MPN software

Preliminary Technical Manual

MPN machines are controlled by a panel pc with a touch screen, installed with Linux and running MPN software.

The MPN software for machine control is roughly speaking divided into two parts.

1. A set of statemachines (software agents) managing a physical part in the machine/workcell and a set of configuration files (input from sensors, loadcells, output to motors, pneumatics, frequency converters etc., parameter for robot paths, product patterns).
2. A binary program written in C/C++ executing the statemachines, communication with external hardware (Advantys for the distributed I/O, Moxa products for shuttle servo, frequency converters and loadcells), updating input/output for usage in statemachines, calculation of robot paths from parameters in configuration file and uploading the paths to the Trio Motioncontroller. The binary part also handles the graphical user interface (GUI).

If changes are required in the binary part a Linux system is needed with a valid software environment for compiling the software. The following chapter will describe how a Linux system is installed and set up for compilation.

The first part does not necessarily need a Linux system since all these files are regular text files (ASCII) and can be edited from any editor and transferred to the panel pc using scp (secure copy). The files can also be edited directly on the panel pc by logging on the panel pc using ssh (secure shell) and the editor vi (editor). On Windows a program called putty can handle ssh connections and 'pscp' can handle secure copying (Both can be downloaded here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)

Table of Contents

1 Installing Fedora Core 5

Alternatively it is possible to use VMware Player (http://www.vmware.com/products/player/) and a preconfigured Fedora Core 5 image (e.g. http://www.thoughtpolice.co.uk/vmware/#fc5). If this solution is selected section 1.1 Preparation and 1.2 Install can be skipped. It might be necessary to install some development packages.

Dual boot with Windows is preferred since some of the software tools used be MPN equipment is Windows only (Motion Perfect 2 for the Trio Motion controller, MacTalk for the shuttle servo motors and Advantys software for the distributed I/O).

1.1 Preparation

Use Partition Magic or similar partitioning tool to prepare the hard drive for installation. First acquire some free space, 8-9 GB should be sufficient.

Create an extended primary partition. In the extended partition create a logical partition on 100 MB for boot directory. Create a partition on 7-8 GB for installation partition. And last create a partition for swap memory (equivalent to pagefile.sys in Windows). The size should be the same as the amount of physical system memory or more.

1.2 Install

Insert disc 1 and setup the computer to boot from cd.

Press Enter to start graphical installation.

Select language during installation.

Select keyboard.

Select 'Create custom layout' and select the drive prepared for the installation.

On the list select the 100 mb boot partition and press edit. In 'Mount Point' select '/boot'.

On the list select the installation partition and press edit. In 'Mount Point' select '/'.

Press Next.

On the next screen select which operation system to boot by default and press next.

Setup the network. To setup for use in MPN network use IP address 192.168.0.XX (15-140 is most likely not to conflict) Netmask 255.255.255.0

Select Time zone.

Enter root password (root is equivalent to administrator).

Select 'Office and Productivity' and 'Software Development'.

Wait for dependencies check to finish.

Click Next to begin installation (this will approximately take 20-30 minutes).

After reboot there will be some last setup of the installation. Accept license agreement, leave Firewall and SELinux on default, set the time and date, set the resolution of your display, create a user (do not use root as default user), test the sound card.

1.3 Setting up the software environment

Login with the user just created (the windowmanager will be Gnome, if KDE or some other is desired it must be installed first)

In the top left corner select Applications → Accessories → Terminal. Execute '`su -`' and enter the root password. You are now logged in as root

To install an editor execute `yum install emacs`. This will install the needed packages.

Execute `yum install SDL_image-devel` to install libraries required to compile the binary part of the MPN software. Execute `yum install qt-devel` to install libraries required by the offline pathplanning tool, mpnguide.

1.3.1 SGE library

The SGE library is used by the graphical interface (Bx)

To compile the SGE library it is necessary to edit an include file, execute:

```
emacs /usr/include/freetype2/freetype/freetype.h &
```

and insert `#include <ft2build.h>` above `#ifndef FT_FREETYPE_H` (should be on linenumber 19). Save by pressing ctrl + x, ctrl + s and exit emacs by pressing ctrl + x, ctrl + c.

Download by executing:

```
wget
http://www.etek.chalmers.se/~e8cal1/sge/files/sge030809.tar.gz
```

Extract by executing: `tar zxvf sge030809.tar.gz`

Switch to directory with `cd sge030809` and execute `make install`

There is a bug in the SGE library that causes unicode/utf8 encoded strings (e.g. Russian) to be displayed wrong. To fix this download libSGE.a and libSGE.so.0.030809 (\\beta2\mvj\MPN software\lib\) and copy them to /usr/lib/

1.3.2 FFTW library

From http://www.fftw.org/index.html: FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST).

Download by executing: `wget http://www.fftw.org/fftw-3.1.2.tar.gz`

Extract by executing: `tar zxvf fftw-3.1.2.tar.gz`

Switch to the extracted directory by `cd fftw-3.1.2` and execute:

```
./configure
make
make install
```

2 Sandbox

Sandbox is the root directory of all MPN software (All rights reserved).

Download sandbox (\\beta2\mvj\MPN software\), unzip sandbox.zip

To be able to use Russian characters copy sandbox/rs/data/arial.ttf to sandox/rs/data/decker.ttf.

2.1 mnt/database/

In the Sandbox the directory mnt/database/ contains statemachines, language, error messages and configuration files. In the following sections the subdirectories and their files will be explained. The text [wid] can be replaced with a 3 digit workcell id number. The workcell id number is a unique identifier for a MPN machine. 300 series for robots, 500 series for checkweighers, 700 series for case packers, 800 series for case magazines and 900 series for miscellaneous.

Saransk: 318 for packed cans and jars palletizer, 319 for filled cans and jars depalletizer, 320 for filled cans and jars palletizer and 321 for empty cans and jars depalletizer and empty cans palletizer. 904 for pallet shuttle transporting pallets to/from 320 and 321.

2.1.1 system/

Directory for top level configuration files. Named [wid]system.ini (e.g. 318system.ini). Contains:

- Which plant and workcell to load.
- IP address assigned to panel pc at boot (pcIPaddr) and which network interface to use (useNic). Changing these can result in no contact to the panel pc after reboot.
- IP address of the trio motion controller (mcIPaddr). Only valid if there actually is a trio in the system.
- Touch screen values. By setting 'touch.initialized=0' the touch screen will be recalibrated at the next startup. 'touchport' specifies which port the touch controller is connected to, can be /dev/ttyS1, /dev/ttyS2 or /dev/ttyS3 depending on the panel pc brand.
- Debug options
- Robot speedfactor (in percent) at startup (startSpeed)
- Default user and password at startup
- Which GUI main screen to load (mainDialog)

2.1.2 plant/

Directory for plant files. Defines which workcells to load. Since there always is only one workcell this file is not necessary but has not yet been removed. The idea was to be able to have more workcells in one plant and control it from one panel pc.

