "hands on" experience of programming with examples provided.

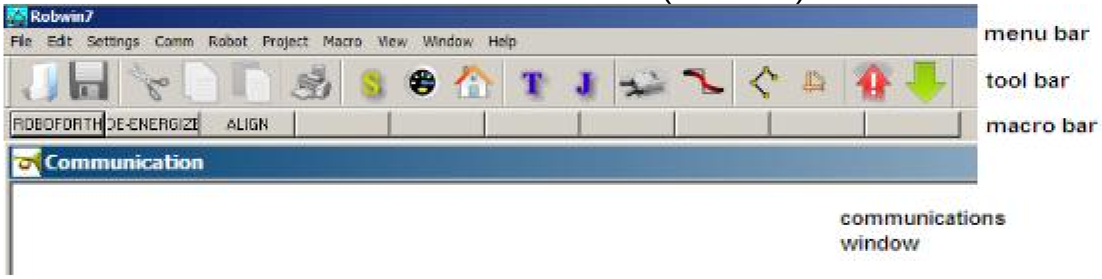
"Tell me and I will forget, show me and I will remember, involve me and I will understand" Aristotle

Some sections are marked **==advanced==** these are less useful and more complex programming. If you want you can just skip these to the next section.

Not all the features of RoboForth are described in this tutorial. For more information see the RoboForth II manual and the controller manual.

Launch RobWin

Double-click on the ROBWIN icon. Along the top are the menu buttons, below that is the robot tool bar and below that is the macro bar (see later).

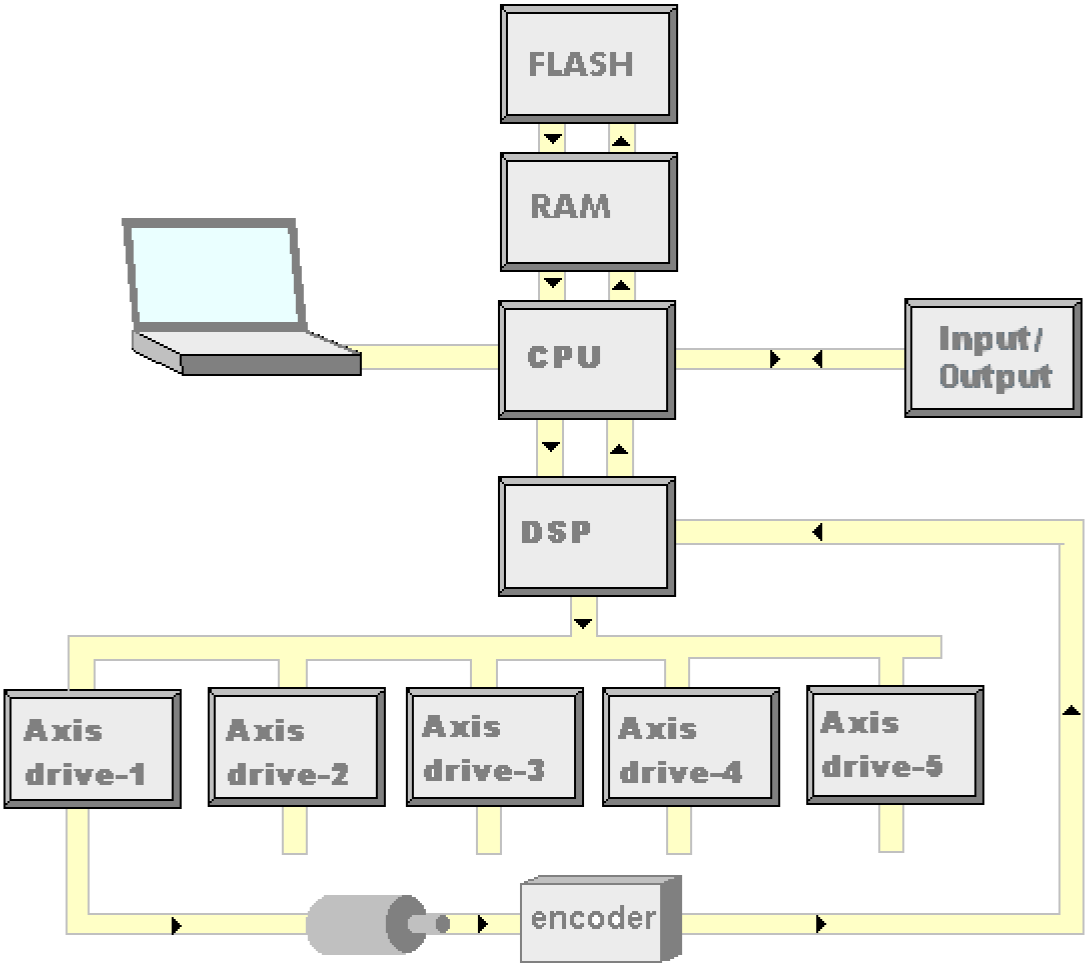


Load/check settings: On the menu bar click settings, open file. If you have an R17 click on R17.CFG; if an R19 click on R19.CFG, if an R12 click on R12.CFG
Caps lock – **All RoboForth commands are in UPPER CASE, ensure CAPS LOCK is on**.
Switch the key switch to cold start and **switch on the robot controller**.
In the communications window (the console) you should see a herald which includes the words ‘cold start’ and a chevron prompt **>**.
Type
ROBOFORTH

NOTE: You should always close RobWin before you switch off or disconnect the controller.

Basic technology

System architecture: a robot arm and its connections.

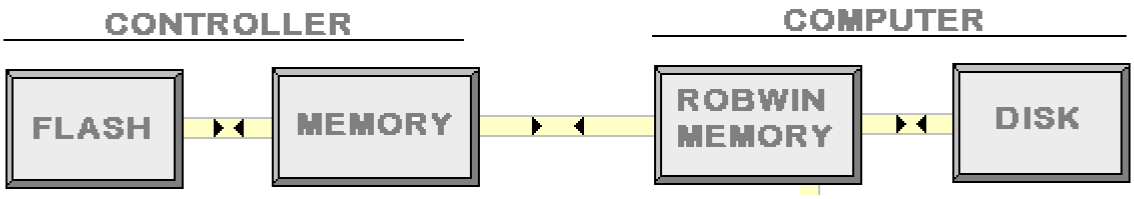


Computer, CPU, DSP, Axis drives, encoders

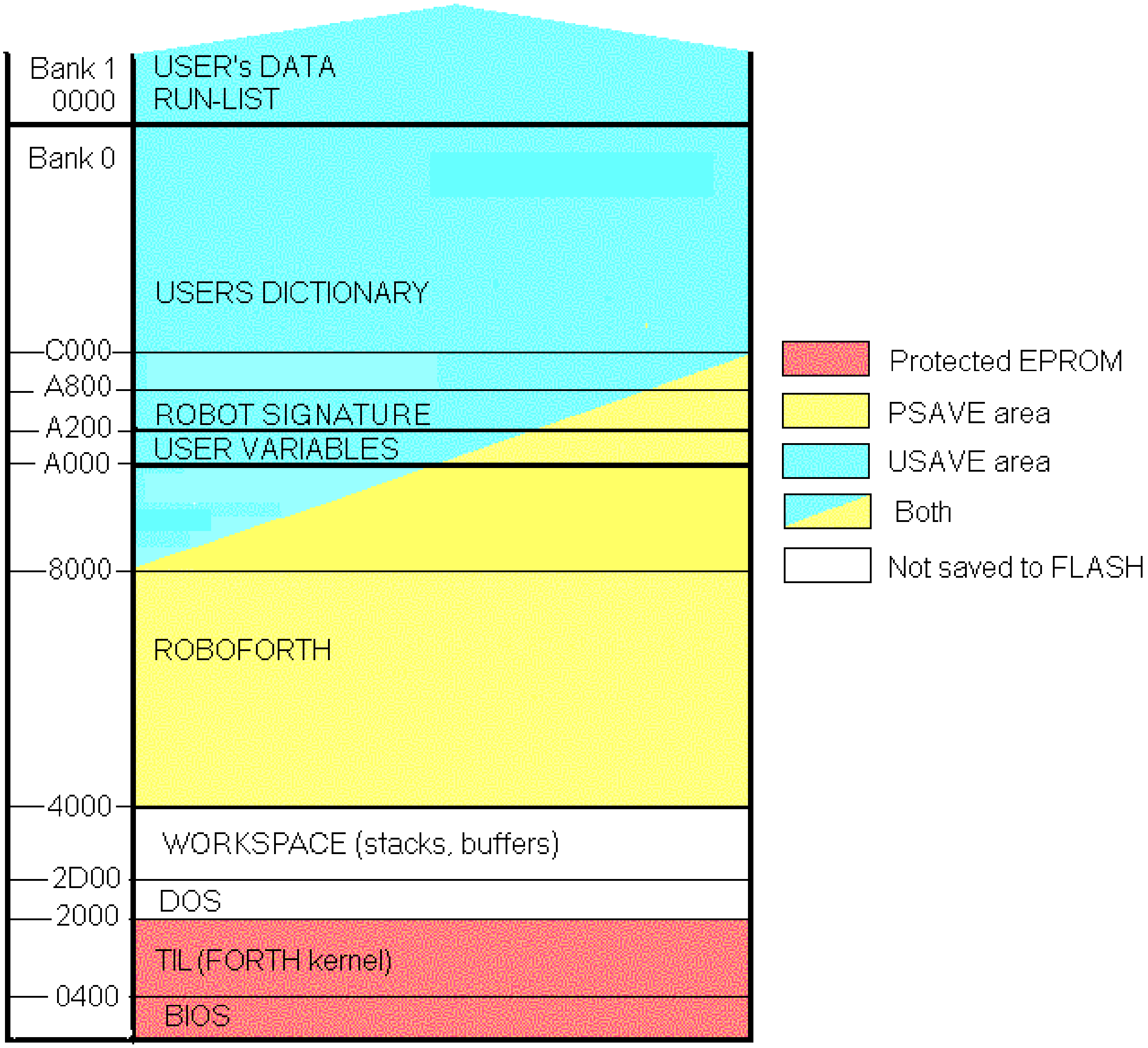
The K11R controller can have up to 6 axis drives for example for a 5-axis robot on a track.

The CPU sends motion commands to the DSP which in turn controls motion via intelligent drive modules, thus freeing the CPU for overall program control and I/O. Optical incremental encoders connect back to the DSP which sends this information back to the CPU on completion of each move. The CPU compares these counts with the counts it sent and halts motion in case of error. For example in the event of a stall robot motion is aborted and an error is announced (which is safer than most robot systems which continually try to get into position).
Both the programming system and user's software are stored in fast RAM but can be saved back to flash EPROM with the command USAVE. When power is switched on the contents of flash are copied to RAM.

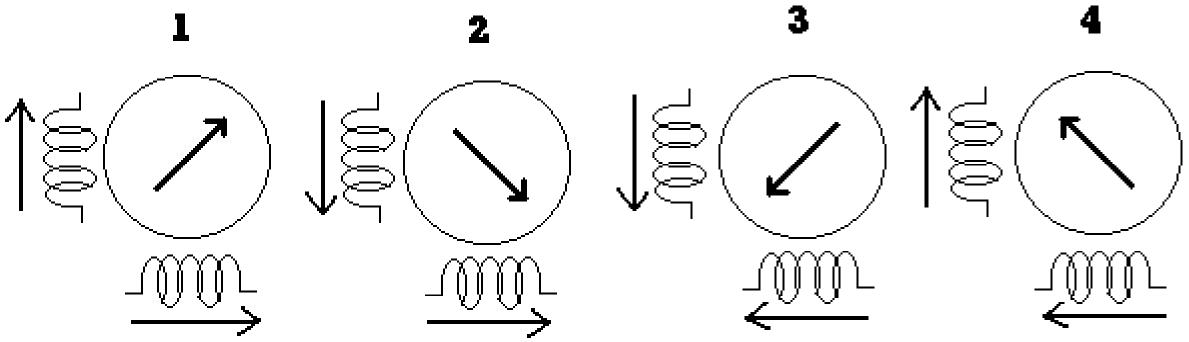
A computer is needed to program the controller which can be removed after programming is complete. Connection is via serial RS232 or USB (option). During programming data and procedure may be saved both to disk and to controller Flash EPROM. Once programmed the controller will run on its own without a computer.



In the K11R generally program and data are separated in two memory banks. The basic input-output system (BIOS), Forth, RoboForth and the users software are all in bank 0 and the data i.e. learned positions are in bank 1. The BIOS and Forth itself are in a protected part of Flash that can never be corrupted.



How stepping motors work, stepping motor counts, micro-stepping.



This is a hypothetical motor that moves 90 degrees per step. By alternately switching the polarity of the vertical and horizontal magnetic coils the magnetic field switches round 90 degrees at a time and the rotor moves 90 degrees to follow it. This means there are 4 steps per rev of the motor.

In a real motor as in an ST robot the rotor is made up of 50 poles so each step is 1.8 degrees. This results in 200 steps per rev.

Additionally the K11R controller breaks each step up into 10 micro-steps. So that instead of a step suddenly changing by 90 degrees as in the hypothetical motor or by 1.8 degrees as in an ST motor the field is progressively rotated sinusoidally from one step to the next. This results in 2000 micro-steps per rev. The ST software then scales this up using an array of constants called MICROS. If the value in the array for that particular axis is 2 then there are only 1000 CPU-steps per rev. It were 4 then there would be only 500 CPU-steps per rev. The usual values for MICROS are 1 or 2.

As the frequency of the steps increases the motor runs faster. Because there is mass and inertia, not just of the rotor but of the whole robot the frequency must be ramped up and then down again to stop. All this is controlled by the DSP.

When the motor is running fast it generates a back-emf which works to reduce the current and therefore the torque and power. So the K11R controller uses a high voltage and constant current control to keep the current constant throughout the rev range.

Because of the constant current drives, high voltage and inductive loads the motor cable must never be unplugged with power on. If you do remove a connector it will arc at even higher voltages and damage the connectors, both in the cable and in the robot.

Getting Started

Because RoboForth is built on the computer language Forth it is helpful to have some understanding of Forth itself.

Command Line

Commands are words which you type in followed by the return key. All commands in FORTH and ROBOFORTH are in upper case. Commands may have lower case but they are different commands. It is easier just to press caps lock and stay in upper case.

Throughout this manual the user's entry will be written in UPPER CASE, GREEN UNDERLINED but not the computer's response e.g.:-
TELL OK
You press the return key after typing TELL and the computer answers OK. I won't keep repeating that you need the return key after the command or after a line of commands. And I won’t keep writing the OK.

If it is a command that takes some time to complete, for example a motion command then the reponse OK will not come until the controller (and robot) have completed the command successfully. If there is an error then you won’t see OK, but an error message.

Where commands have ROBWIN shortcuts the ROBWIN key, button or mouse sequences are indicated with a ROBWIN icon.

Sometimes a string of commands are required; these are typed on one line separated by spaces e.g.
(don't type this)
TELL TRACK OK

You press the return key after the last word and the computer answers OK when it successfully completes the whole line - provided there has been no mistake as in the following:-
TELL TRUCK TRUCK NOT DEFINED - the word 'TRUCK' was not understood.

