BlueBotics
Mobile Robots at your Service

Parc Scientifique PSE-C
1015 Lausanne EPFL
Switzerland

Tel +41 21 693 83 14
Fax +41 21 693 83 15
www.bluebotics.com

The WHEELCHAIR Platform

User Guide / REV2359

Table Of Contents

Chapter
1

Introduction

Brief description of the robot and its functionality

1.1 The WHEELCHAIR Platform

The WHEELCHAIR platform is the results of three years of experience gained by BlueBotics during the MOVEMENT project[1]. It is shortly described here with respect to mechanics, electronics and software.

1.1.1 Mechanics

BlueBotics integrates as many standard components as possible from the motorized wheelchair market. Then, the long experience of BlueBotics in building mobile robots has then been used for designing all the structure of the robot resulting in a compact system, which is very robust and powerful.

1.1.2 Electronics

When used for research, the electronics of the robot has not just to be industry proven, but also to be as flexible as possible in order to allow further extension of the presented robot since the robot remains a development platform within the project.

During the MOVEMENT project, BlueBotics has developed an electronics, which is compatible with the former products, but allow also making autonomous this wheelchair as well as many industrial AGVs[2]. This ensures reliability and extensibility accompanied by knowledge about their characteristics and performances.

1.1.3 Software

The reliability and extensibility of the hardware has to be accessible to the end-user. For this, on the top of the hardware a hard real-time operating system has been adopted for all the time critical tasks concerning the movement of the robot.

To ensure flexibility, the functions controlled by this operating system can be accessed by any other system either via HTTP/CGI calls.

[1] Modular Versatile Mobility Enhancement Technology, Specific Targeted Research Project nr. 511670
[2] Automated Guided Vehicles

1.2 Important Disclaimer

The delivered platform remains a system for laboratory use.

The platform should not be used without supervision of a competent person.

1.3 Robot Overview

Figure 1 shows the basic design of the base. The drive wheels are in the middle, while there are two main castor wheels in the rear and two castor wheels on springs in the front. The motor amplifiers are under the wireless communication. In the front there is a SICK LMS-200. The CompactPCI rack is placed over the batteries and fixed to the robot frame.

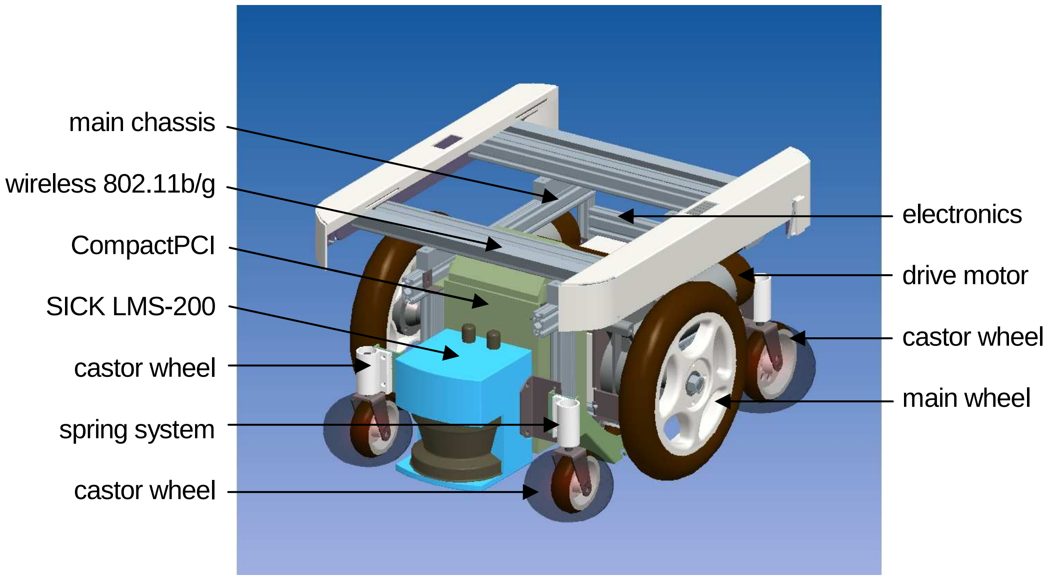

Figure 1: The robotic part with all the components and the chassis. The main components are: Two drive wheels with motors and amplifiers, four castor wheels (two on springs), two batteries, two main electronic prints, one CompactPCI rack and an optional SICK laser scanner.

1.4 Specification

The specification is inspired by the *DIN EN 12184 (ISO 7176) Class A and B standards*:

• Payload[3]:	150kg
• Dynamic stability:	6° (10.5%)[4]
• Climbing ability:	6° (10.5%)[4]
• Static stability:	9° (15.8%)[4]
• Parking brake:	9° (15.8%)[4]
• Max speed:	5km/h
• Max acceleration:	1.35m/s^2
• Minimal clearance:	40mm
• Obstacle ability[5]:	35mm
• Maximal turning radius:	1000mm
• Maximal turning area:	1300mm
• Autonomy:	10km
• Length[6]:	670mm
• Width[6]:	560mm
• Horizontal braking distance:	0.8m from max speed[7]
• Noise:	65dB(A)[8]

[3] Standard for DIN EN 12184 testing and Otto Bock A200 are 100kg
[4] DIN EN 12184 Class B
[5] DIN EN 12184 Class A is 15mm, Class B is 50mm
[6] Without cover
[7] DIN EN 12184
[8] DIN EN 12184 Class A

Chapter
2

Basic Use

Main switch, charging, power supply

2.1 Main Switch

The WHEELCHAIR platform's main switch (on/off) is on the left of the platform. Down is off, up is on.

