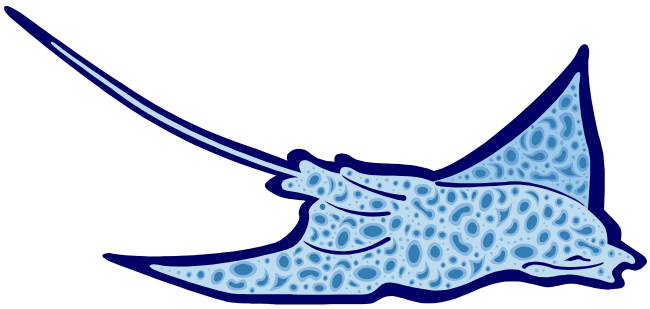
ftComputing

ROBO Pro : Tutorial

Ulrich Müller

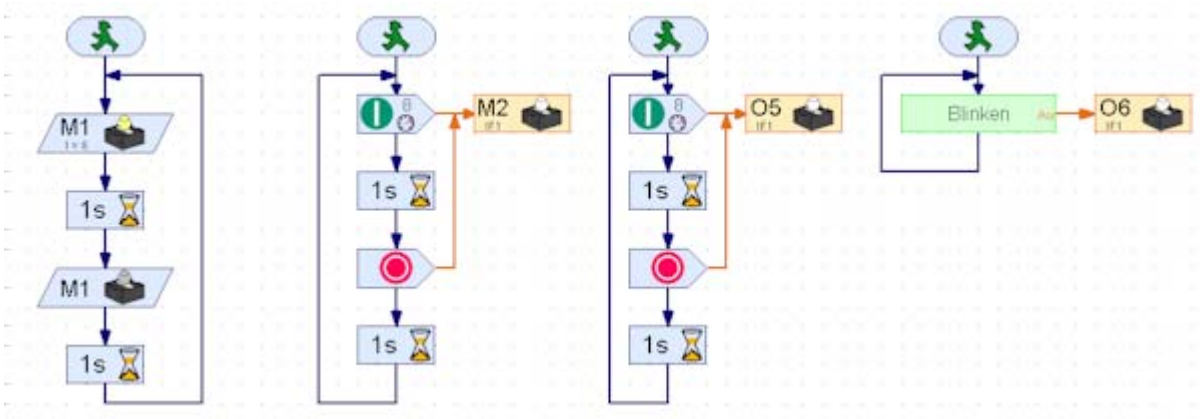


Inhaltsverzeichnis

Tips & Tricks

Techniques

Blinker : Loops / Subs / Outputs



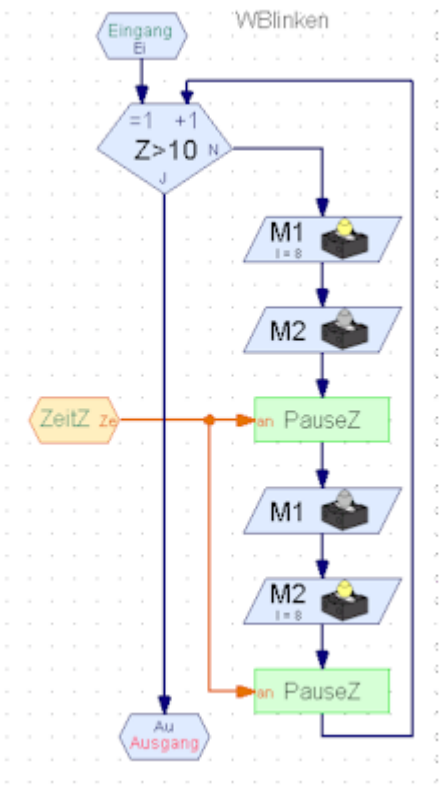
Connected are one lamp to a M-Output or O-Output and ground respectively.

The left diagram level 1 is used, the others use level 3. With level 1 only the basic element can be used, with level 2 in addition subprograms without parameters and with level 3 you can use variable and the yellow data flow lines and many other elements. After a short phase of learning you will mostly use level 3. We use level 3 in general, but it is possible to use the basic elements in any mix, if convenient.

The programs above are variations of the same thing : blinking with one lamp.

1. Lamp on M1, 1 sec on, 1 sec off, running in an endless loop. Stopped by the appropriate button of the IDE.
2. Lamp on M2, like (1.), but with elements of level 3 with special On an Off commands, which send via data flow (yellow lines) messages to the motor output. Each special motor output symbol should be used only once in a main or sub. Multiple use make sense, if the wiring would be to complicated.
3. Lamp on O5 and ground, like (2.)
4. Lamp on O6, functional like (3.), but blinking is done in a sub. The Output to be used is transfered as parameter.

WechselBlinker : Loops / Subs 2

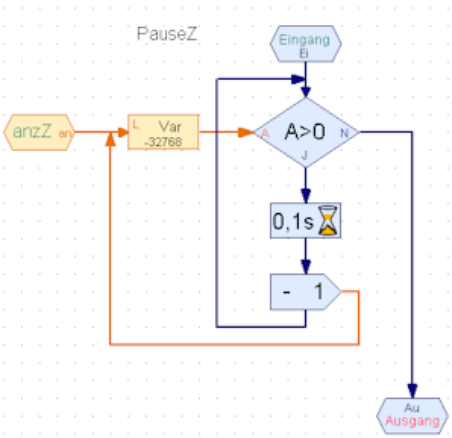


If blinking is not the only purpose of a program, to place blinking, loop included, in a subroutine.

In WBlinken (WechselBlinker : Alternative Blinking) two lamps (M1 / M2) are controlled with level 1 commands.

The blinking loop is controlled by a level 1 counter loop command. It runs 10 times. Therefore the loop counter is set to 1 entering the left pin. On each further entry on the right pin it is incremented by 1 and compared to the max value of 10.

The time delay command has a fix delay value, the delay function was outplaced in a sub PauseZ with a min delay of 0.1 secs. The parameter n gives WBlinken the number of repeats of that delays. WBlinken itself routes it to PauseZ.

