Force control interface for ABB S4/IRC5

/home/isolde/robot/doc/extctr.pdf

Isolde Dressler

July 13, 2009

1. Overview

A standard industrial ABB S4 robot controller has been extended for performing control experiments. The extension makes it possible to read and modify the position, velocity and torque references that are sent from the main controller to the axis controllers (Fig. 1).

A controller modeled in Simulink can be integrated in the robot control using Real-Time Workshop and an opcom interface communicating with controller and S4Cplus via the labcom protocol. Using the interface, controllers can be loaded, controller parameters changed and sensor data be accessed. Controller signals can be logged by an external program.

2. Safety

The customized controller will bypass S4's safety procedure by sending the modified position/velocity reference directly to the axis controller, **so never use a controller unless you have not made absolutely sure that it is robust and stable!** You might damage the robot, the robot lab or worst, harm yourself or other people!
Always go through **all** of the following steps for testing a controller:

1. Test the controller with the robot simulation model in Simulink. Do not only test the ideal case! Check what happens if the controller is activated/deactivated.

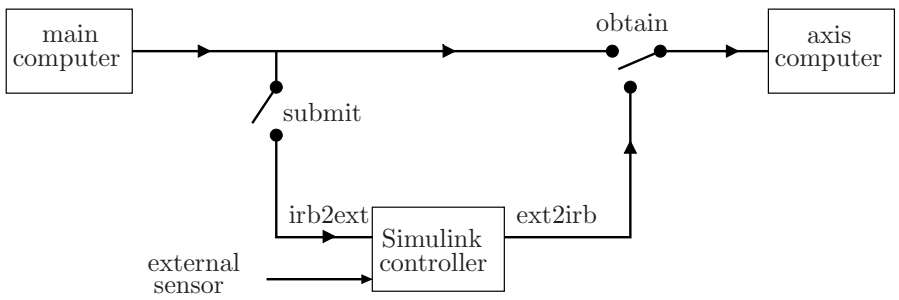


Figure 1 Schematic of reference value path from main to axis computer

2. Load the controller in opcom and test the logged signals in "submit" state. Test the control without environment interaction, i.e. produce contact forces with a broomstick or similar.
3. Finally, the experiment can be performed in "obtain"-state. Test the control first with a soft environment, e.g. a cardboard box.

Pay attention to the following things:

- Use a parameter for activating the controller from the opcom interface. It has to be inactive when starting the controller (obtain), else the controller is in an uncertain state and the robot's movements are unpredictibly when pressing obtain. This parameter will in the following be called `f_switch`.
- Never press "obtain" in opcom if you are not absolutely sure that the loaded controller is deactivated (fswitch 0) and its states (integrators) are 0, else the robot might make an unexpected and perhaps very large jump.
- Never leave the robot and control in an unsafe state when you are not near. Press the emergency stop button, unload the controller in opcom (never leave it in obtain-state!) and eventually exit the extrapid program.
- Integrate a routine in your controller to return the robot/controller states to its initial position/zero with low velocity when the controller is deactivated.
- When running a control experiment, have robot in manual state so that switch has to be pressed during the experiment. Always watch the robot and be ready to abort it by letting go the switch, even if you have run the same experiment before.
- When running extrapid, introduce a signal in your controller that tells if the controller states have returned to zero. The first command in an extrapid program should check this signal and only continue when it is sufficiently small.
- Pay special attention when running an extrapid program in automatic state of the ABB controller.

3. Controller in Simulink

As there are problems with the current version of Simulink and Real-Time Workshop (June 2007), the 2006 version has to be started:

```
VERSION=R2006a matlab -nodesktop
```

The s-function files necessary for the robot control library "extctrl" are only available on gladia in the robot lab. The directories including them are added with:

```
addpath /opt/robot/matlab/;
addpath /opt/robot/matlab/irb/mex/;
addpath /opt/robot/matlab/irb/mex/mex_IRB2400_16;
```

For other robots than IRB 2400 the corresponding directory has to be included.

The commands can conveniently be executed by including them (and others) in an executable text comment in Simulink. An available template file contains such a text comment, otherwise it can be obtained by choosing "Annotation properties" in the text comment's context menu and choosing "Use display text as click callback".

As the above pathes should be available when the model is opened, it has to be reopened after executing the commands, else the linking between Real-Time Workshop and Orca is not done properly.