Words and definitions

Forth has no “syntax” as such. All the commands are just words that are organized as a linked list or "dictionary". With each word in the dictionary is kept its definition, as with a conventional dictionary. Each word in the command line is looked up in the dictionary and its "meaning" is executed. If FORTH cannot find the word it tries it as a number and converts it to a value. If that fails the word is rejected and the rest of the line ignored. Note that sometimes two English words make up one ROBOFORTH word e.g. GOTO or ISAT.

You can easily add to this dictionary by defining your own words and that is how programming in Forth is done. To add a new word to the dictionary you use a defining word, of which the most common is the colon. The colon says add the next word to the dictionary and the words after that are its definition. The definition ends with a semi-colon.

For example enter

```
>: HI .” HELLO “ ; OK
```

The new definition of HI **begins with a colon and ends with a semi-colon**. Press return after the semi-colon to get OK.

Now the word HI has been added to the dictionary you can use it thus: Just type HI and press return.

```
>HI HELLO OK
```

⚠ WARNING: All words are stored in the dictionary as a length plus the first 5 characters of the word. Therefore ambiguities are possible and should be avoided (such as ROBOT1 and ROBOT2 which are the same length and have the same 5 starting characters).
In particular avoid creating words similar to RoboForth words for example if you create a word RECEIPT this has the same first 5 characters and the same length as RECEIVE. Since RobWin uses this word if you redefine it in this way then communication between RobWin and Roboforth is not possible.

Words may be made up of any characters and any number of characters. It is a convention that many of the more basic words in the FORTH 'kernel' have agreed pronunciations, for example . (dot) is the shortform (there is no long form) for PRINT and is pronounced "print", ! is pronounced "store", @ is pronounced "fetch" and so on. Pronunciations will be given as they arise.

A word you have created in this way may be removed from the dictionary with FORGET e.g.

```
>FORGET HI OK
```

Now try HI and you will get

```
>HI HI NOT DEFINED
```

==advanced==

When you type words into the command line those words are collected and interpreted by a master word called the outer interpreter.

Each word in the dictionary has a structure comprising the length of the word, the first 5 characters, the code field address or CFA, then the parameter field address or PFA. When you include a word in a new definition the outer interpreter looks that word up in the dictionary and compiles it's code field. When a word is executed it's code field is sent to the inner interpreter which actually executes the code. The word it execute may well be another definition of another list of words each one of which is sent to the inner interpreter to be execute. Finally words which are pure machine code end the thread. These words are called "primitives".

The PFA can be found with a single quote pronounced as tick.

' HI (tick HI) looks up HI in the dictionary and leaves the PFA on the stack. If it can't be found it says `HI NOT DEFINED`

The CFA can be found with FIND e.g.

FIND HI X. results in the CFA of HI being found and printed in a hex format (X.)

If HI can not be found it will leave zero.

Once a CFA is found it can be executed with EXECUTE, i.e.

FIND HI EXECUTE is the same as just typing HI.

Arguments

Arguments are values used by any function or command e.g. in
100 STEPS
the argument is 100 and is typed in *before* the command (STEPS) which needs it.

The BASIC expression 5 + 6 uses INFIX notation. In FORTH we write 5 6 + which is POSTFIX notation. This is because the values 5 and 6 are placed on a STACK for use by the word '+'
A stack is an area in memory into which values are temporarily stored on a last-in first-out basis. The answer is left on the stack for use by following words for example a period which means print the value on the stack. For example:-

```
>DECIMAL (return key) OK
>5 6 + . (return key) 11  OK
```

You could just as easily have written the above numbers and words on separate lines. Forth doesn't care whether you use a space or a new line. For example

```
>5 (return key) OK
```

At this point you have done nothing but put the value 5 on the stack.

```
>6 (return key) OK
```

Now there are two values on the stack. The top value is the 6 with the 5 next one down.

```
>+ (return key) OK
```

The word "+" removes the two values from the stack, adds them together and leaves the result on the stack. Now there is only one value on the stack, the value 11

```
>. (return key) 11 OK
```

The dot is shortform for print. It takes the value off the stack and prints it on screen. Now the stack is empty.

So because all Forth words that use values expect to find them on the stack, those values must be entered or computed first. For example

```
>1000 STEPS (return key) OK
```

moves a selected joint 1000 motor steps -- or rather 1000 CPU steps (see above).
Don't worry, nothing should move because we haven't yet selected an axis.

```
>1000 MOVE (return key) OK
```

is the same but the system keeps a count of how many steps have been sent.
You can't write MOVE 1000

Integers

The standard representation of a number in ST Forth is a 16 bit integer made up of 2 bytes. A single stack entry is a 16-bit value. Arithmetic is done with 16-bit twos complement values in the range -32768 to +32767 unless unsigned integers are specified (0-65535). If necessary 32 bit values may be used (4 bytes). These occupy 2 stack places and are normally only used to assist with scaling i.e. mixed precision arithmetic. Floating point arithmetic is also possible in FORTH but integer arithmetic is preferred. Even trigonometry can be performed using integers (see controller manual) with an implied decimal point.

FORTH words you need to know

The FORTH words are not part of ROBOFORTH but exist at the bottom of the dictionary. They support the higher level ROBOFORTH words. While many of these words are esoteric there are some that are really useful or even essential for writing any kind of software that goes beyond the basics, for example changing speed, I/O etc. The following FORTH words are worth knowing. For details of all the FORTH words see the system manual. It is possible to do really complex tasks using both FORTH and ROBOFORTH.

Although the stack is used extensively to pass arguments, variables also exist and are handled with the words **!** (pronounced "store") and @ (pronounced "fetch"). A variable is just a word which leaves its address on the stack. The actual address is unimportant.

A new variable is created with the word VARIABLE e.g.
VARIABLE TOPSPEED
creates a new variable with the name TOPSPEED. VARIABLE is a defining word.

The word **!** (exclamation mark and pronounced “store”) means "store the second value down on the stack into the address which is the top item on the stack", for example 12 5000 ! means "store the value 12 into address 5000". In the process both numbers are used up leaving the stack empty.

To put a new value into the variable TOPSPEED use the syntax e.g.
2000 TOPSPEED !
Which means store the value 2000 into the address TOPSPEED
Remember TOPSPEED is just a location in memory which has a name.

@ (at-sign, pronounce “fetch”) means "fetch the contents of the address on the stack and leave it on the stack". In the process it uses up that address. For example
TOPSPEED @
gets the current value of TOPSPEED and leaves it on the stack for another word to use up, for example the period **.** which means "print".
The current value of TOPSPEED could be found with
TOPSPEED @ **.** (pronounced "topspeed fetch print")
or the easier form
TOPSPEED ?
Data can be transferred from one variable to another with e.g.
TOPSPEED @ SPEED !

You can also create a variable with the word USER.
When you use VARIABLE (word) the actual value of the variable is stored in the parameter field of the new word as per standard Forth practice. See the structure of a Forth word in the controller manual. However if the text around this word is edited then recompiled (reloaded) then the actual address of the data may change. Therefore any previous value will be lost.
There is another way of creating a variable with
USER TOPSPEED
This stores an address in the parameter field which points to a location in a reserved memory area called the User memory. The contents of this are unchanged after a reload.

If USAVE is used then the contents of user variables are saved to FLASH ROM and reloaded from FLASH after a reset or power-up.

There is a pointer to this memory UVP which is incremented each time a new USER variable is created. If you FORGET a user variable the pointer is restored to the previous user variable. Typing the word ROBOFORTH restores the pointer to the start of the user memory.

Control words -1

Control words are IF...THEN, BEGIN...UNTIL and DO...LOOP and others. These words cannot be used in command mode but must be incorporated into new words (new definitions). They are a bit harder to understand so you may wish to revisit this section at a later time.

As already stated arguments are passed on a stack. A condition is a numerical argument for such words as IF, WHILE and UNTIL. Any non zero value (positive or negative) is treated as true, and zero is false.

The structure of an IF statement is this: (condition) IF (action) THEN
If the condition is true (non-zero) then all of (action) is executed but if false (zero) then the program flow branches immediately to the words following after THEN. The word THEN closes the IF statement like ENDIF in Pascal. In BASIC the close of an IF statement is the end of the line.
A condition may be set up with comparing words such as = > and < but the result of any calculation may be used.
Examples of conditions are:
2 4 = leaves zero on stack i.e. false
2 2 = leaves a 1 on stack i.e. true
PRESSURE 100 > leaves true (non-zero) if the word PRESSURE leaves any value on the stack which is greater than 100.
Example:

```
: PRELIEF PRESSURE 1000 > IF VALVE ON THEN ;
```

Try this definition:

```
>: ISIT6? 6 = IF .” YES “ ELSE .” NO “ THEN ; OK
>6 ISIT6? YES  OK
>5 ISIT6? NO  OK
```

It is often useful to set up a continuous loop which ends only if a condition is met. Structure:
BEGIN (action) (condition) UNTIL
This executes action in a loop and repeats until the condition is true.
Example (don’t try it): This repeats a robot motion until the escape key is pressed

```
: TASK
BEGIN
  P1 GET
  P2 PUT
?TERMINAL UNTIL
;
```

Don’t try this yet. Real examples will be given later.

Other control loops are: IF ELSE THEN, BEGIN WHILE REPEAT, and counting loops DO LOOP.
For a full explanation of all the various control words and how to use them please see the system manual, section 10.6

RobWin

Is a computer program to help you program the robot. Most of the buttons do no more than send a command to the robot to save you the trouble of typing it. RobWin is a project manager and once you have a project open some buttons or actions change data in the controller as well as in the computer.

⚠ **WARNING**

Before trying any of the following commands be sure to
KEEP OUT OF THE ROBOT ENVELOPE

Initialization and calibration commands:

To START with enter
START
or click this button in Robwin S
This initializes all the drives, starting values for variables such as SPEED and so on.
CALIBRATE
or ⏲ tells the robot to seek out all the sensors on the axes. The number of steps from the home position to these sensors is known, so that after CALIBRATE is used the robot knows exactly how far it is from the HOME position. HOME is where all the counts are zero. It is also a significant position for Cartesian transformations (kinetics) because it is where X and Y positions of the hand are both zero.
HOME
or click 🏠 tells the robot to go to to the HOME position.
Enter
WHERE
This will report the position of the robot. You will see 3 lines of numbers e.g. for R12/R17

```
      WAIST   SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
          0        0        0       0       0
          0        0        0       0       0
          0        0        0       0       0
```

The top line is the software counts for each axis. Of course these are zero for the HOME position. The middle line is the counts back from the encoders. The bottom line is where the encoders think the robot is i.e. the counts multiplied by a factor depending on gear ratios, encoder resolution etc. In practice these will *not* be zero. They will be close to zero for an R17 but significantly larger for an R12 because on an R17 the encoders are close to the motors and on R12 they are some way down the gear train.

If you want to move the robot by hand use
DE-ENERGIZE
Then to re-energize use
ENERGIZE
But if you have moved the robot while de-energize its position will be invalid so you had best use START and CALIBRATE again.

Moving the robot and teaching the robot are separate operations.

Moving the robot using TELL with relative and absolute commands

Try these
TELL WAIST 1000 MOVE (press enter) (small delay then) OK
The robot moves 1000 steps and when it stops you see OK. Try
WHERE

```
      WAIST  SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
       1000       0       0       0       0
```

1000 MOVE
We are still talking to the waist so it moves another 1000
WHERE

```
      WAIST  SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
       2000       0       0       0       0
```

Another 1000 brings the total movement to 2000
1500 MOVETO
MOVE was a relative command and MOVETO is an absolute command.
Now the waist moves back from 2000 to 1500
WHERE

```
      WAIST  SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
       1500       0       0       0       0
```

-1000 MOVE
Now the waist moves backwards 1000
WHERE

```
      WAIST  SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
        500       0       0       0       0
```

TELL SHOULDER REVERSE 1000 MOVE
Now the shoulder will move backwards
WHERE

```
      WAIST  SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
        500   -1000       0       0       0
```

And so on for the other axes.
You can move any axis with the word STEPS e.g.
TELL WAIST 1000 STEPS
However these will not be counted so when you type WHERE they do not show.

==advanced==
Two flag bytes determine which motors will run and in which direction. In each byte bit 0 (value 1) is motor 1 (waist), bit 1 (value 2) is motor 2 (shoulder), bit 2 (value 4) is motor 3 (elbow) and so on. These two bytes are like variables but instead of using @ and ! as previously explained you must use the 1-byte versions, C@ and C!
MEP – Motor Enable Pattern
MDP – Motor Direction Pattern.
Examples: 1 MEP C! 0 MDP C! 1000 MOVE moves the waist 1000 steps.
2 MEP C! 2 MDP C! 1000 MOVE moves the shoulder 1000 steps in reverse.

Operating the gripper.

GRIP

UNGRIP

The RobWin icon will alternately (toggle) the gripper open and closed.
Time is allowed for the gripper to operate. This is in the variable TGRIP in mSecs.
To see how many mSecs is allowed write TGRIP ?
To increase or reduce the time allowed put a new value in TGRIP e.g.
1000 TGRIP !
TGRIP only applies to pneumatic grippers. The electric gripper is software controlled with acceleration, closing speed and holding force.

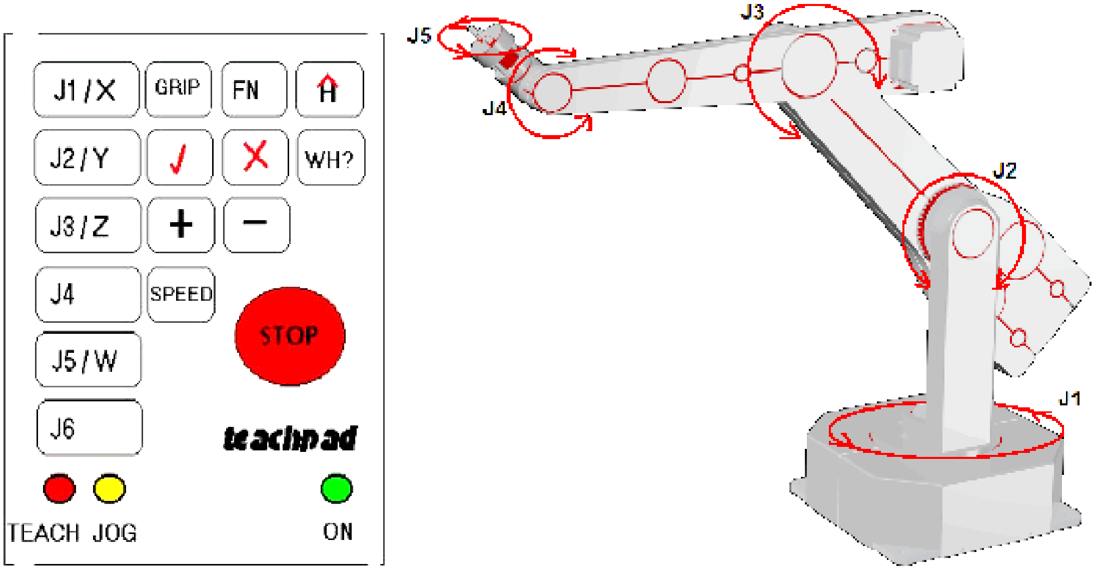
To operate the pneumatic (not electric) gripper without any delay at all use
GRIPPER ON and GRIPPER OFF

The electric gripper uses two outputs which are switched on and off at high speed to control the speed of closing and opening. This is called PWM (pulse width modulation).