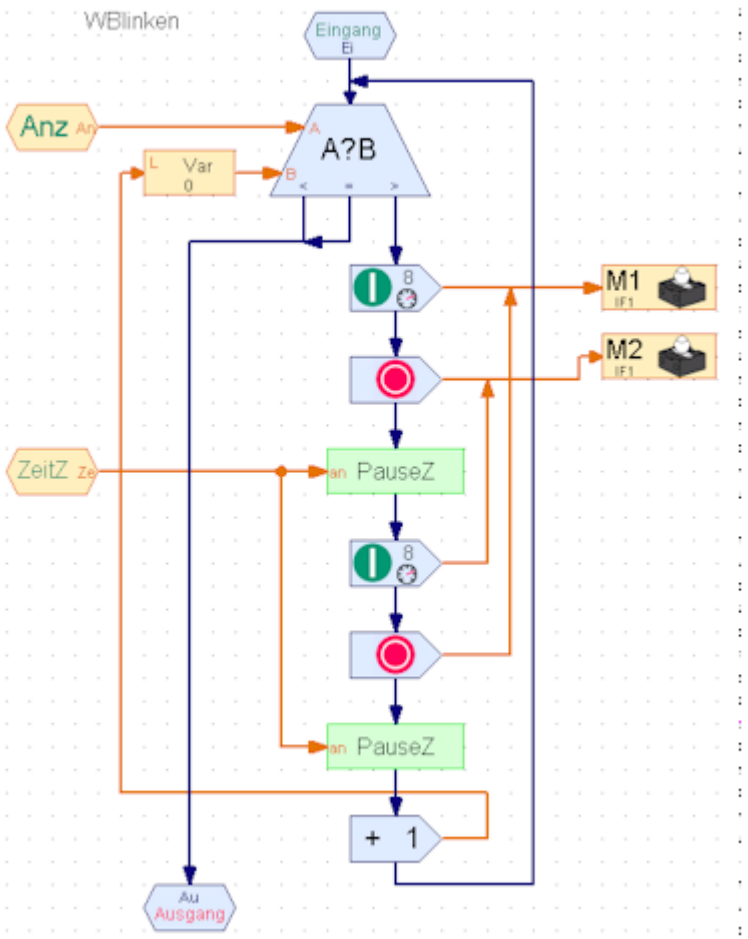


PauseZ (variable time delay) operates with a variable number of loops. The number of loops is given by a parameter stored in a local variable. Before executing the delay it is compared to 0. After the delay the local variable (Var) is decremented by 1.

The local variable is created when entering the sub and destroyed on leaving the sub. Var can't be accessed from outside.

The main program only must call the sub with an adequate blinking frequence. That is done with the constant 2.

WechselBlinker : Loops / Subs 3

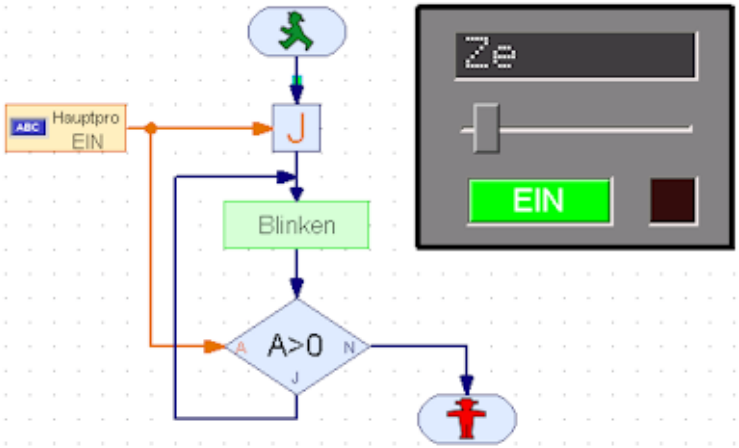


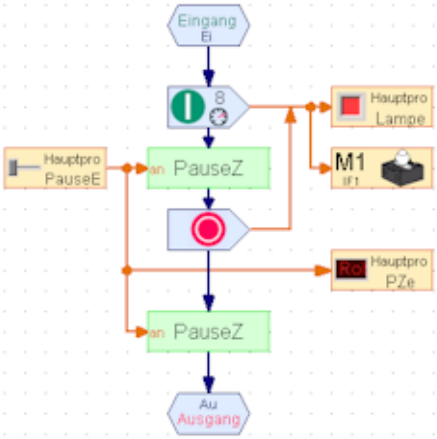
WechselBlinker has become an additional parameter Anz for the number of blinking loops (instead a fix 10). The command A?B compares parameter Anz with the actual value of the local Var (loop counter). Var is set to 0, if entering the sub. A?B has three exit pins. Two of them are connected to leave the sub on Anz<=Var.

This compare could be done too with an A?0 compare (look PauseZ). in that case Var must be decremented.

This sample uses level 3 commands for controlling the lamps.

Bedienung : Panel elements





Panel elements combined for display a value and control a simple blinker. They are situated on a grey form (menu painting). The square therefore must be place in background (menu painting | object to background). The panel element are placed for a better view in the function area of the main. There is a special tab for panel elements too.

The panel elements are suplemented by special panel inputs and panel outputs. In this case there an EIN input corresponding to the EIN button, PauseE for the scrollbar for controlling

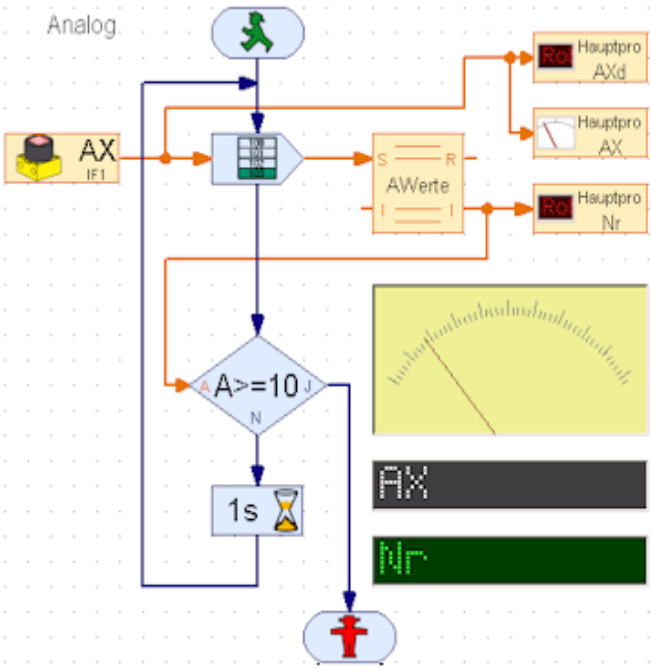
the blinking frequence, Lampe for displaying the lamp state and PZe for displaying the actual pause time.

You can see : inputs and outputs can be placed anywhere in the whole program and displayed on a central place.

Because of the handling of the buttons is a little bit crucious (last run state is stored), it is controlled towtimes in the main, once for starting the blinking (J command) and once for ending (A>0, pushbutton).