2.1.3 workcell/

Directory for workcell configuration files. Named [wid].workcell. Contains which other files to load. Most of them described in the next sections. Also contains IP addresses for remote hosts which are used when searching for and updating link values and states of statemachine not found locally. If more than one language is loaded the first one will be used when the program starts up.

The keywords for loading other files are: `ROBOT`, `TOOL`, `ITEM`, `PATTERN`, `CPPATTERN`, `ACCESSORY`, `PPSCRIPT`, `DIO`, `COMSK`, `LOADCELL`, `MAC`, `STM`, `SCRIPT`, `STATETEKST` `REMOTEHOST`, `BOPTCOEFF`, `LANGUAGE`.

All files are located in their own respective directory which are described in the following sections, e.g. `ROBOT 318.robot` will load the file mnt/database/robot/318.robot. Only exception is `REMOTEHOST`, which is the keyword for supplying IP address for a remote host.

2.1.4 mac/

Directory for MAC800 Servo motor configuration files. Named [wid]Filename.mac.
Communication with the MAC800 motor goes through a Moxa NPort 5232 (Rev. 2) serial (RS485) to Ethernet converter. The file contains:

- IP Address and port of Moxa where the motor is connected
- The address of the motor on the RS485 bus and the number of motors on the bus.
- Initial value for a register written on the form:

  ```
  Init[register number]=[value] "[name]" "[comment]" [valuetype] [factor=[factorvalue]]
  ```

  [register number] can be found in 'MAC800 Integrated Servo Motors Preliminary Technical user manual' and will be set to [value] when the program starts up. [name] and [comment] are strings. [valuetype] can be longint, fixed16 or fixed24 where the correct type can be found for each register in the manual. [[factor]=[factorvalue]] is optional, it is set to 1.0 if not specified. The factor is stated in the manual or there are information from which the factor can be calculated.

  Example:

  ```
  Init3=0 "P_SOLL" "Commanded position" longint
  ```

- Input bits from a specified register (output bits not supported)

  ```
  InBits[register number]="[name]" "[comment]" [valuetype] [factor=[factorvalue]]
  Bit[bit number 1]="[bit name 1]" "[bit comment 1]"
  Bit[bit number 2]="[bit name 2]" "[bit comment 2]"
  ...
  Bit[bit number n]="[bit name n]" "[bit comment n]"
  EndBits
  ```

  [name] and all [bit name x] can be read from statemachines (Input=inname "[name]"). The maximum number of bits depends on the [valuetype], longint is 32 bits, fixed16 is 16 bits, fixed24 is 24 bits. It is not necessary to specify all bits.

  Example:

  ```
  InBits42="CardBoardShuttle Home Mode" "HOME_MODE" longint
  Bit16="CardBoardShuttle HOME_DONE" "Home Done"
  EndBits
  ```

- Input (read) and output (write) registers

  ```
  [key][register number]=[value] "[name]" "[comment]" [valuetype] [factor=[factorvalue]]
  ```

  [key] can be 'In' for input, 'Out' for output and 'InOut' for a combined input/output.

Example:

```
In29="CardBoardShuttle Temperature" "DEGC" longint factor=0.12207
Out2="CardBoardShuttle Mode" "MODE_REG" longint
InOut10="CardBoardShuttle Curpos" "P_IST" longint
```

The 'InOut' keyword makes it possible to both read and write the register.

2.1.5 dio/

Directory for Advantys I/O configuration files. Named [wid].dio. Advantys is the I/O unit which enables read of sensors over Ethernet and setting valves and motors over Ethernet. Advantys is a collection of modules (all modules is called 'island') and consists of a Ethernet module, a power supply, a series of input, output and motor modules (other modules are available but not used in any current MPN configuration). The configuration file describes how the island is physically configured and specifies the names for inputs, outputs and motors for use in statemachines. The file contains the IP address for the island, a value (NumOfModules) for how many modules and a series of module definitions. Examples of input and output modules are shown below.

```
Type=STB3725DDI                   Type=STB3705DDO
Name=D5101                        Name=D5201
InConnect1=""                     OutConnect1="Valve 1"
InConnect2="Photo 1"              OutConnect2="Valve 2"
...                               ...
InConnect16="Photo 16"            OutConnect16="Valve 16"
```

The order must be the same as the physical Advantys island from left to right. The Name value matches the name in the electrical drawings. An InConnect or OutConnect string must be unique or empty. Statemachines can access the input/output by referring to the InConnect/OutConnect names.

An example of a motor module is shown below:

```
Type=EPI2145
Name=D5301
Connect1="Motor 1 Forward"
Connect2="Motor 1 Reverse"
...
Connect7="Motor 4 Forward"
Connect8="Motor 4 Reverse"
Ready1=" Motor 1 Ready"
Tripped1=" Motor 1 Tripped"
Energized1=" Motor 1 Energized"
...
Ready4="Motor 4 Ready"
Tripped4="Motor 4 Tripped"
Energized4="Motor 4 Energized"
```

The motor module connects to motor-starters which handles the actual motor (power and protection). Ready, Tripped and Energized are status information. Whether or not the motor can reverse depends on the motor-starter. Statemachines can access both the Connect and the status informations.

2.1.6 statemachine/

Directory for statemachines. Named [wid]StatemachineFilename.statemachine for statemachines used for a specific workcell and statemachinefilename.statemachine for general statemachines used across workcells (included by workcell specific statemachines). Each machine (workcell) has a set of statemachines which are included in the .workcell file and executed sequential by the include order. When all have been executed the sequence is started over. This is done 127 times per second.

Each statemachine has a number of states where one of the states is the active one. Example of a state:

```
State=ST_NAME
  ;code
  ;more code
END
```

The name, `ST_NAME`, must be unique. A line preceded with a ';' will be treated as a comment and be ignored be the executer. The code in the active state is executed each time it is the statemachines turn to be executed. In states it is possible to read input, read states and values of other statemachines, set output, do calculations, conditional jumps (if-then-else), use timers and switch to other states.

A state switch can be performed by using this command:

```
SETSTATE ST_OTHER_NAME
```

The rest of the code in the current state after the `SETSTATE` command will still be executed, it is not like a return in e.g. C or C++. `ST_OTHER_NAME` will be executed next time the statemachine is active for execution.

Each statemachine has a unique name defined in the top of the file by `Name=MyStatemachineName`. It must be unique for other statemachines to be able to reference to it and the variables in it.

One statemachine can include another with the command `Include=filename`. The statemachine including the other will be a copy of the one included. Definitions with the same name and states with the same name will be overwritten. E.g. if a timer with the same name is defined the new value will be used instead of the one in the file included. This opens the possibility for general statemachines where small changes can be placed in each specific workcell by overwriting a state or variable. This is an advantage if a general change is required because only one statemachine needs to be modified and the others will automatically include the change (of course except if one of the others has overwritten the state where the change resides).