The modeling of a controller is explained in Sec. 3.5 on a simple example (see Figs 5 and 6).

3.1 Labcom files

For communication between the Simulink controller and the robot, you have to write a labcom file (*.lc) which specifies the input/output signals for labcom. See the following example:

```
sample float forceOut[6];
sample float fswitch;
```

As can be seen in the example, the file contains inputs and outputs in random order. Vector signals have to be specified in the shown way. The name of the labcom file has to be added in the RTW options ORCA tab (see App. A).

3.2 Real-Time Workshop

From the opcom interface it is possible to change controller parameters. To be able to do so, these parameters have be declared before building the model. In the model's Simulink menu, go to *Tools* → *Real-Time Workshop* → *Options*. Select *Optimization*, then *Configure* next to *Inline parameters* in the *Simulation and Code Generation* block. New parameters can be declared selecting *New* and typing the name it has in the Simulink model. The parameter type should be "auto".

Screen prints with other settings can be found in App. A.

Once the Simulink controller is modeled, the files used for control can be generated with "ctrl + b" or select *Tools* → *Real-Time Workshop* → *Build Model*.

3.3 Library extctrl

The Simulink blocks available in the extctrl library can be seen in Fig. 2. It contains kinematic blocks for the IRB robot (the uppermost line of blocks) and blocks to calculate between different coordinate systems (the right part). The signal transmission and controller blocks to the left are seldomly used.

It is important to keep in mind in which coordinate system a pose or vector is expressed. More on different coordinate systems follows in Sec. 3.4.

There is also a difference between joint angles expressed on the arm or on the motor side. To change between the two the two blocks "arm2motor" and "motor2arm" exist. Inputs and outputs (ext2irb or irb2ext) of the Simulink model are given in motor angles, inputs to the kinematic blocks in arm angles.

Matrices are always given as an array in which the elements are stored row after row. The vector indices of a homogeneous matrix are then for example:

$$T44 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 2 \\ \vdots \\ 15 \\ 16 \end{pmatrix}$$

Kinematic blocks There is an IRB forward kinematics block (arm angles as input), two different Jacobian matrix blocks (available to give velocities in base or flange frame, $\dot{X} = \mathbf{J} \cdot \dot{\theta}$) and a block that calculates the pseudo inverse of the input (Jacobian) matrix to compute the inverse Jacobian.

Transformation blocks

- **Velocity/force transmission:** As forces/velocities are not only vectors but have a point of attack, the absolute values of momentums and angular velocities might change if they are expressed in another frame having a different origin.
- **arm2motor/motor2arm:** Calculate from joint angles on motor to arm side or vice versa; input to kinematic blocks in arm angles, inputs/ outputs of Simulink model in motor angles.
- **Quaternions and homogeneous representation:** Any transformation between coordinate system is described by a rotation and a translation. A quaternion respresentation has 7 elements, the first 3 are the translation vector and the remaining 4 quaternions describing the rotation. Two blocks are available to change between the two representations. More on quaternions and homogeneous coordinates can be read in [1].
- **Quaternion/T44 inversion:** If the input $T44$ of this block fulfills $P_{\text{base}} = T44 \cdot P_{\text{flange}}$ (P_{flange} are the coordinates of point P given in the flange frame), the output $T44^{-1}$ fulfills $P_{\text{flange}} = T44^{-1} \cdot P_{\text{base}}$. Similar for quaternions.
- **Quaternion/T44 multiply:** The block outputs the upper input multiplied with the lower one. Suppose the upper input $T1$ fulfills $P_{\text{base}} = T1 \cdot P_{\text{flange}}$ and the lower one $P_{\text{flange}} = T2 \cdot P_{\text{TCP}}$, then the output $T3 = T1 * T2$ fulfills $P_{\text{base}} = T1 \cdot P_{\text{flange}} = T1 \cdot T2 \cdot P_{\text{TCP}} = T3 \cdot P_{\text{TCP}}$. Similar for quaternions.

3.4 Coordinate systems

It is important to keep in mind in which coordinate system a pose is given, otherwise the robot might act unpredictibly. There are normally 4 different coordinate frames: the base frame, the flange frame and the sensor frame. Figure 3 shows base and flange frames. Sensor and TCP frames can be defined manually corresponding to the used sensor or tool.