In sub Blinken the scrollbar input (PauseE) is used with 3 yellow lines for PauseZ parameters and for displaying purposes.

Analog : Storing to a List, Analog Display

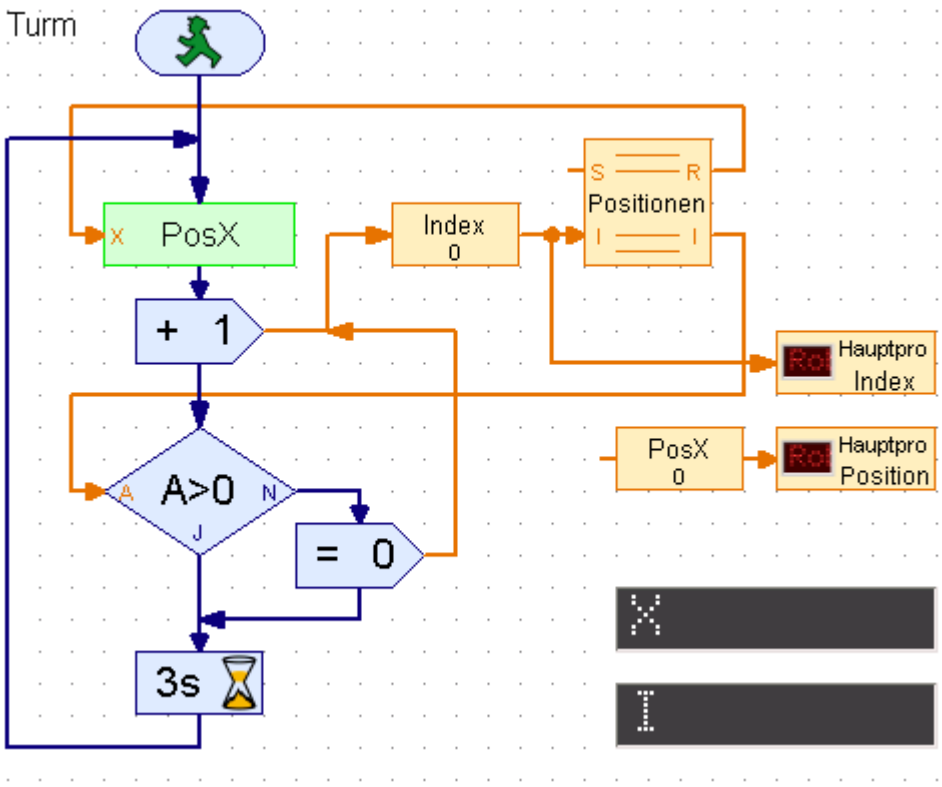


Analog scans in second intervals the actual value of the AX-Input with is connected to a photo resistor. It is displayed on the analog control and digital too. The actual list size is displayed also.

Storing to list AWerte is done by the special command append value via the S-Entry of AWerte. The analog values are send from the AX-Input to the pin of append value (right click to get him). At beginning of the program the list is empty an can contain max 100 values (default). It is possible to store start values in the list (right click : table). The max size can be changed also.

Every changing of the list via S-Entry, the I-Exit contains the new number of list elements (on program start too). In cause of exceeding the max number of 100 elements, the I-Exit contains the value 0. In the sample excat 10 value are stored, numbered from 0 to 9. To show them 1 sec delay.

Turm : Access to a List



Turm moves the pile of an Industry Robots (Rob 2, 3, 4) or Computing Starter (Welding Robot) in an endless loop beginning with position 0 to 3. Therefore the list Positionen is filled with coordinate values (right click, table : 0, 45, 150, 30). The movement of the pile is done with the library routines PosX / Pos.

The variable Index the actual position value is choosen and transfered via R-Exit of the list as a parameter to PosX. The R-Exit offers always a new value in case of a change of the I-Input (done by variable Index). The same happens on program start. In that case the value for index 0 is available, even if there is no Index connected. Therefore the incrementing is done after the first position processing.

PosX has a special procedure for position 0 : moving strictly to the end switch without counting and than set variable PosX to 0. The usual call Home can be dropped in this case.

After incrementing Index it is compared for Index>0 (A>0) to decide for remaining list values. If not 0 is retrned. Caution the R-Exit than conains the value –32767.

For an endless processing Index is than set to 0 (on program start it is done automaticaly : local variable) and the loop continous with a pause of 3 seconds for having a right view to the spectacle.

Notice : Storing and accessing should be done in separate subs, because of possible complications on use of the list Entries and Exits.

Industry Robots

General

Usual Assigning of Motors / M-Outputs and Switches / I-Inputs

X-Axis (Pile) : M1, I1 (end switch), I2 (impulse switch)
Y-Axis (Arm horizontal) : M2, I3, I4
Z-Axis (Arm vertical) : M3, I5, I6
4-Axis (Grip) : M4, I7 (end switch : grip open), I8 (impulse switch)

The impulse switch is a special (fourTooth) wheel, situated on a motor axis. It controls a switch connected to an I-Input. The 0/1 and 1/0 changing of ist state is counted and used for determining the position of the operateted component.

Working Area

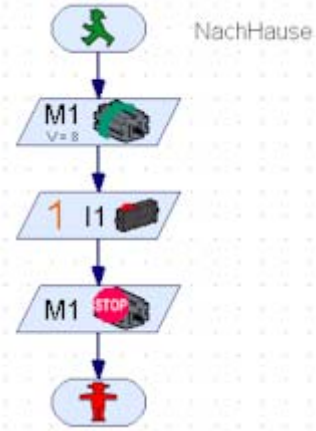
The working area of an Industry Robot is counted in number of impulses up from the end switch. It has values up from 90 to 230, depending on the component and the cable routing.

The get a coordinate 0, the robot must move to the end switch, at min, if starting the program. That is done with **left revolving** (test panel : click to left button). If he doesn't do it, change the wiring of the appropriate motor.

On reaching of an end switch, the corresponding (global) position variables should be set to 0. The reversed procedure seems me to be better solution (set to max working area an decrementing) but it is not usual. In this document home has the value 0.

The Tower only

NachHause : Moving until end switch bump



The robots are using impulse wheel for controlling their actual position. The impulse wheels are mounted on an axis of the motor gear and are operated by a switch. The impulse wheel has 4 cams – hights and downs – makes eight changes between on and off. The actual position ist counted as impulsed from up the end switch (position 0).