Every statemachine should at least have the following states:

`ST_IDLE`: The initial state of a statemachine when the program is started. Also here the statemachine should be when the workcell is paused, a semi-safe state (power can still be on) where all output should be turned off and nothing will happen before the workcell is started again. Some statemachines, e.g. a statemachine controlling a blinking light does not need to follow the rules of `ST_IDLE`. Statemachines can be forced into this state from the screen of the panel pc.

`ST_RESET`: A state used for initial and one-time code after `ST_IDLE` and before “real” running states. Can be skipped if nothing needs to be set up or initialized after `ST_IDLE`. Statemachines can also be forced into this state from the screen.

`ST_HALT`: A state used for manual operation of outputs owned by a statemachine. In `ST_IDLE` the outputs might be blocked because they are switched off during pause and can therefore not be manually set. `ST_HALT` is usually empty and the outputs are not blocked. Statemachines can also be forced into this state from the screen. Should be forced to `ST_IDLE` first.

`ST_TIMER`: A special state used in connection with timers. See section about Timeout.

There are some statemachines that are present in every workcell and often similar to one another in different workcells. Some of the most common are described here:

Emergency: Simple statemachine indicating whether the emergency stop is activated or not. The status is updated in different ways on different machines. The emergency statemachine gives an identical interface to the emergency status for all other statemachines needing it.

ContinueKnap: A statemachine acting as a virtual button. The 'Go' button on the screen of the panel pc activates this button. Used by the Workcell statemachine.

Workcell: A statemachine showing which state the workcell is in, error, paused, starting, running, etc. Uses the ContinueKnap to start the workcell. Is also responsible to check for other statemachines to be in `ST_IDLE` before the hole workcell can be declared in pause (semi-safe).

Parent: A statemachine working as a shortcut to the Workcell statemachine indicating whether or not the machine is running in a simpler way. Other statemachines can use this to switch from `ST_IDLE` and go to an an active state when the workcell is started and go to `ST_IDLE` from an active state when the workcell is going to pause.

IOError: A statemachine watching the external hardware for connection problems. If a timeout occurs this statemachine will report error. This statemachine is updated by the drivers of the hardware and is thereby only a link between the statemachines and the binary part of the MPN software.

ErrorCode: Each statemachine has the possibility to report an error and this statemachine is used to gather all the different errors into one simple interface for the Workcell statemachine

TripWatch: Watches for motor overheating and failures and reports error.

Program: Contains parameters for different products. E.g. how many layers to palletize.

TrafficLight: Controls the 3 coloured light tower based on the state of the Workcell statemachine.

The following statemachines are only present when the workcell contains a robot.

ProgramB2: This statemachine decides how the robot should move and act. It decides which path to use in specific situations and the progress of the palletizing and/or depalletizing.

Robot: Functions as a link between statemachines and the Trio Motion Controller. The Trio driver updates variables in the statemachine where some of them are used by the statemachine to reflect the state of the robot through the states of the statemachine. Other variables are used by other statemachines.

Tool: Contains options for the tool mounted on the robot. In some cases the tool is controlled by the Trio (open and close) where the statemachine is a dummy introducing a open and close time delay so the ProgramB2 statemachine does not initiate a path too early. Some times the tool is more advanced than just open and close and the statemachine controls the tool instead of the Trio.

RoboError: Handles error from the Trio (crash, follow error, emergency stop activated while running). Variables updated be the Trio driver.

2.1.6.1 *Types and commands*

Before a type can be used inside a state it must be defined outside the states, normally grouped in the top of the file. The following section describes each of the types that exists in the statemachine environment. Furthermore the commands that can be used with the types are described.

Value and Const

Value is defined using one of the two:

```
Value=name 0
Value=fname 0.0
```

The first will be an integer and the second will be a floating point number. The number will be the initial value when the program is started.

Const is defined with

```
Const=name 0
Const=fname 0.0
```

The only difference between Value and Const is that the initial value of Const can be changed from the screen and will be remembered next time the program starts, the value entered on the screen will be written in the statemachine file. Changing a value of Value from the screen will only be temporarily and will be reset next time the program starts.

Value and Const can be used in conditional jumps in states of a statemachine. An example is shown below including the `SET` and `CALC` commands:

```
TEST name = 3
  ;jumps here if condition is true
  SET name fname
ELSE
  ;jumps here if condition is false
  CALC name = fname + 1
ENDTEST
```

`TEST` is the same as an if-statement in other programming languages. The statement will be evaluated and if name has a value of 3 the `SET` command will be executed and otherwise the `CALC` command will be executed. The `TEST` statement supports these operators: `=`, `!=`, `>`, `>=`, `<`, `<=`

The `SET` command will copy the value of fname (the source) to the value of name (the destination). In this case where name is an integer and fname is a float the value is rounded down. The `CALC` command will evaluate the expression on the right side of the equal sign and copy the result to the be the value of name. Possible operators are `+`, `-`, `/`, `*` and `%`. Only one operator at a time is supported. The `TEST` statement can be expanded with more conditions by added `AND` or `OR` statements. See example below:

```
TEST name != 3
AND fname < 2
OR name = 1
AND fname > name
  ;true
ENDTEST
```

Each extra `AND` or `OR` statement must be on a separate line. The `AND` statement binds harder than the `OR` statement and is evaluated first. This means that the above example is equivalent to:

```
(name != 3 AND fname < 2) OR (name = 1 AND fname > name)
```

The last option in connection with the `TEST` statement is the `ELSETEST` statement. It can replace a `TEST` statement in the `ELSE` part of another `TEST` statement. See the example below, the two are equivalent:

```
                                   TEST name = 3
TEST name = 3                        ;true
  ;true                            ELSE
ELSETEST name = 2                    TEST name = 2
  ;true                 ↔              ;true
ELSE                                 ELSE
  ;false                               ;false
ENDTEST                              ENDTEST
                                   ENDTEST
```

linkValue

A link to a definition in another statemachine. Defined with one of the two:

```
linkValue=localName remoteName remoteStatemachine
linkValue=remoteName remoteStatemachine
```

When localName is left out the local name will be the same as the remote name. The linkValue can be referenced to like it was locally defined. If the remoteName is not located in remoteStatemachine or the remoteStatemachine is not on the local host (the same panel pc) the program will search for the remoteName in the remoteStatemachine on remote hosts (other panel pc in the network) using the IP addresses given in the .workcell file. When using linkValue from a remote host only Value and Const are supported.

Timeout

Timers in the statemachine environment are called Timeout and is defined with:

```
Timeout=name 500000
```

The value is in microseconds. It is only possible to have one active timer per statemachine, if a new timer is activated the old one is overwritten. A timer is started with the two commands shown below:

```
TIMEOUT name ST_STATE_NAME
SETSTATE ST_TIMER
```

The first command starts the actual timer with the value defined in 'name' and indicates that when the time is up the active state will be `ST_STATE_NAME`. The second command switches the actual avtive state to `ST_TIMER`, which is a special state that should be presence in every statemachine (it can be empty). The state `ST_TIMER` automatically watches the timer and switches to the state given in the `TIMEOUT` command (in this example, `ST_STATE_NAME`) when the time is right.