In the example Simulink controller, the coordinate frames are given and used in the following way (check!!):

- **Base frame:** Attached to the robot base, the only constant one, z-axis upwards.

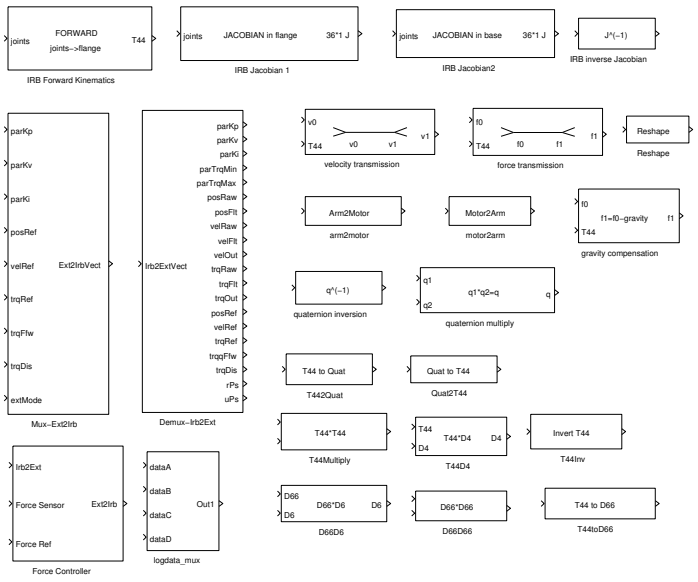


Figure 2 extctrl library

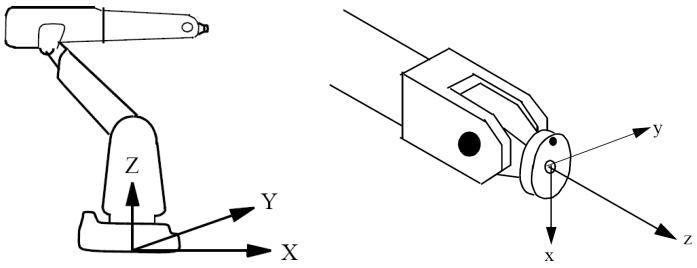


Figure 3 Coordinate systems: base (left) and flange (right)

- **Flange frame:** Attached to midpoint of flange, output of kinematics function blocks is transformation base to flange. $P_{\text{base}} = T44 \cdot P_{\text{flange}}$
- **TCP frame:** Transformation TCP to flange. $P_{\text{flange}} = T44 \cdot P_{\text{TCP}}$
- **Sensor frame:** Transformation flange to sensor. $P_{\text{sensor}} = T44 \cdot P_{\text{flange}}$

Example Figure 4 shows the part of the robot simulation model which calculates the force sensor output. It is a nice example for calculations between the different coordinate frames.

The surface normal vector, and hence also the force direction, is given in the base frame, but has to be transformed to the sensor frame. So the input to the velocity transmission block in the right of Fig. 4 should be the transformation matrix which fulfills $P_{\text{sensor}} = T44 \cdot P_{\text{base}}$.

Inputs in the subsystem are 3 different coordinate systems: the transmission

- from flange to TCP: $P_{\text{TCP}} = T1 \cdot P_{\text{flange}}$,
- from flange to sensor: $P_{\text{sensor}} = T2 \cdot P_{\text{flange}}$