The tower motor is on M1 with end switch on I1 and impulse switch on I2

To determine home position (position 0), the tower must run to 0 : the end switch. That is done with the small program above :

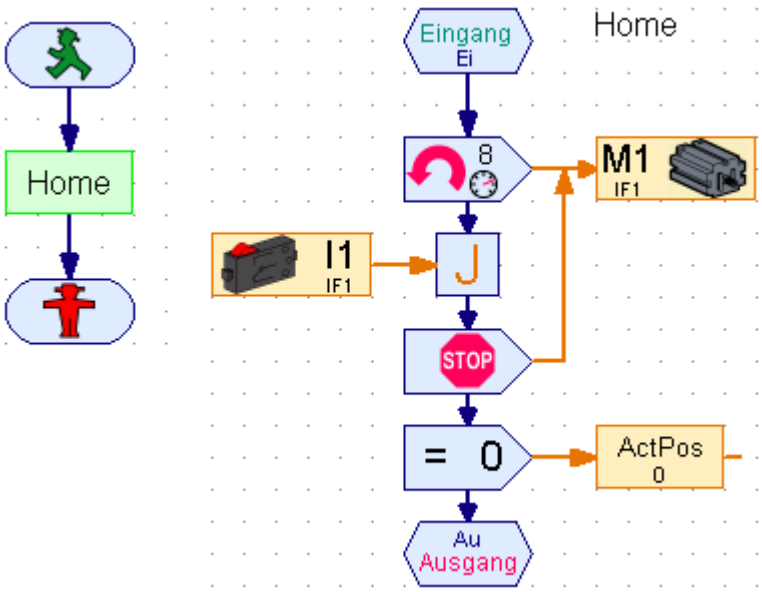
Switch on motor M1.

Pulse counter in mode wait for 1 : waiting for I1 to be closed

Switch off motor M1.

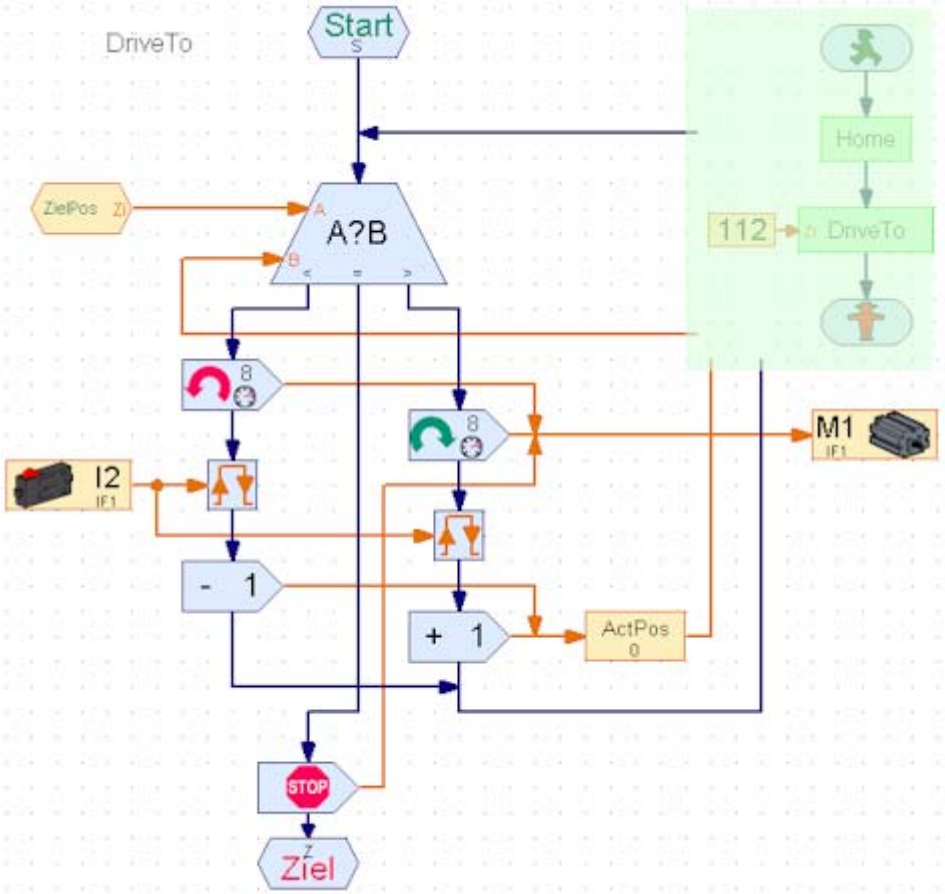
For a good program a global variable Pos must be set to = 0

Home : Go Home using a Sub



The function 'Drive to Home' now is placed in a sub using level 3 commands. A global variable ActPos is added. With larger programs the program gets more clear. Alternatively your can use the library routine PosX.

DriveTo : A special Position

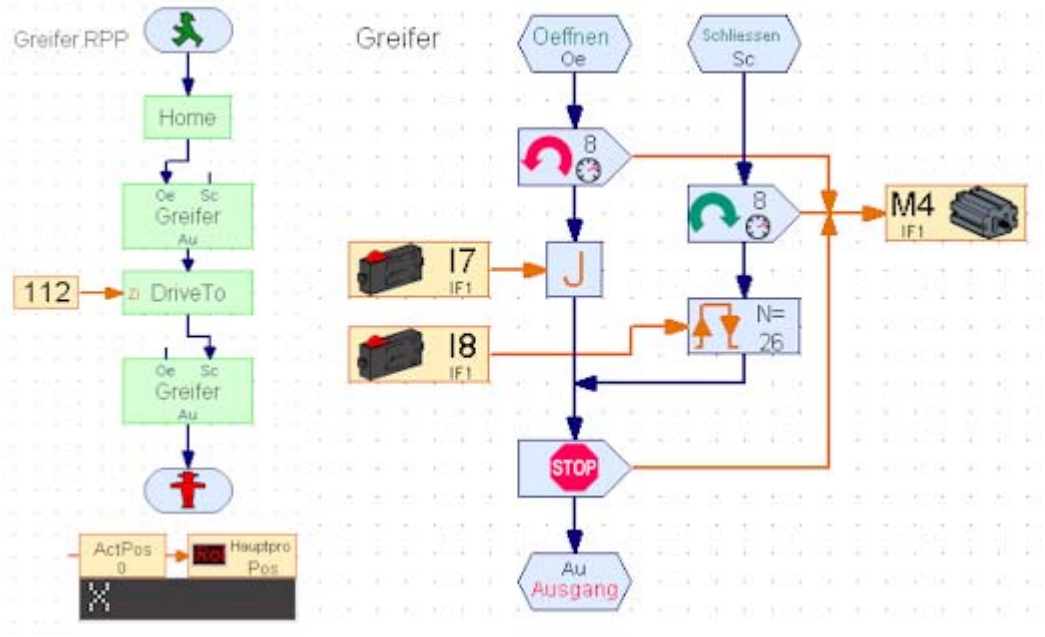


The position is noted by the parameter ZielPos (DestinationPosition). The actual position is stored in the global variable ActPos. ActPos must not to be seen in the main, but in Home. The most interesting command is A?B which compares ZielPos with ActPos. If equal, destination position is reached, motor off, return.

