ERA Mobile Robot User's Manual

Rev H, October 2009, Player Software v. 2.0.4

©2009 Videre Design LLC

Table of Contents

Figures and Tables

1. Introduction

The Erratic (ERA) mobile robot is a full-featured, industrial mobile robot base. The name comes from the Latin *errare*: *to wander*. The ERA is compact and powerful platform, and capable of carrying a full load of robotics equipment, including an integrated PC, laser rangefinder, and stereo cameras.

We have designed the ERA to be both simple, reasonably priced, and highly expandable. ERA was created with the experience of many years of teaching an advanced robotics course (cs225b.stanford.edu) at Stanford University, as well as participation in DARPA mobile robot projects such as Centibots (www.ai.sri.com/centibots) and LAGR (Learning And Ground Robots, www.darpa.mil/ipto/programs/lagr/vision.htm).

The base platform is designed to work with an on-board PC. The PC runs the Player/Stage open-source software system (www.sourceforge.net/playerstage), as well as user programs for intelligent behavior. The PC has a wireless internet link for communication with off-board computers.

ERA supports a full range of sensors, including IR floor sensors, sonars, laser rangefinders (LRFs) and stereo vision. The on-board PC has sufficient processing power to run sophisticated processing algorithms for these sensors – for example, to perform 3D analysis for localization and mapping.

1.1. Base Characteristics

Base platform size	40 cm (L) x 37 cm (W) x 18 cm (H)
Wheels	15 cm diameter (driven) 6.25 cm diameter (caster) Polymer core, soft non-marking rubber tread
Wheelbase	33 cm
Drive type	Differential, single rear caster
Maximum speed	2.0 m/sec, 720 deg/sec
Motors	DC reversible with gearhead 72 W continuous power
Encoder resolution	500 cycles per motor revolution
Controller	16 bit microcontroller Integrated controller / motor driver Analog, digital, and servo interfaces
Power	12V, 7AH lead-acid batteries (x3) 5A charger
Weight	4.5 Kg (base) 12 Kg (base + 3 batteries)
Payload	20 Kg

1.2. *Onboard PC Characteristics*

PC size	16cm (L) x 16cm (W) x 5.5 cm (H)
Processor	Intel Celeron-M 1.4 GHz Optional Core 2 Duo Processor
RAM	512 MB standard Optional up to 1 GB
Storage	40 GB hard disk Optional up to 100 GB
Ports	2x USB 2.0 1x IEEE 1394a (FireWire), 6 pin VGA, DVI
Removable drives	CD-ROM drive
Wireless	IEEE 801.11b/g
Power	30W maximum 20W normal use
Weight	1 Kg

1.3. *Warranty Information*

The ERA-MOBI is sold with a one year warranty. We check the robot's functionality thoroughly before shipping, but due to the variability of shipping and end user conditions, we cannot guarantee performance. This warranty does not included damage from customer misuse, mishandling or disassembly. Examples of misuse can include, but are not limited to damages sustained when disassembling the robot, intentional modification of the robot, or damages sustained during extreme use. If you have any questions about the warranty, do not hesitate to contact Videre Design (sales@videredesign.com).

2. Robot Setup

The ERA comes with batteries and computer separate from the robot base, for protection during shipping. If the Hokuyo URG laser rangefinder or SICK laser rangefinder have been ordered, then they are also un-mounted from the robot base. Sonar and IR floor sensors are already installed. Please follow the directions in this section to set up your ERA robot for use.

2.1. *Components*

Unpack and locate all of the components of the robot. You should have:

- Robot base with 12V batteries (3)
- Battery charger
- Compact PC
- Cables (USB and computer power)
- Laser rangefinder (if ordered)

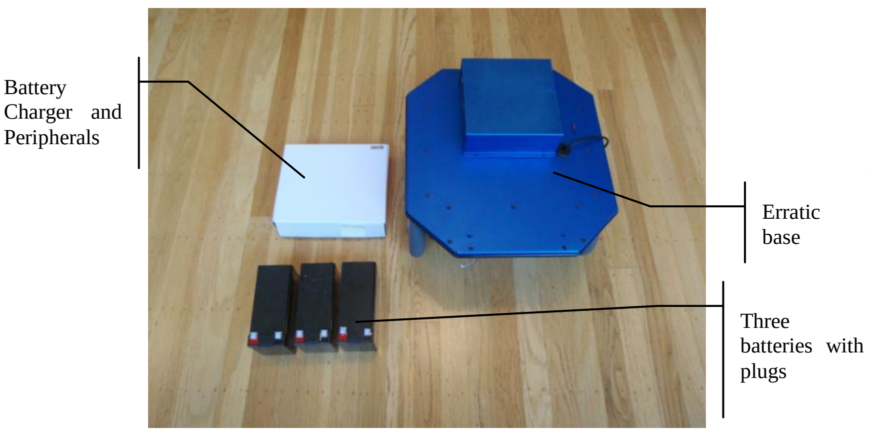


Figure 2-1 Components for the Erratic robot. Unpack the small box to access the PC, cables, and battery charger.

Refer to Figure 2-2, which shows the robot base and other components to be mounted onto the robot. The PC, charger, and cables are in the box.

The cabling for the computer and other devices is connected to the robot base, and held in place with blue tape. Remove the blue tape wherever you find it on the robot base.

For installation, the main power switch must be in the OFF position. Leaving the switch ON/RESET during installation can result in damage to the robot and injury to the user.

2.2. *Connecting the computer*

The computer is installed onto the robot base prior to shipping, but with several cables disconnected for safety reasons. Locate the antenna connector and power connector on the side of the PC (see Figure 2-3). The power connector should be on the *right* side of the robot, looking from the front (see Figure 2-2). In the white box came antenna. It must screwed onto the Wireless Antenna Terminal (see Figure 2-2) for wireless operation.

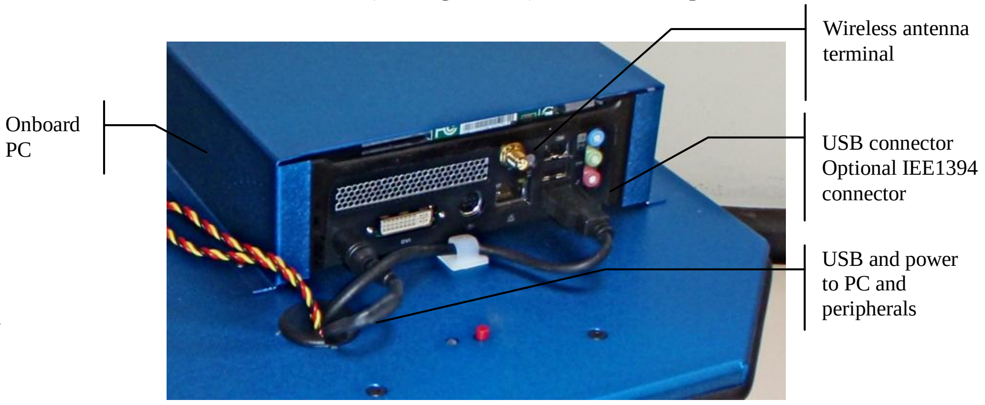

Onboard PC
Wireless antenna terminal
USB connector Optional IEE1394 connector
USB and power to PC and peripherals

Figure 2-2 Top of the robot, populated with a computer and cable connections. Front of robot is at the bottom of the image.

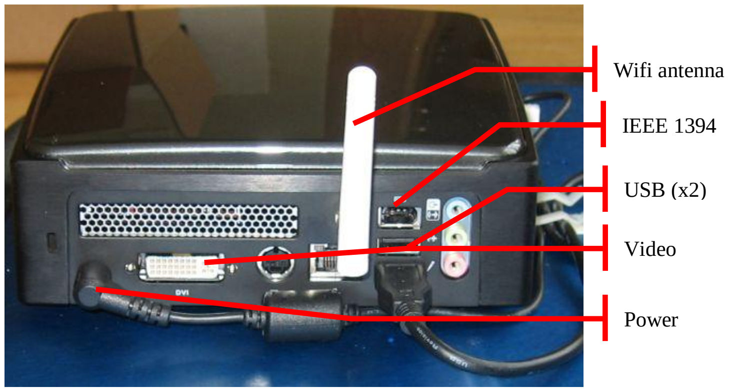


Figure 2-3 Port side of the PC. USB, IEEE 1394, Ethernet, video, stereo, and microphone ports. Antenna is attached to the wifi connector.

After removing blue tape, connect the power and USB cables from the base to the PC. See Figure 2-3 and Figure 2-4 for the connections. Plug the 8-pin connector into the Erratic base, and the USB cable into the lowest port on the PC.

2.3. LED indicator

The LED indicates the status of the robot. When the robot is powered, it is lit. The color indicates the status of the robot controller:

- Green when the controller is on, but there is no power to the wheel motors.
- Orange (red+green) when the controller is on and there is power to the wheel motors.
- Flashing when the battery is low (< 11V), otherwise solid.

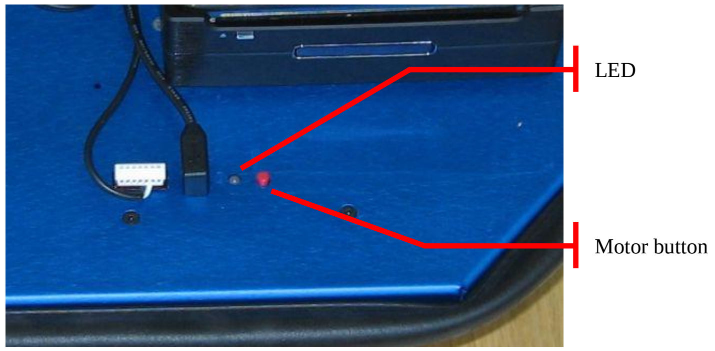


Figure 2-4 Ports, LED, and power button on top of the robot.

2.4. Motor power button

The red button is an emergency switch for stopping the motors when the robot is moving. If you press this button and let it go, power to the motors is cut off, and the motors go limp.

This button can also be used to perform a basic motor/controller test of the robot – see Section 2.7.

2.5. Installing the batteries

Make sure that the main power switch (Figure 2-7) is turned OFF before proceeding!

The robot is not shipped with the batteries installed. The following description should be consulted when removing and re-installing the batteries; for the initial setup, please read and follow carefully.

Batteries are installed from the back of the robot. Locate the caster wheel in the back. There is a slot to the right of the caster wheel for inserting the batteries (refer to Figure 2-5). The back plate of the robot can be removed with the two silver thumb screws on the rear sides of the robot. Insert the first battery in the slot, then slide it over all the way to the left, until it

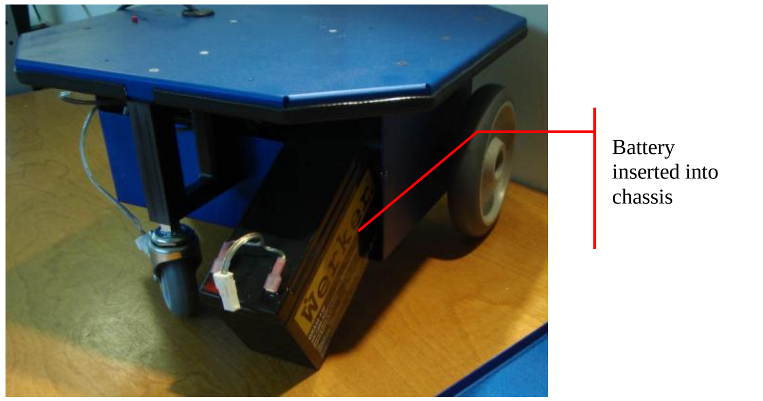


Figure 2-5 Inserting the first battery into the slot on the right side of the chassis.

contacts a bumper. Insert the second and third batteries in a similar manner. You should now have three batteries in the robot.