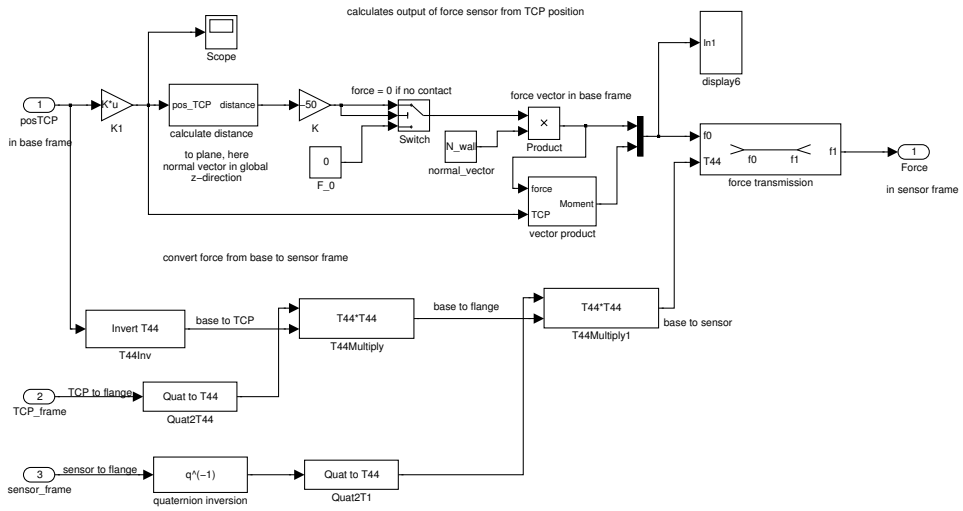


Figure 4 Part of the robot simulation model as example for coordinate system calculations

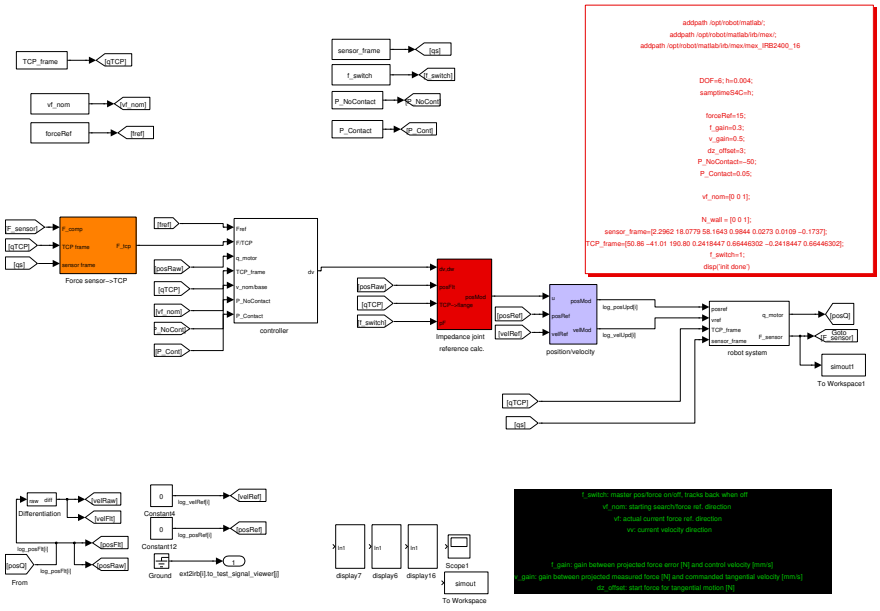


Figure 5 Example: a simple controller – simulation model

- and from base to TCP: $P_{\text{TCP}} = T3 \cdot P_{\text{base}}$.

The coordinate transformation is obtained multiplying $T2 \cdot T1^{-1} \cdot T3$. For used blocks see Fig. 4.

3.5 Controller example

Figures 5 and 6 show the example controller, once with a robot model for simulation and once the controller model for running experiments.

The controller block is the same for both Simulink models. A simple P controller is implemented. As soon as a force is measured, the robot moves in the direction given by the force vector until the absolute value of the force reaches