On ZielPos > ActPos the tower (M1) will turn right waiting for impulses of I2, each impulse is added to ActPos. With ZielPos < ActPos the tower is on left turn in direction of the end switch.

It makes sense to place a compare for end switch true or to use the library subs PosX / Pos.

Greifer : Sub with two Entries, Show Position



The grip differs from the other components in its operating, it is only opened and closed. But the angle to be closed is given with number of impulses. If transporting every time the same 'yellow barrell', a fix value can be choosen, may be 26.

Opening the grip is only done with a 'Wait for J' command.

In this case there are no parameters for opening and closing, it is done with the two different entry points to Greifer (grip). This is a more oldfashioned method.

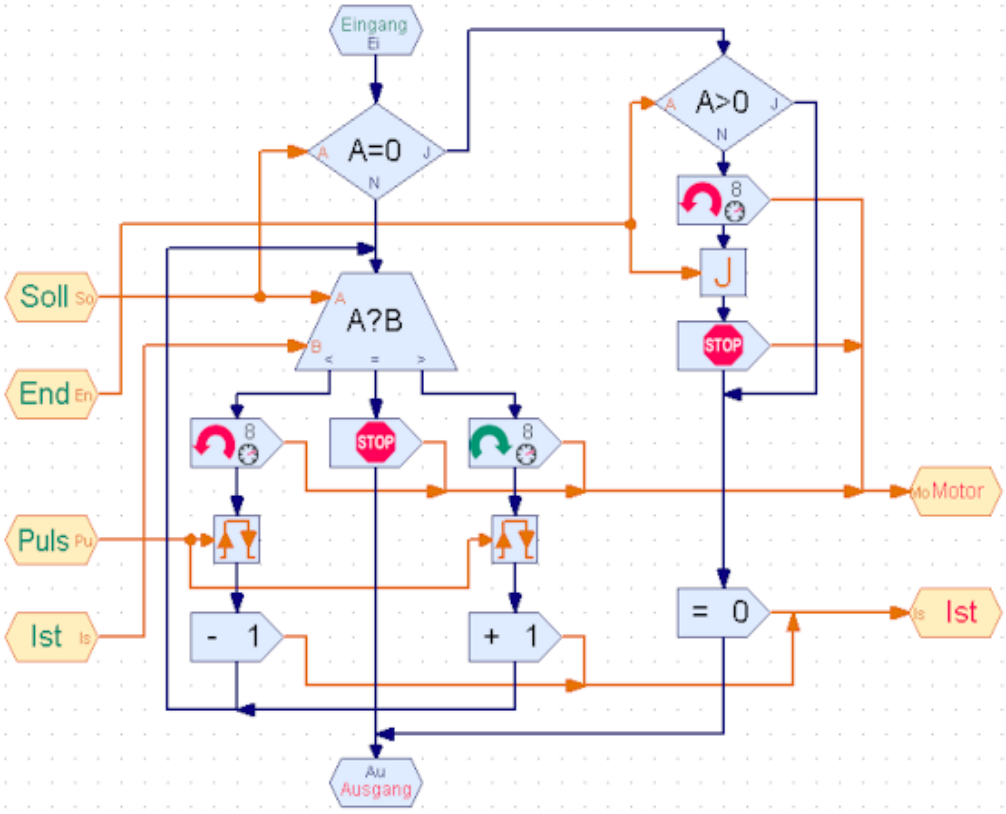
The main uses at the begin a separate Home followed by a Greifer. A Call Greifer can be placed in Home alternatively.

The actual value of the global variable ActPos (actual position of the tower) is displayed in main. A pair of additional Lamps for right / left turn would be nice. Can be done with the '=' command.

Some Notices about the Library Position ES

Position ES contains subs for moving to positions by counting impulses on the appropriate impulse switches. For fixing the home (0) position an end switch is used, if true it is decided to be home. The motor turn left (in term of ROBO Pro commands), if going in the direction of the end swtch. The addition ES in the library name means end switch for pointing to this feature.

Pos : Move to a single Position

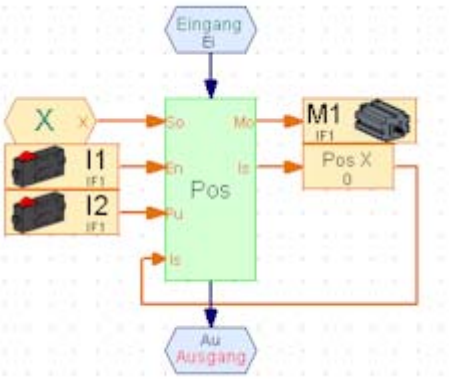


Parameter
- Soll : The destination position (variable or constant)
– Ist : Actual position, must be a global variable.
the same for in and out parameter, done to use Pos more flexible.
look to PosX.
- End : End switch
– Puls : Impulse switch
– Motor : Motor of that component.

On Soll = 0 it is a special case to rund to the end switch directly, without impulse counting (turning left), afterwards ActPos is set to 0.

With Soll > 0 there is a loop beginning with an A?B comparision. In case of Soll > Ist : right turn and incrementing, while Soll < Ist left turn and decrementing.

PosX : Move to a single Position with "X-Axis"

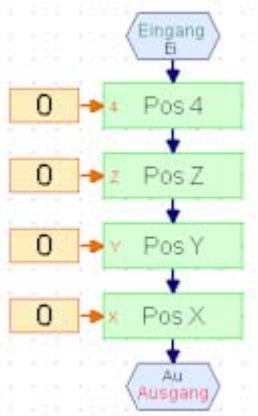


Work is done in Pos, PosX only organizes the actual parameters for the special case X-axis. The parameter X for destination position is passed through from outside.

The handling of the actual position is of some interest. It is stored in the global variable PosX. Each changing of parameter Ist causes a change of PosX, this new PosX is given back to Pos.

In this manner it is possible to use Pos for PosX, PosY, PosZ and Pos4.