Before proceeding, ensure that the robot power switch is OFF.

Now connect the batteries to the internal power plugs of the robot. There are three power plugs that are suspended from the underside of the robot top plate. Connect one to each of the plugs on the batteries. The plugs are polarized, so they can only be connected in the correct way.

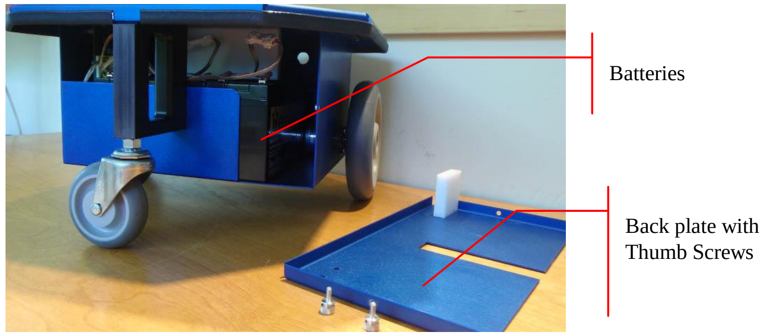


Figure 2-6 Three batteries installed in the back of the robot, with the back plate on the right side.

When you are finished, the installed batteries should look like Figure 2-6. To test that they are working, turn on the main power switch. You should see the green LED light up on the motor switch (Figure 2-2).

Turn the main power switch OFF.

2.6. Charging the batteries

The batteries are charged before they are shipped, but they should be re-charged before using. Make sure the battery charger is turned off before plugging it in – there is a power switch on the side of the charger. Plug the power cord of the charger into a power socket. Plug the charging plug of the charger into the charging port on the side of the robot (see Figure 2-7). Turn the charger on. The red LED indicates that the charger is working. The other LED will be orange when the batteries are being charged, and green when they are fully charged. Charging takes 1-2 hours for a full charge, depending on the state of the batteries.

The charger supplies 5 amps, which is strong enough to charge the batteries even when the PC is running (the PC will draw about 2 amps). You may operate the robot PC while the charger is plugged in. However, be careful not to run any programs that cause the robot to move.

2.7. Basic motor test

To test the motors and the controller, either put the robot on blocks so its wheels are off the floor, or put it in the middle of the floor with the charger cord disconnected. During the test, the robot will move slowly in a small circle, so you don't need a lot of space.

Turn on the robot, and note that the green LED becomes lit. Then, press and hold the motor button down for about 3 seconds. The green LED will switch to orange, indicating the motor test will begin when you let go of the button. The robot will now test each motor in sequence:

Right motor power
Left motor power
Right motor control
Left motor control

In the first two tests, each motor is supplied with power. You should see the robot first turn to the left (right motor turning), then to the right (left motor turning).

In the second two tests, the controller sets the motor speed using feedback from the encoder. Again, you should see the robot turning to the left and then the right. It turns at a very slow speed, and the speed should be constant. If it turns slowly at first, and then starts to speed up, there is a problem with the encoder or controller for that motor.

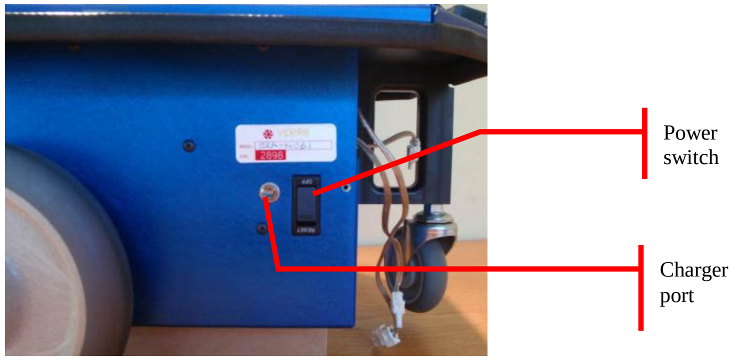


Figure 2-7 On/off switch and charger port on the left side of the robot.

2.8. Hokuyo URG installation

If you have the Hokuyo URG mounted on a tilt platform, please see the next section. This section is for the URG on a fixed mount.

If your robot was ordered with the Hokuyo URG laser, the cables for this device are already installed on the robot base. You must remove the blue tape that secures the cables, and connect the cables to the URG and to the PC – refer to Figure 2-2.

There are two cables that are mounted on the robot base. One is an 8-pin cable that connects into the URG. It supplies power to the URG. The other end is connected to the robot base power port, inside the robot.

The other cable is a USB A to mini-B cable. The mini-B end should be plugged into the USB port of the Hokuyo. The other end should be plugged into one of the USB ports on the PC.

The Hokuyo URG is now ready to be used.

2.9. Hokuyo URG with tilt unit

The Hokuyo URG can come equipped with a tilt unit for tilting the rangefinder forward or backwards (refer to Figure 2-8). The maximum amount of travel is 60 degrees forward and 90 degrees backward, so that the URG can sweep the ceiling.

For shipping, the cables are unplugged, and the URG is removed from the top plate. Remove the tape that holds the URG to the top plate, and any other packing materials. Using the four small nuts on the bottom of the URG, install the Tilt unit to the matching holes on the top plate. The miniature USB connector plugs into the USB slot of the URG, and supplies communication to the device. The black cable power connector plugs into

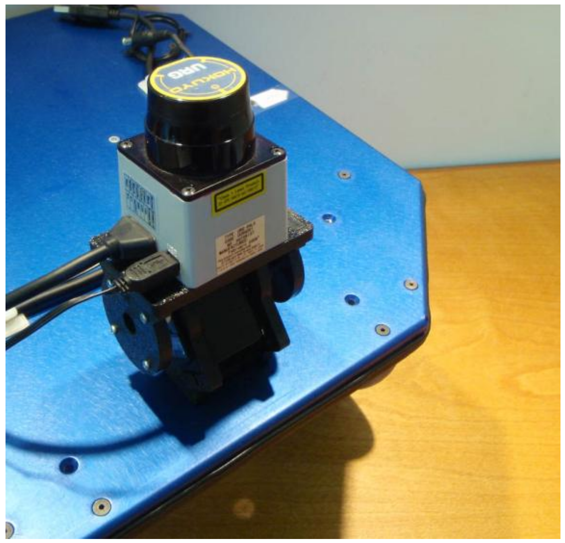


Figure 2-8 Hokuyo URG mounted on a tilt unit.

the 8-pin slot of the URG.

The tilt unit can be tested by turning on the power to the robot. It should move to an upright position and stay there.

When the robot is turned off, it is recommended to put the URG into the "stowed" position, 90 degrees backwards. In this position, the URG is less likely to be damaged by hitting something. The tilt unit can be moved by pushing gently on the base of the URG.

2.10. STOC installation

If you have a STOC stereo camera (or other camera device) with your robot, and it is mounted on a fixed base, then follow these steps for installation.

Unpack the STOC, and locate the ¼" x 20 threaded hole on the bottom of the STOC. This hole will be used to mount the STOC on the mounting bracket. Put the STOC on the thread of the mount, and turn it to engage the threads and hold the STOC to the top of the bracket.

After mounting the STOC, connect the grey IEEE 1394 cable to the STOC port. This cable supplies both power and communications to the device. The other end of the cable plugs into the IEEE 1394 port of the computer (see Figure 2-3).

Please refer to the STOC User's Manual from the downloads section of the Videre Design stereo vision website for information on the operation of the STOC.

2.11. STOC installation with pan-tilt unit

The STOC (or other camera) can be mounted on a pan-tilt unit (see Figure 2-9). The pan-tilt unit provides +- 90 degrees of panning motion, and +60, -90 degrees of tilt.

To install the STOC device, remove all packing tape and other materials from the pan-tilt unit. There is a ¼"-20 bolt in the top deck of the tilt unit, which fits onto a thread hole in the STOC. Locate the threaded hole on the bottom of the STOC, mount the STOC on the tilt unit, and screw the bolt into the STOC.

Power and communications is supplied by a white IEEE 1394 cable. The cable is attached to the top plate. Insert one end into the IEEE 1394 port on the STOC device, and the other end into the IEEE 1394 port on the computer. Make sure the cable is free around the STOC, since it must move when the pan-tilt unit moves.

The pan-tilt unit can be tested by turning on the power to the robot. It should move to an upright position and stay there.

When the robot is turned off, it is recommended to put the STOC into the "stowed" position, 90 degrees backwards. In this position, the STOC is less likely to be damaged by hitting something. The tilt unit can be moved by pushing gently on the base of the STOC.

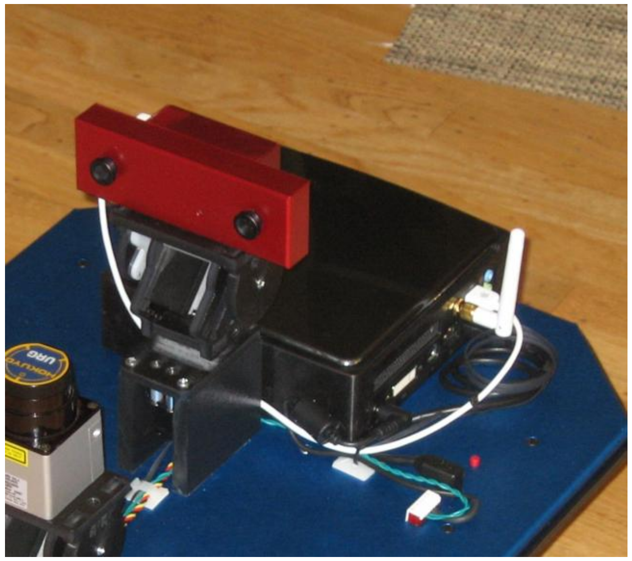


Figure 2-9 STOC stereo device mounted on a pan-tilt unit.

(note: Older model PC pictured)

Please refer to the STOC User's Manual from the downloads section of the Videre Design stereo vision website for information on the operation of the STOC.

3. Quick Start

After setting up the robot (Section 2), you should set up the Erratic PC to interact with your computer environment, and run some sample programs to test the functionality of the robot.

3.1. Networking environment

The Erratic PC comes pre-configured with an installation of Ubuntu Linux (www.ubuntu.com), Player/Stage robot control software (http://playerstage.sourceforge.net), and all drivers necessary for running the peripherals on the PC, most notably the wireless device. Figure 3-1 shows a typical setup for using the robot within a networked environment.

The robot PC has a wireless 802.11 b/g link already installed. This link can communicate to a wireless *access point*. The access point is part of a wired local network, and other PCs are connected to the wired network.