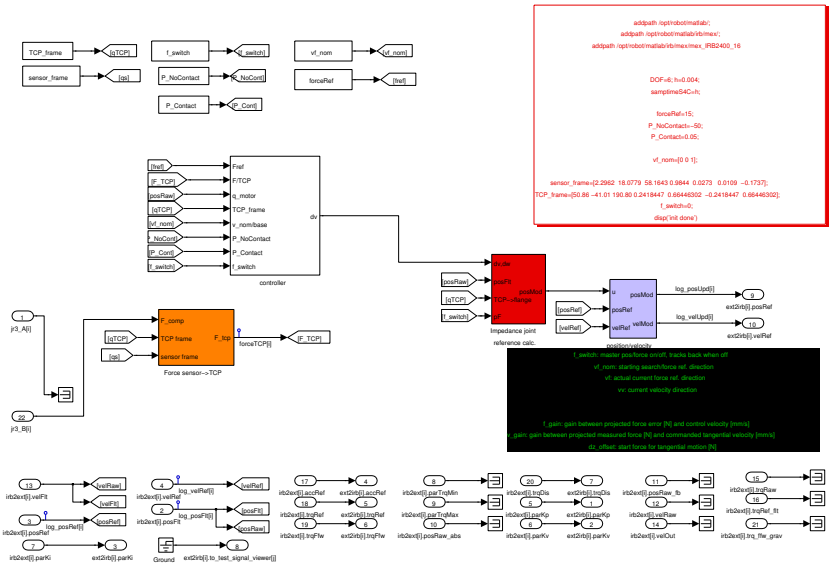


Figure 6 Example: a simple controller – model for control

a reference value. If not in contact, the robot moves in a given search direction. Output of the block is an incremental velocity change given in the TCP frame. In order to transform the global search direction to the TCP frame, the motor angles and TCP frame have to be inputs to the block.

The following block transforms the TCP velocity into joint velocities using the robot's Jacobian matrix.

The next block modifies the references sent to the robot. In this block, backtracking in case of controller deactivation is implemented. It is also convenient to implement other safety features here, e.g. a maximum reference change and what to do if this maximum is attained.

For the real controller, the changed references are sent to the robot, in the simulation model they are inputs to the robot model.

The model for the real controller has inputs/outputs. For these, an*. lc file has to be written (see Sec. 3.1). The names of vectors have to be written as name[i]. The same is valid for logged signals (see Sec. 5).

comment to run before controller, close reopen. Here pathes to the kinematics library are added and parameters like the TCP frame are defined. Should be available in the template file.

The force measurement is given in the sensor frame and has to be transformed to the TCP frame. In the block in front of that one, a constant offset is substracted from the force measurement.

Parameters that you want to change from opcom have to be added as inline parameters.

As safety features, the controller should always have a parameter to activate and deactivate the controller as well as a routine what should be done after deactivation. The controller states should then return to zero. It is also very convenient when working with extrapid to have a variable that measures if the states are returned to zero. It is also good to introduce a maximum possible modification of the robot position.

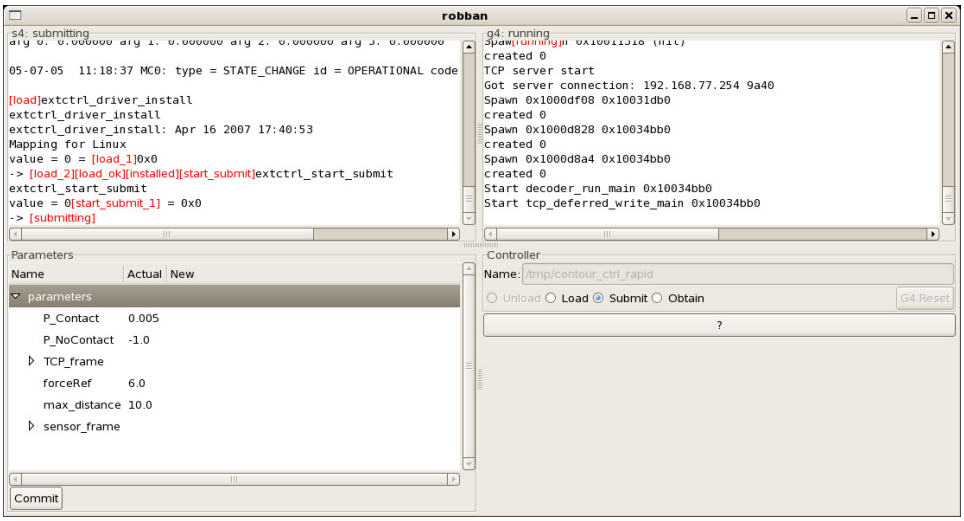


Figure 7 Opcom interface

Use preferably common Simulink blocks, not all might be translated in the right way or implemented.

4. Control interface and experiment

After the Simulink controller model has been built, the controller can be loaded and used with the opcom interface. It is started in the following way:

```
cd /home/robot/project/extctrl/opcom
./robban -v
```

For the IRC 5 cabinet which controls the Gantry-Tau robot, the opcom interface is started exchanging `robban` for `tekla` or `yoda` (in directory `yoda_m`).

The interface can be seen in Fig. 7, it consists of 4 parts. The upper left window is for communication with the S4 system and the JR3 force sensor, the upper right window for accessing files on the g4 Power PC in the ABB controller cabinet. In the lower left area the inline controller parameters can be read and changed and in the lower right area the controller can be loaded.

