

# The Mobile Robot User's Manual

March 29, 2000

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Features and specifications</b>	<b>2</b>
2.1	General . . . . .	2
2.2	Locomotion . . . . .	2
2.3	Computing & communication . . . . .	2
2.4	Sensing . . . . .	2
2.4.1	IR sensors . . . . .	2
2.4.2	SONAR . . . . .	3
2.5	Power . . . . .	3
<b>3</b>	<b>The Monitor Program</b>	<b>3</b>
3.1	Locomotion . . . . .	3
3.2	Dead reckoning . . . . .	4
3.3	Communication . . . . .	4
3.4	Sensing . . . . .	4
<b>4</b>	<b>Basic operation</b>	<b>4</b>
4.1	Power and charging . . . . .	4
4.2	General operation . . . . .	5
<b>5</b>	<b>Interactive use</b>	<b>5</b>
<b>6</b>	<b>Programming the robot</b>	<b>5</b>
6.1	Files . . . . .	6
6.2	Library functions . . . . .	6
6.2.1	Locomotion . . . . .	6
6.2.2	Dead Reckoning . . . . .	7
6.2.3	Sensors . . . . .	7

## 1 Introduction

This document describes the mobile robot hardware, basic operation of the mobile robots, interactive operation (teleoperation) of the robots, and how to program the robots.

These are small differential drive robots with a single caster in the front. There are five IR sensors on the front and sides of the robot and one SONAR sensor on the front of the robot which can be panned from side to side. They have an on-board microcontroller which controls the motors,

reads the sensors, and communicates with a desktop computer or other robots via a wireless radio link.

Ultimately, this system should consist of a set of five (working) robots, a desktop computer, and a vision system to measure robot position via an overhead camera. Users should (ultimately) be able to write robot programs and cross-compile them on the desktop computer, download them to the robots via the radio link, and then execute and monitor the robots from the desktop computer.

## **2 Features and specifications**

The following subsections describe in some detail the hardware components of the robot. You may want to just skim this section the first time you read this document and continue to the "Basic Operation" section.

### **2.1 General**

The robot is 7 inches wide, 8.25 inches long, and about 7 inches tall (not including the antenna). It should weigh somewhat less than 1.5 Kg.

### **2.2 Locomotion**

The robot is differential drive (i.e. there are two independently driven drive wheels). Unlike many other differential drive robots, however, the drive wheels are towards the back of the robot, and there is a single (passive) caster at the front of the robot.

The wheelbase is approximately 5.5 inches, and the wheel diameter is approximately 6 cm. The maximum speed of the robot is approximately 50 cm/sec.

The motors have magnetic encoders which provide 16 pulses per motor shaft revolution in quadrature. There is a 41:1 planetary gearhead between the motor and the wheel.

### **2.3 Computing & communication**

The onboard computer is a F16mite v. 3.0 made by Intec Automation. It features a 25 MHz Motorola 68HC16 microcontroller. (For those familiar with the 68HC11, this is similar but faster, with more features, and also is a 16 bit processor.) The microcontroller provides digital I/O lines, A/D conversion, two PWM outputs, input capture, serial output hardware, etc. It has 128K of FLASH and 128K of RAM.

The computer plugs into a "motherboard" which connects the processor I/O to the robot components. There are connectors on this board to access the serial port of the computer as well as a hardware-debug port. These connectors, however, should not be needed unless you are doing some low level software or hardware development.

The RF module is a Radiometrix RPC module which works at a frequency of 433 MHz. It can transmit at approximately 2.5 Kbaud and interfaces to the computer via digital I/O lines.

### **2.4 Sensing**

#### **2.4.1 IR sensors**

There is one IR sensor on the front of the robot and two on each side. The IR sensors consist of a separate IR LED and an IR photodetector. When enabled, the IR LED emits IR light at 880 nm (?) and the photodetector circuit produces a voltage proportional to the amount of IR light reflected. This voltage is sampled by the A/D conversion on the computer to produce a number between 0 and 255. The IR sensors are not constantly enabled to conserve battery power.

## 2.4.2 SONAR

There is one SONAR sensor (a Polaroid 6500 ranging board with an instrument grade transducer) mounted on the front of the robot on a hobby servo which can pan the SONAR side to side (total range of about 180 degrees, 90 degrees to each side).