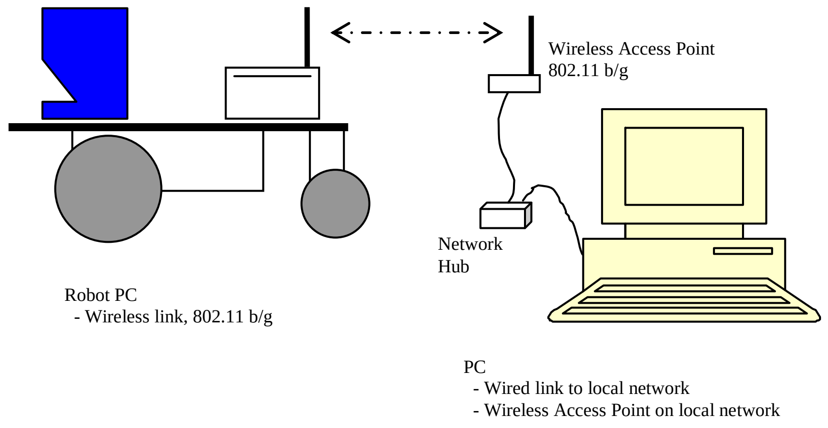

Wireless Access Point
802.11 b/g
Network
Hub
Robot PC
- Wireless link, 802.11 b/g
PC
- Wired link to local network
- Wireless Access Point on local network

Figure 3-1 Typical setup for integration of Erratic into a computer environment.

Thus, the robot can communicate with any computer on the local network through the wireless access point, and if the local network is on the Internet, with any computer connected to the Internet.

3.2. Accessing the Onboard PC

You must access the onboard PC to set up the wireless network. There are two modes of access:

- Keyboard, mouse, and monitor
- Wired Ethernet connection

For users who do not have much experience with Linux, the first mode is recommended.

3.2.1. Keyboard, mouse, and monitor

This is the simplest way to access the PC. Plug in a PS/2 mouse and keyboard in the appropriate slots on the back of the PC. Plug in the monitor to the VGA connector on the back of the PC.

Start up the PC by turning on the power switch of the robot, and pressing the Power button on the front of the PC. You should see the PC boot taking place on the monitor. At the end of the boot process, there will be a Ubuntu Welcome screen for login. Log onto the PC using:

User name: `erratic`
Password: `erratic`

Go to Section 3.3 to continue.

3.2.2. Wired Ethernet connection

To set up the onboard PC via wired Ethernet, you must have a local network that uses the DHCP protocol to assign internet addresses. Plug an Ethernet cable into the Ethernet port on the back of the PC. Then start up the PC by turning on the power switch of the robot, and pressing the Power button on the front of the PC. The disk activity LED will flash for awhile, and it will take about 2 minutes to boot the PC.

To access the PC, you have to determine the IP address that has been assigned to the PC. Often you can find this information by looking at the log files of the DHCP server – contact you System Administrator for this information. If you cannot determine the DHCP address, you will have to use the first method (keyboard, mouse, monitor) of accessing the PC.

If you find the IP address of the onboard PC, then you can connect to the PC using SSH (Secure Shell) from any computer on the local network (even Windows computers, if they have SSH installed). Start a command window, and use the command:

```
ssh erratic@xxx.xxx.xxx.xxx
```

Here xxx.xxx.xxx.xxx is the IP address of the onboard PC, and "erratic" is the user name. You should now be asked for a password, which is "erratic" again. At this point, you will be talking to the onboard PC.

3.2.3. Wireless access

This is the normal way to access the robot. See Section 4 for details on how to set up the wireless Ethernet connection that is built into the onboard PC.

3.3. Root access

For many operations, such as changing the wireless configuration, the user must be *root*. The `erratic` user has root privileges, using the `sudo` command. In a shell, you can issue any command as root by prefixing it with the word "sudo", e.g.

```
sudo mkdir /etc/myfile
```

You will be asked for a password the first time you issue the sudo command; the password is the erratic user password.

You can also become the root user by using the following command:

```
sudo su
```

After this, all commands will be performed as root.

3.4. Player software

The onboard PC has a complete installation of the Player/Stage software. The directory structure for the development software is:

```
/home/erratic/player-2.0.3   Player server and clients
                config/              configuration files
                examples/libplayerc  C code examples
                examples/libplayerc++ C++ code examples
/home/erratic/stage-2.0.3    Stage robot simulator
                worlds/      simulation config files
```

In addition, the core executable files and libraries for Player and Stage are in the standard directories `/usr/local/bin` and `/usr/local/lib`, and useful configuration files are in `/usr/local/share/player/config` and `/usr/local/share/stage/worlds`. The sample client executable files are in the `examples/libplayerc(++)` directories.

3.5. PC Client software

To run clients on a remote PC, you must install Player/Stage on that PC. Please see the Player/Stage documentation for information on how to do this. Player/Stage can run on Linux and Apple systems.

There is also a java client available for all systems: MS Windows, Linux, and Apple. This software can be downloaded from http://java-player.sourceforge.net. Videre Design provides a Java client viewer that is useful for checking out the robot, and can serve as a platform for future development.

3.6. Running the Player server

The software system for controlling the robot is the Player/Stage system (http://playerstage.sourceforge.net). The onboard PC has a full installation of the latest version of this software.

Figure 3-2 shows a typical system for controlling the robot. The onboard PC runs the *Player server*, which is a program for controlling the robot's motors and sensors. The server takes commands and exports information to *clients* that can run on the onboard PC, or remotely on other PCs connected to the network.

Before any clients can connect to the robot, the Player server must be started on the robot. This is always the first step in running robot programs. To start the server, open a shell on the robot PC, and run the following program:

```
player /usr/local/share/player/config/erratic.cfg
```

This will start the player server, connecting to the microcontroller via the USB port. The server will print a startup message, indicate that it has established communication with the microcontroller, and then listen for connections from clients.

To stop the Player server, use ctrl-C or a kill command to it process. The server can be restarted at any time.

An alternative method is to have Player start on the robot during the boot process. In this way, there is no need to open a shell on the robot to start Player. See Appendix A for information on autostart of Player.

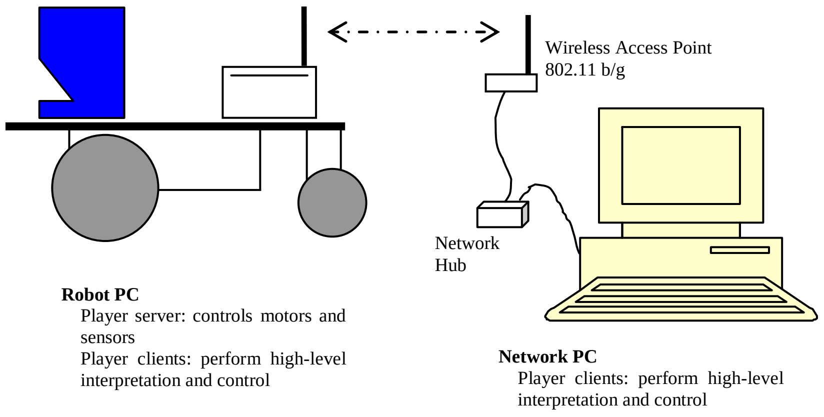

Wireless Access Point
802.11 b/g
Network
Hub
Robot PC
Player server: controls motors and sensors
Player clients: perform high-level interpretation and control
Network PC
Player clients: perform high-level interpretation and control

Figure 3-2 Typical setup for Player server and clients.

3.7. Services

The server for Erratic offers the following services, depending on the peripherals that are installed.

- **position2d**
  This is the basic service for controlling the motion of the robot, and publishing the integrated position of the robot based on the motor encoders (dead reckoning).
- **power**
  Provides information about the battery voltage
- **analog**
  Analog voltage levels from the wheel current sensors, and user analog and digital I/O
- **ir**
  Infra-red floor sensors, detecting the floor in front of the robot
- **sonar**
  Sonar sensor service, provides information about sonar ranges from the front sonar array, if installed.
- **laser**
  The laser service controls the Hokuyo URG or SICK LMS 200 laser rangefinder, and publishes range scans from the laser.
- **ptz**
  The ptz service controls up to 3 hobby-type servos. These servos are typically connected as a pan-tilt unit, for cameras or the Hokuyo laser.

3.8. Running standard Player clients

Once the Player server has been started on the onboard PC, *clients* can connect to the server to control the robot and show information about the robot. These clients will run on Linux and Apple systems.

A client can be run on any PC that has an Ethernet connection to the robot. If the client produces graphical output, then typically it will run on an off-board PC, because transferring the graphics information across the Ethernet connection can consume a lot of bandwidth, especially for the wireless connection. But for a quick start, you can run the clients from the onboard PC. If you have an ssh connection to the onboard PC, and the connection exports X windows, then the client graphics will show up on your screen.

There are two clients that can be used to check the basic functionality of the robot.

- **playerjoy**
  This client controls the robot via a joystick or the keyboard. Start the client with the command:

  ```
  playerjoy -g [<xxx.xxx.xxx.xxx:6665>]
  ```

  The optional argument `<xxx.xxx.xxx.xxx:6665>` is used when playerjoy is started on a remote network PC – xxx.xxx.xxx.xxx is the IP address of the robot PC. The –g argument uses incremental keyboard control, which is a useful way to move the robot.

  If the robot is tethered to a keyboard and monitor, or to a wired Ethernet, then be careful to only move the robot a small amount.

  These are the key assignments for motion.

  - i – move forward
  - , -- move backwards
  - j – rotate left
  - l – rotate right
  - k, <space> -- stop

  Pressing any of the motion keys more than once will increase the speed in that direction.

- **playerv**
  This program presents a graphical interface for viewing the robot's laser rangefinder, and for moving the robot.

  ```
  playerv [-h xxx.xxx.xxx.xxx]
  ```

  The optional argument is used when playerv is started on a remote network PC -- xxx.xxx.xxx.xxx is the IP address of the robot PC.

  Playerv will open a graphic window that shows the robot's motion and sensor information. You must *subscribe* to the robot's services to display them, or to control the robot. To subscribe to the laser device, use the Devices->laser menu item. You should see a representation of the robot's laser scan after you subscribe.

  To control the robot, subscribe to the following three items:

  Devices->position2d->Subscribe
  Devices->position2d->Command
  Devices->position2d->Enable

  You can now control the robot by dragging the control icon in the middle of the window.

  If the robot is tethered to a keyboard and monitor, or to a wired Ethernet, then be careful to only move the robot a small amount.

3.9. *Running the Javaclient Viewer*

The Javaclient viewer is available from Videre Design – check the Download pages under Support. You must have JRE (Java Runtime Environment) 1.5 installed to run the Viewer. For developing programs with Javaclient, you also should install the JDK (Java Development Kit). See http://java-player.sourceforge.net.