Moving the robot using the teach box.

To use the teach pad click the T icon in RobWin or enter TEACH (then press enter twice)

The following keys now apply:



After entering TEACH you are now in "TEACH mode". To move the arm first select the joint to move, J1 for waist, J2 for shoulder, J3 for elbow, J4 for hand, J5 for wrist twist. On selecting a joint the terminal/computer will beep. Then press and hold pressed either **+** or **-** for motion in a positive or negative direction.

A slow speed is ideal for careful positioning. To change the speed press the SPD key then + to increase or – to decrease. If you used the T icon in RobWin then you can also click the pending dialog box and change the speed there to anything you want between 1 and 255.

To operate the gripper press the key marked 'GRIP', then to close the gripper press the **+** key and to open the gripper press the **-** key. Exit TEACH mode by pressing the escape (ESC) key on the computer keyboard.

If you press the Home key (an H with a red roof) the robot goes back to Home position.

Changing speed and acceleration.

The robot speed is set with the variable SPEED. The default value (after typing START) is 10000. You can change the speed with e.g.
5000 SPEED !
The ! means store. This line stores the value 5000 into the variable SPEED.
Try this: create a new definition thus:

```
: TEST TELL WAIST 5000 MOVE 0 MOVETO ;
```

Then try different speeds e.g.

```
1000 SPEED ! TEST
20000 SPEED ! TEST
10000 SPEED ! TEST
```

The maximum value of speed for an R17 is 32767 and for an R12 65535. High speeds should only be used with caution.

The motors have to ramp up in speed and ramp down at the end of the move. This is the variable ACCEL (acceleration)
Try

```
10000 SPEED !
2000 ACCEL ! TEST
500 ACCEL ! TEST
1000 ACCEL ! TEST
```

```
SETTINGS
```

Allows you to change speed and/or acceleration. Press enter to retain the existing value. Also shown is the speed limit of the track.

```
NORMAL
```

Restores the normal speed and acceleration settings.

It’s often useful to define words for speeds to suit your application, for example

```
: FAST 30000 SPEED ! 2000 ACCEL ! ;
: SLOW 2000 SPEED ! 1000 ACCEL ! ;
```

These can be used in later definitions.

==advanced==
If you want to temporarily increase speed and then restore the original value, you can save the original value on the stack, then recover it later, for example:

```
: SLOMO
SPEED @                    ( SAVE SPEED ON STACK )
1000 SPEED !               ( SET A LOW SPEED )
TELL SHOULDER 1000 MOVE    ( MOVE SHOULDER 1000 AT LOW SPEED )
SPEED !                    ( RECOVER ORIGINAL SPEED FROM STACK )
;
```

Calibration -2

The RoboForth word CALIBRATE seeks out the sensors and changes the counts to those in an array LIMITS. Thus the robot knows exactly where it is in relation to the HOME position.
VIEW LIMITS
Once it is calibrated there is no actual need to go to the HOME position. You can go directly from the calibrate position to any other learned position.

There is another word
CALCHECK
That merely checks the calibration and does not correct anything. If the robot is out of calibration more than a set amount then system stops and reports the error.

==advanced==
You can seek out the sensor on an axis with DATUM, for example
TELL WAIST DATUM
Of course the waist may simply move further away from the sensor so you need to use VIEW LIMITS to determine which way to search. If the value for waist is positive then from the HOME position
TELL WAIST DATUM will find the sensor.
If the value is negative then
TELL WAIST REVERSE DATUM will find the sensor.

Introduction to Cartesian mode.

In Cartesian mode it is possible to position the robot to any X-Y-Z position it can reach. To set Cartesian mode type

CARTESIAN or click
To return to joint mode enter
JOINT or click

At the HOME position type
WHERE

```
      WAIST  SHOULDER  ELBOW  L-HAND  WRIST   OBJECT
          0       0       0       0       0
```

Now enter
CARTESIAN or click
WHERE

```
          X       Y       Z    PITCH      W   OBJECT
          0       0    750.0   -90.0      0              (for R17)
          0       0    500.0   -90.0      0              (for R12)
```

The position referred to is the intersection of the hand pitch and hand roll (W) axes.
The place where X Y and Z are all zero (the origin) is at the intersection of the shoulder axes and the axis of rotation of the waist. This is clearly not at bench level. The distance between these two positions when the arm is dead straight (as it is in the HOME position) is 500mm for an R12 and 750mm for an R17.

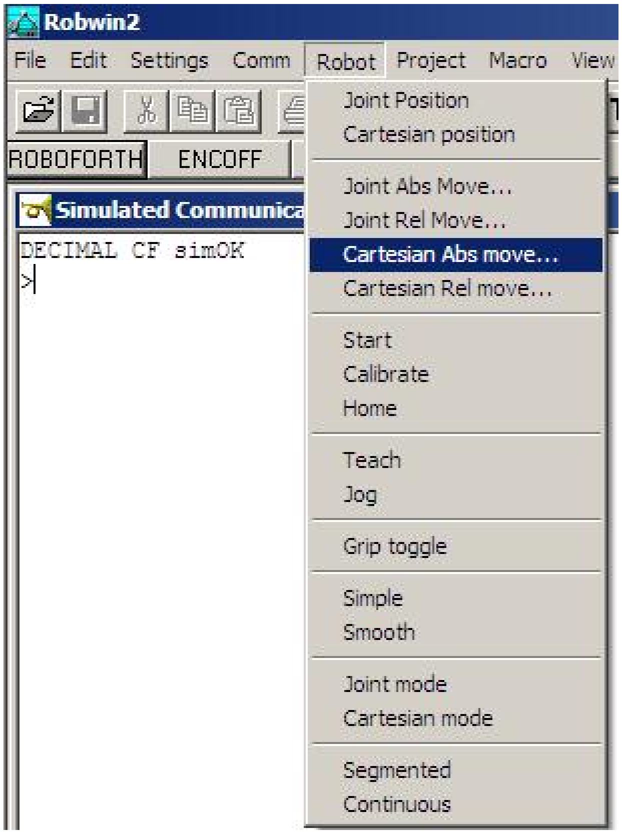
A PITCH value of –90.0 degrees is with the hand pointing up. 0 degrees is horizontal and 90.0 degrees is pointing down.

Important: RoboForth is an integer system. There are no decimal places. 500mm is expressed in RoboForth as 5000 i.e. 5000 units of 0.1mm each. 90 degrees is expressed as 900 i.e. 900 units of 0.1 degrees.

To start using Cartesian mode it is best to use RobWin to set up the required X-Y-Z position along with workable values for PITCH. Starting from the HOME position try

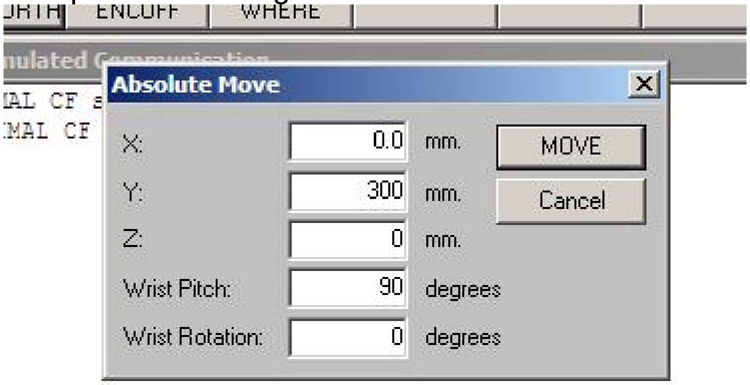
CARTESIAN or click

On the RobWin menu bar click **Robot**, then select **Cartesian Abs move**.



```
DECIMAL CF simOK
>
```

Complete the dialog box with



X:	0.0	mm.
Y:	300	mm.
Z:	0	mm.
Wrist Pitch:	90	degrees
Wrist Rotation:	0	degrees

And click **MOVE**

The robot will move to a position in front of HOME position with the wrist level with the shoulder. The position shown is the location of the center of intersection of the wrist roll and hand pitch joints.

0,0,0 is the center of the intersection of shoulder and waist axes.

So with Z=0 the wrist center is on the same level as the center of rotation of the shoulder.

In the communication window enter
WHERE

```
          X       Y       Z    PITCH      W   OBJECT
          0   300.0       0     90.0      0
```

Now in the communications window try
0 1000 0 MOVE (a relative command)
The robot will move forwards on the Y axis a further 100.0 mm. (remember 1000 is 1000 times 0.1mm units). (actually you can enter 100.0 – it makes no difference but looks better. Just 100 on its own would be 10.0 mm not 100mm)
WHERE

```
          X       Y       Z    PITCH      W   OBJECT
          0   400.0       0     90.0      0
```

0 3500 0 MOVETO (an absolute command)
WHERE

```
          X       Y       Z    PITCH      W   OBJECT
          0   350.0       0     90.0      0
```

The Cartesian system is top down. When you enter a Cartesian command you are merely changing the values of variables X Y Z and so on. The commands MOVE and MOVETO and other commands look at these variables and perform “reverse kinematics” to compute the motor values for each axis. However if you move any axis using a JOINT command then these variables are not updated, but stay at their previous values, now invalid. To update these variables from any robot position (forward kinematics) use COMPUTE. Even if the robot is in JOINT mode you can see the robot’s Cartesian position with CARTWHERE. So for example you could enter
JOINT TELL SHOULDER 1000 MOVE
COMPUTE CARTWHERE

Points

You can record temporarily or permanently any Cartesian position the robot is at. Just enter POINT and a name for example
POINT P1
You can return to this position at any time with
P1 GOTO

Using the teach pad in Cartesian mode

Once you have the robot in a convenient Cartesian position above you will find it easier to position the robot using Cartesian commands than Joint commands and the same applies to the teach pad. This time click the J button (or enter JOG). You will see an increment shown in a dialog box, default value is 10.0mm. You can change this to anything from 0.1mm to 25.5mm (note that 25.4mm is one inch).
Select an axis e.g. X Y or Z.
Now press **+** or **–** and the robot will move in that direction along that axis by the amount set as increment. It just moves once then stops. Release the key and press again for another movement. You can increase/reduce the increment by pressing SPD then **+** or **–**
The increments are set values – to pick a value between them use the dialog box.
J4 is the hand pitch. Default is 10.0 degrees per increment.
J5 is wrist rotate, default is 10.0 degrees per increment.

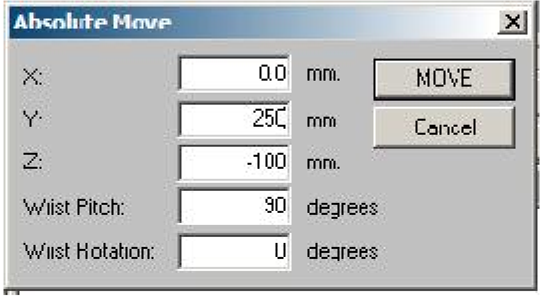
ALIGN mode

Try selecting X and move in one direction. As the robot moves along this axis you will see that the shoulder and elbow adjust to lengthen the distance between the shoulder and hand and thus keep the hand on the axis i.e. constant Y value.
You will also see that the gripper goes round with the robot because it is not moving relative to the arm. If the gripper is vertical (PITCH is 90.0 degrees) then ALIGN mode can keep it in line with the X and Y axes and not rotate with the arm.
Exit JOG mode with the escape key.
Enter ALIGN
You will see the wrist immediately aligns with the X axis.
Enter NONALIGN and the wrist returns to original position.
Type WHERE – the value under W is the angle of the wrist to the robot. In ALIGN mode it is the angle to the X axis.

Walk-through of ALIGN mode.

1. Put the robot in a suitable starting position for this demonstration. Type `HOME` and `CARTESIAN` then click **robot, Cartesian absolute move**. Send robot to, say X=0, Y=250mm, Z=-100mm, wrist pitch=90 degrees, wrist rotation=0 as in fig.1. Click MOVE. The robot should look something like fig. 2:

fig. 1



Absolute Move

X: 0.0 mm.

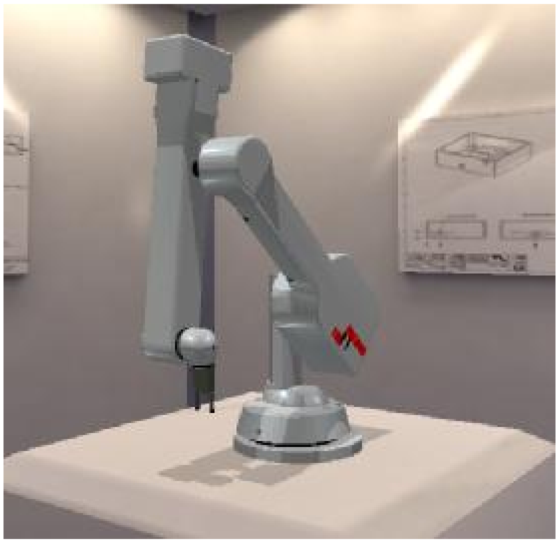
Y: 250 mm

Z: -100 mm.

Wrist Pitch: 90 degrees

Wrist Rotation: 0 degrees

fig. 2



2. Now move the robot sideways using the communication window. Enter

2000 2500 –1000 MOVETO

You'll notice that as the waist rotates the shoulder, elbow and hand have all had to adjust to keep the gripper at the same Y distance. You'll also notice that the wrist has rotated with the waist as in fig.3.