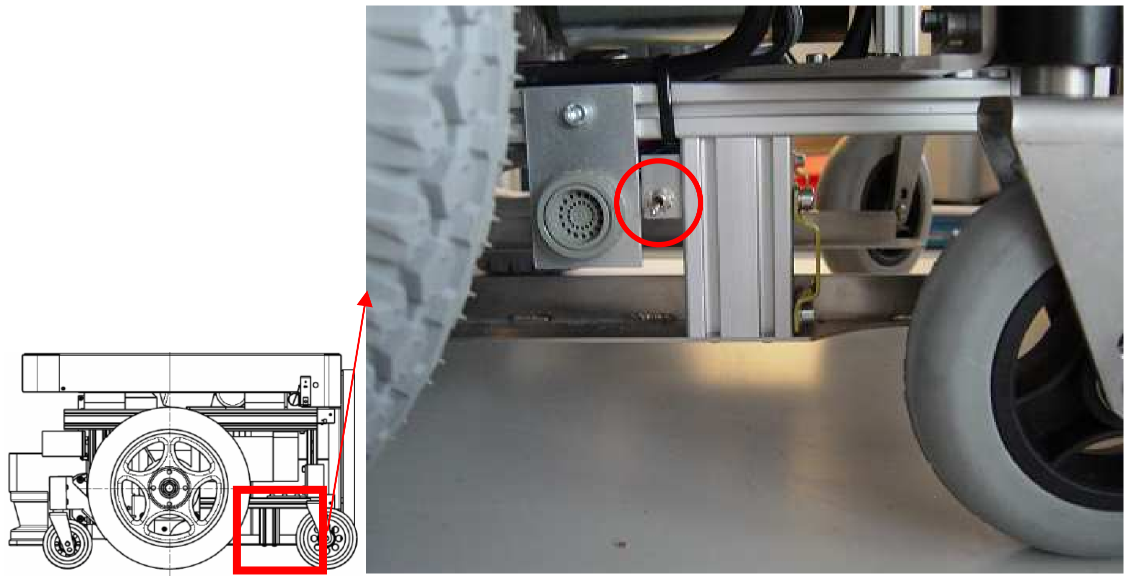

Figure 2: Main switch on the WHEELCHAIR platform.

2.2 Charging

This section briefly describes how to charge the robot's batteries.

2.2.1 Material

Charger: Curtis charger, black box for charging the batteries.

2.2.2 To Charge the Robot

1 - Switch off the robot (see Figure 2).

2 - Connect the charger to the plug on the robot (Figure 3).

3 - Turn on the charger.

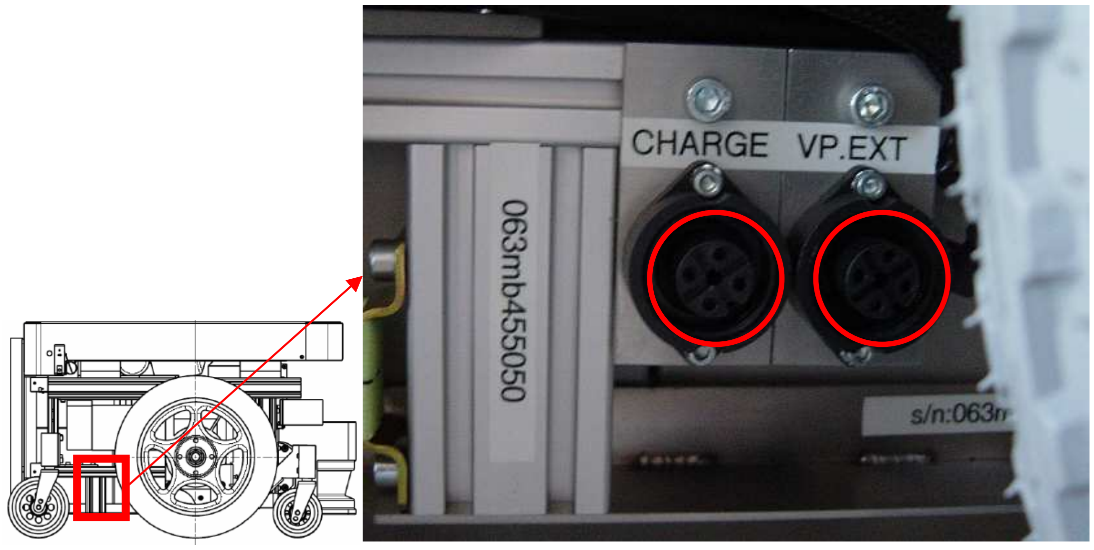

Figure 3: Connector for charging. External 24V power supply.

Note that you can also work without batteries by connecting an external 24V power supply. In this case the platform will not be allowed to move.

2.3 Communication with the Firmware

If you have to configure the firmware of the PowerPC card (see 3.1.3 Configuration of Firmware on page11), you have to connect to the serial port in the front left of the rack.

2.4 Modeling of the Laser Bumper

The WHEELCHAIR platform implements a virtual bumper based on the laser scanner data. Three areas are defined:

- Warning area (orange line) – If the laser detects a point in this area, it informs the safety system, which turns the flash on.
- Emergency area (red line) – Points detected in the emergency area cause an emergency message to the safety systems, which stops the platform.
- Exclude area (green line) – The exclude area are permits to define an exception to the emergency area. More concretely, in the current setup of the WHEELCHAIR platform, the front castor wheel can be in the view of the laser scanner (when the platform moves backward). The area where the castors can be seen by the laser scanner is excluded from the emergency area (but not from the warning area).

The configuration of these areas for the WHEELCHAIR platform is shown in Figure 4