PosInit : Go Home



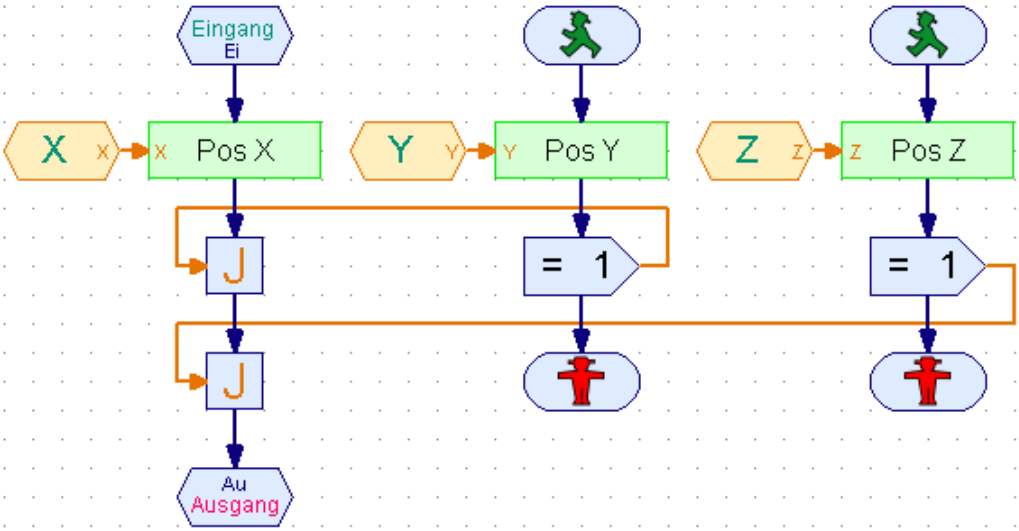
The home position of the complete robot is reached step by step with a call sub for each component. In this manner there is a good change to operate under narrow circumstances.

PosInit is simple, it is using the special function with position 0 of Pos for a direct move to the appropriate end switch.

Anzeige : Panel Elements

Gathers up all global variables (PosX, PosY, PosZ and Pos4) to be displayed and there there panel outputs. They displayed on the panel with panel elements.

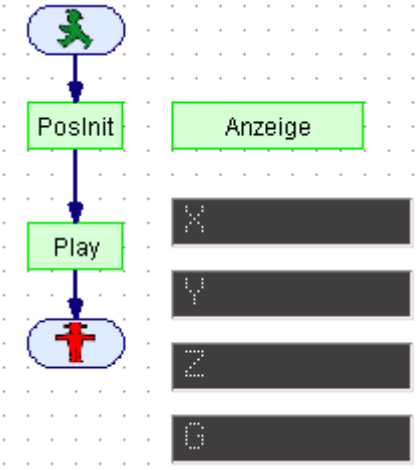
Pos XYZ : Simultaneous Move to Position X/Y/Z



Moving to positions for the X-, Y- and Z-axis is done simultaneous in three different prozesses : The main prozess and two new one, created and – later on detroyed – while the sub is running. Each process (thread) executes one sub for postioning a robot component. The main process – after doing ist positioning work – is waiting for the two other processes to get ready. This is done with an wait for J command.

Rob 3 / 4 : List operated

iRobListe : Main



Is a very simple one. It contains the position displays and the call PosInit (Home) and call Play.

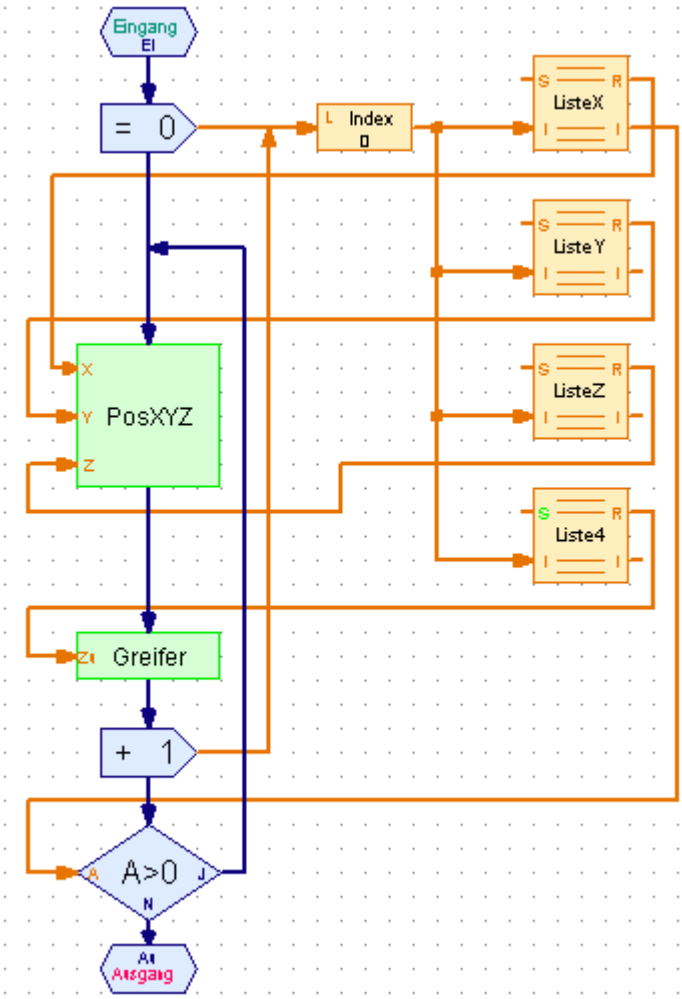
The modified library sub Anzeige contains the assigns from the global variables to panel outputs, the panel displayed are erased.

The globla variables PosX (pile), PosY (arm horizontal), PosZ (arm vertical) and Pos4 (grip), which are prepared in sub Anzeige for display.

Sub PosInit moves the whole robot to ist home position. That's the position the appripriate end switches are true. They will moved to left turning, first 4(grip), than Z, Y, X.

Play is that who does the most interesting work. The lists for the components are used to move the whole robot to new positions. It is done straight forward, if it is not enough, it can be done in an endless loop. With or without PosInit within (to have the exact home in the loop) the loop.

Play : Working with the Lists



Each robot component has its own list which contain the positions to be moved to. A unchanged position never the less must be a list element.