3. The angle of the wrist is still zero but to the arm not to the real world. To make it zero to the X axis enter

ALIGN

as in fig 4.

fig.3

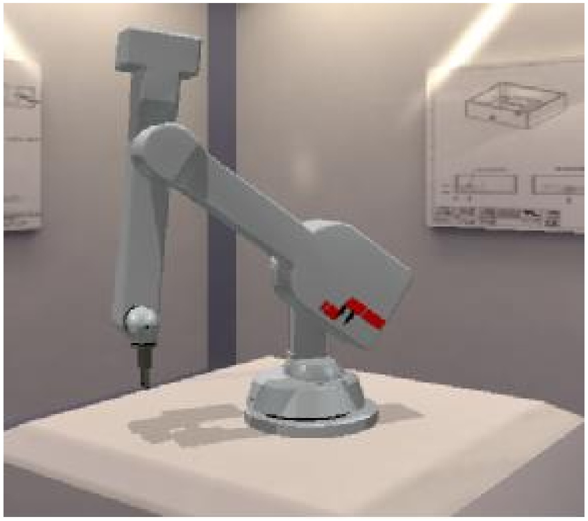
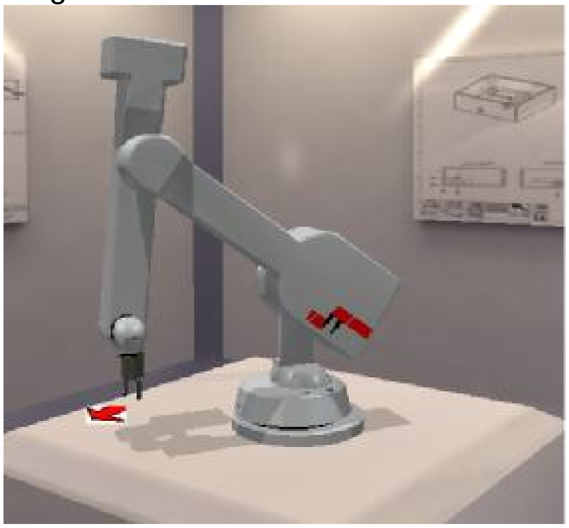


fig.4



4. Now move the robot to the other side. Enter
-2000 2500 –1000 MOVETO
The wrist rotates the other way to hold zero angle to the X axis as in fig. 5
5. Try moving the robot down 50mm, enter
0 0 –500 MOVE
The hand remains vertical and the wrist remains at the same angle to the X axis as in fig. 6

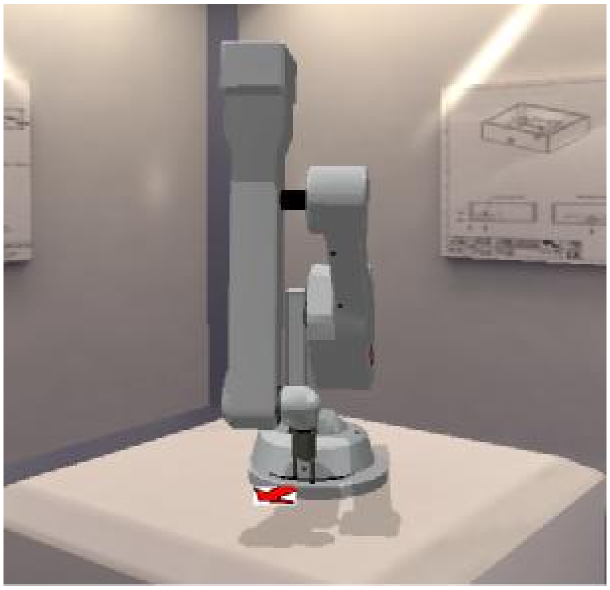


fig.5

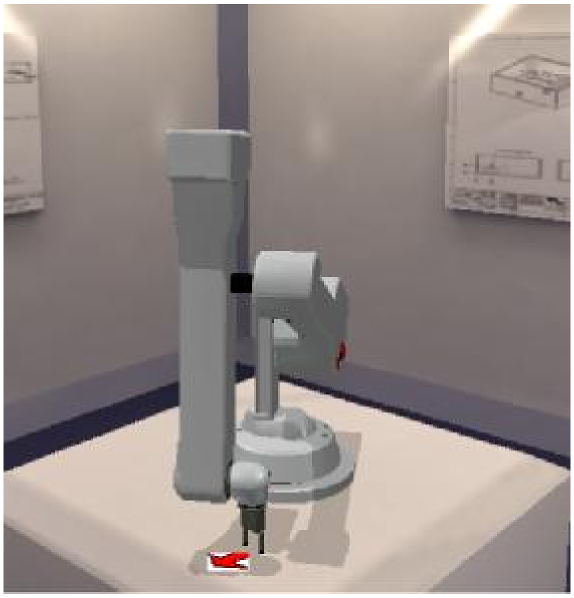


fig.6

From this you can see that moving the robot left or right, up or down, the wrist will hold a constant attitude to the X-Y and Z axes.

Tool transformations

Besides ALIGN mode you can also change the value of a variable TOOL-LENGTH e.g.
1000 TOOL-LENGTH ! sets the length to 100 mm.
This is the distance between the wrist center (intersection of hand pitch and roll axes) and the end of the tool. By changing this value you position the end of the tool at the chosen X-Y-Z position and not the center of the hand.
TOOL invokes a text dialog in the communication window that asks certain questions.

```
WRIST PITCH
```

enter a value for wrist pitch in degrees times 10 or just press return to retain the existing value

```
WRIST ROLL (W)
```

enter a value for wrist roll in degrees times 10 or just press return to retain the existing value

```
TOOL LENGTH
```

enter a value in mm times 10 from the center of the hand to the tool tip, or just press return to retain the existing value

`ALIGN ROLL?` Y to invoke ALIGN mode or N for non-align.

`EXECUTE?` Y to execute the above changes immediately.

Or press enter to defer the changes. They will take place the next time you use a MOVE or MOVETO command.
So to invoke ALIGN mode you could just type TOOL, then press enter, enter, enter, Y, Y
For normal use the value of TOOL-LENGTH is left at zero.
Also try changing the angle of the hand using TOOL then enter
1000 PLUNGE – the robot moves 100.0 mm in whatever direction the hand is pointing.

Walk-through using TOOL-LENGTH

1. Return to starting position as in the example of use of ALIGN mode above. Click **robot, Cartesian absolute move**. Send robot to, say X=0, Y=250mm, Z=-100mm, wrist pitch=90 degrees, wrist rotation=0. Click MOVE.
2. Bring the hand up to level. Click **robot, Cartesian absolute move**. Change wrist pitch to 0 degrees. The hand is now level but the gripper or your end effector is no longer at the chosen Y distance (fig. 7).

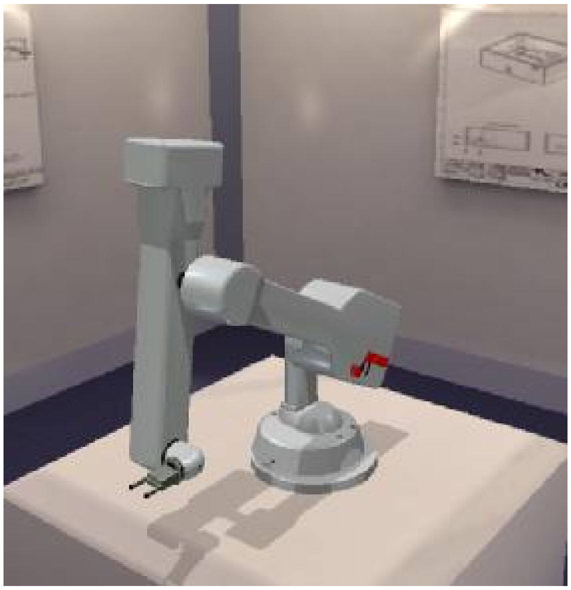


Fig. 7

3. Measure the distance from the center of the wrist/hand joint and the tip of the gripper or centerline of your end effector. Suppose this is, say 100mm. Enter

```
1000 TOOL-LENGTH !
```

4. Now click **robot, Cartesian abs. Move**, and just click ok
The robot should pull back so that the gripper is over the chosen position, as in fig.8.
5. Move the robot sideways with

```
2000 2500 –1000 MOVETO
```

Even though the hand has to point in a different direction the gripper (or your end effector) will still be on the same Y-distance line. The robot should look as in fig. 9.

fig. 8

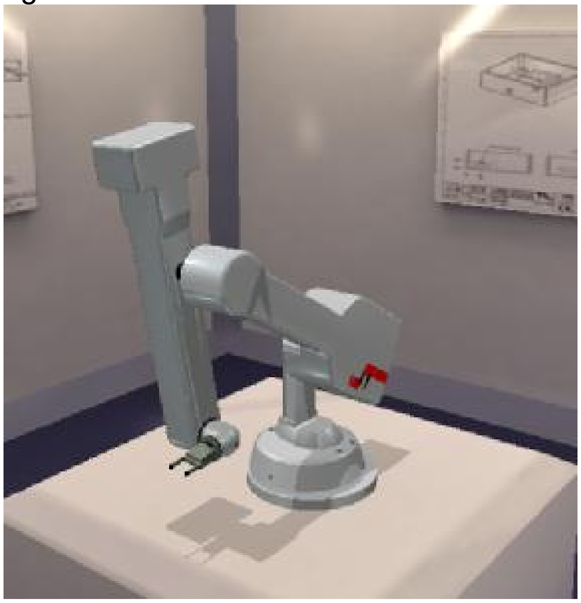
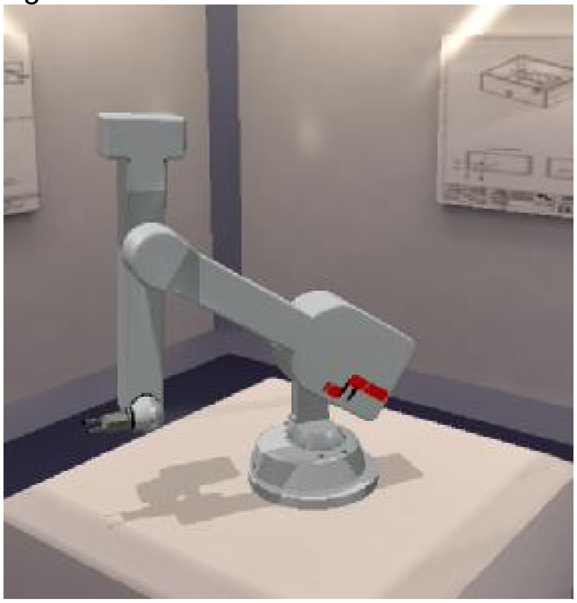


fig. 9



Learning positions

You can create named positions and lists directly in the communications window. However you need to use Robwin so that your positions are also saved on disk together with all your programming.

Start a project

Start by creating a new project. Go to RobWin **project, new,** and choose a name e.g. **myprog.** Three more windows will appear – the myprog.ed2 text window, Places and Routes. You are now ready to program the robot and not just move it about.

Programming using PLACE names.

If you wish you can create a place directly in the communications window with the command PLACE e.g. PLACE JIG creates an entity called JIG. The coordinates of JIG are the joint values of wherever the robot happens to be when you created it. Just using the name JIG at any future time will send the robot back to those coordinates.

Use RobWin to create a place called JIG.
First use the teach pad to guide the robot to the required position. Using CARTESIAN mode and the Jog button J is best. In the Places box click Add New. Enter JIG as the place name.
Now type HOME – robot goes back to HOME
Now type JIG – robot goes back to the learned coordinates of JIG
You can add an approach position to JIG. Simply use the teach pad to move the robot up away from JIG a small amount. In the Places window highlight JIG and click Set Approach. Now when you type JIG the robot goes first to the approach position then to the target position.
To move back again to the approach position use
WITHDRAW
Note that the coordinates stored in JIG are Joint values not Cartesian values.
If you wish to change the coordinates at any time guide the robot to the new position then click Set to Here. The approach position also moves, following the change in target position. This is because the coordinates learned for the approach are relative coordinates.

Now create a second place say 300mm to the left called FEEDER and a third place further still called BIN

If you want to see the actual data in a PLACE name then go to the Places listbox and click **Show data.** You can also VIEW a place, e.g.
VIEW FEEDER
This will show the coordinates and the relative coordinates of the approach position (marked with an R).

Objects - 1

You can define objects that the system keeps track of as they are moved around. There are a number of advantages – 1. finding unique objects and 2. registering whether a position or lots of positions are occupied with an object or a certain type of object.

You can not use RobWin to create an object, but must do it in the communications window or the myprog.ed2 window (see later)

Suppose you have an object called PART

```
OBJECT PART
```

You can set a starting place for this object with

```
PART ISAT FEEDER
```

Now you can move the part with

```
FEEDER  GET
JIG  PUT
```

```
WHEREIS PART JIG OK
```

To repeat this you need to put a new PART in FEEDER and empty the JIG

```
PART ISAT FEEDER
0 ISAT JIG
FEEDER GET
JIG PUT
```

If you try this again but leave out 0 ISAT JIG you will get an error

```
JIG PUT POSITION OCCUPIED
```

```
VIEW FEEDER
```

will show the coordinates and the relative coordinates of the approach position (marked with an R) and any object that is there.

If an object is being held by the robot then WHERE will show the name of the object.

WHEREIS PART tells you where the part is, or if it is being held by the robot.

To tell the robot what it is holding enter

```
HOLDING PART
```

Timers

You can create a time delay with MSECS e.g.
1000 MSECS (1 seconds delay then) OK
You can also measure the time something takes…
RESETMS resets the timer to zero. It starts running right away
MSECS? leaves the value of the timer in milliseconds on the stack. You can then print it with dot.
Example:
RESETMS TELL SHOULDER 1000 MOVE MSECS? **.**
will print the time it took for the shoulder to move 1000 steps.
(reminder: dot on the end means print)
There is also a microsecond timer e.g.
500 USECS gives a delay of 500 microseconds.

Creation and downloading of text file for pick-and-place routine. (.ED2 file)

Although you can type a definition of a new word directly into the communications window, if you do you will not be able to edit it or save it to disk. So new definitions should always be entered into the ed2 window, (project-name.ED2). When you have added or edited a definition you then download it into the controller with the green down-arrow ⬇ which simultaneously saves the text to disk, a text file project-name.ED2

The above programming with objects is best typed into the myprog.ed2 window where you can edit it or add words later.

You already have three PLACEs created – JIG, FEEDER and BIN
If you are starting here then create those 3 places again as described previously.

In the ED2 text window enter the following text:

```
( MYPROG )
OBJECT PART
: TASK
PART ISAT FEEDER
FEEDER GET
JIG PUT
5000 MSECS
JIG GET
BIN PUT
0 ISAT BIN
;
```

Now click the green down-arrow ⬇ to download this into the controller.
In the communications window enter

```
TASK
```

To run the whole program.

If you want to edit it say to change from 5 seconds to 4 seconds then make the edit and click the red arrow again. The code in the controller is replaced with the new code.

Save the project with **project, save.**
NOTE: You should save the file and shut down RobWin *before* you disconnect from the controller or switch off the controller.

To save the project in controller flash memory enter

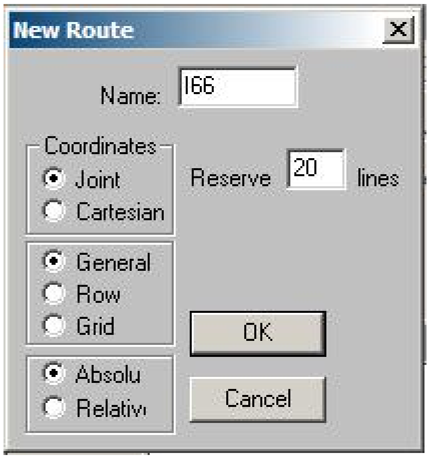
```
USAVE
```

Programming using named lists called ROUTEs.

A route is a list of positions. Originally it was just a path that the robot followed from one line to the next but the same construct was later used to create a row of equally spaced positions or a matrix.

A route may be a list of Joint positions or a list of Cartesian positions. In RoboForth you could create a route directly in the communications window as ROUTE (name). But RobWin should be used. When created the system also asks how much space you will require (reserved lines).

Create the simplest route using RobWin called, say PATH1 as follows:
Click the routes Window.
(tip: if you have accidentally closed the routes or places windows then go to project, routes or project, places to remake the window.)
Go to the Routes box and click **New**.