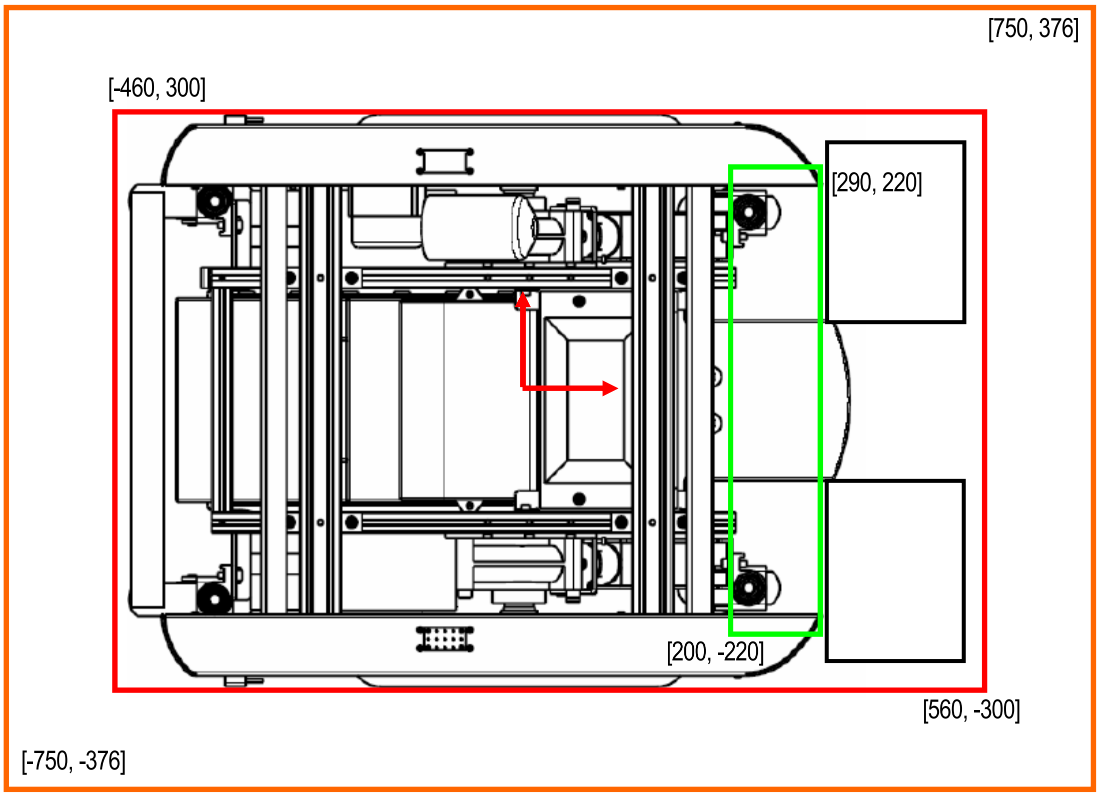

Figure 4: Configuration of the warning, emergency and exclude areas for the laser bumper.

The configuration of the bumper can be done at boot time using the following procedure:

- Create an ASCII file named *LaserBumper.conf* containing the following command: `LaserBumper.DefineAreas BUMPER wXmin wXmax wYmin wYmax emXmin emXmax emYmin emYmax exXmin exXmax exYmin ~` where `w`, `em` and `ex` correspond respectively to the Warning, Emergency and Exclude areas. `Xmin, Xmax, Ymin, Ymax` define the coordinates of the rectangular areas.

- Copy the *LaserBumper.conf* file in the directory of the TFTP server defined as the "default host" (see section 3.3). The file will be loaded at boot time and used to configure the bumper.

Here's an example of a (fictive) LaserBumper.conf configuration file:

```
LaserBumper.DefineAreas BUMPER -0.49 0.49 -0.376 0.376 -0.36 0.36
-0.276 0.276 0.20 0.29 -0.22 0.22 ~
```

2.5 Shape Modeling for Obstacle Avoidance

If you are using ANT®[9] for navigation, you can run the obstacle avoidance (ANT® – motion) based on the SICK laser scanner. This approach uses a multi-layered dynamic path planning and collision avoidance based on physically meaningful representations (actuator speeds and accelerations, robot geometry in work-space). To do that, the system has to know the shape of the robot. This is described here.

The shape configuration for obstacle avoidance is stored in the file *Peripherals.conf*. In case you want to use ANT® – motion on the platform with a docked module, contact BlueBotics.

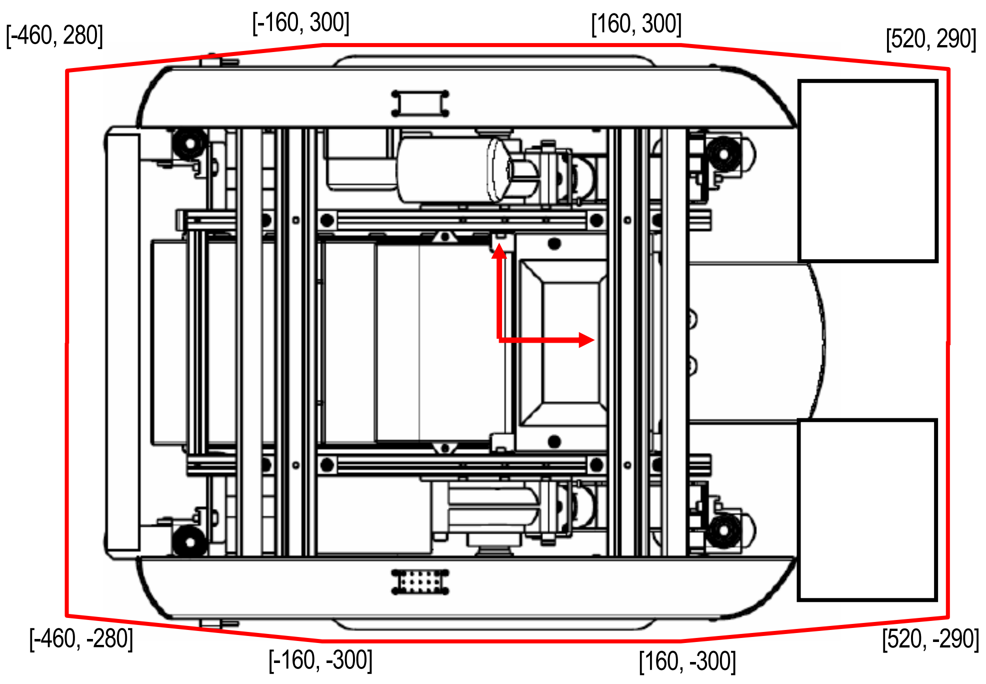

Figure 5: Shape modeling of the platform for obstacle avoidance.

The configuration command in *Peripherals.conf* is then:

```
Common.PolygonAddPoints HULL 0.520 -0.290 0.160 -0.300 -0.160
-0.300 -0.460 -0.280 -0.460 0.280 -0.160 0.300 0.160 0.300 0.520
0.290 ~
```

Changing the position of the footrests will affect these parameters. Don't do that in the autonomous mode!