The SONAR emits a burst of ultrasonic sound and then listens for an echo. If the echo is “strong enough” the SONAR sensor indicates that it has received the echo. The time between sending the burst and receiving the echo is measured by an input capture circuit on the microcontroller (in ticks of a clock whose frequency in the processor frequency divided by 256).

## 2.5 Power

The robot is powered by a single 12V 1.2 AH sealed lead acid battery which should provide power for 2–3 hours. There are DC-DC converters on the “motherboard” separately regulated power for the electronics and for the SONAR and the motors.

# 3 The Monitor Program

The on-board computer has a monitor program which takes care of operating the robot hardware and provides certain services to user programs. User programs should not access the hardware directly but should do so through an appropriate monitor function.

This following subsections detail the monitor capabilities and services. Detailed interface routines are described in the programming section.

## 3.1 Locomotion

There are several different ways of commanding robot motion:

1. “displacement” control — you can tell the robot to go forwards or backwards some distance, tell it to turn “in place” some angle, or tell it to stop. There are several caveats to this mode of control:
  - The robot is not guaranteed to go exactly in a straight line.
  - The robot is not guaranteed to travel the exactly distance or angle that you specify.
  - A turn rotates the robot about a point on the wheel axis, midway between the two wheels. Since the wheels are in the back of the robot, it does not turn in place.

This mode of control can either be either blocking (i.e. wait until the motion is completed) or nonblocking. Nonblocking calls allow the user program to perform other computations while the robot is in motion. The program (at a later time) can check to see whether the motion is complete or can wait for the motion to finish.

2. velocity control — you can command a net translational and rotational velocity for the robot. Note that there are some dynamics to the robot — it cannot accelerate or decelerate instantaneously.
3. wheel speed control — you can specify a wheel speed for each wheel. Note that there are some dynamics to the robot — it cannot accelerate or decelerate instantaneously. This mode of operation is not recommended; use one of the other two unless there is some special reason that you need this mode.

The actual implementation has a velocity controller that uses feedback from the encoders to maintain the reference velocity for each wheel. The velocity control mode (above) transforms the net velocities into wheel velocities and applies some more gently acceleration and deceleration. The “displacement” control creates a velocity profile that should achieve the given displacement; this profile is fed (point by point without feedback) to the velocity controller. Ultimately, we will have real position and/or trajectory control (i.e. a controller that keeps the robot on a given path or drives it to a given point).

### **3.2 Dead reckoning**

The robot keeps track of the readings from the motor encoders to calculate its position. When the robot is turned on, its orientation is 0, and it is at the origin in its local coordinate frame. The  $x$  axis of this frame points straight ahead; the  $y$  axis points to the robot’s left.

The position of the robot can be set arbitrarily; future robot movement will increment the position from its current location.

Remember that dead reckoning error accumulates with robot movement due to wheel slip and inaccurate calibration.

### **3.3 Communication**

The monitor implements a simple protocol for the wireless radio transmitters that can send a message of upto 400 (?) bytes. The recipient of the message can be specified (including sending a message to everyone) but the hardware inherently provides broadcast communication. Robots will simply ignore any message not intended for them.

### **3.4 Sensing**

Both IR and SONAR readings take time: the IR readings require a analogue to digital conversion, and a SONAR reading must wait for the echo to return. There will ultimately be blocking and nonblocking versions of commands to get these readings.

## **4 Basic operation**

### **4.1 Power and charging**

When you come into the lab to use the robot, you will find it at the charging station. To use the robot, do the following steps:

1. Make sure the power switch is off! The power switch is located at the back of the robot.
2. Unplug the charger from the back of the robot.
3. Start your program.
4. Turn on the robot.

When you are finished using the robot:

1. Turn the robot off
2. Return the robot to the charging station
3. Plug the charger into the charging jack

Some clarifications and caveats regarding power and charging:

- The charger has two modes: a *fast* charge mode and a *trickle* charge mode. If the “fast” light on the charger is off, then it is in trickle charge mode, and the battery is fully charged. Otherwise, it is in fast charge mode. It will take 4–5 hours to completely charge a battery, so it is important to plug in the robots when you are done.
- The sealed lead acid battery is not like nickel-cadmium or nickel-metal-hydride batteries — it should never be completely discharged, and it does not have any memory effect.
- Do not attempt to change the battery!
- When the charger is plugged in, there is no power going to the robot electronics.
- There is a reason for starting your program before turning the robot on: if the battery voltage is too low, then the robot will send a message saying so, and then it will not accept any commands. If you haven’t started your program, then you won’t see this message, and you will be confused. The `init()` routine (which your program must start with) will listen for and report this message. You should then turn off the robot, return it to the charging station, and plug the charger in.