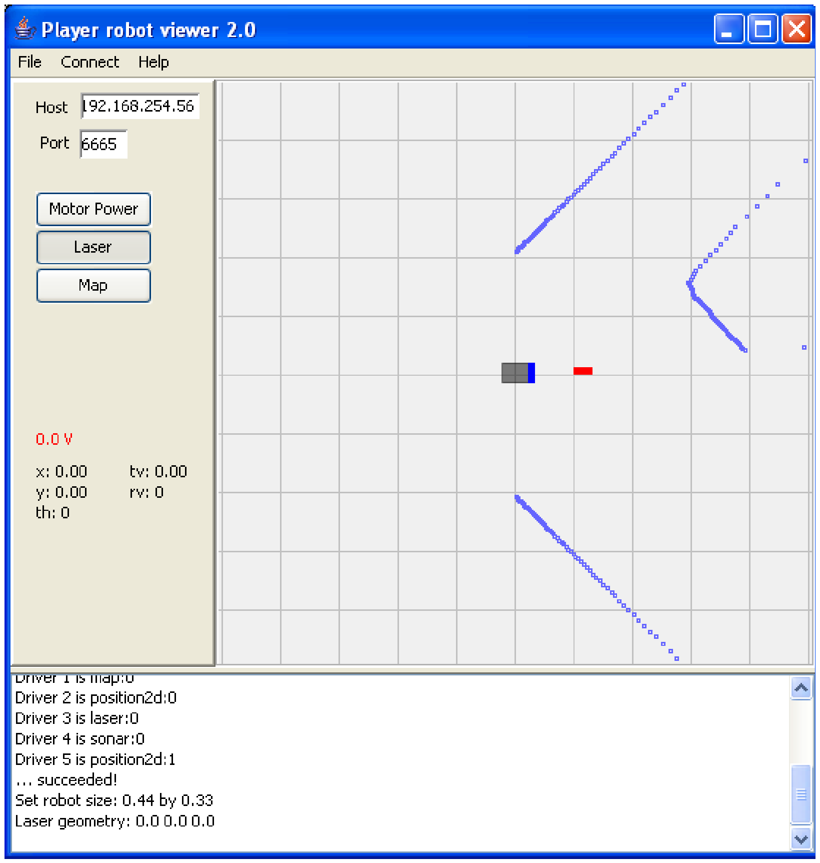
To start the Viewer client, on Linux systems run

```
java -jar DoViewer.jar
```

and on MS Windows systems just double-click on `DoViewer.jar`.

The Viewer GUI will start, looking like Figure 3-3. The `Host` field is for the address of the robot PC – typically it will be a local DHCP-derived address, unless you have set up the robot with a DNS name. If you are running Viewer on the robot, use `localhost`.

After specifying the host address, use the `Connect` menu to open a connection to the robot. If the address is correct, and Player is running on the robot, then Viewer will open the connection, and list the services available in the text viewing area at the bottom of the GUI. The buttons on



Host 192.168.254.56
Port 6665

```
0.0 V
x: 0.00    tv: 0.00
y: 0.00    rv: 0
th: 0
```

```
Driver 1 is map:0
Driver 2 is position2d:0
Driver 3 is laser:0
Driver 4 is sonar:0
Driver 5 is position2d:1
... succeeded!
Set robot size: 0.44 by 0.33
Laser geometry: 0.0 0.0 0.0
```

Figure 3-3 Javaclient Viewer program.

the left will be enabled for the corresponding service.

To view the sensors, click on one of the buttons on the left, for example, the `Laser` button. Now, the laser readings will appear in the Viewer graphics pane.

To move the robot, press the Enable button to enable motor power. Then, click in the graphics pane, and use the i,j keys to increment or decrement the forward velocity, and the j,l keys to increment or decrement the rotational velocity.

If the robot is tethered to a keyboard and monitor, or to a wired Ethernet, then be careful to only move the robot a small amount.

As the robot moves, you should see the sensors outline the shape of objects around the robot.

3.10. Example client programs

There are two simple client programs that show some interesting behavior of the robot. These programs are found in the `examples/libplayerc++` directory of the distribution.

- **laserobstacleavoid**
  This program moves the robot forward, and turns if it encounters an obstacle ahead of the robot.
- **randomwalk**
  This program is similar to the previous one, but after a short interval it turns the robot in a new direction.

4. Configuring the wireless interface

The wireless interface is the best way to observe and control the robot. Once the robot is connected over a wireless link, you can connect to it from any PC on the network, and run client programs to control the robot.

4.1. Configuration

To configure the wireless interface, you must login to the onboard PC. You must also have the following information from the systems administrator:

- ESSID of the wireless network
- WEP key, if WEP security is used

If you are logged in using a keyboard and monitor, you can use the Wireless administration tool to configure the wireless device. Go to the menu at the top of the page, and navigate to `System -> Administration -> Networking`. You will then see a dialog with the Ethernet devices listed. Choose the Wireless device, and go to the Properties tab. You can change the ESSID and WEP key there.

If you are logged in using SSH, you must edit the file (as root):

```
/etc/network/interfaces
```

Both `emacs` and `vi` editors are available. In the interfaces file, you will find the following lines for the wireless device:

```
# USB wireless
iface ath0 inet dhcp
wireless-essid MyNetwork
wireless-key MyKey
wireless-mode Managed
auto rausb0
```

Change the ESSID and key to the appropriate values for your wireless network. If there is no key, then remove the wireless-key line.

The wireless interface should now be configured for your network.

4.2. Testing the wireless interface

The wireless interface is configured to start automatically on boot. You can check if it is enabled by using the `ifconfig` command. This will list all of the currently enabled Ethernet interfaces.

If you need to start the wireless interface manually, for instance, to check if the configuration is working, you can use the ifup and ifdown commands.

```
ifdown ath0
ifup ath0
```

The first command stops the interface, and the second starts it, applying the new configuration parameters.

If the wireless interface has successfully connected to the local network access point, it will have an IP address and be accessible over the network. You can check this by trying to ping another computer on the network, e.g.

```
ping hostname
```

is a command for pinging the computer named “hostname”. If the computer does not respond, check the wireless configuration settings.

4.3. Finding the robot on the network

The robot PC uses DHCP to find an address using the local network's DHCP server. The DHCP server will assign a random IP address to the robot PC. In order to communicate with the robot from an external PC, it is necessary to know this IP address. There are two ways this can be done.

- Force the DHCP server to assign a fixed IP address to the robot. This is possible with some DHCP servers, using the MAC address of the wireless interface. See the system administrator for more information.
- Find the assigned IP address. Typically, the DHCP server will assign the same IP address to the robot PC every time it connects to the network. You can find this address by looking at the logs of the DHCP server.

4.4. Connecting to the robot

There are two ways to connect to the robot from a remote network PC.

- SSH connection. The remote PC opens a shell on the robot PC, and then runs programs on the robot PC from the shell. This method is similar to the Ethernet method of Section 3.2.2. It is recommended to start the Player server on the robot PC using this method.
- Network connection. In this method, the remote PC runs a client application, which connects to the robot PC via the network. For example, the user could start playerv to view the sensor readings of the robot:

```
    playerv -h xxx.xxx.xxx.xxx
```

  Here the string "xxx.xxx.xxx.xxx" is the IP address of the robot PC. The playerv application is running on the remote PC, getting data from the robot via the network connection. Using the network connection, many different client programs can run on remote PCs. It is recommended to run GUI programs using this method, since sending graphics information on the wireless interface is limited by bandwidth.

5. Robot Interfaces

The robot controller connects to several sensors and actuators on the robot. Through the USB connection, the controller accepts commands to control the actuators, and sends information it collects from the sensors. User programs on the PC can communicate directly with the controller using the packet protocol described in Appendix D.

Each sensor or actuator can be thought of as an *interface*. The packet protocol is one way to control the interface, the lowest level of control available. An alternative is to use a more abstract view of the interface, in which the details of the packet communication are hidden from the user. Robot-control software on the PC presents these more abstract interfaces to the user. Erratic uses the Player/Stage project for abstract interfaces, which is an open-source, widely-supported software platform.

This section documents the interfaces of the Erratic platform, as well as the Player driver abstract interfaces that correspond to them. From the client side, we will use the Player C++ proxy (libplayerc++ library) as an example of how to interact with the Player driver.

The Player proxy generally uses units of meters for distance, and radians for angles.

5.1. Robot parameters and Player configuration file

Many of the functions of the controller are modified by a set of configuration parameters. These parameters are stored in flash memory on the controller, and most can be changed via controller commands. Appendix A is a full listing of the robot parameters for reference. Here, we describe parameters in more detail in the relevant subsection.

If you are using the Player interface, many of the parameters can be changed using the Player driver configuration file for Erratic.. The configuration file is read by the Player driver on the onboard PC, to determine the resources of the robot base. Typically Player is invoked using the command

```
player <config_file>
```

This reads in the `<config_file>` and starts the driver using its parameters. The standard configuration is listed in Appendix B; the actual file is located on the onboard PC at

```
/usr/local/share/player/config/erratic.cfg
```

In the subsections below, we will describe the relevant Player driver configuration parameters.

5.2. Communication port

This is the PC port over which Player (on the PC) or any other user program communicates with the Erratic controller. The standard name for this port is

```
/dev/erratic
```

This device name is a symbolic link to the actual device, which is a USB serial device (typically `/dev/ttyUSB0`). The PC uses `udev` rules, set at the factory, to link the port name to the correct USB serial device.

If a user program is communicating from a MS Windows machine, the port name will be a TTY comm. port, e.g., `COM6`. For example, the test program for the robot (Appendix D) will run on a MS Windows machine, and it accepts the name of the comm. port as an argument.

The Player driver finds the port name in the configuration file:

```
# Serial port to use for connecting to robot
port "/dev/erratic"
```

5.3. Robot type information

There are several parameters relating to the robot type. Most of these are set at the factory. The user may change the name of the robot using a utility program (A.1).

Parameter	**Default Value**
Name	`Erratic-1`
Type	`Erratic`
Subtype	`Rev E`
Serial	`XXXX` (factory setting)

The <name> parameter is the user name for the robot, and can be any string of 20 characters or less. The user may change this parameter.

The <type> parameter is the class of robot. For this robot, the type is "`Erratic`", a differential-drive small mobile robot. This parameter should not be changed.

The <subtype> parameter specifies a revision or subclass of the robot. The current revision is "`Rev E`". The revision number is useful when updating the controller firmware, since it specifies the exact cabling and other particulars of the robot and its controller. This parameter should not be changed.

The <serial> parameter is a unique serial number assigned to the robot. The serial number should be used in any communications with the support team at Videre Design. This parameter should not be changed.

These parameters are available from the low-level packet interface by sending the configuration packet command. The controller responds by sending a configuration response packet (see Appendix E.8).

The Player driver reads the robot name and type information from the controller and prints it out when it starts up, but otherwise does not use it. The Player configuration parameter <name> is the name of the driver itself, not the robot. This name is used as the driver name by clients that attach to the Player driver.

```
# Instance name of the driver
name "erratic"
```

Currently (Player 2.0.3), there is no Player interface to the type information of the robot.

5.4. Player interface list

The Player driver provides interfaces to the various resources of the robot base. These include motor control, battery power, analog and digital I/O, IR floor sensors, and sonars. Note that the laser rangefinders are handled as separate drivers connected to the PC, and are not configured through the Erratic robot base driver.

All interfaces presented by the Player driver must be listed in the interface list.

```
# Standard player option, tells the driver to expose all
# interfaces, and assigns them IDs
provides [ "position2d:0"
           "power:0"
           "aio:0"
           "ir:0"
           "sonar:0"
           "ptz:0" ]
```

5.5. Position control

The Erratic controller drives the robot wheels using an integrated dual H-bridge motor driver. The driver is able to supply up to 30A at 12V to the motors, although in practice the current is limited to less than this. The motors are rated at 75W continuous power, and are able to drive the robot at speeds up to 2 m/sec and 720 deg/sec. Each wheel is controlled independently, allowing the robot to turn as well as drive forwards or backwards (differential drive).

The controller also connects to the motor encoders, which count 500 cycles per revolution of the motor axis (effectively 38400 counts per revolution of the wheel). With this resolution, the controller can implement very stable feedback motion control, even at very slow speeds (< 1cm/sec).

In addition, the controller integrates the motor encoder values into 2D pose estimates (X, Y, and angle coordinates). This pose estimate is called the *odometry pose*, or sometimes *dead-reckoned pose*. The raw encoder values are also available to user programs.

Position control can be achieved at the low level through the use of packet commands sent directly to the controller. From Player, the `position2d` interface implements position control commands, and returns odometry information. For the C++ proxy, the class is `Position2dProxy`.