The hull can be customized at boot time by providing a file named *CustomHull.conf* on the default host's TFTP server, with the following content:

```
Common.PolygonSetPoints HULL x0 y0 x1 y1 x2 y2 ... ~
```

Where (x0, y0), (x1, y1), … are the points composing the hull as shown above. For example, the following file could be used for a docked module (note the ~ at the end of the point list):

```
Common.PolygonSetPoints HULL
   0.52 0.25
```

[9] ANT® is the navigation product of BlueBotics SA, Lausanne. ANT® stands for Autonomous Navigation Technology and is a registered trade mark of BlueBotics.

```
    0.276 0.34
    -0.53 0.34
    -0.58 0.29
    -0.58 -0.29
    -0.53 -0.34
    0.276 -0.34
    0.52 -0.25
~
```

2.6 Map for Localization

When using the localization system (ANT® – Localization) you have to provide a model of the environment. This model is called "map". Note that for the moment:

ANT® – Localization is only active when you give mission commands to the robot (see section 4.1).

In the future the localization will be completely stand-alone. This means that you will only have two modes (i.e. speed control or position control with obstacle avoidance) and you will turn on/off localization separately.

Chapter
3

Configuration of the Hardware

How to make the robot work

3.1 The Inova Firmware (PowerPC BIOS)

3.1.1 Bootfile

The bootfile is a file, which is automatically downloaded by starting the robot. The robot searches for that file via FTP on a specified server.

3.1.2 Server & Host IP

The robot is directly accessible on the network via radio Ethernet. It therefore requires an IP address. Furthermore, as explained in the last section, it needs a FTP server to start-up. The IP of the server has also to be specified.

3.1.3 Configuration of Firmware

By connecting a serial cable (null-modem) from the back of the rack (see 2.3 Communication with the Firmware on page 7) to a standard terminal (bps: 9600, data bits: 8, parity: none, stop bits: 1, flow control: none) the firmware can be configured. Turn on the system and press a key during booting.

```
Press any key to stop auto-boot...
```

This will lead to a prompt.

```
[VxWorks Boot]:
```

The *"c"* command allows then configuring the following options. The "host name" should **never** contain a dot (".").

```
'.' = clear field;  '-' = go to previous field;  ^D = quit

boot device          : dc0
processor number     : 0
host name            : host
file name            : xo2r106mov
inet on ethernet (e) : 192.168.8.100:ffffff00
inet on backplane (b):
host inet (h)        : 192.168.8.25
gateway inet (g)     : 192.168.8.1
user (u)             : ftpUser
ftp password (pw) (blank = use rsh): ftpPassword
flags (f)            : 0x0
target name (tn)     : robotName
startup script (s)   :
other (o)            :
```

3.2 Configuring the Netgear Station Adapter

You have to configure the station adapter to access your own network. For this, you have to connect with a web browser on the station adapter via the Ethernet cable on the platform (remember that you have to be on the same subnet). Then you log in (user: admin; pass: none), and finally, you choose your Wireless Network Name (SSID).

3.3 The XO/2 Real-Time Operating System

The XO/2 operating system has a configuration facility, which is embedded in the *Preferences* module. It is preferable not to play with this module, but if changes in the network configuration (DNS, SMTP server, mail address, search domain, etc.) are required, this module allows for simple configuration by means of the following commands (via telnet):

- Preferences.Show – Allows seeing the current configuration. Use this before changing the parameters!
- Preferences.SetBoardName – Configures the name of the PowerPC board.
- Preferences.SetBootfileName – Not working with this board (see section 3.1.1).
- Preferences.SetClockDST – Sets the *daylight saving time*.
- Preferences.SetClockDiff – Defines the time zone.
- Preferences.SetDefaultHost – This is an important command, which defines from which IP address the robot will search for compiled modules if they are not already on the robot, when the dynamic loading takes place. It also defines where the robot will look for its initial map file. You have to set this option to your computer when you are developing.
- Preferences.SetNetworkParameters – Not working with this board (see section 3.1.2).
- Preferences.SetPrimaryDNS – Configures the primary DNS for your network.
- Preferences.SetSMTPServer – Defines the SMTP server the robot will use to send emails.
- Preferences.SetSearchDomain – Sets the search domain for you network (i.e. epfl.ch)
- Preferences.SetSecondaryDNS – Configures the secondary DNS.
- Preferences.SetServerAddress – Not working with this board (see section 3.1.2).
- Preferences.SetTimeSever – Defines the time server within your network.
- Preferences.SetUserMailAddress – Defines the email address the robot will use when writing emails. This option can be used to recognize from where the email is coming or to have a reply to the responsible of the robot.
- Preferences.SetUserName – Sets the name of the user, which is then used for login.
- Preferences.SetUserPassword – Defines the password, which is required for login.

3.4 Loading the Map from a TFTP Server

In order to allow you changing the map in every moment, the text file describing it is not in the bootfile. The *Init.map* file is directly loaded from a TFTP server defined as the "default host" (see section 3.3).

Note that if your computer is defined as "default host" you don't need to log in with user and password via telnet before using the robot.

3.5 Step by Step Configuration

To summarize you have to follow these steps:

0. Configure the Netgear station adapter to access your Wireless Network Name (SSID). See section 3.2.
1. Connect the serial cable. See section 2.3.
2. Connect with a serial terminal (bps: 9600, data bits: 8, parity: none, stop bits: 1, flow control: none). See section 3.1.3.
3. Turn on the robot and press a key. See section 3.1.3.
4. Configure the robot name (target name) and address (inet on ethernet) and the FTP server name (host name), address (host inet), user name (user) and password (ftp password). You may also need to configure the gateway (gateway inet) and bootfile name (file name). See section 3.1.3.
5. Put the *bootfile* in the FTP directory and the *Init.map* in the TFTP directory of the server. See section 3.4.
6. Restart the platform. The operating system should start. If not, recheck points 4 and 5.
7. Finally, you have to configure the Default Host to your TFTP server: Preferences.SetDefaultHost "IP_adress". See section 3.3.
8. Enjoy…

Chapter

4

Basic Software

Robot drivers, controllers and interfacing

4.1 Introduction

As explained in the design of the platform[10] the software is deployed on two levels:

- On the top of the hardware a deadline driven hard real-time operating system operates all the time critical tasks accessing the robot basic hardware.
- Another level of abstraction is added for flexibility: All functions controlled by the embedded real-time operating system can be accessed by any other system via HTTP/CGI calls.