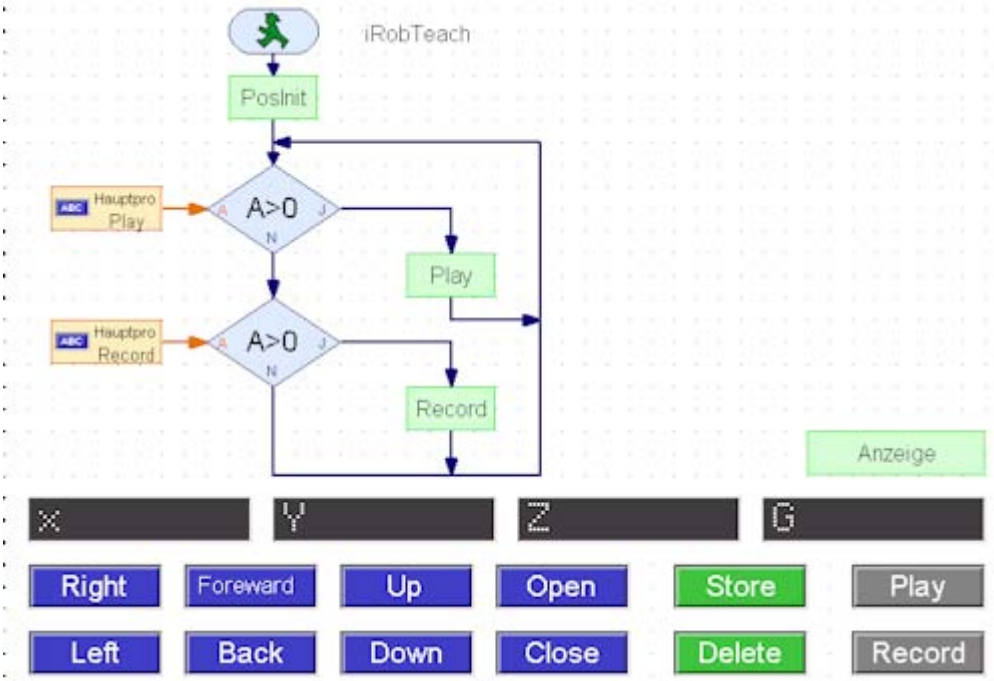
The local variable Index, set to 0 at sub start, points to the actual position within the single list. Each changing of Index positions the appropriate position value to the R-Exit of the list and than to the parameter entry of PosXYZ and Greifer. The new position is moved to simultanously, if done the grip is operated. Grip operating is done separat, because of it must open or close on an predicticable position. The grip gets the values 0 and 1 for open/close. Greifer internally uses the library sub Pos4, the Pos4 parameter than contains the real parameter value for closing.

After call Greifer done, Index will be incremented by 1. The I-Exit sends a 0, if Index is >= number of list elements (end of list), otherwise the number of list elements (the manual is not exact in describing the list variable).

Notice : The library subs use names which contain blanks. That doesn't work well for long time. In the program are no blanks in the names (or are that even two or nothing?).

TeachIn for Rob 3 / 4

iRobTeach : The Main



Diagramm

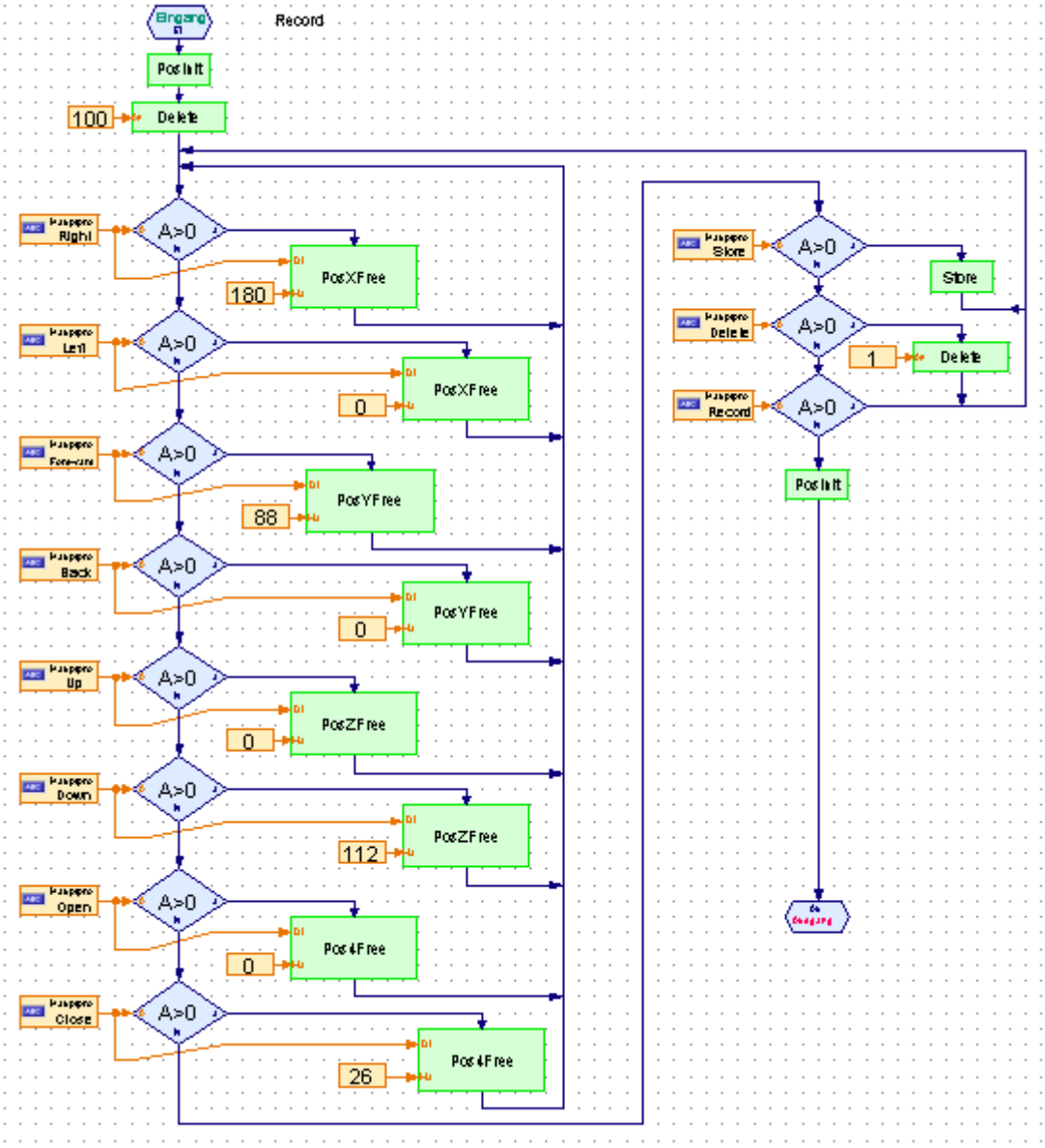
The robot moves to home position. Than, in an endless loop, the (gray) 'Play' and 'Record' buttons are scanned and, if true, the Play or Record sub called. The buttons are pushbuttons, they will stay on until they are pressed again. Using the (modified) library routine Anzeige (Display) the actual position is displayed vis PosX ---. The robot positions are stored in global lists ListeX ... At start they contain a predefined position list – ready to play --.