5.5.1. Odometry coordinate system

The controller maintains an internal estimate of the pose of the robot in a global coordinate system (Figure 5-1). The pose is calculated based on integrating the motor encoder readings as the robot moves. The accuracy of the estimate depends on a number of factors, including the type of floor (smooth tile is better than carpet), the speed of the robot, wheel slippage, bumping obstacles, etc.

The robot position within this coordinate system refers to the position of the midpoint of the axle between the two wheels. For Erratic, this position is slightly forward of the center of the robot body. The dimensions of the robot, and the position of the coordinate reference, are given in Appendix G.

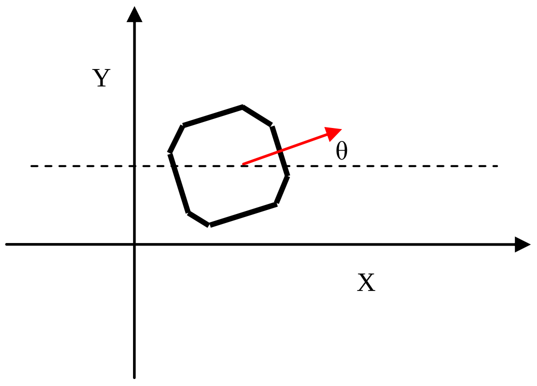


Figure 5-1 Odometry coordinate system. Positive angles are counter-clockwise.

This system is initialized to (0,0,0) when the robot is opened by a client, that is, the robot is at the origin, pointing in the +X direction.

The controller accepts one low-level command, to reset the odometry to (0,0,0).

The Player driver maintains an additional transform on the odometry system, so that it is possible to set the robot's pose to an arbitrary place within the odometry system. There are two position2d interface commands that are relevant:

```
ResetOdometry()
SetOdometry(x,y,theta)
```

The first function resets the robot pose to (0,0,0). The second sets it to (x,y,theta) within the coordinate system (in meters and radians).

The raw cumulative odometry readings (24-bit number of counts) are available from the low-level interface, using the packet commands (see Appendix E.9).

5.5.2. Enabling the motors

Before any motor control commands are accepted, the motors must be enabled. There is a packet command for enabling the motors. The Player Position2dProxy has `SetMotorEnable(bool)` will enable or disable the motors.

When the motors are enabled, the Erratic platform will resist movement by outside forces. To push the robot by hand, disable the motors, either under program control, or by pressing the motor control button.

5.5.3. Motor control modes

The controller presents two different modes of controlling the 2 degrees of freedom available from the differential wheel drive. In TR mode, the controller sets the translational and rotational speeds or positions independently. In VEL2 mode, the controller sets the speed of each wheel independently. Typically, user programs will use the TR mode as the most convenient.

5.5.4. TR mode

In TR mode, the translation and rotation of the robot are treated as independent values. For example, the forward velocity can be set to 0.3 m/sec, while setting the rotational velocity to 20 deg/sec.

Within TR mode, both rotation and translation motion can be determined by either velocity or position commands. For example, while commanding a forward velocity of 0.3 m/sec, the user could also command the controller to achieve a heading of 120 degrees with respect to the odometry coordinate system. Specifying translation velocity and rotation position (heading) is a particularly useful mode.

The controller automatically switches to TR mode whenever one of the TR packet commands is issued to the controller (see Appendix E).

The Player driver uses a configuration parameter to distinguish between TR and VEL2 (independent wheel velocity) modes. In the configuration file there is a line:

```
# Set to 1 to have the driver use individual wheel control.
direct_wheel_vel_control 0
```

If this parameter is set to 0, then all position2d commands are interpreted as rotation/translation commands.

Player provides a subset of the available TR modes:

Rotation and translation velocity [`SetSpeed` functions]
Rotation heading and translation velocity [`SetVelHead` function]

With `SetSpeed(xvel,yawvel)`, a user program sets the desired velocities for translation and rotation. <`xvel`> is in m/sec, <`yawvel`> is in rad/sec. Positive values for rotation velocity turn the robot counter-clockwise, negative clockwise. Negative values for translation velocity move the robot backwards.

With `SetVelHead(xspeed, yawangle)`, a user program sets the desired velocity for translation, and the desired heading, relative to the odometry system. For example, if <`yawangle`> is specified as 85 degrees, and the robot's odometry says that it is currently at 30 degrees, then the robot will move 55 degrees counter-clockwise, and hold that heading.

5.5.5. VEL2 mode

The controller supports independent wheel control, in which the velocities of each wheel are given. There is a low-level packet command for sending these velocities – the controller automatically switches to VEL2 mode when it is received.

The Player driver support VEL2 mode with a configuration parameter:

```
# Set to 1 to have the driver use individual wheel control.
direct_wheel_vel_control 0
```

If this parameter is set to 1, then all position2d commands are interpreted as individual wheel commands. Note that it is not possible to switch between VEL2 and TR mode from within Player – the mode is fixed when the driver starts.

In VEL2 mode, the Player position2d proxy interprets SetSpeed commands as referring to the velocities of the individual wheels:

```
SetSpeed(leftvel,rightvel)
```

<`leftvel`> and <`rightvel`> are the wheel velocities in m/sec.

5.5.6. Acceleration and velocity limits

The controller implements a constant acceleration velocity profile when given a velocity setpoint. That is, when the controller is commanded to achieve a velocity, it uses a constant acceleration to change the velocity in a linear manner, until the setpoint is achieved.

Higher values for these parameters will make the robot more "perky". Higher values may also help simple control algorithms, which depend on a short lag time in achieving a velocity setpoint. However, higher values for acceleration are less safe, since the robot can achieve higher speeds more quickly.

Sometimes it is useful to have a larger acceleration for an "emergency stop". For this purpose, there is a separate deceleration parameter. Typically, it is the same as the acceleration parameter, but can be set to another value. Whenever the robot is commanded to zero velocity, the deceleration value is used.

Often it is desirable to limit the velocity that the robot is allowed, below the maximum achievable – for example, for safety. The robot absolute maximums are approximately 2.2 m/s and 720 deg/s. The limit velocities can be set to any value below this.

The acceleration and velocity limits can be set using low-level command packets, or through the Player driver configuration file. Unfortunately the position2d interface does not have the ability to set acceleration or velocity limits at the present time. The following table shows the relevant variables. Please note that the Player driver uses meters, while the controller uses mm for distance.

Firmware Parameter	**Player driver config**	**Default**	**Effect**
pTransVelMax mm/sec	`max_trans_vel` m/sec	`1000` mm/sec	Maximum rotational velocity allowed
pRotVelMax deg/sec	`max_rot_vel` deg/sec	`200` deg/sec	Maximum rotational velocity allowed
pTransAcc mm/sec^2	`trans_acc` m/sec^2	`400` mm/sec^2	Translation acceleration
pTransDecel mm/sec^2	`trans_decel` m/sec^2	`400` mm/sec^2	Translation deceleration
pRotDecel deg/sec^2	`rot_decel` deg/sec^2	`400` deg/sec^2	Rotation deceleration
pRotAcc deg/sec^2	`rot_acc` deg/sec^2	`400` deg/sec^2	Rotation acceleration

5.5.7. Motor PWM and PID control

The motors are driven by pulse-width modulation, which efficiently throttles full battery current to produce slower speeds. A higher PWM frequency exhibits less noise, but too high can cause overheating of the drivers. 16 KHz is a good compromise – only dogs and small children can hear this motor whine.

The PWM maximum duty cycle cuts the maximum current that can be delivered to the motors. For higher performance and faster speeds, set this number higher.

The controller uses a 200 Hz control loop to achieve velocity setpoints, using feedback from the encoders. The PID parameters control the responsiveness and stability of this loop. These parameters should be left unmodified, unless there is a very good reason for changing them – e.g., the robot carries a heavy payload.

PID values should not be changed by the user without contacting Videre Design. The motor control can easily be made unstable if these values are changed.

The PWM and PID parameters can be set with low-level packet commands, or through the Player driver configuration file. The following table lists the relevant parameter names and defaults.

Firmware Parameter	**Player driver config**	**Default**	**Effect**
pPwmWrap timer count	`motor_pwm_frequency` Hz	`16KHz`	Pulse frequency for motor current
pPwmMax timer count	`motor_pwm_max_on` fraction	`0.7`	Maximum duty cycle for PWM
pTransKp	`pid_trans_p`	`40`	Proportional value for translation PID
pTransKv	`pid_trans_v`	`80`	Differential value for translation PID
pTransKi	`pid_trans_i`	`0`	Integral value for translation PID

Firmware Parameter	Player driver config	Default	Effect
pRotKp	`pid_rot_p`	`30`	Proportional value for rotation PID
pRotKv	`pid_rot_v`	`60`	Differential value for rotation PID
pRotKi	`pid_rot_i`	`0`	Integral value for rotation PID

5.5.8. Saving settings

The values for some robot parameters in the configuration file can be saved into flash memory on the motor controller of the robot. The values that are saved are the rotation and translation acceleration and velocity, and the PID parameters.

```
save_settings_in_robot  0  [0 or 1]
```

If the value is 1, the configuration settings are saved in flash memory. Once saved, it is no longer necessary to specify them in the configuration file.

5.6. Analog and digital sensors

The Erratic controller has a complement of analog and digital sensors. Some of these are attached to fixed functions, such as measuring the battery voltage or the motor current. A few of them are free for use by other equipment that may be put onto the robot.

The analog and digital information may be accessed via the low-level packet interface, or through the Player driver and associated proxies. Some of the fixed functions have special Player interfaces. These are detailed in this section.

5.6.1. Battery voltage

The battery voltage is measured by a fixed analog input on the controller. Its value is returned with every motor information packet, in tenths of a volt. The nominal voltage is 12V. When the voltage goes below 11V, the LED starts blinking, and the batteries should be recharged.

The Player driver has a power interface that makes available the battery voltage to client programs. The `PowerProxy` class has a member function `GetCharge()` that returns the battery voltage as a floating-point number.

5.6.2. Analog and digital inputs

Analog and digital inputs are available through the low-level packet interface. A controller command turns the sending of the packets on or off.

The packet contains information about 4 analog channels and 3 digital channels. The analog channels return a count of [0,1023], which translates to a voltage from [0V,3.5V]. The digital channels return a value of [0,1], which is mapped into a voltage of 0 or 1. The Player `AioProxy` class accesses either of these values.

Input	Function	Value	Player proxy

Analog
Ch 0
Ch 1
	
	Left motor current
	Right motor current
		[0,1023]
		25 mA / cnt
		25 mA / cnt
			`AioProxy`
			`GetCount(ch)` or
			`GetVoltage(ch)`

Analog
Ch 2
Ch 3
	
	User input
	User input
		[0,1023]	`AioProxy`
			`GetCount(ch)` or
			`GetVoltage(ch)`

Digital
Ch 4
Ch 5
	
	Left floor sensor
	Right floor sensor
		[0,1]
		0 = floor
		1 = no floor
			`AioProxy`
			`GetCount(ch)` or
			`GetVoltage(ch)`

Input	Function	Value	Player proxy
Digital Ch 6	User input	[0,1]	`AioProxy` `GetCount(ch)` or `GetVoltage(ch)`

5.6.3. IR floor sensor

Two IR floor sensors at the front of the robot detect drop-offs such as a step down. The sensors have a fixed sensing distance about 8 cm below the floor plane. If they detect the floor, they signal a <0> on digital channel 4 (left) or 5 (right). If they do not detect a floor, they signal <1>.