4.2 HTTP/CGI Communication

Note that for the WHEELCHAIR platforms, we have adopted an 802.11b/g wireless system. We therefore expect a better response time with compared to the old 802.11 BreezeCom AP.40. The response time with the BreezeCom AP.40 wireless module is shown in Figure 6. For low data requests (~ 20 bytes) the response time via the 4 port hub is below 10 ms. For bigger ones like the data from the SICK laser scanner (~15 kB) the time is below 40 ms. However, when the Windows 2000 scheduler interrupts the program, peaks can reach up to 150 ms even for the low data requests. Requests via wireless (not cable on its hub) have a response time of less than 30 ms for low data and less than 110 ms for the scan.

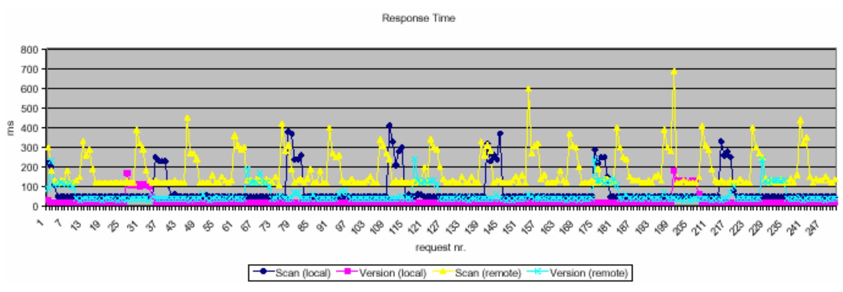

Figure 6: Access of resource into the PowerPC via HTTP/CGI

[10] D6.1 - Multi purpose mobile platform design, UPDATE (a), 15.11.2005

4.3 CGIs with XO/2

The CGIs calls to a machine running XO/2 use the syntax:

http://IPaddress/cgi/ModuleName.CGIName?firstParameter+secondParameter+…+nthParamanter

For the WHEELCHAIR platform a dedicated module accessing all the needed resources has been implemented. All the CGIs are therefore accessed by the *PPCtoPC* module.

Being this module a standard XO/2 module, the CGIs can also be used as standard XO/2 commands via telnet with the syntax:

ModuleName.CGIName firstParameter secondParameter … nthParamanter

When a call returns one ore more values, each value is returned on a separate line.

The status code returned by several calls is a hierarchically-structured string that gives information about the execution status of the call. Read from left to right, each component gives more information. For example:

Ok	Command execution successful
Error.Parameter.x	Parameter parse error on parameter "x"
Warning.SecurityNotRunning	Safety controller not running

4.4 Sensors – XO/2 System

4.4.1 Odometry

The odometry is updated at 100Hz.

odometryGet – Returns the robot position in [m, m, rad].

Syntax:	*PPCtoPC.odometryGet*
Returns:	x, y, theta

odometrySet – Sets the robot position in [m, m, rad]. The syntax is:

Syntax:	*PPCtoPC.odometrySet?x+y+theta*
Returns:	status
Status codes:	Ok
	Error.Parameter.*

4.4.2 Laser Scanner

The laser scanner works at 37Hz, the timestamp is set as the meantime of a scan meaning that the maximum error is approximately 15ms.

lsGetCartesian – Returns the data from the laser scanner in ASCII format with the observation timestamp [ms], the robot position at observation time [mm, mm, mrad], the number of measurements, and all the (x, y) measurements [mm].

Syntax:	*PPCtoPC.lsGetCartesian*
Returns:	timestamp [ms]
	x_r, y_r, θ_r [mm, mm, mrad]
	n_measurements
	x_n, y_n for n = 1 .. n_measurement [mm, mm]

lsGetPolar – NOT WORKING IN THIS REV – Returns the data from the laser scanner in ASCII format with the observation timestamp [ms], the robot position at observation time [mm, mm, mrad], the number of measurements, and all the (phi, rho) measurements [mrad, mm].

Syntax:	*PPCtoPC.lsGetPolar*
Returns:	timestamp [ms]
	x_r, y_r, θ_r [mm, mm, mrad]
	n_measurements
	θ_n, ρ_n for n = 1 .. n_measurement [mrad, mm]

lsGetPolarIntens – NOT WORKING IN THIS REV – Returns the data from the laser scanner in ASCII format with the observation timestamp [ms], the robot position at observation time [mm, mm, mrad], the number of measurements, and all the (phi, rho, int) measurements [mrad, mm, SICKsignalintensity].

Syntax:	*PPCtoPC.lsGetPolarIntens*
Returns:	timestamp [ms]
	x_r, y_r, θ_r [mm, mm, mrad]
	n_measurements
	θ_n, ρ_n, int for n = 1 .. n_measurement [mrad, mm, SICKsignalintensity]

lsGetRawData – Returns the raw data from the laser scanner in the sensor frame (not robot frame like all the other. The ASCII format is: observation timestamp [ms], robot position at observation time [mm, mm, mrad], number of measurements, and all the rho measurements [mm]. The angles in sensor frame are directly given by 180°/number of measurements.

Syntax:	*PPCtoPC.lsGetRawData*
Returns:	timestamp [ms]
	x_r, y_r, θ_r [mm, mm, mrad]
	n_measurements
	ρ_n for n = 1 .. n_measurement [mm]

4.5 Speed Controller – XO/2 System

The speed controller runs at 100Hz.

speedGetSpeed – Returns the platform speed (translation and rotation).

Syntax:	*PPCtoPC.speedGetSpeed*
Returns:	xDot [m/s]
	thetaDot [rad/s]

speedSetSpeed – Sets the speed of the platform (translation and rotation) in [m/s] and [rad/s].