Alternatively it is possible to go directly to `ST_STATE_NAME` but then it is necessary to manually test for the timer in state `ST_STATE_NAME`.

```
TIMEOUT name ST_STATE_NAME
SETSTATE ST_STATE_NAME
```

The timer count down from the value specified and the timer name will contain the actual timer value. When the timer hits 0 the value is set to 1. This indicates that the time is up and can be used with the `TEST` statement:

```
TEST name = 1
  ;time is up
ENDTEST
```

Input

Inputs are used to read values from external hardware, like the Advantys and the MAC800 servo motor. Defined with

```
Input=name "Input Name"
```

The string 'Input Name' is the name defined in e.g. the .dio file for Advantys input. When a statemachine accesses a input the result depends on the type of input. The InConnect from Advantys is integers that can be only 1 or 0 (On the screen shown as ON or OFF). Inputs from MAC800 can be 32 bit signed integer, 16/24 bit signed fixed point. See more in 'MAC800 Integrated Servo Motors Preliminary Technical user manual'. What kind of input is defined in the .mac file.

Inputs can be used in `SET` statements as the source (`SET valueName inputName`), in `CALC` statements as one of the sources (`CALC valueName = inputName1 + inputName2`) and in `TEST`, `ELSETEST`, `AND`, `OR` statements comparing it to numbers, Value's or Const's.

Filter not natively supported by any type.

Output

Similar to input. Defined with

```
Output=name "Output Name"
```

Outputs can be changed with the `SET` command (`SET outputName 1`).

Link

Similar to linkValue but links to one hole statemachine. Defined with one of the two:

```
Link=localName remoteStatemachine
Link=remoteStatemachine
```

When localName is left out the local name will be the same as the name of the remote statemachine. When referring to the Link the result is the name of the state the given statemachine is in. This is used in `TEST` statements:

```
TEST localName = ST_STATE_NAME
  ;The statemachine is in state ST_STATE_NAME
ENDTEST
```

If the statemachine is not found on the local host the remote hosts are searched for the statemachine similar to the way of linkValue. If the statemachine is not found on any remote hosts either the result of a `TEST` statement will be that localName is in state `UNKNOWN`. It is possible to test for this too, `TEST localName = UNKNOWN`. If the statemachine exists and it still returns `UNKNOWN` it can be a network problem or a non-responsive panel pc.

Path

The way to be able to access a robot path from a statemachine. Paths for the robot are defined in the

.boptcoeff file where the different paths are described with parameters and other configuration options, more on this in section 2.1.10 boptcoeff/. The .boptcoeff file must be loaded in the .workcell file before the paths are accessible by statemachines. In statemachines a Path is defined with

```
Path=name mode type
```

The mode and type refers to a specific path in the .boeptfile. The name is only for local use in the statemachine. The mode and type can be accessed by using the subcommands `name.mode` and `name.type`, e.g.:

```
SET name.mode newMode
```

By changing mode it is possible to change the path but keep the name. This is useful if the robot for example picks up object from two different places and needs different path parameters to do this. Then it is enough to change the mode of the path to switch between the two paths from the .boptcoeff file. Thereby some conditional jumps are unnecessary because only one is needed to change the mode and for the rest of the time working with one of those two paths only the name is necessary.

```
LOAD name fromFrame toFrame
```

loads the path with the start point `fromFrame` and endpoint `toFrame` with the parameters and options from the .boptcoeff file defined by mode and type of `name`. The path is calculated by the panel pc and uploaded to the Trio Motioncontroller (MC) but yet not executed. The path will not be executed until the following command is performed by the statemachine:

```
EXEC name
```

Should not be done before the path is done loading which can be checked through the state of the path.

The state of the path can be accessed by `name.state` which indicates what the path is doing and which commands are allowed to use. When `name.state < ST_INACTIVE` it is allowed to change mode and use the `LOAD` command. Each state represent a number which is why it is possible to perform such a test. All the states are shown below:

```
ST_IDLE        1 //inactive state (        : wait for LOAD command)
ST_LOADED      2 //inactive state (        : wait for EXEC or FREE command)
ST_FINISHED    3 //inactive state (        : wait for EXEC,FREE or LOAD command)
ST_ERROR       4 //inactive state (        : wait for FREE command)
ST_INACTIVE    5 //command state  (        : )
ST_LOAD        6 //command state  (STM     : tell LOADER to load path)
ST_EXEC        7 //command state  (STM     : tell EXECUTER to execute path)
ST_BOPTING     8 //active state   (LOADER  : optimize according to parameters, goto LOADING)
ST_LOADING     9 //active state   (LOADER  : send path to MC and goto LOADED)
ST_EXECUTING  10 //active state   (EXECUTER: run path and goto FINISHED)
```

Not all are used in statemachines but only for internal use. Can still be used in statemachines but with no real usage. The `ST_LOADED` state is the state a path must be in before executing the `EXEC` command.

`ST_EXECUTING` means that the path is being processed, the robot is moving. This can be used if something should be calculated or performed while the robot is moving. Could be setting a value to indicate that the robot has moved objects from the pick up point and that new objects can be moved into position for next pick up.

A path state can switch to `ST_ERROR` which means that the path was not finished successfully. Can be caused by upload error (e.g. missing communication to Trio MC), the robots crash sensor activated, emergency stop activated or follow error on a robot servo motor.

`ST_FINISHED` indicates that the path has been executed, the robot has moved and is now finished and ready for the next path (if the state of the next path is `ST_LOADED`). The number is less than the number of `ST_INACTIVE` which means it is possible to load this path again.

Frame

A Frame defines a position of the robot. It consists of 6 values which can describe a position and orientation in 3 dimensions. A frame describes how the robot tool is positioned and oriented. It is defined with

```
Frame=name x=50.0 y=50.0 z=50.0 v=90.0 w=0.0 u=0.0
```

x is the distance from the gantry and out to the tool centre point (tcp). y is the distance from the floor and up. z is the distance from the gantry zero point along the gantry. v is tilt. w is turn. u is not used since the robot only has a total of 5 axis. Each of the above values can be accessed by using subcommands similar to mode and type for Path. The subcommand is the same as the name of the value, `name.x`, `name.y`... The Frame type has its own `SET` command:

```
SETFRAME name name2
```

Which similar to the `SET` command copies the value of name2 to the value of name. All the values are copied, x, y, z, v, w and u.

Item

Item is a physical description of an object used in the workcell (e.g. object palletized by the robot). An Item is described in an .item file. More on this in section 2.1.12 item/. The .item file must be loaded in the .workcell file before the item is accessible by statemachines. In statemachines a item is defined with

```
Item=name NameInFile
```

Usually only the height of the object is used to be able to handle different kind of objects and automatically calculate the pickup position based on one common position (Frame) and measurements of the item. Like Frame there is a `SET` command for Item:

```
SETITEM name name2
```

The subcommands for Item is the same as the parameters in the .item file. See section 2.1.12 item/.