Name: I66

Coordinates
● Joint
○ Cartesian

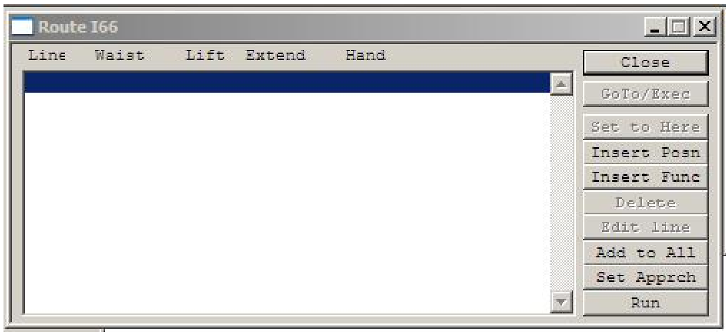
Reserve 20 lines

● General
○ Row
○ Grid

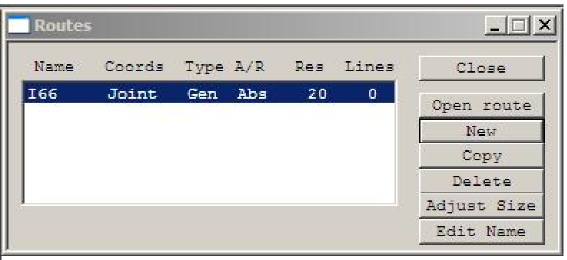
● Absolu
○ Relativ

Give the new route a name, for example **I66**
Note that 20 lines are automatically reserved for this route, but if you are expecting to create a route with more positions than that you can enter a bigger number. If you don't enter a big enough number and need more later you can still make the route bigger at a later time. The maximum number of positions the controller will store is about 4000 which includes any reserved but unused lines. If you end up using more than 4000 and you have some unused reserved lines you can make a route smaller at a later time.

Once you have created your route you will see a new listbox window for I66 and I66 entered as a route in the Routes listbox.



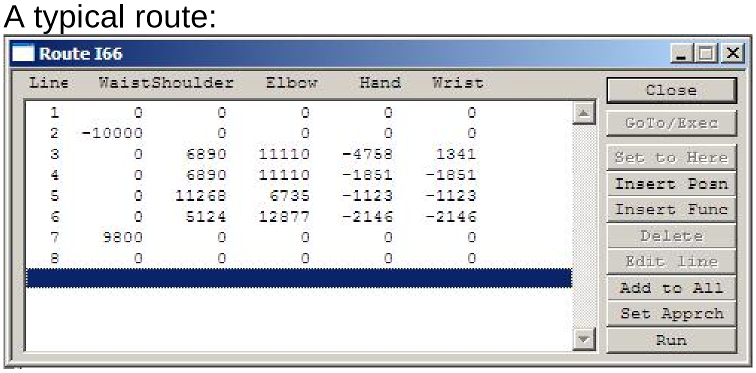
Line	Waist	Lift	Extend	Hand



Name	Coords	Type	A/R	Res	Lines
I66	Joint	Gen	Abs	20	0

To learn positions into the route click the **Insert Posn** button. This learns the current position into the list. The next blank line will be highlighted.

A typical route:



Line	Waist	Shoulder	Elbow	Hand	Wrist
1	0	0	0	0	0
2	-10000	0	0	0	0
3	0	6890	11110	-4758	1341
4	0	6890	11110	-1851	-1851
5	0	11268	6735	-1123	-1123
6	0	5124	12877	-2146	-2146
7	9800	0	0	0	0
8	0	0	0	0	0

Now move the robot again and click **Insert Posn** button and so on.

Although you are using a button marked “insert” you are inserting a line on the*end* of the list.

You can also use the teach pad for this. If you are using the teach pad to move the robot about you can press the tick key and the current position of the robot will be added as the next line of the route. Press the FN and cross (red X) keys together and the last position learned will be removed. (The FN key acts here like a shift key – press and hold it, then the red X.

At this stage be sure that you are in JOINT mode using TEACH - the T icon. Also bear in mind that if you have more than one route that route must be selected before using TEACH tick or cross. The system needs to know which route you are working on – see later.

If you wish to insert a line within the list then highlight where you want it inserted by clicking on that line then click **Insert Posn.** The new line will be inserted in front of the highlighted line.

If you wish to edit a position move the robot to the required position, highlight the line you want to change and click **Set to here**

Another way to edit a line is to click Edit Line then enter the 5 axis values manually.

If you want to delete a line highlight it and click **Delete**

Finally to run your route click **Run**

But clicking Run is for testing, not a way of using the route. In the communications window type RUN You can also compile RUN into other definitions.

You can add the route to a definition in your text window e.g.

```
: TASK                              }
PART ISAT FEEDER                    }
FEEDER GET                          } this text
JIG PUT                             } already
5000 MSECS                          } in the text window
JIG GET                             }
BIN PUT                             }
0 ISAT BIN                          }
;                                   }
: TEST
I66 RUN
HOME
;
```

Click ⬇ and enter TEST in the communications window.
You have two top-level words now – TASK and TEST.
You can retrace your steps in the route, i.e. run the route backwards with RETRACE

Other things you can do:

LEARN
this learns a new position on the end of the route, the robot's current position. It is the same command that is sent by the **Insert Posn** button but the data is only learned in the controller and not in the computer. So you won't see the data change in the route window.

n INSERT
this inserts the current position of the robot as a new line in front of line n. It is the same command that is sent by the **Set to here** button but the data is only inserted in the controller and not in the computer. So you won't see the data change in the route window.

n DELETE
this deletes line n. It is the same command that is sent by the **Delete** button but the data is replaced only in the controller and not in the computer. So you won't see the data change in the route window.

UNLEARN
removes the last position LEARNed.

n REPLACE
this replaces line n with the current position of the robot. It is the same command that is sent by the **Set to here** button but the data is replaced only in the controller and not in the computer. So you won't see the data change in the route window.

L**.** (L dot)
This lists a route on screen.

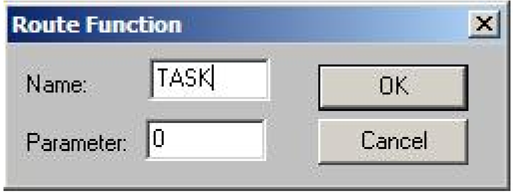
To copy the data up from controller to computer use the red up-arrow ⬆

The REPLACE command may be useful where line n needs to be changed during the flow of the program. It may not be necessary to change the data in the computer because that data is dynamic and changes as the robot runs. Examples later.

n GOTO

The robot will go directly to the position in line n. This is the same as to highlight a line and click **Goto/Exec**

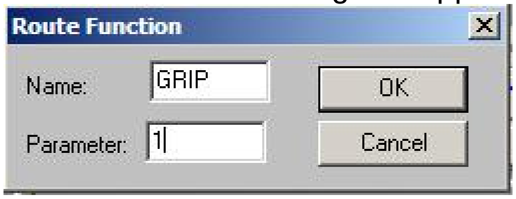
It is possible to have some other action take place at some point in the route using **Insert Func.** A new dialog box will appear.



Name: TASK
Parameter: 0

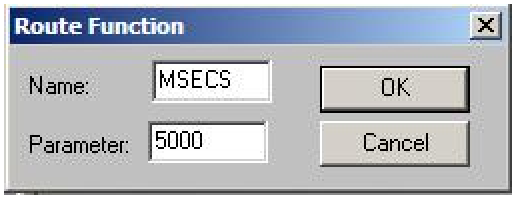
Click line to insert in front of. Enter the word you want executed, for example you could enter TASK and have that executed at line n, then the route would continue after it completes. The dialog box also asks for a parameter – enter zero.

If you want the gripper to operate at a particular point in the route highlight the line and click **Insert Func.** A new dialog box appears. Enter GRIP and either a 1 to GRIP or a 0 to UNGRIP.



Name: GRIP
Parameter: 1

You can also include a delay in the route using MSECS. For example if you want a 5 second delay between lines 4 and 5 then click line 5, click **Insert Func**, enter MSECS in the dialog box and 5000 for the value.



Name: MSECS
Parameter: 5000

All the commands described work on the currently selected route. To RUN the route I66 enter I66 RUN. If you type 5 GOTO the robot will go to line 5 of I66. All the words like RUN, RETRACE, GOTO, REPLACE, LEARN, INSERT, DELETE etc all apply to I66 until you type the name of another route. For example I66 5 GOTO M1 5 GOTO will send the robot first to line 5 of route I66 then to line 5 of route M1. You only need type or use the name of the route once and you can keep using those dependent words until you type or use the name of another route.

I66 5 GOTO M1 5 REPLACE RUN

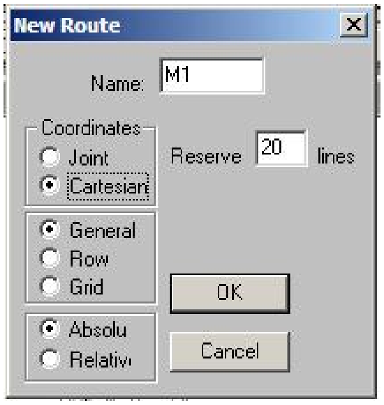
sends the robot to position 5 of route I66, changes the contents of line 5 of route M1 to the same coordinates then RUNs route M1.

Cartesian routes

Using Cartesian coordinates is preferable and you can create a route that uses Cartesian coordinates. First get the system into Cartesian mode with

CARTESIAN or click

In the routes listbox click New. Let’s call this new route M1



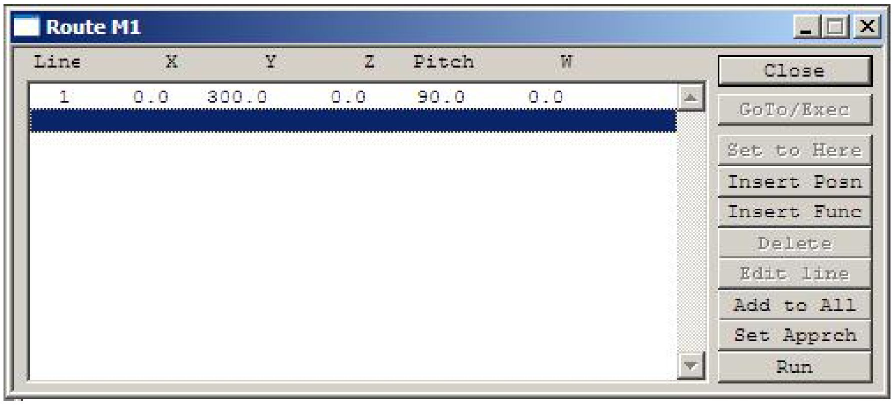
Name: M1

Coordinates ○ Joint ● Cartesian

Reserve 20 lines

● General ○ Row ○ Grid

● Absolu ○ Relativ



Line	X	Y	Z	Pitch	W
1	0.0	300.0	0.0	90.0	0.0

Enter the name of the route and check the Cartesian ‘radar’ button. Click OK and the new Cartesian route opens. As you can see the heading is now in Cartesian terms rather than axis names. Position the robot and press **Insert Posn** to learn the first position, example shown. Continue learning new positions as previously described. If you want to use the teach pad use JOG or the J button. Press the tick to learn the next position in sequence. Press the cross to delete the last position learned.

If you wish to shift the whole route in any direction you can use the **Add to All** button. For example if you wanted everything to be 10mm higher just click **Add to All** and enter 10.0mm in the Z space.

Continuous Path

When you typed RUN you saw that the robot goes from one position to the next, stopping at each line. This is SEGMENTED mode. If you want the robot to go through these positions without stopping enter CONTINUOUS or click
Then enter RUN
Instead of running the route you might get an error “too tight, line….”
This is the reason:
The robot does not actually run through the points as such, but runs past them as it changes direction for the next point. The best analogy is a series of right-angle turns but the maths is the same for any change of direction or speed. See the diagrams below.

Diagram 1

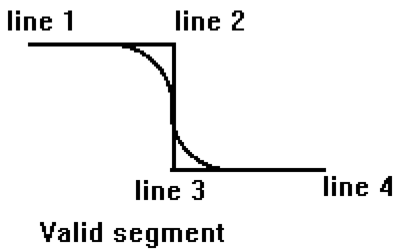
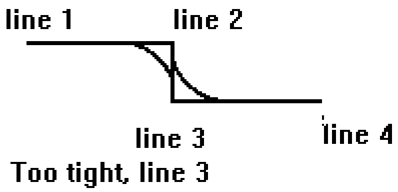


Diagram 2



In diagram 1 the robot rounds off the corner at line 2 successfully in time to start rounding off the corner at line 3. But in diagram 2 lines 2 and 3 are too close together for this to be possible. This would result in the error. The solutions are either to reduce the speed or increased the acceleration. Either would result in a tighter turn that could fit in the time and distance allowed for the turn.

The simplest solution is to enter ADJUST. This hashes the speed down to a value that does not result in the error. It does this by the CPU sending trial speeds to the DSP and asking it if that is OK.

ADJUST reduces SPEED and announces the value. If that is too slow you will need to reset SPEED to a high value, increase ACCEL and try ADJUST again.

You can include ADJUST in a definition, for example

```
: TEST
I66 ADJUST RUN
HOME
;
```

The word RUN has these important features:

1. It asks the DSP if the current value of SPEED is workable.
2. It tells the DSP to run the route.
3. It then waits for the DSP to finish, then you see OK
4. It monitors the stop button and if that is pressed it sends a STOP command to the DSP. Then waits for the DSP to decelerate all the motors and finish.
5. It asks the DSP how much it moved all the motors and updates the counts.

No.1 in the list above can take some time for a very long route. There could be a small but noticeable delay.
ADJUST may also take some time but if the speed is already low enough no time is wasted.
DRY RUN tests for too tight errors but does not actually run the robot.

==advanced==

CRUN tells the DSP to get on and run the route but

1. There is no check for a valid route. Therefore you must be sure you have a valid speed before using it. It may be a good idea to use DRY RUN or ADJUST to make sure you have a valid speed before using CRUN
2. It does not wait for the DSP to finish but gives you OK immediately (or carries on to the next word in your definition). Obviously the CPU can therefore do something else while the DSP is moving the robot.
3. It does not check the stop button. In your following words you need to keep checking the stop button. You can stop motion either because the stop button was pressed or some other event with the word STOP
4. It does not update the counts after the DSP has finished. To do that you need to read back the DSP with DSPASSUME

Try a really low speed and start the robot running with
I66 CRUN OK
While the robot is running enter
STOP
Robot stops. Type WHERE – the counts have not changed. Now type
DSPASSUME
WHERE

How to find out if the DSP has stopped:
Use the word ?RUN – this leaves 0 on the stack if the DSP has stopped.

Example:

```
: WAIT-DSP-STOP
BEGIN
?RUN 0 = UNTIL
;
```

How to find out if the stop button has been pressed and what to do about it:

```
    STOP? IF                         ( stop pressed?
      STOP                           ( tell DSP to stop
      FSTOP C1SET                    ( set stopped flag
      BEGIN ?RUN 0= UNTIL            ( wait for DSP to finish
    THEN
```

Always end with DSPASSUME

Of course if you go too far with these you might just as well use RUN. The reason for using CRUN is so that you can do something else while the robot is running, but somewhere in your code must be a check for stop button and a check for when the DSP has finished.