Syntax:	*PPCtoPC.speedSetSpeed?xDot+thetaDot*
Returns:	status
Status codes:	Ok
	Error.Parameter.*

speedGetWheelSpeeds – Returns the right and left wheel speed in [rad/s].

Syntax:	*PPCtoPC.speedGetWheelSpeeds*
Returns:	speedR, speedL [rad/s]

speedSetWheelSpeeds – Sets the speed of the left and right wheel speed in [rad/s].

Syntax:	*PPCtoPC.speedSetWheelSpeeds?speedR+speedL*
Returns:	status
Status codes:	Ok
	Error.Parameter.*

speedTrapezTranslate – Perform a pure translation motion of the given distance, with a trapezoidal speed profile. Distance is the distance to be traveled in [m] and must be positive. Speed is the maximum speed of the motion profile in [m/s], and must be positive for forward motion and negative for backward motion. Acceleration is the acceleration at the beginning of the motion and the deceleration at the end in [m/s]. THIS CALL ONLY WORKS RELIABLY IN SPEED CONTROL MODE (see securitySetBumpMode).

Syntax:	*PPCtoPC.speedTrapezTranslate?distance+speed+acceleration*
Returns:	status
Status codes:	Ok
	Error.Parameter.*

speedTrapezRotate – Perform a pure rotation motion of the given angle, with a trapezoidal speed profile. Angle is the angle to be traveled in [rad] and must be positive. Speed is the maximum angular speed of the motion profile in [rad/s], and must be positive for counter-clockwise motion and negative for clockwise motion. Acceleration is the angular acceleration at the beginning of the motion and the deceleration at the end in [rad/s]. THIS CALL ONLY WORKS RELIABLY IN SPEED CONTROL MODE (see securitySetBumpMode).

Syntax:	*PPCtoPC.speedTrapezRotate?angle+speed+acceleration*
Returns:	status
Status codes:	Ok
	Error.Parameter.*

speedXchgJoystick – Exchange speed commands and override flag with joystick commands. xDot and thetaDot are the requested platform speeds (translation and rotation) in [m/s] and [rad/s]. If override is 0, the joystick is used for speed commands. If override is 1, xDot and thetaDot are used for speed commands, therefore overriding the joystick. The call returns the current joystick position.

Syntax:	*PPCtoPC.speedXchgJoystick?xDot+thetaDot+override*
Returns:	joystickX [%]
	joystickY [%]

4.6 Position Controller – XO/2 System

The position controller runs at 10Hz.

positionAbort – Aborts the current GoTo command.

Syntax:	*PPCtoPC.positionAbort*	
Returns:	status	
Status codes:	Ok	

positionGetState – Returns the internal state of the position controller in ASCII format.

Syntax:	*PPCtoPC.positionGetState*	
Returns:	state	
State codes:	Warning.SecurityNotRunning	Safety controller is not running
	Ok.Running	Robot is moving
	Ok.Ready	Robot is ready for the next command

positionGetStateInfo – Given a state code, this CGI returns a string explaining the meaning of the value.

Syntax:	*PPCtoPC.positionGetStateInfor?"state"*	
Returns:	stateInfo [string]	
Status codes:	Ok	

positionGoBackward – After executing this command, the robot will move around in backward direction. !!! **WARNING**: The robot has no sensors in the back !!!

Syntax:	*PPCtoPC.positionGoBackward*	
Returns:	status	
Status codes:	Ok	

positionGoForward – After executing this command, the robot will move around in forward direction. This is the default direction.

Syntax:	*PPCtoPC.positionGoForward*	
Returns:	status	
Status codes:	Ok	

positionGoTo – Defines the (x, y, theta) goal position in world coordinates that the robot has to reach.

Syntax:	*PPCtoPC.positionGoTo?x+y+theta*	
Returns:	status	
State codes:	Ok	
	Error.Parameter.*	
	Error.TooFar	Goal position is too far (distance >10m)

positionTurn – Defines the heading goal position in world coordinates that the robot has to reach.

Syntax:	*PPCtoPC.positionTurn?theta*	
Returns:	status	
State codes:	Ok	
	Error.Parameter.*	

4.7 Safety Controller – XO/2 System

The safety controller guarantees that the speed controller and eventually the position controller are always alive. Furthermore, it takes into account the bumpers by setting the speed to (0, 0) and disabling the motor amplifiers when a collision occurs.

securityGetState –This CGI return the state, which allow understanding the current state of the safety controller.

Syntax:	*PPCtoPC.securityGetState*	
Returns:	status	
Status codes:	Ok	
	Warning.SecurityNotRunning	Safety controller is not running
	Warning.VoltageLow	Battery voltage is getting low
	Warning.HotProcessor	Processor temperature is too high
	Emergency.SpeedCtrlHalted	Speed controller is halted
	Emergency.ObstAvoidanceHalted	Obstacle avoidance is halted
	Emergency.VoltageLow	Battery voltage is critically low

securityGetStateInfo – Given a state value, this CGI returns a string explaining the meaning of the value.

Syntax:	*PPCtoPC.securityGetStateInfor?"state"*
Returns:	stateInfo [string]
Status codes:	Ok

securitySetAvoidMode – Allows activating the obstacle avoidance (with the SICK laser scanner) as position controller. This CGI can be used only when the safety controller is stopped.

Syntax:	*PPCtoPC.securitySetAvoidMode*	
Returns:	status	
Status codes:	Ok	
	Error	Mode could not be activated

securitySetBumpMode – Allows stopping the obstacle avoidance to use the speed controller directly. The laser scanner is used as bumper only. This CGI can be used only when the safety controller is stopped.

Syntax:	*PPCtoPC.securitySetBumpMode*	
Returns:	status	
Status codes:	Ok	
	Error	Mode could not be activated

securitySetJoyMode – Stops the obstacle avoidance and puts the platform into joystick-only control mode. In this mode, traffic is relayed bidirectionally between the motor controller and a handcontrol connected to the CAN bus of the docking module. The laser scanner is used as bumper only. This CGI can be used only when the safety controller is stopped.