Pattern

The Pattern type is used to palletize in a specific pattern of items on the pallet. When a robot solution is palletizing cases on 2-5 positions on the pallet and on top of each other it is solved by defining 2-5 Frames and using the height of the item. When the pattern is more complex and/or changing from product to product the Pattern type is used. The pattern is described in an .pattern file, more on this in section 2.1.11 pattern/. The .pattern file must be loaded in the .workcell file before the pattern is accessible by statemachines. In statemachines a pattern is defined with

```
Pattern=name NameInFile
```

Also has a `SET` command:

```
SETPATTERN name name2
```

The .pattern file contains a series of frames each indicating one position in the pattern. Each position can be accessed by first setting the position index by:

```
SET name.idx value
```

Where `value` is the number of the position wanted. When idx is set it is possible to access the position values which are like in Frames: `name.wcp.x`, `name.wcp.y`, `name.wcp.z`, `name.wcp.v`, `name.wcp.w`, `name.wcp.u`. There can be more than one position in each layer so a layer value is also available: `name.layer`.

CPPattern

CPPattern is used for defining how to place products in cases using the Case Packer (CP6000). The Case Packer is provided with one product at a time (bag of potatoes, carrots, tray of tomatoes etc.). A number of products is collected and pushed into a buffer place where one layer is build which takes a number of pushes. When a layer is ready it is dropped into the case and the buffer place is ready for building a new layer. This is repeated until the case is full.

The cp pattern is described in an .cppattern file. The .cppattern file must be loaded in the .workcell file before the cp pattern is accessible by statemachines. In statemachines a cppattern is defined with:

```
CPPattern=name NameInFile
```

Also has a `SET` command:

```
SETCPPATTERN name name2
```

The .cppattern file contains a series of layers describing how many pushes to do in each layer and how many products per push (ppp). Subcommand `name.maxlayers` gives the number of layer in each case. It is possible to set `name.layeridx` which indicates which layer to get information from and `name.pppidx` which indicates the pushnumber in the selected layer to get information from. With these set it is possible to get `name.maxpush` which indicates the number of pushes in the selected layer. `name.ppp` indicates the number of products per push for the selected layer and pushnumber (pppidx). `name.gettime` indicates the time (in microseconds) for the product to move from the photosensor to the place where it is ready for push.

Table

Table is a type for collecting integer or float values. Used to view the history of a value, to calculate the average value of the values inserted and/or to find the maximum value of the values in a specified sampling area. A Table is defined with one of the following:

```
Table=name1 20.0
Table=name2 200.0 AVG
Table=name3 600
```

The value defines the size of the table, and the type of the value defines the type of the elements, float or integer. The size can be changes with `SET name1.newtablesize newSizeValue`. Using the keyword, `AVG`, indicates that the table is for average calculation. Inserting elements in a table is done with the normal `SET` command:

```
SET name1 value
```

The value is inserted after the other elements in the table. It is not possible to index into the table. It works as a circular buffer. When the table is full the insertion is started over and the value goes into the first place in the table. When the non-average table is used as the source, `SET value name1`, the last value inserted is copied to value. When the table is of the average type, the average value of all the values in the table is returned.

To find the maximum value the following can be done:

```
SET name1.topsearchwidth topSearchValue
SET topidx name1.topidx
SET topval name1.topval
```

The table will be searched backwards for the maximum value until topSearchValue number of elements has been searched. topval will be the maximum value. The order of the commands are important. topidx and topval can not be calculated without setting the top search value. topidx invokes the actual calculation and must be done before topval.

Other subcommands: `name1.maxidx` gives the actual insertion index, `name1.size` gives the size of the table.

Other commands

`ROBOTSPEED`: sets the speedfactor of the robot. Equivalent to the speed button on the lower left corner of the panel pc main screen. Should not be set to less than 1.0 and not more than 100.0

`PRINT`: prints any of the types described in the previous sections. To see the prints it is necessary to start the program manually from a console, log on to the panel pc and start the program.

`INLINE`: command to import statemachine commands from a file. In some cases some lines of code are used more than one place in different states. By placing those lines in one file and using the `INLINE` command the code is locating in one place making it easier to make changes if necessary.

2.1.6.2 Example

The example is a statemachine going to `ST_RESET` at startup where a counter and a timer is initialized. The counter is incremented every 2 seconds and printed. When the counter reached a value of 3 it is reset.

```
STATEMACHINEVERSION 3
Name=Test
Timeout=time 2000000
Value=count 0

State=ST_HALT
END

State=ST_TIMER
END

State=ST_IDLE
  SETSTATE ST_RESET
END

State=ST_RESET
  SET count 1
  PRINT count
  TIMEOUT time ST_RUN
  SETSTATE ST_TIMER
END

State=ST_RUN
  TEST count = 3
    SET count 1
  ELSE
    CALC count = count + 1
  ENDTEST
  PRINT count
  TIMEOUT time ST_RUN
  SETSTATE ST_TIMER
END
```

The printed output from this statemachine is shown below, the numbers are timestamps of the `PRINT` command. The three dots indicates that the output continues.

```
00000:514 PRINT VALUE: count=1
00002:530 PRINT VALUE: count=2
00004:542 PRINT VALUE: count=3
00006:558 PRINT VALUE: count=1
00008:566 PRINT VALUE: count=2
00010:579 PRINT VALUE: count=3
...
```

The `ST_RUN` state is only active every 2 seconds and if some other work (like monitoring an input) is required during those 2 seconds, the code must be placed in `ST_TIMER`. This is not possible if other states use `ST_TIMER` for time delays since the code might not be suitable for that time delay. To avoid this and do other work it is possible to check the timer manually in `ST_RUN` and do the other work outside the timer `TEST` statement. The new `ST_RUN` is shown below. For it to be in the same statemachine the state is called `ST_RUN1`:

```
State=ST_RUN1
  TEST time = 1
    TEST count = 3
      SET count 1
    ELSE
      CALC count = count + 1
    ENDTEST
    PRINT count
    TIMEOUT time ST_RUN1
    SETSTATE ST_RUN1
  ENDTEST
  ;work while waiting for timer
END
```

Furthermore it is necessary to change the following in state `ST_RESET`

```
TIMEOUT time ST_RUN        →    TIMEOUT time ST_RUN1
SETSTATE ST_TIMER               SETSTATE ST_RUN1
```

The result is the same as before but now it is also possible to perform other work while waiting for the timer.

2.1.7 statetekst/

Directory for files containing info/error messages to be showed on the screen. Named [language].statetekst. The other files are deprecated. The format for the file is:

```
STATETEKSTVERSION 1
English
00001 Info: Scanning hardware.
...
00232 An error has occurred - the robot is halted.
00232 The robot has dropped the filled cases.
...
05011 Error: Pallet not received in time. Please remove pallet manually.
...
```

The first line is the version of the statetekst file syntax. The second line is the name of the language, in this case English. The rest of the is series of string id numbers and strings with the info/error message. The ID numbers does not need to be sequential. They must be unique or have two affiliated strings, like `00232` in the example above. Only works for error messages since the screen has two available lines for errors and only one for info messages.