## 4.2 General operation

- There is a reset switch (a pushbutton switch) on the top of the robot. This resets the onboard computer.
- Do not let the robot run into things! Either pick it up or hit the reset switch.
- Do not let the antenna for the RF module get bent or mangled. Be careful when the robot is near chairs or tables: the robot might under it but the antenna won’t!
- Do not run the robot on top of a desk or workbench unless you have put the robot in its stand (to keep it from driving off the edge!)

## 5 Interactive use

The `winmon` program in the `/projects/mr` directory that allows you to operate the robot interactively. This is a menu driven program; all commands are preceded by a slash character.

Using this program, you can drive the robot around, read the sensors, etc.

## 6 Programming the robot

Currently, the only mode for programming the robot is to write a program on the desktop (linux) machine that sends commands to the robot (and receives information from the robot) via the wireless radio link. This will sort of be like a program that automatically types commands into the `winmon` program except that the interface is done through a set of C library routines.

Ultimately, we should have a development environment where we can write code that will run directly on the robots, cross-compile this code on the desktop, download the code for each robot through the RF link, and then run and monitor the robots from the desktop.

## 6.1 Files

In the `/projects/mr/example` directory, there is a makefile and an example program that runs the robot. You should copy files these to your home or project directory. You can then compile and run this program with the commands:

```
make example
./example
```

After you have started this program, you can then turn on the robot. Make sure you put the robot down where it will have some room to move. The robot will do some simple task like moving in a square.

This directory also includes subdirectories for include files, the library files, and the library sources.

## 6.2 Library functions

All distances are specified in millimeters, angles in radians times 1000, and time in seconds. Therefore, velocity is specified in millimeters per second, acceleration in millimeters per second squared, and angular velocity in radians times 1000 per second.

### 6.2.1 Locomotion

- “Displacement” control — these calls all block (i.e. don’t return until the motion is complete). The basic motion commands are:

```
- void stop()
- void straight(int dist)
  dist is a signed distance (positive goes forwards, negative goes backwards).
- void turn(int angle)
```

The following commands specify parameters for the velocity profiles that are generated.

```
- void setMaxVelocity(int vel)
- void getMaxVelocity(int *vel)
- void setAcceleration(int acc)
- void getAcceleration(int *acc)
```

The maximum velocity of the robot is 50cm/sec; the maximum acceleration is ??? The defaults will be ...

- Velocity control — these commands give an appropriate setpoint for the velocity of each wheel and then return

```
- void setVelocity(int v, int w)
- void getVelocity(int *v, int *w)
```

- Wheel speed control — these commands set the wheel speed and then return.

```
- void setMotorSpeed(int ls, int rs)
- void getMotorSpeed(int *ls, int *rs)
```

`ls` and `rs` (the left and right wheel velocity) should be signed integers between -255 and 255 inclusive.

## 6.2.2 Dead Reckoning

- `void getPos(int *x, int *y, int *theta)`
- `void setPos(int x, int y, int theta)`

## 6.2.3 Sensors

- IR sensors — the IR sensors can be read with the command:

```
void readIR(int which, int readings[5])
```

which blocks until it gets the readings. You must specify which IR sensors you want to read. (This is set up this way to conserve power.) The which argument can be constructed from the symbols:

```
#define FRONT_IR      8
#define FRONT_RIGHT_IR 4
#define BACK_RIGHT_IR 16
#define FRONT_LEFT_IR 1
#define BACK_LEFT_IR  2
#define ALL_IR        31
```

The symbols for individual IR sensors can be logically OR'ed together to specify multiple sensors to read. The ALL\_IR symbol reads all the IR sensors. The specified sensors are all read simultaneously.

Needless to say, if you don't specify that a sensor should be read, then the value returned in that position in the array isn't valid. (What is the order of the values in the array?)

- SONAR — the SONAR sensor can be read with the blocking command:

```
void readSonar(int *delay)
```

where the delay is the number of input capture clock cycles between the ultrasonic burst and when the echo was detected.

The sonar can be panned from side to side with the commands:

```
void setSonarPan(int angle)
void getSonarPan(int angle)
```

where angle is a signed integer; an angle of 0 is straight ahead. We don't get any feedback from the servo that positions the sonar, so there is a small time delay to ensure that the SONAR has reached its final position before this routine returns.