Syntax:	*PPCtoPC.securitySetJoyMode*	
Returns:	status	
Status codes:	Ok	
	Error	Mode could not be activated

securitySetJoyOverMode – Stops the obstacle avoidance and puts the platform into overridable joystick control mode. In this mode, traffic is relayed bidirectionally between the motor controller and a handcontrol connected to the CAN bus of the docking module, but the speed target values can be overridden using the xchgJoystick LOS call or the speedXchgJoystick CGI call. The laser scanner is used as bumper only. This CGI can be used only when the safety controller is stopped.

Syntax:	*PPCtoPC.securitySetJoyOverMode*	
Returns:	status	
Status codes:	Ok	
	Error	Mode could not be activated

securitySetJoyMotionPar – Sets motion parameters for joystick drive. The joystick sends drive commands as integers in the range [-100; 100]. The X axis drives the platform rotation, the Y axis its forward motion. maxX and maxY limit the range of commands allowed from the joystick in both directions, and must be in the range [0; 100]. maxAccX and maxAccY limit the rate of change on each axis per 20ms, when accelerating. Deceleration cannot be trimmed for safety reasons. The default values are maxX=30, maxY=100, maxAccX=1, maxAccY=1. This means acceleration is at its minimum value, and the platform needs 2 seconds to get to full forward speed.

Syntax:	*PPCtoPC.securitySetJoyMotionPar?maxX+maxAccX+maxY+maxAccY*
Returns:	status
State codes:	Ok
	Error.Parameter.*

securitySetMailer[On / Off] – NOT SUPPORTED ON THIS PLATFORM – Turn on/off the mailer option. When on, the robots sends its errors to the defined mail address (see securitySetMailAdr in this section). In order to use this option the robot has to be connected to the Ethernet network (not a local one) and have access to the defined SMTP server, which is set in the board ROM by Preferences.SetSMTPServer (see section 3.2).

Syntax:	*PPCtoPC.securitySetMailerOn* and *PPCtoPC.securitySetMailerOff*	
Returns:	status	
Status codes:	Ok	
	Error	Mailer could not be turned on/off

securitySetMailAdr – NOT SUPPORTED ON THIS PLATFORM – Allows to set the email address to use with the mailer option.

Syntax:	*PPCtoPC.securitySetMailerAdr*?"addr"
Returns:	status
Status codes:	Ok
	Error.Parameter.*

securityStart – Starts the safety controller.

Syntax:	*PPCtoPC.securityStart*	
Returns:	status	
Status codes:	Ok	
	Error	Safety controller could not be started

securityStop – Stops the safety controller. This **must** be used **before turning off** the robot.

Syntax:	*PPCtoPC.securityStop*	
Returns:	status	
Status codes:	Ok	
	Error	Safety controller could not be stopped

securityVoltage – Returns the voltage of the batteries in [V].

Syntax:	*PPCtoPC.securityVoltage*
Returns:	voltage [V]

4.8 Mission – XO/2 System

Mission allows moving around by **using the map graph**. You can either go to a (x, y, theta) goal position or to a Node defined in the map.

missionAbort – Aborts the current GoTo command.

Syntax:	*PPCtoPC.missionAbort*	
Returns:	status	
Status codes:	Ok	

missionGoTo – Defines the (x, y, theta) goal position in world coordinates that the robot has to reach.

Syntax:	*PPCtoPC.missionGoTo?x+y+theta*	
Returns:	status	
Status codes:	Ok	
	Error.Paramter.*	
	Error.TooFar	Goal position is too far (distance > 10m)
	Error.Busy	Robot is already executing a GoTo command

missionGoToNode – Defines the node ID that the robot has to reach.

Syntax:	*PPCtoPC.missionGoToNode?nodeID*	
Returns:	status	
Status codes:	Ok	
	Error.Paramter.*	
	Error.Busy	Robot is already executing a GoTo command

missionReset – Resets the odometry position of the robot to the home node position. This is helpful when the robot is not started on the home node or if the robot get lost.

Syntax:	*PPCtoPC.missionReset*	
Returns:	status	
Status codes:	Ok	

missionGetMotionState – Returns the current state of the navigation algorithm. A state starting with "Moving" means the algorithm is still in control, whereas a state starting with "Stopped" means the robot has either arrived at the requested position or has given up.

Syntax:	*PPCtoPC.missionGetMotionState*	
Returns:	state	
State codes:	Moving.Ok	
	Moving.Collision	A collision was detected with the bumpers
	Moving.EmgButton	The emergency button is pressed
	Stopped.Ok	The goal has been reached
	Stopped.SecurityNotRunning	The security controller is not running
	Stopped.LocalizationError	There was an error in the robot localization
	Stopped.PlanError	An unknown node was encountered
	Stopped.Aborted	The motion was aborted by missionAbort
	Stopped.GoalBlocked	The obstacle avoidance cannot reach the goal
	Stopped.Lost	The robot got lost
	Stopped.Undefined	An undefined error has happened

4.9 Localization – XO/2 System

The localization can be started and stopped using the following commands.

localizationStart – This command starts the localization.

Syntax:	*PPCtoPC.localizationStart*
Returns:	status
Status codes:	Ok
	Error

localizationStop – This command starts the localization.

Syntax:	*PPCtoPC.localizationStop*
Returns:	status
Status codes:	Ok
	Error

4.10 Revision History

4.10.1 REV2195

REV2195 is the first version used for the teaching.

4.10.2 REV2359

Added features:

- Added bumper configuration procedure.
- Added localizationStart and localizationStop CGI calls.
- Added speedTrapezTranslate, speedTrapezRotate and speedXchgJoystick CGI call.
- Implemented lsGetRawData CGI call.
- Added boot-time hull customization.

Chapter
5

Hardware Extension

Power supply and serial ports

5.1 +5V, +12V, +24V Power Supply

The WHEELCHAIR platform allows for the connection of several +5V, +12V, and +24. The position of the connectors is shown in Figure 7.