Backspace does not work, use CTRL-h instead, even if it appears not to work.

4.1 S4-Terminal and force sensor

This window is mostly for communication with the two force sensors, which are connected through channel 0 and 1. The following commands can be executed:

- **extctrl_jr3_set_filter 1,3**
  Change filter for force sensor signal: in example filter for channel 1 (0 or 1) is changed to filter 3 (0–6, cutting frequency is cut in half with increasing number, mostly set to 0)
- **extctrl_jr3_get_filter 1**
  Read filter in channel 1

- **jr3_init**
  Resets force sensor in current position; not to be done in "submit" or "obtain"!
- **jr3_test_read_force 1**
  Read force for channel 1

4.2 G-4 Terminal

Here communication with the G4-power-PC is done. Its harddisk can be accessed.

4.3 Loading of the controller

There exist 4 different states in the process of loading a controller:

- **unload**
  Basic state active when interface is loaded, no communication.
- **load**
  After having entered the controller name in field above activate "load". Controller is saved by RTW compiler in directory "/tmp/".
- **submit**
  Communication from S4 to interface active, parameter values can be read, but nothing sent back to controller/axes controller. Try your controller first in this state, it doesn't affect the robot but you can look at the logged signals and see if it's behaving as it should.
- **obtain**
  Communication is active in both directions, parameter values can be changed and controller run. Modified references are sent to axes computers.

4.4 Parameters

Parameters that have been defined as inline parameters in Simulink can be read and changed here. Click on parameter, change value in right field, parameter will first be sent to controller when clicking on commit.

4.5 Carrying out an experiment

- **Model controller in Simulink:** test with simulation model, add parameters, write *.lc files, build
- **Open opcom**

  ```
  cd /home/robot/project/extctrl/opcom
  ./robban -v
  ```

- **Load controller and submit**
- **Test logged signals without running the robot**, force sensor signals are read, so you can simulate environment interaction by pressing TCP with a stick
- **Obtain**, if previous experiment gave expected result
- **Loosen brakes and hold dead man's switch**

- **Activate controller**
- **Be ready to push emergency stop**

5. Data logging

Independently from the control execution, another program can be run to log signals. All input and output signals of the Simulink model (irb2ext or ext2irb) as well as specified signals are logged. To specify that a signal should be logged, click on the signal line and choose *Signal Properties* in the context menu. Activate *Test point* in the first tab *Logging and accessibility* and choose a signal name. If the signal is an array, the name has to end with "[i]" (really "i" and not the signal dimension!), e.g. `testsignal[i]`.

To start the program "log" located in Anders Blomdell's home directory, type

```
cd /home/andersb/work/robot/log
./log -t 10 -d 2 --host prpmc1.m --port 2000 -o /tmp/logA
```

The right host for a specific robot controller can be found in the *.fsm file of the controller's opcom interface as the value of `G4_IP_ADDRESS`.

This will log signals specified in the Simulink model during 10 s (`-t 10`). The `-d 2` command specified that every second sample is logged. The sampling time is 4 ms and the resulting file can get large easily. The logged data is saved in the file "/tmp/logA". To convert the logged data in a format readable by Matlab, run the following program:

```
./log2matlab /tmp/logA > /tmp/logA.txt
```

To read data in Matlab from the data file the program "readlog" can be used. It can be found in

```
/home/tolsson/research/flexaa/newsys_test/readlog.m
```

The program is used in the following way:

```
[data,names]=
   readlog('/tmp/logA.txt', {'log_forceComp','posRef'});
plot(data)
```

The program loads all signals that include the specified strings 'log_forceComp' or 'ext2irb.posRef' in their names from the specified file "/tmp/logA.txt" and saves each signal in a column of the matrix `data`. The struct `names` contains the signal names in the same order as they appear in the matrix.

6. Using EXTRAPID

It is even possible to use a customized Simulink model controller together with a exetended RAPID program. Besides usual RAPID commands, an EXTRAPID program can contain commands for setting or comparing parame-

ters in the Simulink model. With the aid of these parameters, the Simulink controller can be activated and deactivated and the RAPID program flow controlled.