GOTO has variants.
n GOTO goes to line n of the currently activated route. But you can also write
n LINE GOTO
n LINE computes the actual memory address of the data and leaves it on the stack for GOTO to use and send the robot there.
–GOTO is a special case. It will go to the coordinates of any memory location at all that contains valid coordinates. It only works with motor coordinates and not Cartsian. For example
LIMITS –GOTO sends the robot to where the calibration values are stored.

Another DSP command is DSPSMOOTH. This is the same as GOTO but passes the command to the DSP and returns control immediately to the CPU in the same way that CRUN does, but for a single position, not a route.

Examples

```
LIMITS DSPSMOOTH OK
```

Robot goes directly to the calibrate position and CPU says OK while robot is still moving.

```
5 LINE DSPSMOOTH OK
```

robot goes directly to line 5 of the selected route and CPU says OK while robot is still moving.
As with CRUN you must interrogate the DSP to find out where it got to with
DSPASSUME

You can move a single axis in a similar way e.g.

```
TELL SHOULDER 1000 DSPMOVE OK
```

Similarly use DSPASSUME when the motion is finished.

==advanced==

Structure of data in RoboForth

The coordinates of any robot position are stored in 16-byte arrays – view them as 8 16-bit elements.

0,1	2.3	4.5	6.7	8.9	10,11	12,13	14,15
Waist/X	Shoulder/Y	Elbow/Z	Motor 4 /PITCH	Wrist/W	Motor 6 (track)	EXAD	OBAD

Note that bytes 6 and 7 are referred to as motor 4 not hand pitch. This is because hand pitch requires two motors to pitch – motors 4 and 5 together. Motor 5 on its own is wrist rotate.
OBAD contains the CFA of any object that is at that location.
EXAD contains an optional code. This is:
0 means the coordinates are motor values and represent an ABSOLUTE JOINT position.
Case: a hypothetical line 5. If the EXAD contained 0 and you wrote
5 GOTO
then the robot would go to a position where all the counts were the same as the first 5 values in line 5.

1 means the coordinates are motor values and represent a RELATIVE JOINT position. So if you enter
5 GOTO
The values in line 5 would be ADDED to the current position of the robot and the robot would move BY that amount and not to the values themselves.
2 means the coordinates are Cartesian values and represent an ABSOLUTE CARTESIAN position.
5 GOTO sends the robot to that X-Y-Z position.
3 means the coordinates are Cartesian values and represent a RELATIVE CARTESIAN position. So 5 GOTO would take whatever position the robot was currently at, add the values in the line to obtain a new position and go there. In other words the robot moves BY those values and not to the values as absolute coordinates.
If the EXAD value is much larger then it must be the actual CFA of a word. For example if you had GRIP as line 5 then the 12th,13th bytes of the strip of data for line 5 would contain the CFA (described earlier).
Now if you enter 5 GOTO the gripper will operate.

Not so advanced

The upshot of the above is that there are 5 possible types of position:
ABSOLUTE JOINT
RELATIVE JOINT
ABSOLUTE CARTESIAN
RELATIVE CARTESIAN
FUNCTION (a word learned with **Insert Func**)
If a line contains a relative position you will see it listed with an R by it.
If you GOTO a line which is a relative position then the robot does not go to that position but moves BY that amount. So if you are in CARTESIAN mode and if you were to see a line listed thus:

```
Line      X       Y       Z    Pitch      W
05      0.0     0.0    10.0      0.0    0.0 R
```

then you were to type
5 GOTO
the robot would move up 10mm in the Z axis. Each time you type 5 GOTO it would move up another 10mm.

The RoboForth word **ADD** adds two lines together. It can add an absolute position to a relative position (marked R) resulting in another absolute position. Or it can add two relative positions together to make another relative position.

Example

L. (L dot)

```
Line      X       Y       Z    Pitch      W
01      0.0   300.0    50.0     90.0    0.0
02      0.0     0.0     0.0      0.0    0.0
03      0.0     0.0     0.0      0.0    0.0
04      0.0     0.0     0.0      0.0    0.0
05      0.0     0.0    10.0      0.0    0.0 R
```

1 LINE 5 LINE ADD GOTO WHERE

```
    X       Y       Z    Pitch      W
  0.0   300.0    60.0     90.0    0.0
```

START-HERE and END-THERE

There may be situations where you want to do the exact same sequence of moves but in different parts of the workspace. For example you might have a number of different starting positions but from there on the motion is the same. Simply learn your route from the first starting position (or indeed from anywhere). Send the robot to the desired starting position. Then use the route name and START-HERE. The first line of the route will change to the same as the actual position of the robot and all the other lines will shift by the same amount. So you can use for example

M1 START-HERE RUN

A similar function is END-THERE but that is harder to use. You can't send the robot to the ending position and then run the route, so you have to specify that position first. Suppose you have a point P1. You would write P1 then the name of the route then END-THERE, for example

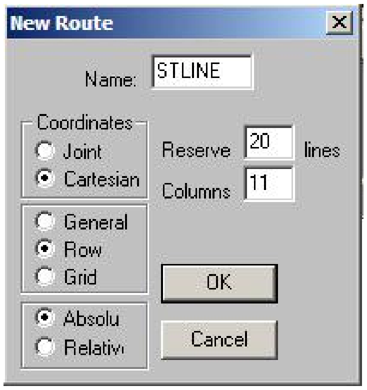
P1 M1 END-THERE RUN

Or if, for example, the ending position of M1 needed to be the starting position of another route, say M2 then you would write

M2 1 LINE M1 END-THERE RUN

Creating a straight line

When moving from one XYZ position to another the robot does move in a straight line but takes the easiest path with least motor motion which will be a curved trajectory. If you want a straight line then in the Routes listbox click New. Select Cartesian Row



Name: STLINE

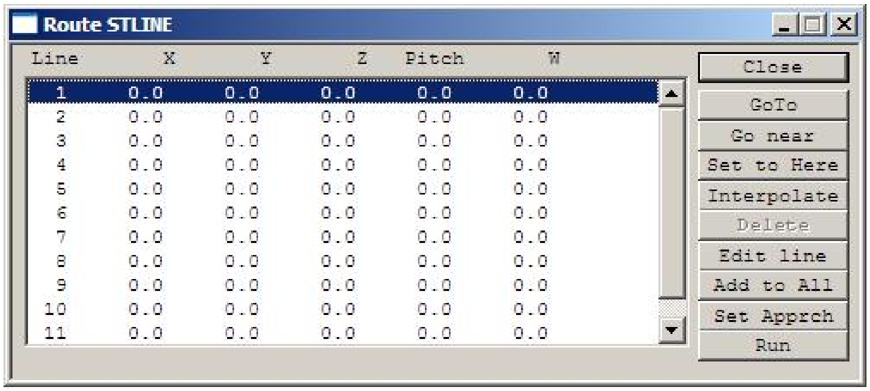
Coordinates
○ Joint
● Cartesian

Reserve 20 lines

Columns 11

○ General
● Row
○ Grid

● Absolu
○ Relativ



Line	X	Y	Z	Pitch	W
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0

A straight line is merely made up of a number of shorter moves. Each move is itself a slight curve so you need to decide how many segments you require. There will of course be 1 more line than segments. In the above example there are 10 segments, that's 11 lines or columns. If you wanted a much straighter route then pick say 100 segments – 101 columns. In that case you would also need to reserve more lines than 20, say 110.
You now see all 11 lines but all are zero.

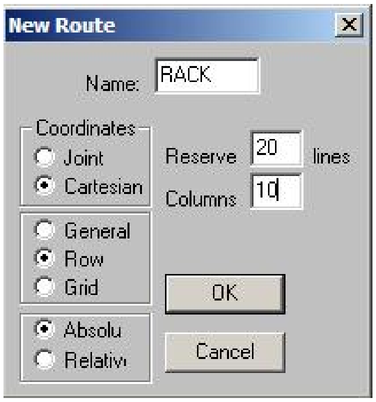
Now move the robot to the start of the line. Highlight line 1 and click **Set to Here**
Next move the robot to the end of the line. Highlight line 11 and click **Set to Here**
Finally click **Interpolate** and all the lines between are computed.
Guide the robot back to the start of the line and enter RUN.
The route will progress line by line. For a continuous path enter SMOOTH. This can result in SPEED = some value that is lower than what you have set. This is because when lines are very close the DSP algorithm can only run the motors slower.

How close to make them?
It depends on how straight you want the line to be. The robot arcs slightly from one point to the next. If you want the deviation to be smaller use more points. There is a small exe program to help you that can calculate the number of points required for any maximum given deviation. Just run straightline.exe.
Enter the total length of move you require in mm times 10.
Enter the distance from the wrist to the nearest center of rotation. This will normally be the length of the fore-arm so for an R12 enter 2500. For an R17 enter 3750.
Enter the maximum accepted deviation. For example if your line must not deviate more than 0.5mm then enter 5
Click calculate. This gives you the total number of moves – i.e. the total number of segments in the route. The number of lines required will be one greater. For example if calculated number of moves is 4 then make your route with 5 lines.

Rows and matrices

Case: A rack of 10 test tubes.

You can create a row exactly as above where the first and last lines are each end of the rack. Obviously you would not RUN this route or it would create a lot of broken glass. But you can GOTO individual positions, for example 1 GOTO would send the robot to the first tube. Create a new route with 10 lines for 10 tubes. For larger numbers make sure the amount reserved is the same plus at least one. Use JOG J to create each position in the rack.


New Route

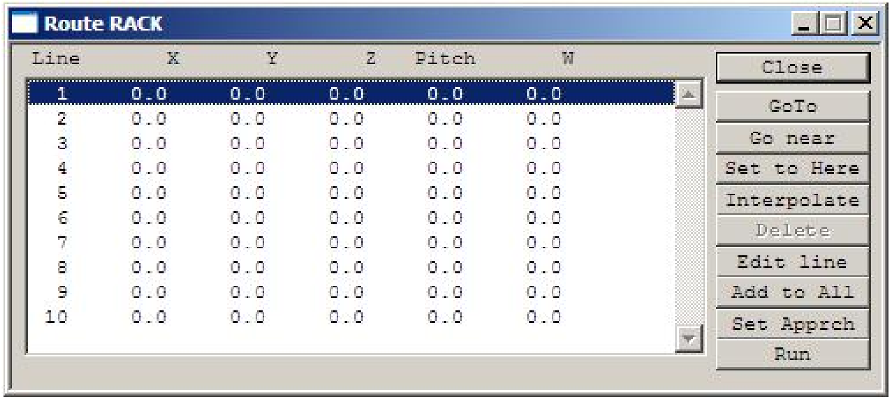
Name: RACK

Coordinates
○ Joint
● Cartesian

Reserve 20 lines
Columns 10

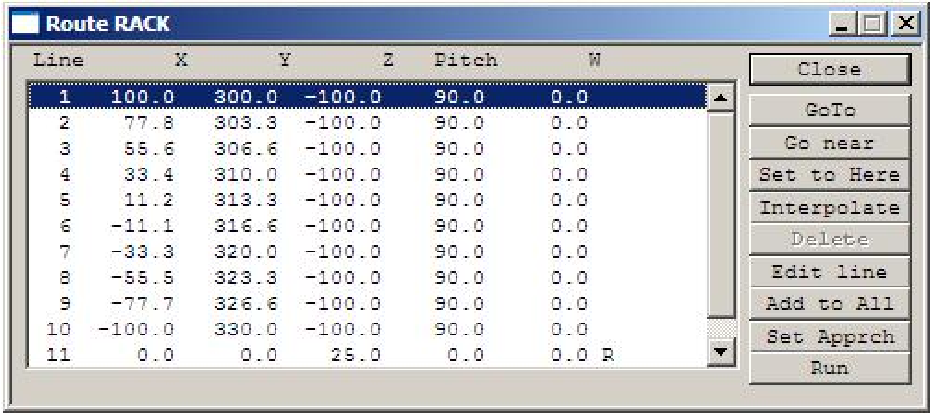
○ General
● Row
○ Grid

● Absolu
○ Relativ


Route RACK

Line	X	Y	Z	Pitch	W
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0

It is necessary to approach each tube from above (or side etc) so you need another 10 positions above the tubes. The way to do that is to create a common approach position that is relative to the others. Finally set a large enough increment and use JOG to move up once only to a position above the tube. OR you can use for example 0 0 250 MOVE which moves the robot up 25mm. Now click **Set Apprch.** You will see an 11th line with 0 for X, 0 for Y and 25.0 for Z. At the end of the line is an R flag meaning this is a RELATIVE route. The system can now ADD this line to all the other 10 to effectively create 10 more lines 25mm up on Z. If this value turns out to be wrong you can edit it using **Edit line.** The finished result will look something like this:

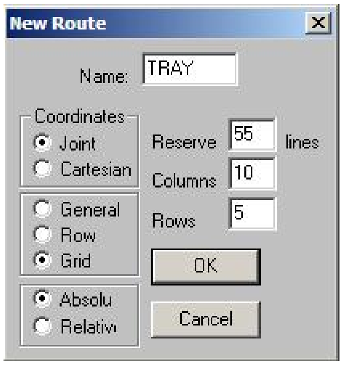

Route RACK

Line	X	Y	Z	Pitch	W	
1	100.0	300.0	-100.0	90.0	0.0	
2	77.8	303.3	-100.0	90.0	0.0	
3	55.6	306.6	-100.0	90.0	0.0	
4	33.4	310.0	-100.0	90.0	0.0	
5	11.2	313.3	-100.0	90.0	0.0	
6	-11.1	316.6	-100.0	90.0	0.0	
7	-33.3	320.0	-100.0	90.0	0.0	
8	-55.5	323.3	-100.0	90.0	0.0	
9	-77.7	326.6	-100.0	90.0	0.0	
10	-100.0	330.0	-100.0	90.0	0.0	
11	0.0	0.0	25.0	0.0	0.0	R

Again if you wish to shift the whole route in any direction you can use the **Add to All** button. For example if you wanted everything to be 10mm higher just click **Add to All** and enter 10.0mm in the Z space.