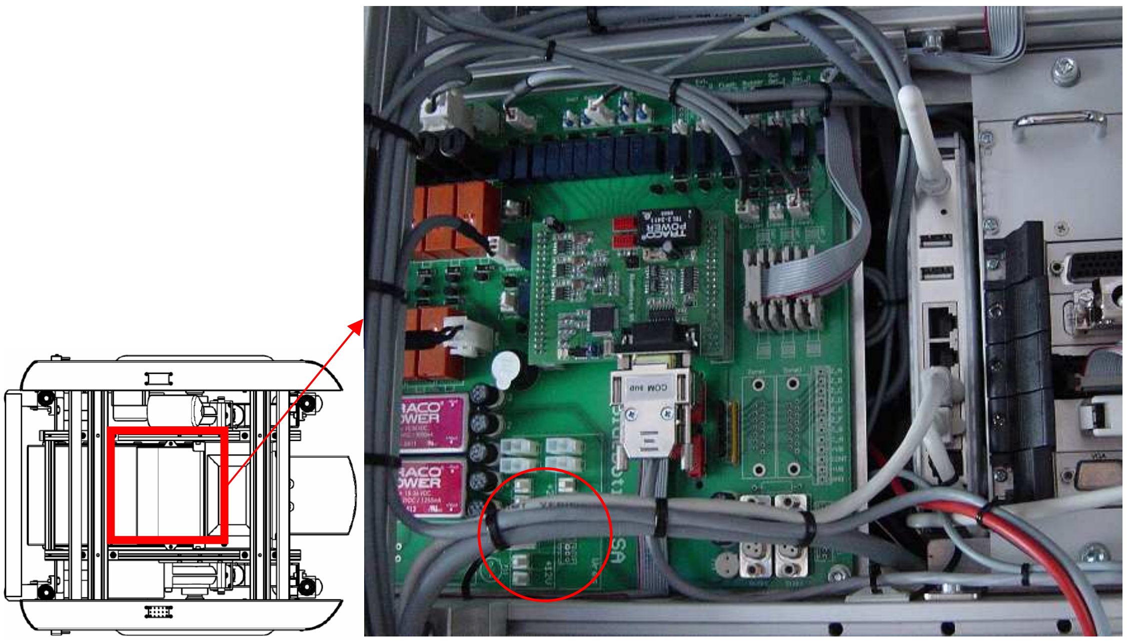

Figure 7: Connectors for +5V, +12V, and +24 power supply.

5.2 Serial Ports

The WHEELCHAIR platform has some I/Os that are not required by the basic system. Figure 8 shows the board allowing connecting new peripherals to the robot via the configurable RS-232/422 ports.

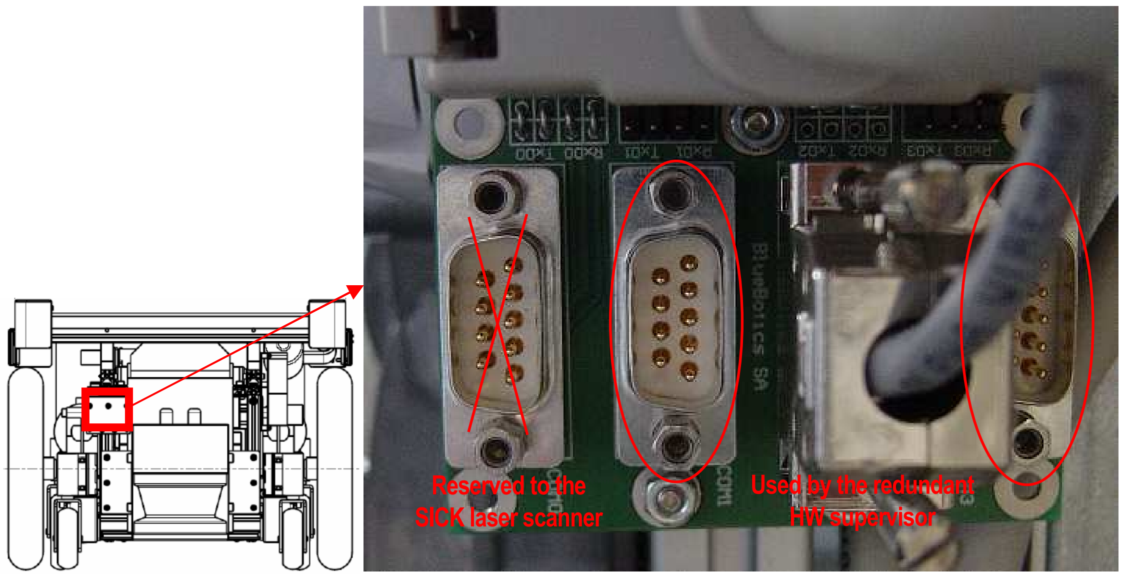
Reserved to the SICK laser scanner
Used by the redundant HW supervisor
BlueBotics SA
COM0
COM1
RxD0 TxD0
RxD1 TxD1
RxD2 TxD2
RxD3 TxD3

Figure 8: The configurable RS-232/422 ports.

Chapter

6

Troubleshooting

What to do when

Problem	Cause	Solution
No response from the RS232 (configuration of the PowerPC)	RS232 settings	9600bps, 8 data bits, no parity, 1 stop bits, no flow control
"	Null-modem need	Add a null-modem converter
Platform beeps and does not power up when I switch on	Batteries are low	Switch off and charge the batteries.
Set speed not respected (robot stops after speed command)	Position control running	Stop the security, set bump mode, start security
"	Bumper contact	Bumper contact cause set speed(0, 0). You have to displace the robot to avoid the contact.
Everything looks fine but the robot does not move neither in autonomous nor in joystick mode	Unbrake system	Release the unbrake system.

Chapter
7

Miscellaneous

Recommendations

7.1 Recommendations

Demounting or displacing parts and components on the robot is not permitted. Defects arising to a component after or during a displacement could cause the loss of the warranty.

The structure (chassis) of the robot is in aluminum. Some edges are sharp and may be dangerous. Pay attention when using the robot.

Part of the electronics is not covered. Avoid any contact with this surface.

7.2 Important Disclaimer

The delivered platform remains a system for laboratory use.

The platform should not be used without supervision of a competent person.