6.1 Example program

An example for such an EXTRAPID program is the following:

```
MODULE EXAMPLE

  PROC main()

    ! This should always be done to ensure that the controller
      is initially in a safe state!!!
    FORCE
     FORCESET f_activate := 0;
     FORCEWAITUNTIL trackedBack <= 0.0001;
     WaitTime \InPos, 1.0;
    ENDFORCE

    MoveL [[106.58,801.18,1178.86],[0.706217,0.008091,-0.707823,
     -0.01267],[0,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
     ,v10,z0,tool0;
    FORCE
     FORCESET vX := 0;
     FORCESET vY := 0;
     FORCESET vZ := 1;
     FORCESET signvX := 2;
     FORCESET signvY := 2;
     FORCESET f_activate := 1;
     FORCEWAITUNTIL forceOut >= 5;
     FORCESET param1 := 1;
     FORCESET f_activate := 0;
     WaitTime \InPos, 3.0;
    ENDFORCE
  ENDPROC
ENDMODULE
```

The first command block is a very important safety feature. If an experiment is aborted while the Simulink controller is active, the controller states might continue to grow instead of decreasing to 0. One possibility to deactivate the controller and check its states is to use the manual communication programs, which will be discussed later.

If this is forgotten the robot may do an unpredictible jump when the emergency brake is released while the opcom interface is still in "obtain" mode and the controller activated.

To prevent that, this block should be inserted in any EXTRAPID program. The variable `trackedBack` is the sum of the controller states' absolute values, and the program only continues when it is sufficiently close to 0 after deactivating the controller. The opcom interface may only be switched from "submit" to "obtain" mode, when the robot starts to move, which shows that the controller is in a safe state.

In the second part, the robot is commanded to move to a certain point, where the Simulink controller is activated. The controller will move the robot in a specified direction until a contact force is measured and then control the contact force.

Before the controller is activated, different controller parameters are set.

6.2 Safety

When using EXTRAPID, 3 systems are interacting: the ABB controller, the Simulink controller and the EXTRAPID program. This complicates the manipulation and

6.3 How to make and run an Extrapid program

The EXTRAPID files (2007-08-14) are located in `/work/robot`. An example program can be found in `/work/robot/extrapid-build/example.prg`.

To run an EXTRAPID program type the following:

```
source /work/robot/extrapid/environment
cd /work/robot/extrapid-build
```

Then copy your program file to `/work/robot/extrapid-build/.` and run it with

```
java Execute example.prg
```

But before you can run your EXTRAPID program together with a Simulink controller, you have to create an interface between EXTRAPID and controller:

```
source /work/robot/extrapid/environment
cd /work/robot/extrapid/src/s4cplus/extcomm
```

Do the following things:

- Create a directory for the new controller (use `drillcomm` as an example).
- Change the names from `drill_xx` to fit the new controller.
- Change the lc-file to contain the parameters of the Simulink controller (a duplicate may be in the Simulink directory).

For the Makefile,

- compile the lc-file using Anders Blomdells labcomm-compiler
- update drill_in.c and drill_out.c to handle the parameters
- compile c-files
- put the binary file so that EXTRAPID can access it.

To build and install the rap interface towards S4C+ type:

```
source /work/robot/extrapid/src/
cd /work/robot/extrapid/src/s4cplus/rapcomm
make
```

To build EXTRAPID:

```
source /work/robot/extrapid/src/
cd /work/robot/extrapid/src/lang/extrapid\_20070318\_0918
ant
```

Do not add any files to `/work/robot/extrapid-build`, it is an automatically generated directory.

To run a new controller you need to make changes in the class `ExtCtrlComm` under `/work/robot/extrapid/src/lang/extrapid_20070318_0918` and compile extrapid (ant).

Corresponding lines in `ExtCtrlComm.java`:

```
String[] drill\_in = {"drill\_in","prpmc1.m","2000"};
String[] drill\_out = {"drill\_out","prpmc1.m","2000"};
```

`drill_in` and `drill_out` handle in and out parameters and can be run manually from a terminal

At the moment, EXTRAPID cannot handle negative parameters or vectors.