To have an error message shown on the panel pc a statemachine can define a value called errorCode:

```
Value=errorCode 0
```

When the errorCode value is set to a different number than 0 the string equivalent to the ID number will be displayed on the screen. All statemachines are searched for the `errorCode` value. Similar for info messages, only the value is called `message`. The only difference of setting `errorCode` and `message` is where the string is displayed on the screen.

2.1.8 language/

Directory for files containing strings for the GUI. Named [language].language. The format for the file is:

```
LANGUAGEVERSION 1
Dansk
0000 "System" "System"
...
0277 "Antal Produkter" "Number of products"
...
```

First line is the version of the language file syntax. The second line is the name of the language, in this case Danish, which will appear on the screen when selecting language. The rest of the file is a index number for the string, the string in the selected language and the string in English.

2.1.9 robot/

Directory for robot configuration files. Named [wid].robot. Contains robot parameters which should not be changed. Also contains robot calibration values, example:

```
m_jointCalib.a=667.08,b=57.71,c=93.55,d=-27.10,e=5.42,f=0.00
```

These values will be changed if any of the calibration steps are performed. See the calibration manual. Should not be changed manually and be careful not to overwrite the file while e.g. updating by copying the complete mnt/database directory from a local computer to the panel pc.

2.1.10 boptcoeff/

Directory for robot path parameter files. Named [wid].boptcoeff. Learn more about boptcoeff and robot paths in the manual 'MPN banefremstilling' (in Danish).

2.1.11 pattern/

Directory for pattern files. The .pattern file contains a series of frames each indicating one position in the pattern.

```
PATTERNVERSION 2
2Kg_80
width=600
length=800
layer=0,wcp=290,0,170,0,-90,0
layer=0,wcp=430,0,170,0,90,0
layer=0,wcp=170,0,170,0,-90,0
layer=0,wcp=590,0,170,0,90,0
layer=0,wcp=430,0,405,0,90,0
layer=0,wcp=310,0,405,0,-90,0
layer=0,wcp=590,0,405,0,90,0
layer=0,wcp=170,0,405,0,-90,0
layer=1,wcp=230,50,290,0,180,0
layer=1,wcp=510,50,290,0,0,0
layer=1,wcp=170,50,290,0,-90,0
layer=1,wcp=590,50,290,0,90,0
...
```

First line is the version of the pattern file syntax. The second line is the name of the pattern. The variables `width` and `length` are only used for visualization in the pattern editor screen, it describes the size of the pallet being palletized using the pattern. The following lines are the actual pattern described with a layer variable and a wcp variable which like frames are described with a x, y, z, v, w, u variable: `wcp=x,y,z,v,w,u`.

2.1.12 item/

Directory for item files. Contains a physical description of an object.

```
 VERSION 3
 BOX Netto dx=604.00,dy=221.00,dz=403.00,sy=212.50,gy=185.50,my=198.00,gx=576.00,weight=2.0
```

First line is the version of the item file syntax. The second line is the type of the item (deprecated?) and the name followed by different dimensions of the item described here:

dx is the length of the item. Found by measuring the item along the tool of the robot when the item is moved.

dy is the height of the item. Found by placing the item on a plane surface and measure from the surface to the top if the item.

dz is the width of the item. Found by measuring the item across the tool of the robot when the item is moved.

sy is the height of the item when it is stacked. Some items fit into one another when stacked and the stacked height is therefore not equivalent to the height. Can be found by stacking 10 items, measure the height of the stack and divide it by 10.

gy is the height of the item where it is gripped by the tool. Usually in the middle of the carrying holes if the item is a case.

my is the height of the item when it is lifted and moved by the tool. Usually in the top of the carrying holes of the item is a case.

Griplength (gx) is the length of the item where it is gripped by the tool. Useful when the bottom of the item is not the same length as the top and the item is stopped in the pick up position by a physical endstop at the bottom of the item.

weight is the weight of the item. Only used in the dynamic model in the offline pathplanning program, mpnguide.

2.1.13 tool/

Directory for tool files. Named [wid].tool

```
      VERSION 2
      311 dx=1210.00,dy=20.00,dz=390.00,weight=25.0,itemsperlift=2
```

First line is the version of the tool file syntax. The second line is the name followed by the physical dimension, the weight and how many items it can carry. The tool file is only used by mpnguide to visualize.

2.1.14 accessory/

Directory for accessory files. Named [wid].tool

```
VERSION 1
320ConveyorOut
dimention x=800.0,y=-100.0,z=2000.0
transformation x=534.0,y=1250.0,z=2715.0
material diffuse r=0.2,g=0.8,b=0.2,a=1.0
material ambient r=0.2,g=0.8,b=0.2,a=1.0
material specular r=0.2,g=0.8,b=0.2,a=1.0
shininess=1.0
```

The accessory file is only used by mpnguide to visualize objects in the workspace besides the robot, like conveyors, obstacles, pallets etc. First line is the version of the accessory file syntax. The second line is the name. The two next lines are the physical dimension of the object and the position in the workspace. The rest is for the appearance.

2.1.15 Other directories

The remaining directories are not relevant for Saransk and therefore only described shortly.

The comsk directory contains configuration files for Commander SK General Purpose AC Drive (frequency converters) from Control Techniques.

The cppattern directory contains files with patterns how to place products in cases using the Case Packer (CP6000).

The icpcon directory contains configuration files for a new distributed I/O module. Simpler, smaller and cheaper than Advantys. Has not yet been used in any MPN machines but good for small machines which only require limited number of I/O (e.g. 8 in, 3 out)

The loadcell directory contains configuration files for loadcells in the checkweigher.

The ppscript and script directory contains configuration files used earlier together with boptcoeff. Now the scripts are deprecated.

The udpserver directory contains configuration files for a communication protocol over UDP Ethernet.

2.2 robostacker

Binary part

In the sandbox directory execute `make` to compile. The executable is placed in the rs directory. The code is located in different directories.

Bx contains the code for rendering the screen.

libmpnrobot/libmpnrobot contains the core of the program, functions for loading files, calculating paths, functions to access information from statemachines etc.

hmi contains code for handle input from the touch screen and.

io contains code for hardware interfacing (drivers)

stm contains code for executing the statemachines. Also contains code for the server/client model for communication between two or more panel pc's.

rs contains code for each different screen (rendered by Bx), loader and executer for paths and the code that binds all the other things together.

In the rs directory there should be a file, 'mntdatabase.txt' containing:

```
../../mnt/database/
XXXsystem.ini
```

where XXX is a workcellnumber. It defines which database to use and which workcell in that database to use.