The IR floor sensors are mapped by the Player driver to the IR interface. User programs can access these values using the `IrProxy` class. The values returned for members of this class are given in the table below. The IR channel for the left sensors is 0, and the channel for the right is 1.

IrProxy function	Interpretation
GetRange(ch) ch 0 = left ch 1 = right	0.1 m = floor 1.0 m = no floor
GetVoltage(ch) ch 0 = left ch 1 = right	0V = floor 1V = no floor

The placement of the IR floor sensors can be found with the `GetPose(ch)` function.

5.7. *Sonar sensors*

Erratic has an optional ring of 8 sonar sensors in the front of the robot. The placement of these sensors is given below in Figure 5-2.

The sonar sensors are MaxBotics EZ-1 sensors. They are moderately sensitive, able to detect objects out to 5 meters. However, it is recommended to only use readings that are less than 2 or 3 meters, since the probability of false positives is very high for longer readings in typical indoor environments.

The acceptance angle for the sonars is well-focused (around 30 degrees), with minimal side lobe response. The technical specifications for the sonars are on the Videre website.

There is a low-level packet interface to the sonar sensors. Sending a sonar command will turn the sonars on or off. When they are on, the controller continuously sends sonar data packets. Each packet contains one or more sonar range readings (in mm), indexed by the sonar number. The data rate for the sonars is 20 Hz.

The Player driver has a sonar interface that accumulates sonar readings and makes them available to client programs. The `SonarProxy` class can be

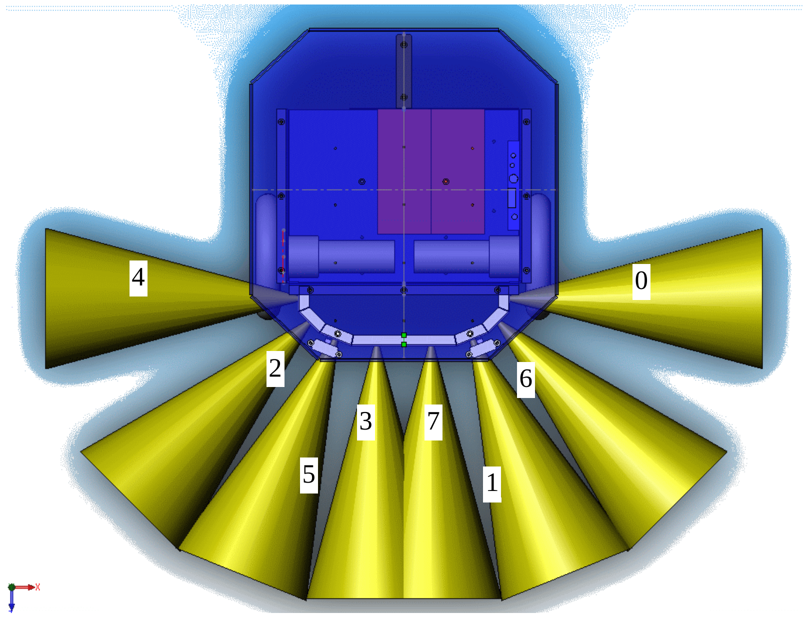


Figure 5-2 Sonar placement. The sonar numbers are given in the figure.

used to get the readings. Use the function

```
double SonarProxy::GetScan(int ch)
```

to retrieve the range of sonar <ch>. Range values are in meters. The placement of the sonars is as in Figure 5-2 above – the coordinates of each sonar can be retrieved with the `GetPose(ch)` function.

5.8. Servo motors

The Erratic controller supports a connection to 3 hobby-type servo motors. These motors can be controlled through low-level packet commands, or through the Player driver via the PTZ interface.

The low-level packet interface is documented in Section E.5. Each of the servos can be individually set to a position using these commands.

The Player PTZ interface can be used to control the servos. There are two PTZ interfaces, one for a full pan-tilt device on servos 1 and 2 (`ptz:0`), and one for a tilt-only device on servo 0 (`ptz:1`).

6. Technical Support

For technical support, please contact Videre Design by email or FAX.

Videre Design
865 College Avenue
Menlo Park, CA 94025
Fax: (650)323-3646

Email: support@videredesign.com and please CC: sales@videredesign.com

Appendix A Robot parameters

Parameters for the robot change the behavior of the robot in various ways, e.g., they set the maximum velocity for the robot.

Parameters are held permanently in flash memory on the embedded controller board of the robot. When the robot is turned on, these parameters are read into RAM on the controller, where they take effect.

Parameters can be changed in RAM by issuing various packet commands (Section E.6). The current parameters in RAM can be written to flash for permanent storage, again by a packet command.

All the parameters can be read out by requesting a configuration packet via a controller command – see Section 0.

Some of the parameters are available from the Player interface, via the configuration file (Section 5 and Appendix B).

A.1 Robot name parameters

The robot base has a set of parameters that define the robot's basic type. These parameters can be set by the user, but we recommend only changing the Name parameter, to identify your robot. There is a utility program for changing the name.

Parameter	Type	Default Value
Name	Null-terminated string	`Erratic-1`
Type	Null-terminated string	`Erratic`
Subtype	Null-terminated string	`Rev E`
Serial	Null-terminated string	`NNNN` (factory setting)

A.2 Firmware version

The version of the controller code is held in flash memory.

Parameter	Type	Default Value
Version	int	`1`
Subversion	int	`0`

A.3 Motion parameters

Motion parameters control the acceleration and velocity of the robot. These parameters act on rotation and translation as independent controls. For motion regimes where the velocity of each wheel is controlled independently, the translational settings are used.

All these values can be changed with the configuration file when Player starts up.

Firmware Parameter	Player driver config	Default	Effect
pTransVelMax mm/sec	`max_trans_vel` m/sec	`1000` mm/sec	Maximum rotational velocity allowed
pRotVelMax deg/sec	`max_rot_vel` deg/sec	`200` deg/sec	Maximum rotational velocity allowed
pTransAcc mm/sec^2	`trans_acc` m/sec^2	`400` mm/sec^2	Translation acceleration
pTransDecel mm/sec^2	`trans_decel` m/sec^2	`400` mm/sec^2	Translation deceleration
pRotDecel deg/sec^2	`rot_decel` deg/sec^2	`400` deg/sec^2	Rotation deceleration

Firmware Parameter	Player driver config	Default	Effect
pRotAcc deg/sec^2	`rot_acc` deg/sec^2	`400` deg/sec^2	Rotation acceleration

A.4 Motion control parameters

These parameters affect the underlying PID control loop of the robot, and the PWM (pulse-width modulation) frequency and maximum duty cycle.. This loop runs at 200 Hz, and provides fine motion servoing for velocity and position control of the wheels.

For testing purposes, these parameters can be changed by Player using the configuration file.

PID values should not be changed by the user without contacting Videre Design. The motor control can easily be made unstable if these values are changed.

Firmware Parameter	Player driver config	Default	Effect
pPwmWrap timer count	`motor_pwm_frequency` Hz	`16KHz`	Pulse frequency for motor current
pPwmMax timer count	`motor_pwm_max_on` fraction	`0.7`	Maximum duty cycle for PWM
pTransKp	`pid_trans_p`	`40`	Proportional value for translation PID
pTransKv	`pid_trans_v`	`80`	Differential value for translation PID
pTransKi	`pid_trans_i`	`0`	Integral value for translation PID
pRotKp	`pid_rot_p`	`30`	Proportional value for rotation PID
pRotKv	`pid_rot_v`	`60`	Differential value for rotation PID
pRotKi	`pid_rot_i`	`0`	Integral value for rotation PID

A.5 Saving parameters to flash

The current parameters can be saved to flash by issuing the `Save Params` packet command (Section E.6). Some of the parameters can also be saved via the Player driver configuration file (Section 5.1). Once saved, they become the default parameters when the robot is turned on.

A.6 Parameter structure

Parameters are held in a structure that can be read out on request to the controller. For reference, here is the structure.

```
struct parameters
{
  char pName[20];
  char pType[20];
  char pSubtype[20];
  char pSN[10];

  uint8_t pVersion;
  uint8_t pSubVersion;

  uint8_t pBaudHost;
  uint8_t pBaudAux;

  uint16_t pRevCount;
  uint16_t pWatchDog;         /* ms */

  int16_t pLowBattery;        /* tenths of volts */

  int16_t pStallVal;          /* PWM counts */
  int16_t pStallCount;        /* how long we stall, in
                              PERIOD ms increments */

  int16_t pPwmMax;            /* PWM counts; 0 will not
                                 work the motors */
  int16_t pPwmWrap;           /* PWM timer wrap value */

  /* settings for onboard control */
  int16_t pRotVelMax;
  int16_t pTransVelMax;
  int16_t pRotAcc;
  int16_t pTransAcc;
  int16_t pRotDecel;
  int16_t pTransDecel;

  /* PID parameters */
  int16_t pTransKp;
  int16_t pTransKv;
  int16_t pTransKi;
  int16_t pRotKp;
  int16_t pRotKv;
  int16_t pRotKi;

  /* sonar parameters */
  int16_t pSonarConv;   /* converts from cycles to
                           256*mm */

  /* some future expansion might take place through this
mechanism */
  char reserved[8];
};
```

Appendix B Player robot configuration file

This appendix contains the Player configuration file for the Erratic Rev G base. Other drivers, such as the Hokuyo or SICK Laser Range Finders, require their own sections in the driver.

See Section 5.1 and Appendix A for more information on the configuration file parameters.

```
# Example file instantiating the Videre robot driver
# All options you might want to use are instantiated here

# In general, the player standard units are
# translational velocity: m/s
# rotational velocity: deg/s

driver
(
  # Instance name of the driver
  name "erratic"

  # Standard player option, tells the driver to expose all
  # interfaces, and assigns them IDs
  provides [ "position2d:0"
             "power:0"
             "aio:0"
             "ir:0" ]

  # Serial port to use for connecting to robot
  # NOTE: using udev rules to set up this symlink
  port "/dev/erratic"

  # Maximum translational and rotational speeds accepted
  # during operation.
  # This is also capped by robot's internal hard limits,
  # potentially lower than
  # given here. For Erratic, it's about 2 m/s and 720 deg/s
  max_trans_vel 0.7
  max_rot_vel 100

  # The robot uses these acceleration values to achieve
  # desired velocity. There is a hard minium acceleration
  # as well as a top one, set by the robot.
  trans_acc 1.0
  rot_acc 200

  # If set, these specify the robot to use another acceleration
  # when target velocity is zero.
  # trans_decel 3
  # rot_decel 300

  # Parameters for the PD control loop onboard the robot.
  # When in individual wheel control, translation parameters
  # are used for each wheel control.
  # Warning: Changes can have unexpected effects, including
  # truly erratic behavior.
  pid_trans_p 40
  pid_trans_v 100

  pid_rot_p 40
  pid_rot_v 100

  # These set how to drive the wheel motors. Frequency affects
  # noise, and perhaps power efficiency.
  motor_pwm_frequency 16000
  motor_pwm_max_on 0.7

  # Set to 1 to have the driver use individual wheel control.
  # PD settings for translation will be used. This might be
  # useful depending on the kind of control loop in your
  # application, but probably you want to leave it at 0
  # (default).
  direct_wheel_vel_control 0

  # Setting this option to 1 saves the settings in the robot,
  # making permanent whatever settings state it is in after
  # settings of this config file are applied. The settings can
  # then be left out of other configuration files.
  # Note that some of the settings here are driver settings
  # and not robot ones, and are therefore not affected by a
  # save.
  save_settings_in_robot 0
)
```

Appendix C Player server autostart

The Player server can be autostarted at boot time on the robot. The advantage to doing this is that there is no need to ssh in to the robot to start Player. The disadvantage is that if there is a problem, there is no way to tell what went wrong. In this case, it is always possible to start a shell on the robot and check if Player is running, and if not, to start it and see what the problem is.