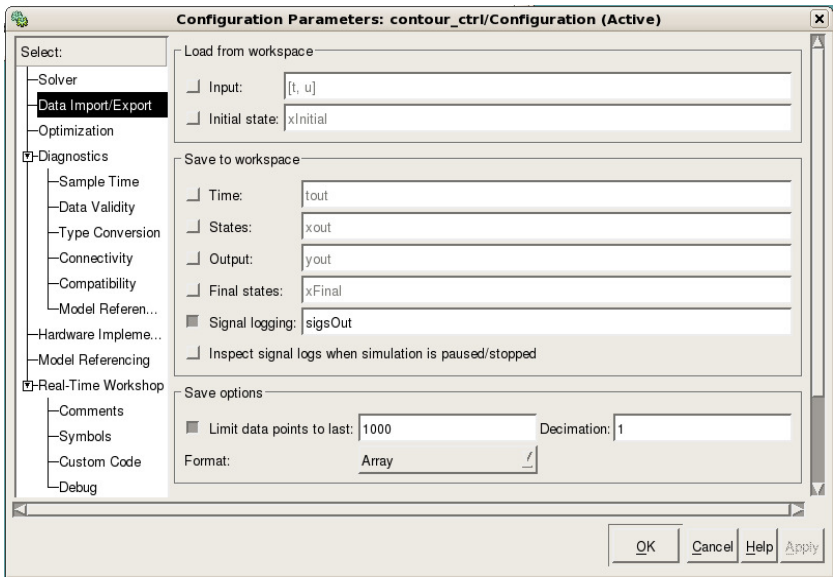
To run an EXTRAPID program, the abb controller has to be in automatic state.

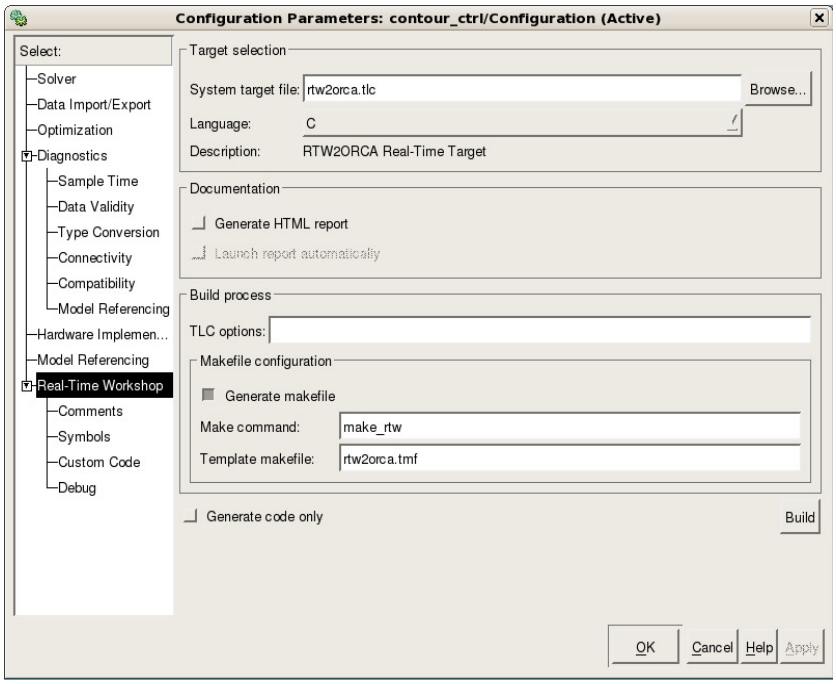
6.4 Safety

As the ABB controller has to be in automatic state to run EXTRAPID, special attention has to be paid to safety issues. In the automatic state, All points cited above should be followed and

Run the programs for in and out parameters manually to check if the controller is working properly.

A. Real-Time Workshop Options





B. Accessing harddisk on robot server

- `ncftp -uu -pp robban.m`
- `lcd robot/control/example/`
- `! ls`
- `get VOLVO070618A.prg`
- `quit`

List of all possible commands with `help`

C. How to change system on ABB controller

- backup old system (/home/robot/project/SMErobot/backup_tekla/ for IRC 5), use ncftp
- 2 files important: mc/mc.sym (compiled system, in folder bin of robot system) and install.cmd (points to which robot to load, in folder robots of robot system), actually used file only copy of one of group of several possible files, choose right file, check that right robot is pointed to
- power down on control cabinet, now cables can be exchanged if another robot should be connected to the controller cabinet
- power up
- restart - advanced - X-start - choose system that should be booted
- eventually additional I-start to erase all old information

[1] R.M. Murray, Z. Li and S.S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC Press, Boca Raton, FL, 1994.