To start the program execute `./robostacker` in the rs directory. It will load the workcell defined in 'mntdatabase.txt'. Some options can be given to the program:

```
-s          Simulate all
-s mouse    Use mouse instead of touch
-w id       Override workcell id given in mntdatabase.txt
-db path    Override database path given in mntdatabase.txt
```

The `-s` option will use the mouse instead of touch and simulate all hardware which mean no connection will be made to the different hardware in the setup (Advantys, Trio, MAC800 etc.) This is useful when running the program on a regular pc for testing statemachines or other things not requiring the hardware. The `-s mouse` option will connect to the hardware but use the mouse.

2.3 bxbuilder

Program to generate GUI skeleton code based on ui files. The ui files are text files describing how the page/screen/form is build up of different elements (buttons, listboxes, checkboxes etc.) in a language similar to XML. The files can be created and modified with Qt Designer (will automatically open when a ui file is double-clicked in bxbuilder). Obviously needs to be installed first, install by executing `yum install qt-designer` as root (su -).

The program is located in sandbox/bxbuilder/bxbuilder/ where it can be compiled by executing `make`. The program is launched by executing `./bxbuilder`. Open the file sandbox/rs/robostacker.bxp from bxbuilder to load the robostacker project. This file contains which ui files are used.

To create a new screen copy an existing ui file under a new name and add it to the bxp file (Alternatively add it from bxbuilder from the menu Edit → Add). Open the project (bxp) in bxbuilder and open the ui file by double-clicking on it in the listbox and modify it with the Qt Designer. Remember to change the main dialogue name since bxbuilder uses it when generating function and variable names. It might cause a conflict with the code generated if 2 forms has the same name. Save and close Qt Designer and in bxbuilder select File → Generate and press Generate. This will generate the following files for each ui file:

```
FormName.h
FormName.c
FormNameCode.c
FormNameUser.h
FormNameUser.c
```

FormName is the name of the main dialogue. The User files are only generated if they do not already exists. These files contains user code for events on the form like clicking a button and is therefore not generated since code will be lost. Before the form can be used the names of the C files must be added to sandbox/rs/Makefile in the `C_SRCS` variable. To compile the new code execute `make` in sandbox/rs/.

2.4 mpnguide

Program used for off-line path planing. The paths are defined in the boptcoeff files and this program is used to visualize the paths. Learn more about boptcoeff and robot paths in the manual 'MPN banefremstilling' (in Danish).

The program visualizes the workcell in 3D and needs glut libraries to compile, install by executing `yum install freeglut-devel` as root (su -). Execute `make` in sandbox/mpnguide/mpnguide. Like in the rs directory there should in this directory be a file defining which workcell to use, 'mntdatabase.txt'. This program has no options so the desired workcell id must be entered in the file.

The program is started by executing `./mpnguide`.

3 Panel PC

3.1 Creating flash disk

The procedure for creating a flash disk is described in the file sandbox/flash/HOWTO.

3.2 Files and directories

The first partition contains Linux and when booted a ram disk is created with the Linux file system from the ramfs.igz file. The second partition contains the MPN software and is mounted in /mnt/ and contains the following:

- extras:
  - scripts to get information from the statemachine through the binary part. Useful for remote debugging. getStatus.sh prints the status of the machine and the current state of the Workcell statemachine. AllStates.sh prints the current state of all statemachines. AllVal.sh prints all values in a given statemachine.
- database: The directory described in section 2.1 mnt/database/
- robostacker: the binary program.
- mntdatabase.txt: text file indicating which workcell from mnt/database/ to start.
- data: directory containing images, font and resource file used by robostacker
- startup: script executed at panel pc start up. Starts robostacker and restarts robostacker when it is restarted from the screen (Menu → STM Control → Menu → Restart Program)
- connect2mpn.sh: script to make a connection to the MPN server (mpnrobotics.dyndns.org) so it is possible to log on remote even if the panel pc is on a private network behind a router.

3.3 Log in

To log in on a panel pc the network on the pc used must be set up correct. The IP address should be in the range 192.168.0.2-140 and the netmask should be 255.255.255.0.

Log in with the command `ssh root@192.168.0.XXX` where XXX can be found in mnt/database/[wid]system.ini (usually in the range 151-155). The password is 'Robotten'

On Windows a program called 'putty' can handle ssh connections (can be downloaded here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)

For debugging it can be useful to start the program manually since it will print diagnostic messages which help to locate a problem. First terminate the running program with the command `killall robostacker`. Switch to the /mnt/ directory (`cd /mnt`) and execute `./robostacker`. The program can be terminated by pressing Ctrl + c. To return to normal operation execute `reboot`.

3.4 Editing and updating

When logged in files can be edited with a simplified version of the editor vi. It is started with `vi filname` and will start up in command mode. In this mode it is not possible to edit anything. By pressing 'i' the mode is changed to insert mode and it is now possible to edit the file. When done editing press esc twice to return to command mode. The following basic commands should be enough to do simple editing of a file:

dd	Delete the current line
:w	Save changes
:wq	Save changes and quit
:q	Quit
:q!	Quit without saving changes

When changes are made directly on the panel pc the changes should also be transferred to the local file on the pc/server.

Generally it is important to keep the panel pc and the local version synchronous. Variables can be changed from the screen of the panel pc and these changes will not automatically be transferred to the local files. One semi-automatic way to synchronize the files is to mount (share) the local sandbox over the network on the panel pc and use the script sandbox/extras/kompareworkcell.sh. Both described in the next two sections.

3.4.1 Mount

Mount is a term from Linux which is the name of what is needed to access a device (harddisk etc.), to mount a device into the file system for access. It is also possible to mount a directory from another pc into the local file system so it can be accessed like it was local. This requires a NFS server on the other pc which is usually installed together with any Linux distribution.

On the panel pc execute the following command to mount the your local copy of the sandbox:

```
mount -t nfs -o nolock,vers=2 [IP]:/path/to/sandbox /tmp
```

The directory /path/to/sandbox/ on the computer with IP address [IP] is mounted locally in the directory /tmp/. It is best to mount it in /tmp/ because this directory on the panel pc is made for this purpose. Now it is actually also possible to start the version of the robostacker program located on the local computer by entering the /tmp/rs/ directory on the panel pc and execute `./robostacker`. When this is done the local sandbox is used (only if mntdatabase.txt contains the correct database) which means that it is possible to perform changes on the local pc instead of using vi on the panel pc. Afterwards it is of course important to copy changes to the panel pc (backup data, robostacker and database on the panel pc first). Full update:

```
cp -r /tmp/rs/data/ /mnt/
strip /tmp/rs/robostacker
cp -r /tmp/rs/robostacker /mnt/
cp -r /tmp/mnt/database/ /mnt/
```

/tmp/rs/data The `cp -r` command copies a file or a directory recursively to a destination. The `strip` command removes debug information from a executable and thereby reducing the size. It is a good idea to synchronize before updating.

3.4.2 Synchronize

When the local sandbox is mounted on the panel pc it is possible to see the difference between 2 workcells in the 2 versions of the sandbox using the script /tmp/extras/kompareworkcell.sh. To compare 2 workcells execute:

```
sh /tmp/extras/kompareworkcell.sh -sync [wid]
```

which will compare the workcell with id [wid] from /tmp/mnt/database/ (from local pc mounted on the panel pc) with the workcell with the same id but located in /mnt/database/ (on the panel pc). The above command only shows lines that differ. It is also possible to shown the full files with indication of which lines that differ. See other options to the script by supplying the option `--help`.