There is a script installed in `/etc/init.d/player` that will turn on autostart of the Player server during boot. This script has the Player command `/usr/local/bin/player` commented out. To enable autostart, uncomment this line.

Appendix D Test program

There is a standalone test program that exercises the robot controller via the low-level packet interface. The test program is a good way to check the functions of the robot, and can be used as an example program for low-level access via the packet protocol.

If the robot is tethered to a keyboard and monitor, or to a wired Ethernet, then be careful to only move the robot a small amount.

To invoke the program, connect the PC to the USB port of the controller (or use the onboard PC), and use:

```
./robot_client <comm port>
```

The <comm port> argument is

`/dev/erratic` or `/dev/ttyUSBn` for Linux

`comN` for MS Windows

If the controller is alive, the program will print a confirming message, and also some information about the robot (type, subtype, name, serial number, battery voltage).

Single-character commands send command packets to the controller. The character must be followed by <CR> to take effect.

Command	**Effect**
R,r	Turn on/off raw packet printing. The bytes of all packets sent from the controller are printed.
z	print a summary of the robot state, once
A,a	turn on/off analog packet printing
S,s	turn on/off sonar packet printing
i	increment forward velocity
m	decrement forward velocity
j	increment left turn velocity
l	decrement left turn velocity
space, k	stop motors
B,b	right/left drift increment

Appendix E Erratic controller packet communications

This Appendix documents the format of the Erratic controller communication protocol. Further details can be found in the source code for the Erratic driver, in `player/server/drivers/mixed/erratic`.

E.1 Communication channel

Communication is through a USB 1.1 connection. Host PC drivers convert the USB connection into a serial connection. The name of the connection on Linux systems is

```
/dev/erratic
```

This device name is a symlink to `/dev/ttyUSBn`. The symlink is performed automatically at boot-up by rules in `/etc/udev/rules.d/60-symlinks.rules`.

On MS Windows systems, the connection is

```
comX
```

where X is the next highest serial port number.

The device communicates at a native speed of 38.4 Kbaud, 8 bits per byte, no parity, 1 stop bit.

E.2 Special byte definitions

Some bytes have a special meaning at certain points within a packet (see subsequent sections). These are given symbolic names as follows.

SYNC0	0xFA
SYNC1	0xFB
END	0xFE
ARG	0x3B
NARG	0x1B
SARG	0x2B

When integers are sent as arguments, they are always inserted into the byte stream as 2 bytes. The first byte is the low byte, the second byte is the high byte of the integer.

E.3 Packet protocol

The protocol is based on command packets that are sent to the controller, and information packets that are received by the host PC. All packets have the following format.

SYNC0
SYNC1
count
count-2 bytes of data
checksum (2 bytes)

E.4 Checksum calculation

The checksum is calculated on the full packet, starting after the SYNC bytes, and includes the byte count. The checksum algorithm is given here in C code. The argument "n" is the number of bytes, and "b" is the vector of bytes in the packet.

```
int calc_chksum(int16_t n, unsigned char *b) {
  uint16_t c = 0;
  while (n > 1) {
    c += (*(b)<<8) | *(b+1);
    n -= 2;
    b += 2;
  }
  if (n > 0) c = c ^ (uint16_t)*b;
  return c;
}
```

E.5 Command packets

A command packet consists of a command byte, followed by an argument. The command argument can be either NULL, a 16-bit positive integer, a

No arg	Positive int arg	Negative int arg	String arg
SYNC0 SYNC1 command END	SYNC0 SYNC1 command ARG low byte high byte	SYNC0 SYNC1 command NARG low byte high byte	SYNC0 SYNC1 command SARG count N < 100 byte 0 … byte N-1

Table -1 Format for command arguments.

16-bit negative integer, or a string of bytes. Table -1 shows the format of these command types.

Commands should not be sent to the robot at a rate faster than 20 Hz.

E.6 Controller command reference

A listing of the commands that can be sent to the robot, and the interpretation of their arguments, is given in the table below.

Command	Num	Arg	Effect
Open	1	None	Resets origin of odometry, turns motors off. Should be issued when first connecting to the robot.
Close	2	None	Turns off motors. Should be issued when disconnecting from the robot.
Config	18	None	Returns a configuration packet (see Section 0).

Command	Num	Arg	Effect
Enable	4	ARG	Enables or disables commands to control the motors. With an argument of 0, stops the motors and disables motor power (motors go limp). With an argument of 1, enables motor power (motors are engaged, the robot responds to motion commands). This command must be issued before any motion takes place.
Set acc	5	ARG	Set translational acceleration, in mm/sec/sec.
Set max vel	6	ARG	Sets the maximum translational speed of the robot, in mm/sec.
Set racc	23	ARG	Sets rotational acceleration, in deg/sec/sec.
Set max rvel	10	ARG	Sets the maximum rotational speed of the robot, in deg/sec.
Stop	29	None	Stop all translation and rotation.
Trans vel	11	ARG NARG	Set translational velocity, signed mm/sec Switches to TR mode
Rot vel	21	ARG NARG	Set rotational velocity, signed deg/sec Switches to TR mode
Trans pos	8	ARG NARG	Set translational distance, signed mm relative to current position. Switches to TR mode
Rot pos	12	ARG NARG	Set rotational heading, signed deg in odometer system.

Command	Num	Arg	Effect
			Switches to TR mode
Rot relative pos	22	ARG NARG	Set rotational heading, signed deg relative to current heading. Switches to TR mode
Wheel vel	32	ARG	Set velocity of two wheels independently. First byte is signed left velocity (2 cm/sec), second is signed right velocity (2 cm/sec). Switches to VEL2 mode
Origin	7	None	Reset odometry origin to (0,0,0).
Encoder	19	ARG	1 turns on encoder packets, 0 turns them off.
Analog	71	ARG	1 turns on analog info packets, 0 turns them off.
Sonar	28	ARG	1 turns on sonar info packets, 0 turns them off.
Servo	75	STRING	Sets a servo motor position. First byte is the servo number (0-2), second two bytes are the position as a 2-byte integer.
PWM freq	73	ARG	Sets motor PWM frequency. Values should be from 4000 to 16000.
PWM duty	74	ARG	Sets max PWM duty cycle, in percent.
Trans PID P	80	ARG	PID proportional parameter for translational motion.
Trans PID D	81	ARG	PID differential parameter for translational motion.
Trans PID I	82	ARG	PID integral parameter for translational motion.
Rot PID P	83	ARG	PID proportional parameter for rotational motion.
Rot PID D	84	ARG	PID differential parameter for rotational motion.
Rot PID I	85	ARG	PID integral parameter for rotational motion.
Robot name	90	STRING	Robot name, max 20 chars.
Robot type	92	STRING	Robot type, max 20 chars [Erratic]
Robot subtype	93	STRING	Robot subtype, max 20 chars [Rev E]
Robot serial	91	STRING	Robot serial number, max 20 chars
Save params	72	None	Save current controller parameters to flash storage.

E.7 Motor info packets

Motor packets are sent continuously from the controller at the rate of 20 Hz. These packets are always sent, there is no way to turn them off.

Motor packets contain basic information about the state of the robot and its motors, as well as integrated odometry information. Values larger than a byte are always inserted with the least-significant byte first.

Type	Value
byte	Header: 0x8n. <n> is 2 if the robot is not moving, and 3 if it is.
3 byte integer	Odometry X position, in signed mm

Type	Value
3 byte integer	Odometry Y position, in signed mm
2 byte integer	Odometry heading, in degrees [0-360]
2 byte integer	Left wheel velocity, signed mm/sec
2 byte integer	Right wheel velocity, signed mm/sec
byte	Battery voltage, 1/10 volts
byte	Motor stall status – bit 2 set for left wheel stall, bit 3 set for right wheel stall
2 byte integer	Heading target, in deg [0-360]. Only valid if rotational position control command has been sent.
2 byte integer	Distance target, in signed mm. Only valid if translational distance control command has been sent.

E.8 Configuration packet

A configuration packet is sent in response to a configuration command (18). The configuration packet has all the information of the configuration structure (Section A.6)

Type	Value
byte	0x20
byte	Parameter version
N bytes	Configuration structure, as a sequence of bytes

E.9 Encoder packet

Encoder packets are sent continuously when the encoder packet command has been issued with argument 1. Encoder packets give the raw 24-bit encoder values for the left and right wheels.

Type	Value
byte	Header: 0x90
3 byte integer	Left wheel encoder value.
3 byte integer	Right wheel encoder value

E.10 Analog packet

Analog packets are sent continuously when the analog packet command has been issued with argument 1. Analog packets give the state of the 4 A/D converters, as well as 3 digital inputs.

Analog values are from 0-1023 (10 bits). This specifies a linear range from 0 – 3.5V.

Type	Value
byte	Header: 0x9A
byte	Number of analog readings [4]
2 byte integer	First analog reading – left motor current. Current is approximately 25 mA x Value.
2 byte integer	Second analog reading – right motor current. Current is approximately 25 mA x Value.
2 byte integer	Third analog reading – user input.
2 byte integer	Fourth analog reading – user input.

Type	**Value**
2 byte integer	Digital I/O bit 0 – left floor sensor. 0 is floor, 1 is no floor bit 1 – right floor sensor. 0 is floor, 1 is no floor bit 2 – user input

E.11 Sonar packet

Sonar packets are sent continuously when the sonar packet command has been issued with argument 1. Sonar packets give the state of the 8 sonars in the front sonar ring.

Sonar values are in mm.

Type	**Value**
byte	Header: 0x9B
byte	Number of sonar readings in the packet.
byte	Sonar number
2 byte integer	Sonar range, in mm
byte	Sonar number
2 byte integer	Sonar range, in mm
…	

Appendix F Power Port

The power port on the robot controller board is an 8-pin connector that provides power for peripherals. The header is a polarized AMP MTA header, with spacing of 0.1 inches.

The power pins assignments are given in Figure A-1. Normally, peripherals that are ordered with the robot are connected to the power port. The user can also connect his own peripherals.

If no peripherals are connected, an empty connector should be plugged in, to prevent any metal from shorting the power pins.