To automatically use this approach position do not use GOTO but use INTO e.g.
1 INTO sends the robot to position 1 via its approach position 50mm higher.
To withdraw from the position do not use WITHDRAW which is a PLACEs word, but simply use
UP
If you wanted to go to the approach position only use
1 NEAR

Case: a rack of tubes in a matrix 10 by 5, total 50 tubes.
In the routes window click **new**.
In the new route dialog choose Cartesian, Grid. Colums will be 10 and Rows 5. You will need to reserve space for at least 50 lines, choose 55



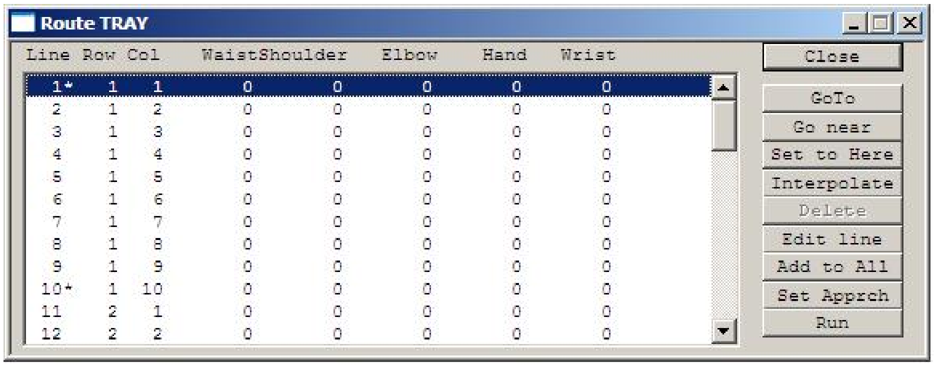
Name: TRAY

Coordinates
● Joint
○ Cartesian

Reserve 55 lines
Columns 10
Rows 5

○ General
○ Row
● Grid

● Absolu
○ Relativ



Line	Row	Col	Waist	Shoulder	Elbow	Hand	Wrist
1*	1	1	0	0	0	0	0
2	1	2	0	0	0	0	0
3	1	3	0	0	0	0	0
4	1	4	0	0	0	0	0
5	1	5	0	0	0	0	0
6	1	6	0	0	0	0	0
7	1	7	0	0	0	0	0
8	1	8	0	0	0	0	0
9	1	9	0	0	0	0	0
10*	1	10	0	0	0	0	0
11	2	1	0	0	0	0	0
12	2	2	0	0	0	0	0

You will now see rows and columns numbered as well as line numbers. There is an asterisk by 3 lines which represent 3 corners of the matrix. JOG the robot to the first corner, escape, highlight line 1 and click **Set to Here**. Then JOG across to the next corners, 10 columns across, line 10 and learn that, then up 5 rows to the 3rd corner which is line 50 and learn that. Click **Interpolate** and all the intermediate positions are calculated (including the 4th corner). Finally use JOG or MOVETO command as in the description of the ROW creation above and click **Set Apprch**. This will create a 51st line with the R flag for a relative position. The system can add this to the other positions to effectively create another 50 positions equally displaced from the originals. The new commands are
n INTO to go to a position via the approach
UP to withdraw from it
n NEAR to go to only the approach position

Objects – 2

It is possible to record objects in route positions just as in PLACES. Use the form
PART ISAT LINE 5 e.g. example program to take a part from FEEDR and put it in line 5:
PART ISAT FEEDR
FEEDR GET TRAY 5 INTO PUT
Now when you type L. you will see the object listed next to line 5.
If you try to do it again you will get an error 21, `POSITION OCCUPIED`
You can take out an object from a line with the words:
5 INTO GET
If there is no object there you will get error 26, `NOTHING TO GET`

You can gradually fill up an array by repeating the line in a loop as follows:
First define a word that just

```
: PUT-IN INTO PUT ;
```

now you can use that word to put any held object into any route/matrix.

```
FEEDR GET 1 PUT-IN
```

gets an object (PART) from the place FEEDR and puts it in line 1 of the selected route (TRAY)

```
VIEW LINE 1
Line       X       Y       Z    Pitch      W
01       0.0   300.0     0.0      0.0    0.0 PART
```

You could automate that for all the lines in a route.
Note that MOVES is a pseudo-variable (acts like a variable) containing the number of lines in the selected route. It is in high (extended) memory so must be fetched with E@ not @. It contains the number of lines *including* the approach line i.e. one too many which is exactly what we need for a DO LOOP. Remember I is the index of the loop.

```
: FILL-TRAY
TRAY
MOVES E@ 1 DO
  PART ISAT FEEDR
  FEEDR GET
  I PUT-IN
LOOP
;
```

Now when the robot has finished you can write L. to see a PART in every position, for example:

```
Line       X       Y       Z    Pitch      W
01       0.0   300.0     0.0      0.0    0.0 PART
02       0.0   350.0     0.0      0.0    0.0 PART
03       0.0   400.0     0.0      0.0    0.0 PART
04       0.0   350.0     0.0      0.0    0.0 PART
```

etc.
To clear the route of objects to start again (say with a fresh tray) use

```
TRAY EMPTY
```

The above process is often called palletizing. The reverse is called de-palletizing i.e. to take a part from the tray one position at a time and put them somewhere else, for example to get from the tray and put every part into the bin…

```
: REM-TRAY
TRAY                    select tray
MOVES E@ 1 DO           from 1 to the size of the route
  I LINE GET            go to line I (starting at 1) and get what's there
  BIN PUT               go to BIN and put it there
  0 ISAT BIN            tell BIN it is empty
LOOP                    do it again for the next line
;
```

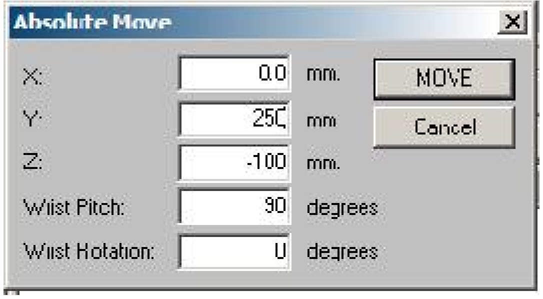
How to tell the system that the tray is full of parts for a new batch:

```
PART TRAY FILLUP
```

Walk-through to create a 2-D matrix

1. Restore the robot to a suitable starting position for the demonstration as in figs.1 and 2

fig. 1



Absolute Move

X: 0.0 mm.

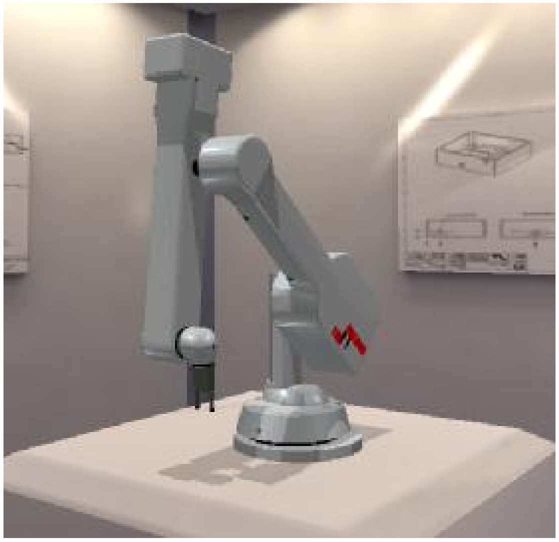
Y: 250 mm

Z: -100 mm.

Wrist Pitch: 90 degrees

Wrist Rotation: 0 degrees

fig. 2



2. Click the J (jog) button and use the teach pad in Z axis only to move the gripper down to bench level.

3. Enter ALIGN

4. Use Jog to move as in fig. 6.

5. Mark out or visualize a 3 x 2 matrix as in fig. 10

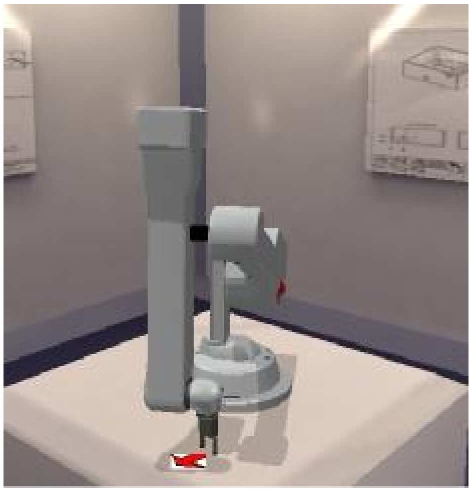


fig.6

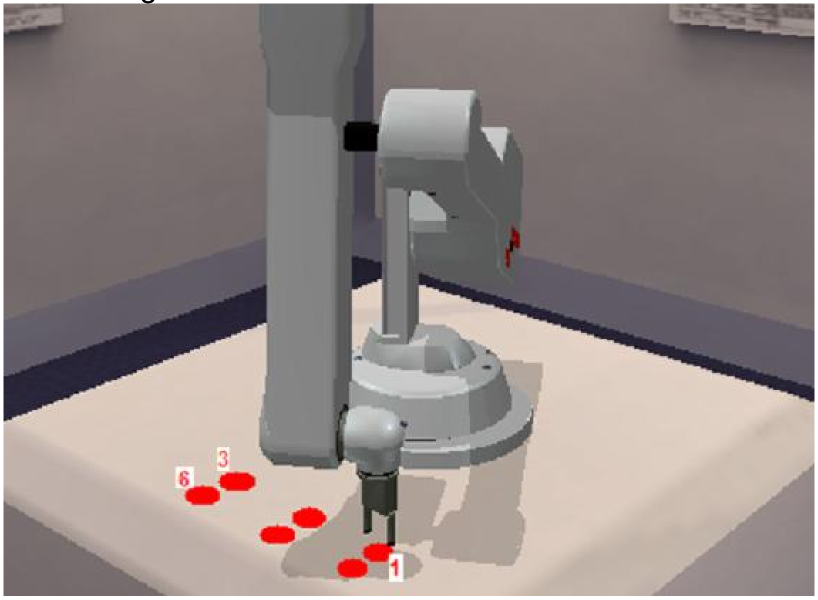
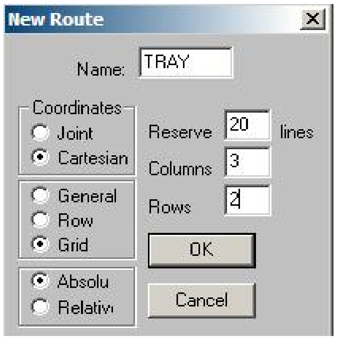


fig. 10

6. Find the routes box (assuming you have the project open) and click **New**. Create a new route called, say TRAY as a Cartesian Grid, 3 columns and 2 rows as in fig.11. Click OK and you should see a further box as in fig. 12



New Route

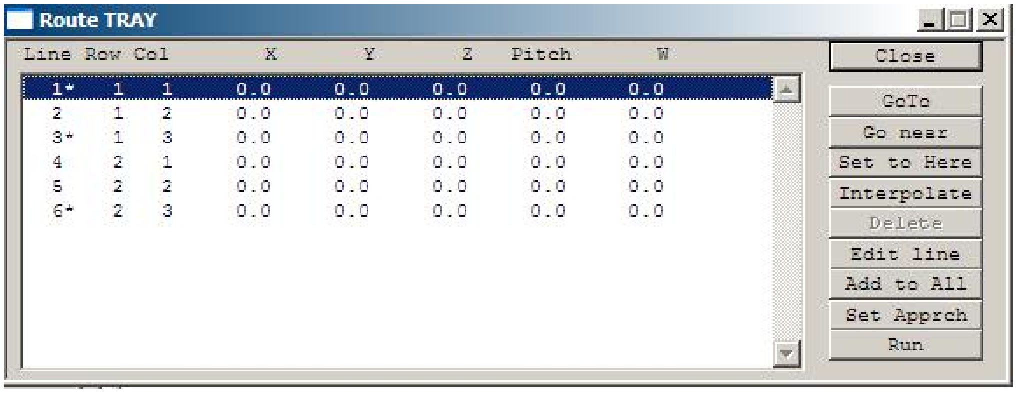
Name: TRAY

Coordinates
○ Joint
● Cartesian

○ General
○ Row
● Grid

● Absolu
○ Relativ

Reserve 20 lines
Columns 3
Rows 2



Route TRAY

Line	Row	Col	X	Y	Z	Pitch	W
1*	1	1	0.0	0.0	0.0	0.0	0.0
2	1	2	0.0	0.0	0.0	0.0	0.0
3*	1	3	0.0	0.0	0.0	0.0	0.0
4	2	1	0.0	0.0	0.0	0.0	0.0
5	2	2	0.0	0.0	0.0	0.0	0.0
6*	2	3	0.0	0.0	0.0	0.0	0.0

fig.11 fig.12

7. Assuming the robot is in the correct position (if not use Jog to adjust it) and with line 1 highlighted click **set to here**

8. Move the robot to the next position as indicated on fig 10, position 3, using MOVETO commands or Jog. If using Jog press esc to exit the teachpad. Highlight line 3 and click **set to here**.

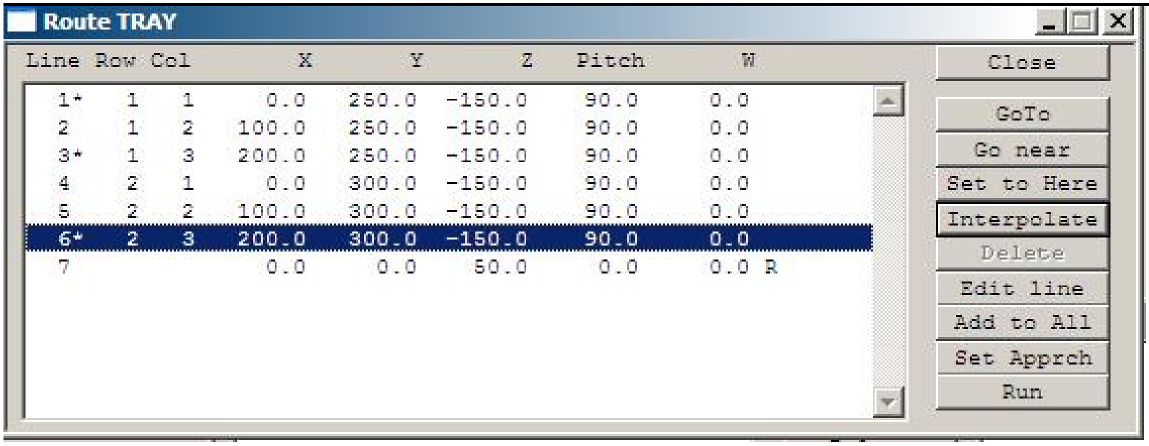
9. Move the robot to position 6 as indicated in fig. 10. Highlight line 6 and click**set to here.**

10. Click **interpolate** and the remaining positions will be calculated.

11. You can optionally set an approach position that is common to all the other positions. Move the robot vertically using Jog or for example using

```
0 0 500 MOVE
```

12. Click **set approach**. You will now see an additional line as in fig.13, line 7 which shows the 50.0 move in Z and is marked with an R which means this is a relative position. All the others are absolute positions.



Line	Row	Col	X	Y	Z	Pitch	W
1*	1	1	0.0	250.0	-150.0	90.0	0.0
2	1	2	100.0	250.0	-150.0	90.0	0.0
3*	1	3	200.0	250.0	-150.0	90.0	0.0
4	2	1	0.0	300.0	-150.0	90.0	0.0
5	2	2	100.0	300.0	-150.0	90.0	0.0
6*	2	3	200.0	300.0	-150.0	90.0	0.0
7			0.0	0.0	50.0	0.0	0.0 R

fig.13

13. A relative position may be added to an absolute position to create another absolute position. In this case the controller can add line 7 to any of the other 6 positions to achieve 6 new positions 50mm above the originals. You can go to any of these new positions with n NEAR where n is the line number or position number. Let's try line 1:

1 NEAR

The robot ends up 50mm higher than position 1.

1 INTO

The robot moves via the approach (NEAR) position of line 1 then into the target position, line 1 itself.

UP

The robot moves up from the target position to the approach position.

1 GOTO

The robot goes directly to line 1 but it could be unsafe to do this. To go from position 1 to position 2 you should write UP first.