Only files starting with [wid] are compared. A workcell may still use files which are not compared, the script is under development and therefore not foolproof. The option `-gendiff` will compare the directories most likely to contain differences; item, pattern, statetekst and language.

4 Windows software

The software described here should not be used to connect to any components of a running system.

4.1 Motion Perfect 2

This program is used to connect to the trio motion controller which is the part of the system that executes paths from the panel pc into physical motion of the robot through the HAC and SAC drives (The 2 big green components in the electrical cabinet).

Install_MotionPerfect_2_2_2_2.exe, can be downloaded from http://www.triomotion.com/

Options → Communications → Add → Ethernet and enter the IP address of the trio. COM1 can be deleted. Controller → Connect. A dialogue will appear where the program in the controller should be saved on the computer. This can take some time, then done there will on the left be a list of programs. Green means the program is running. The latest version is located at sandbox/motionperfect/Robostacker309/

Never click 'Drives Enabled'. It will release the brakes of axis B and axis C before the motors are ready. The robot will collapse.

For diagnostics: Stop running programs: Right click → Stop (it might be necessary to change the Controller Status to editable), reset the emergency stop (on the stand with the panel pc), open a terminal for monitoring the output from the program: Tools → Terminal → Select 0. Double click the program TESTSLM and if it in the beginning has the following:

```
loop:
MOVE(-200) AXIS(6)
WAIT IDLE
MOVE(200) AXIS(6)
WAIT IDLE
GOTO loop
```

remove it and select Debug → Run. The program tries to detect and reset each of the drives (not enable, no movement will be performed). The terminal will print the status for each axis. If it prints 'not detected' or 'unable to reset drive' there is a problem with that axis. Unfortunately this can be caused by several different things. The SLM cards in the trio, the grey cables from the trio to the drives, the drive, the green encoder cable from the drive to the motor or the motor.

It can also be caused by overheating of the drives and the first thing to try is to turn off the power (turn switch on the door of the electrical cabinet), open the doors and wait for 10-20 minutes and try the TESTSLM again.

If that does not help the next step is to try the TESTSLM with another green encoder cable for the axis in question. Before removing the cable all power to the trio and the drives must be turned off. Also the backup power. First turn off the big switch on the electrical cabinet, then turn off the switches below the green UPS system in the top left corner. The UPS will go into battery mode which is turned off by turning the small rotary switch on the UPS to 'service'. Doing this will require a recalibration when the problem has been fixed and the robot is started up again. Wait 30 seconds before removing the cable. For the TESTSLM it is not necessary to exchange the cable completely, disconnect the old cable from the drive and the motor and let the new one go by air. With the new cable applied turn the rotary switch on the UPS back, turn on the switches below and power up the electrical cabinet and try the TESTSLM again. If the TESTSLM returns that all axis are OK the cable should be guided through the cable carrier etc. as the old cable. Remember to turn off all the power before disconnecting the cable. In all the drive connectors there are 2 pins soldered together, reefer to the electrical documentation to see whether or not it should be on the axis with the new cable (axis B and axis C should have it). Also possible to just check the connector on the old cable.

Try with a new motor. Remember the power.

Switch drive.

Switch trio.

4.2 Advantys

Advantys is the I/O unit which enables read of sensors over Ethernet and setting valves and motors over Ethernet. Advantys is a collection of modules (all modules is called 'island') and consists of a Ethernet module, a power supply, a series of input, output and motor modules (other modules are available but not used in any current MPN configuration). If any changes to the configuration or module parameters are required the 'Advantys Config Software' is needed. A 21 days trial can be downloaded from http://www.telemecanique.com/

If the Ethernet module (NIP 2212) is new it needs configuration of IP address, how the modules are connected and parameters for each module. One way to set the IP address is by DHCP (this requires access to the DHCP server to see which address is asssigned, if this is not possible other methods are described in the help section of the Advantys software). To use DHCP set both the rotary switches to a number (doesn't matter which numbers) and power up the module. Acquire the IP address from the DHCP server.

Use a browser to connect to the webpage of the Advantys on the assigned IP address. On the webpage change the IP to 192.168.0.17X, netmask 255.255.255.0. If there are more than one machine in the network make sure there are no conflicting IP addresses. If the module is replacing a defective module use the same IP address as the old module. Unpower the module and set the bottom rotary switch to internal (the island will use IP address set from webpage). Repower.

In the Advantys software first set up the connection settings: Online → Connection Settings → TCP/IP → Settings → Remote IP Address. Enter the IP assigned in the previous step. Select Online → Connect. If it is a new module it should automatically detect the input, output and motor modules. If the island has previously been configured the old configuration is loaded. Select Online → Disconnect and Island → Lock (the island must be unlocked to be able to modify it).

If it is a new the Ethernet module manually add the power module(s) from the Catalog Browser on the right Catalog → Power → STBPDT3100. Drag and drop it to the position matching the physical position. Also add Catalog → Accessories → STB XMP 1100: Termination Plate to the right end of the island. If the module is configured add/remove the desired modules.

The input module, DDI3725 and the output module DDO3705 needs no further configuration.

For each motor module, EPI 2145, Double-click and change the parameter 'Fault Recovery Response' from 'Latched Off' to 'Auto Recovery'.

When configuration is done select Island → Build and Online → Connect and select 'download'. Reconnect and let the software load the current configuration in the island to check that it is correct.

4.3 MacTalk, MAC800 Servo motor and Moxa Nport 5232

A windows setup and diagnostics tool for the MAC800 Servo motor.

Communication with the MAC800 Servo motor goes through a Moxa NPort 5232 (Rev. 2) serial (RS485) to Ethernet converter. To connect to the motor with MacTalk over Ethernet this can be used: Serial to Ethernet connector (trail version: http://www.eltima.com/products/serial-over-ethernet/). MacTalk connects to the MAC motor through serial so this program is necessary to emulate serial over Ethernet.

No setup is needed for the motor so MacTalk is only for diagnostics if something happens that can not be handled from the panel pc.

Moxa Nport 5232 has a build in web server so any browser on any operating system should suffice. The default IP address is 192.168.127.254 and for first time configuration it is necessary to be on the same subnet, e.g. having the IP address 192.168.127.253. Change the following and leave others to default values:

Network Settings:

- IP address: 192.168.0.20X. If there are more than one Moxa in the network make sure there are no conflicting IP addresses. If the module is replacing a defective module use the same IP address as the old module.
- SNMP: Disable

Serial Settings:

- Baud rate: 19200
- Stop bits: 2
- Flow control: None

Operating Settings:

- Operation mode: TCP Server Mode
- Local TCP port: 400x. The Moxa has 2 serial ports and if both are used set the port number to 2 different number and use these in the .mac files.