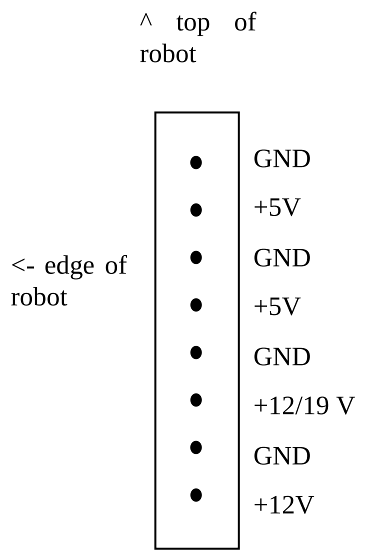
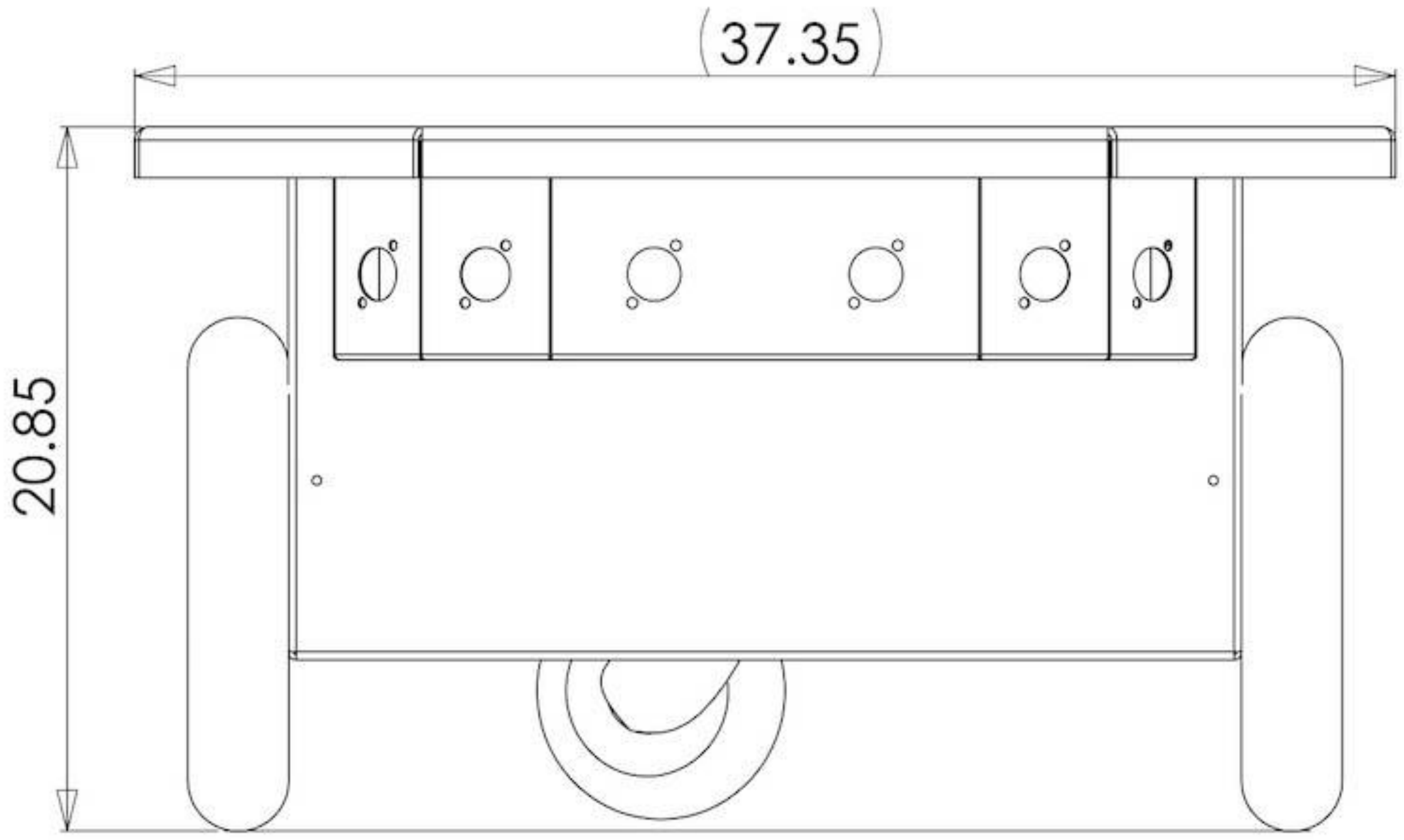
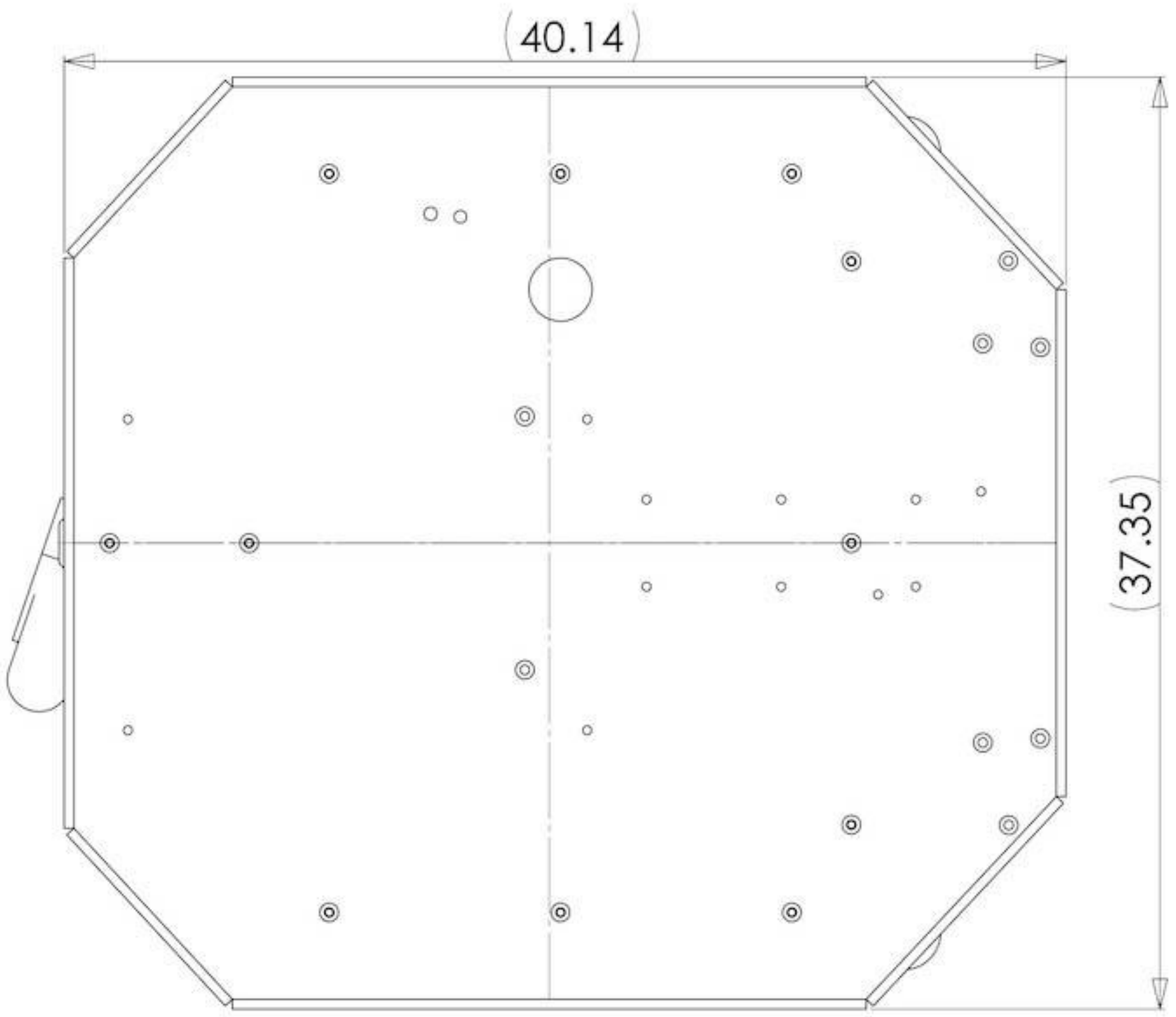
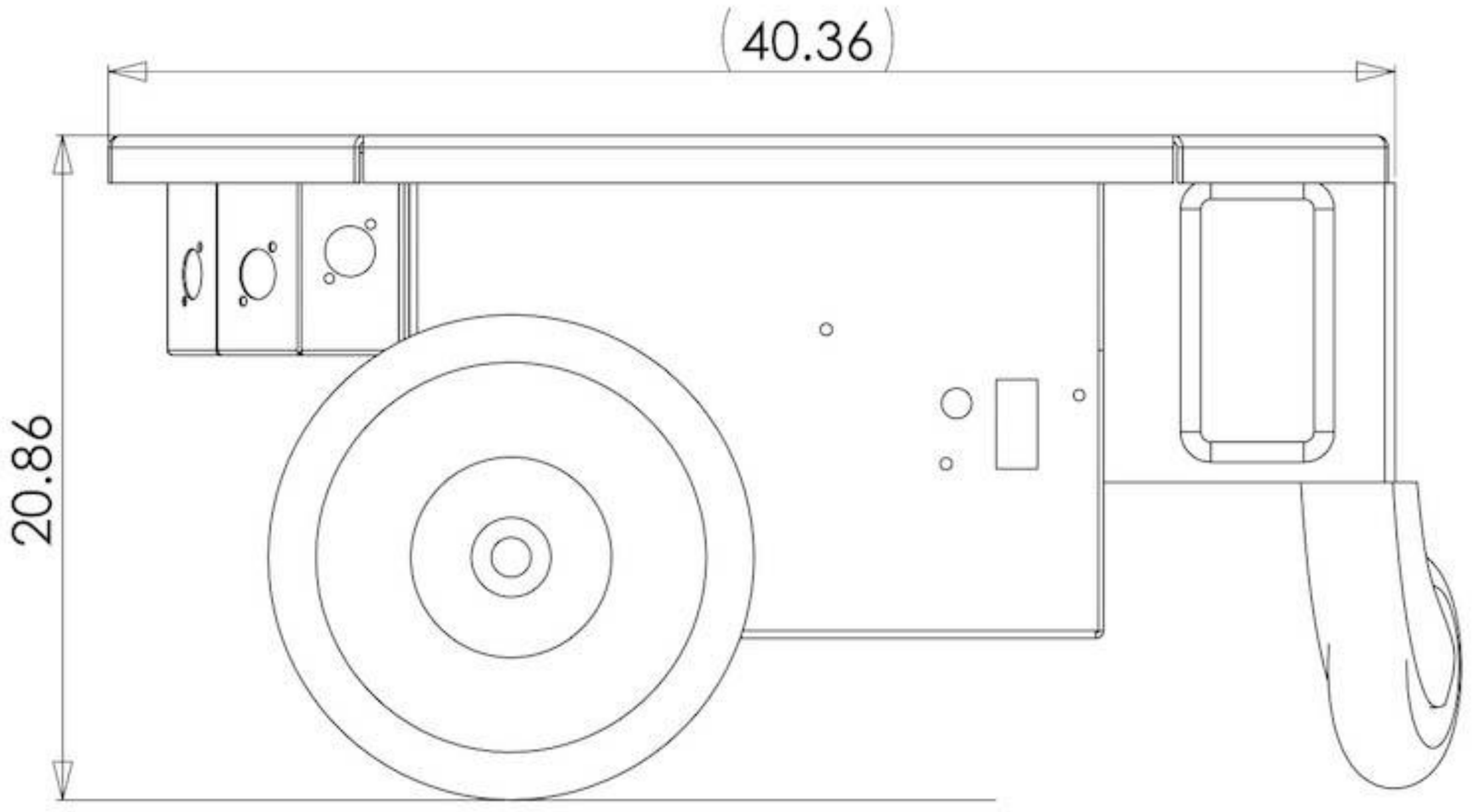


Figure A-1 Power port on top of robot. Pin spacing is 0.1 inches.

Appendix G Erratic base dimensions




40.14
37.35


40.36
20.86