Some subs from the library Position ES are used in this program too (names without blanks) and additional some iRobListe subs.

Panel

The panel elements are placed dirrectly on the main form, it is something more commode. The upper row displays the actual position, down under them left a block with the TeachIn buttons, they only work as long as they are pressed. Followed by buttons for store actual and delete the last position. Right buttons for PayBack and Record.

Record : Storing the different Robot Positions



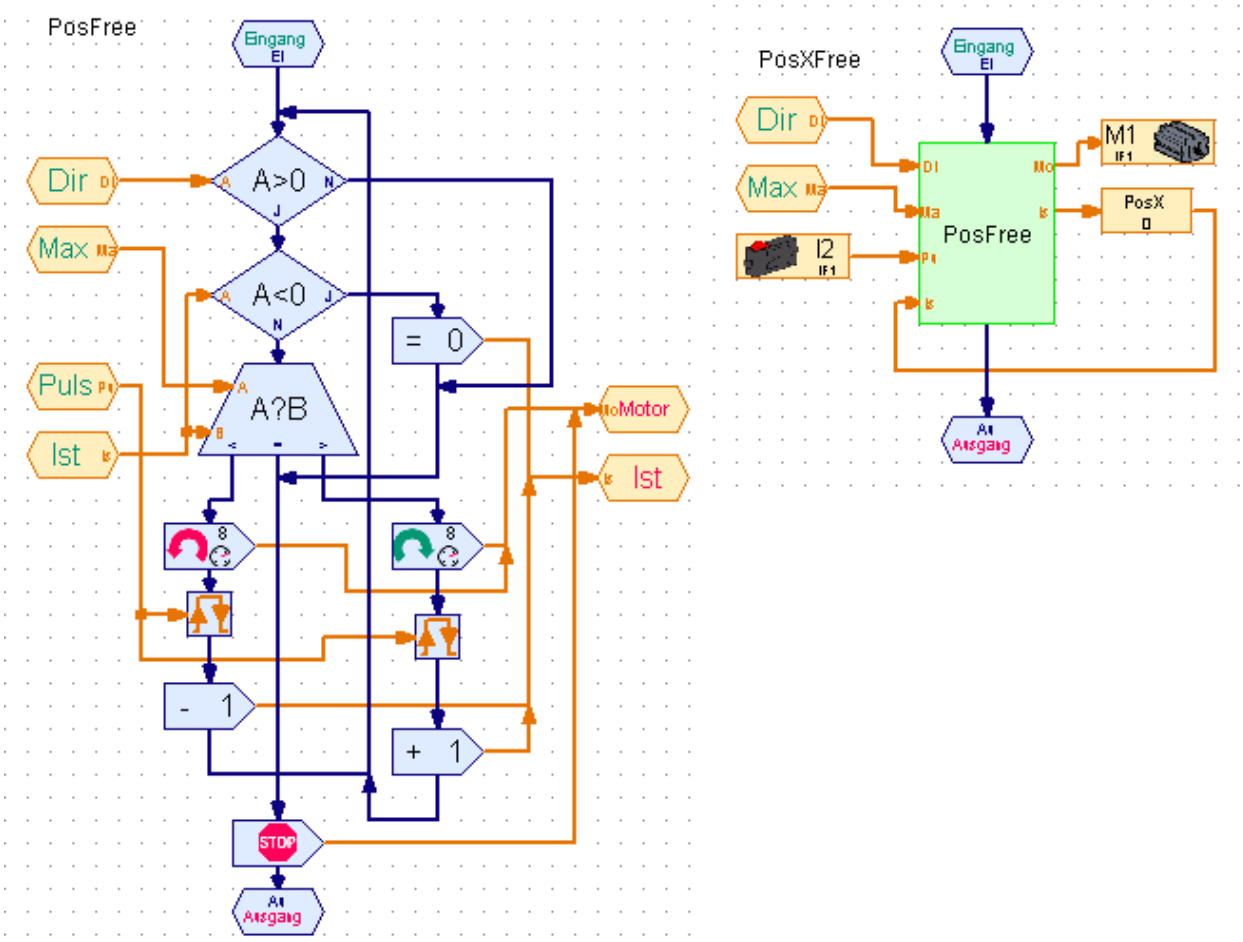
Starts with the move to home position (PosInit) and clearing the position lists (Delete, one list can contain at max 100 elements). The remaining part is dominated by a large loop in which the TeachIn buttons are scanned and the appropriate subs are called. The subs Pos_Free have as parameter the calling button and the max-position. The max-position determine the direction of movement to, the calls therefore are always pairwise.

The end of the loop is occupied by the buttons Store / Delete. They contain a pause to simulate a click event. In each cause lift your finger as quick as possible otherwise position is stored multiple (men with a calm steady temperament should choose a longer pause time).

Notice : Independent from the value of the close position PlayBack closes the grip to position 26, you can change the constant value in Greifer.

If the button Record is no longer pressed, the loop ends, the robot goes back to home.

PosFree / Pos_Free : Move to a Position



PosFree is the general sub for free movement in direction of an max position, their construction is similar to the library sub Pos (for details look their). It runs as long the button (parameter Dir A>0) ist pressed or the minimax value becomes true.

A compare with the end switch is not implemented because of it very hawkful to handle (the button is pressed for more than one impulse). Instead of this positions less than 0 (Ist <0) cause an program end with position 0.

Pos_Free (PosXFree ...) provides PosFree with the special parameters for the component. A fix motor number, special global variables, actual position – with a nice turn from Out to In – logic of the yellow lines. See also library sub PasX ..

Some Point you should be angry with

1. The actual index of the lists is not displayed.
2. Ther is no status text (Record Mode, Tower rotates ...)
3. The Pause in Store and Delete is a quick shut.
4. The PosInit integrated in Record / Delete not alway is a nice solution.
5. and many more

All that should be done very much better – if angry enough for doing that.