A typical routine to access all positions in a matrix:

The robot gets a tube from the tray, takes it to a filling station and replaces it in the tray:

```
: FILLTRAY
TRAY
ALIGN
7 1 DO
    I INTO     ( go to each tray position in turn
    GRIP       ( and grip the tube
    UP         ( remove the tube
    FILLSTN    ( go to where a dispenser is located )
    FILLUPTUBE ( hypothetical command to the dispenser )
    WITHDRAW   ( away from the dispenser
    I INTO     ( back to the tray, same position
    UNGRIP     ( put the tube back in the tray
    UP
LOOP
;
```

Overview of encoder system, hardware/software.

As the robot moves its motion is tracked by incremental optical encoders. The encoder counts are multiplied by constants, a different constant for each encoder. When you type WHERE you see 3 lines. The top line is the motor count. The second line is the counts received back from the encoders and the bottom line is where the encoders think the robot ought to be by multiplying the encoder counts by the constants. On an R17 the encoders are very close to the motors and should match the motor counts within 1 or 2. On an R12 the encoders are down the gear trains and there will be much larger differences between the top and bottom lines.
If the difference between top and bottom exceeds a tolerance then the robot stops with the error ENCODER-STEPPER MIS-MATCH, AXES followed by which axis or axes gave an error. Type WHERE and you will see how much error there is.
Situations where you would see such an error include:
A collision.
Too much payload.
Too high a speed set.
Mechanical problem.
The primary function of the encoders is as watchdogs. However if an R17 robot stalls for a benign reason (not a mechanical problem) then you can read back from the encoders and continue. The command for that is
ENCASSUME
This is not recommended because there is always some difference of opinion between the motors and encoders so ENCASSUME might introduce an error. Generally the motors will always be right and the encoders track with some error. This error is much larger with an R12. So if you use ENCASSUME you will see the error at the next CALIBRATE.

==advanced==
A bit-pattern EFP shows which encoders are fitted (Encoder Fitted Pattern)
(version 11 constant) EFP **.** `31  OK` (version 12 variable) EFP ? `31  OK`
Means the robot has encoders fitted to axes 1,2,3,4,5 (1+2+4+8+16)

You can disable all the encoders with ENCOFF and re-enable with ENCON

You can functionally check the encoders: first DE-ENERGIZE the robot then enter ENCTEST then move the robot by hand. Press escape to exit ENCTEST.

If a single encoder fails it is possible to continue using the robot by disabling that encoder, for example if waist encoder has failed then the EFP you want is 2+4+8+16 = 30 or if the shoulder encoder failes you want 1+4+8+16 = 29.
So for R12
29 EFP !
for R17
29 ' EFP !

Lead-by-the-nose is a technique whereby you move the robot by hand and learn its position. First create a route to take the values. De-energize the robot with
DE-ENERGIZE. Then move the robot by hand and when you are in a desired position type ENCLEARN
OR – use the teach pad. When you are in a desired position press the tick key.
However this method is not recommended for two reasons – one is that the encoders are not accurate enough and the second reason is that it is very difficult to position the robot by hand with any accuracy. Using the teach pad in the normal way is much better and in JOG mode you can move at 0.1mm increments.

Vectored execution

Earlier an explanation of the CFA was given. An example was given:

```
FIND HI EXECUTE
```

Finds the CFA of HI and executes that, i.e. exactly the same as just typing HI
But you could also create a variable, say HIVEC

```
VARIABLE HIVEC
FIND HI HIVEC !
```

Then later use

`HIVEC @ EXECUTE` to execute HI

One example of how this us useful is the FN key on the teach pad. If you do

```
FIND HI FN !
```

Then when you use the teach pad pressing the FN key executes HI
(or any other definition you wish to make the FN key execute).
You can also re-program errors, for example normally pressing the stop button just stops the robot with error 3. But you may need a more complex action to take place. Simply define that action as a word and put the CFA of the word in STOPVEC. For example:

```
: STOP-ACTION
CR ." Stop button has been pressed”
CR ." Press space bar to abort or any other key to continue”
KEY ASPACE = IF
   STOPABORT
THEN
;
```

You can then write

```
FIND STOP-ACTION STOPVEC !
```

Or in more simply

```
SET STOPVEC STOP-ACTION
```

You can put this line in your initialization. A typical initialization would look like this:

```
: INITIALIZE
START
CALIBRATE
SET STOPVEC STOP-ACTION
;
```

Other vectors you can change are:
ENCVEC – for alternative action in case of an encoder error,
DSPVEC – for action to take place during RUN
INTVEC – for action to take place if there is an interrupt.

Input-output

All inputs and outputs are arranged as 8-bit ports. The standard ports are PA PB and PC and the expansion ports are QA QB QC, RA RB RC etc.

PA is an output port, PB is an input port and PC is 4 bits in and 4 bits out.

Outputs

To send a value out of port PA

(value) PA out.

To manipulate an individual bit you can use

PA n ON and PA n OFF where n is a bit number from 0 to 7, for example

PA 0 ON will turn on the gripper. PA 0 OFF will turn it off.

Thinking Forth-wise and modular suppose you have a pump controlled by port PA bit 5, you could define

```
: PUMP PA 5 ;
```

Thereafter you can use

PUMP ON and PUMP OFF

Inputs

You can read the value from any port with IN e.g. PB IN – this leaves the value of that port on the stack.

```
PB IN . 0 OK
```

You can test an individual bit with

PB 5 BIT?

This will leave a true if the input is high and false i.e. zero if the input is low. A true is ANY non-zero value. So for a high input on bit 5:

```
PB 5 BIT? . 32 OK
```

You can hold up a program for an input to change state with WAIT

```
PB 5 1 WAIT
```

waits for bit 5 of port PB to change to a 1.

WAIT has a stop button check in it so if the input is faulty you can still get control by pressing the stop button.

Analog

When fitted there are 4 input channels to the ADC 0-3 and 2 DAC output channels A and B.

To read an input

0 ADC reads channel 0 and leaves the value on the stack.

0 DACA sends 0 volts out of DAC channel A

Second serial port

The second serial port is invoked with IOFLAG C1SET and switched back with IOFLAG C0SET. With the IOFLAG set to 1 all control is transferred to the second port so any command you wish to type you need to type it into the second serial port. Therefore it is sensible to define words that use the second port to start with IOFLAG C1SET and end with IOFLAG C0SET, thus returning control to port 0 after use.

See the controller manual for more details including setting Baud rate and changing port 0 Baud rate.

Control words –2

The BEGIN – UNTIL loop is useful for testing. A RoboForth word ?TERMINAL leaves a true if the escape key is pressed.

```
: TEST
BEGIN
  TELL SHOULDER 1000 MOVE
  0 MOVETO
?TERMINAL UNTIL
;
```

TEST will continually rock the shoulder back and forth 1000 steps until you press escape. The robot will not stop immediately, only after the 0 MOVETO command when the escape key is then checked.

You might want such a loop to be dependent on an input, say PB 7 changing from low to high:

```
: TEST
BEGIN
  TELL SHOULDER 1000 MOVE
  0 MOVETO
PB 7 BIT? UNTIL
;
```

Example of a word that performs two actions depending on the value of an input (say PB 6)

```
: TEST
BEGIN                    begin a loop
  TELL SHOULDER          select shoulder
  PB 6 BIT? IF           is PB 6 high?
    1000 MOVE            move 1000 steps
  ELSE                   else it is not high
    2000 MOVE            so move 2000 steps instead
  THEN                   end of IF-THEN loop
  0 MOVETO               finally go back to zero
?TERMINAL UNTIL          repeat from BEGIN until the escape key is pressed.
;
```

If PB 6 is high the shoulder moves 1000 steps and if it is low it moves 2000 steps.

For WHILE and REPEAT see the controller manual.

Counting loops are achieved with DO… LOOP.

```
: TEST2
TELL SHOULDER  select shoulder
10 0 DO        do it 10 times
 1000 MOVE
  0 MOVETO
LOOP
;
```

This will cycle the shoulder 1000 steps 10 times.
Note that the upper limit of the loop – 10 – is never reached. The loop runs from 0 to 9 i.e. 10 times and exits when the index reaches 10.
I -- this is the actual value of the index of the loop which increments each time round.

```
: TEST3
TELL SHOULDER  select shoulder
10 0 DO        do it 10 times
 I .           print the index
LOOP
;
```

This is the same but prints the values of the index each time around

```
TEST3 0 1 2 3 4 5 6 7 8 9 OK
```

If you want to count from 1 to 10 then it's

```
: TEST4
TELL SHOULDER  select shoulder
11 1 DO        do it 10 times
 I .           print the index
LOOP
;
```

You can leave a loop early using LEAVE e.g.

```
: TEST5
TELL SHOULDER  select shoulder
11 1 DO        do it 10 times
  I .          print the index
 1000 MOVE     move the selected joint 1000
  0 MOVETO     move back to zero
  PB 5 BIT? IF LEAVE THEN    if PB 5 goes high end the loop early
LOOP           do it again or leave if PB 5 was high.
;
```

This will cycle the shoulder 10 times but if you put the input PB 5 high before reaching 10 then the robot will finish its current cycle and stop.
For loops that increment more than 1 or negative see the controller manual.

Outer Interpreter

When you are typing commands into the controller from the communications window your commands are being collected and handled by the outer interpreter which itself is a word. The word is OUTER. It is possible to invoke this word again in a program so you can effectively get more commands in the middle of another command in progress.

Example:

```
: TEST6
TELL SHOULDER select shoulder
11 1 DO do it 10 times
   I . print the index
  1000 MOVE move the selected joint 1000
   0 MOVETO move back to zero
   KEY ASPACE = IF OUTER THEN if you press space run the outer interpreter
LOOP and again
;
```

What TEST6 does is to cycle the shoulder 10 times. If at any time you press the space bar the program will enter the outer interpreter just after the shoulder moves to zero. It will say OK just like it did the first time, but it is actually in the middle of TEST6 as the counts will show. Type EXIT to continue with TEST6

You can also get the stop button to invoke OUTER – see earlier section on vectored execution. Just enter
SET STOPVEC OUTER
Usefulness – when the stop button is pressed the system invokes OUTER. This gives you the opportunity to investigate why someone pressed the stop button and if you type EXIT the robot will continue.

⚠ Warning: the robot may not be able to continue from exactly the position it reached. If a route is being run in continuous path then the system skips the rest of that route and continues from there. This is because a route in continuous path is treated as a single move.

Interrupts

Interrupts work like vectored execution. When an interrupt occurs, if enabled then the word whose CFA is in INTVEC is executed.
Proceed as follows:
1. Define the word you want executed on the interrupt. Suppose the word is MEASURE. Test it before use.
2. Define a new word that has just MEASURE but ends in RETURN (return from interrupt) e.g.

```
: RMEASURE
MEASURE
RETURN
;
```

You can not test RMEASURE in the communications window. It is only suitable for use with interrupts. That is why you need two definitions, one to test and one to use.
3. Put the CFA in INTVEC:

```
SET INTVEC RMEASURE
```

Timer interrupt
Set the interval of interrupts e.g. for an interrupt every 100 mS use

```
100 INT-TIME !
```

Start the timer with

```
START-TIMER
```

and stop it with

```
STOP-TIMER
```

Example to beep the terminal every 500 mSecs even though something else is already running. First define RBEEP which is the same as BEEP but ends in RETURN.

```
: RBEEP
BEEP
RETURN
;
: TEST7
SET INTVEC RBEEP set RBEEP as the word that executes when there is an interrupt.
500 INT-TIME ! set the time to 500 mSecs
START-TIMER start the timer
BEGIN start a loop
  TELL SHOULDER 1000 MOVE move the shoulder back and forth
  0 MOVETO
?TERMINAL UNTIL until the escape key is pressed.
STOP-TIMER then stop the timer.
;
```

The shoulder will rock back and forth until you press escape. While it moves every ½ second the terminal will beep. Press escape to stop the motion and the interrupts.
Event interrupt
For an interrupt from an input a link must be put on the I/O card. See controller manual for details.

HELP! An error message!

While getting started

```
>START NOT DEFINED! ABORTED
```

Your keyswitch is in cold start position. If you already have a project in the controller simply turn the key to warm start and press reset. If you have not yet started programming then enter ROBOFORTH then turn the key to warm start. After you have programmed something enter USAVE to save your program to controller flash memory.
If your keyswitch is not in the cold start position then your program may be corrupted. In that case switch it to the cold start position, press reset then enter ROBOFORTH and turn back to warm start.

On loading a project or reloading the ed2 text using the green down-arrow:

```
>|  >>>>>>>XXXX NOT DEFINED!   ABORTED
```

and a dialog box that says **>expected** with an **OK** button.
(where XXXX is just an example)
Click the OK box and you see

```
>IOFLAG C0SET
>
```

This is because a word you have used in your text file was not defined already. Perhaps it is mis-spelled. Try to find the word in your text file and correct it.

If the word that is undefined is a word after a colon, for example

```
NEWPART UNDEFINED!  ABORT
```

then the most likely reason is a missing semi-colon from your previous definition. The system is still in compile mode when it got to your new word. For example:
: SETUP
words
words
missing semi-colon
: NEWPART
etc

```
>| >>>>>>>> ABORTED
>
```

but no reason given. also a dialog box that says **>expected** with an **OK** button.
The most probable reason is a stack underflow resulting from an improper use of loop words, or a missing loop word, for example UNTIL with no BEGIN.
You can see better where the error occurs like this: go to settings, configuration and un-check hide mode. Now load the file again. If the error comes right after a word like UNTIL or LOOP or THEN then that is the reason – missing BEGIN or missing DO or missing IF.

While using the robot

Using RUN or any word that contains RUN you get
ILLEGAL! ABORTED
or
STACK UNDERFLOW! ABORTED
You must select a route first. Either type its name now or make sure the name of the route is in your code.

Too tight line x ABORTED

See the RoboForth manual, section 7.2.6. If two lines are right close to each other in coordinate values the DSP can not change the direction in the time allowed at the SPEED and ACCEL values given. Simply reduce SPEED or increase ACCEL and try again. There is a word ADJUST which reduces speed automatically to a value that works.

If you have a SPEED value embedded in the route as a function then ADJUST will not work. It will hang and you need to press escape to get out. Your only recourse is to edit the speed values in the route or increase the value of ACCEL.

Another recourse is to list the route and see if two lines are very close to each other in values and maybe edit those two positions further apart.

You might possibly by mistake have two lines next to each other one being a duplicate of the other. This can never work. Delete one of them.

STOP BUTTON PRESSED but it wasn't. Check the jack in the rear of the controller.

ENCODER-STEPPER MISMATCH AXES n n n

Where n n n is a list of the axes which failed.
If all the axes have failed then maybe you didn't type START at the beginning, or the sensor cable is not properly plugged into the robot.

One or more axes could be a result of running too fast – reduce the value of SPEED and ACCEL. If you lose track of what SPEED or ACCEL you have then try NORMAL or start afresh with START CALIBRATE
Or it could be the tolerances are too tight – type WHERE and compare the top and bottom lines. The top line is the target position and the bottom line is where the encoders think the robot is. If there is a difference of 5 or more for an R17 or more than 50 for an R12 it could be all you need is to increase the tolerances. See Help Sheet 12.

Run file too short message from RobWin when loading a project. This means the project is corrupted but it may be possible to recover the project from the controller. See instructions on Help Sheet 12.