PROGO

Applications Manual

for the

TJ PRO Robot

ICC11 Version

By

Keith L. Doty

Gainesville, Florida Phone 407-672-6780

AGREEMENT

This is a legal agreement between you, the end user, and Mekatronix™. If you do not agree to the terms of this Agreement, please promptly return the purchased product for a full refund.

1. **Copy Restrictions. N**o part of any Mekatronix™ document may be reproduced in any form without written permission of Mekatronix™. For example, Mekatronix™ does not grant the right to make derivative works based on these documents without written consent.

2. **Software License.** Mekatronix™ software is licensed and not sold. Software documentation is licensed to you by Mekatronix™, the licensor and a corporation under the laws of Florida. Mekatronix™ does not assume and shall have no obligation or liability to you under this license agreement. You own the diskettes on which the software is recorded but Mekatronix™ retains title to its own software. You may not rent, lease, loan, sell, distribute Mekatronix™ software, or create derivative works for rent, lease, loan, sell, or distribution without a contractual agreement with Mekatronix™.

3. **Limited Warranty**. Mekatronix™ strives to make high quality products that function as described. However, Mekatronix™ does not warrant, explicitly or implied, nor assume liability for, any use or applications of its products. In particular, Mekatronix™ products are not qualified to assume critical roles where human or animal life may be involved. For unassembled kits, you accept all responsibility for the proper functioning of the kit. Mekatronix™ is not liable for, or anything resulting from, improper assembly of its products, acts of God, abuse, misuses, improper or abnormal usage, faulty installation, improper maintenance, lightning or other incidence of excess voltage, or exposure to the elements. Mekatronix™ is not responsible, or liable for, indirect, special, or consequential damages arising out of, or in connection with, the use or performances of its product or other damages with respect to loss of property, loss of revenues or profit or costs of removal, installation or re-installations. You agree and certify that you accept all liability and responsibility that the products, both hardware and software and any other technical information you obtain has been obtained legally according to the laws of Florida, the United States and your country. Your acceptance of the products purchased from Mekatronix™ will be construed as agreeing to these terms.

MANIFESTO

Mekatronix™ espouses the view that the personal autonomous agent will usher in a whole new industry, much like the personal computer industry before it, if modeled on the same beginning principles:

- Low cost,
- Wide availability,
- Open architecture,
- An open, enthusiastic, dynamic community of users sharing information.

Our corporate goal is to help create this new, exciting industry!

Gainesville, Florida Phone 407-672-6780

WEB SITE: http://www.mekatronix.com
Address technical questions to tech@mekatronix.com
Address purchases and ordering information to an authorized Mekatronix Distributor http://www.mekatronix.com/Distributors

TABLE OF CONTENTS

LIST OF FIGURES

1 SCOPE

This manual is intended for teachers, educators, university and advanced high school students, hobbyists, researchers and anyone else interested in advancing the infrastructure of robotics and advanced technology in our society. The goal of this manual is to enable teachers and professors, from middle school through the university level, to develop exciting, stimulating, and entertaining instructional materials for hands-on laboratories using embodied, intelligent, autonomous mobile robots.

PROGO™ provides a vehicle for beginners to write sophisticated programs for the TJ PRO™. Examples in this manual will illustrate all the PROGO™ statements along with the *TJ PRO™ Robot Kernel*, which consists of TJ PRO™ motion and sensor commands.

This manual assumes that you have a copy of the *PROGO™ Language Reference Manual* on hand. A quick read of that manual will help you to grasp the language quickly. That manual also lists the *TJ PRO™ Robot Kernel*. If you want to jump into programming with PROGO™ immediately, you can refer to the language summary at the end of that manual and work through the applications here without delay. Ultimately, you may have detailed questions about the language syntax and may need to consult the *PROGO™ Language Reference Manual* more perspicaciously.

1.1 Requirements to run PROGO™

You can use either the DOS or the WIN95 version of the ICC11 C-compiler to run PROGO™. This manual assumes you have the DOS version of ICC11, operate from WIN95, and that you use the Mekatronix WIN95 application program called the High-Speed Serial Downloader (HSSDL11) to load your robot from a personal computer COM port.

If you work strictly in a DOS environment, you will need the PCBUG11 freeware package to download "`.s19`" files into the robot. PCBUG11 is shipped to you in the TJ PRO™ software distribution package. Further, in order for your robot to talk to the PC, you will need a terminal simulation program like Kermit (not shipped or included) to run on your PC.

For complete details about configuring the TJ PRO™ with your computer, see the *TJ PRO™ Users Manual* (available free from www.mekatronix.com) or the *TJ PRO Robot Education Manual Using ICC11* (available from our distributors).

1.2 TJ PRO™ Specific PROGO™ Applications

Each Mekatronix robot requires separate underlying robot sensor and actuator driver software, called the *Robot Kernel* to support PROGO™. This manual supplies the software package required to drive the TJ PRO™ Robot. Those applications that use specific TJ PRO™ Robot Kernel commands not shared by other Mekatronix robots will not execute properly on the other robots. The PROGO™ language statements, however, will remain unchanged across platforms. Porting a robot program across robot platforms can be as simple as changing the robot specific

commands to as complex as reprogramming substantial portions of the application, depending on the resource similarities between the robots.

2 INTRODUCTION

An autonomous mobile robot, similar to its distant carbon-based life forms, exhibits three principal features, 1) machine actuation, 2) machine perception, and 3) machine cognition. In anthropomorphic terms, these translate to acting, sensing, and thinking. The philosophical and scientific issues as to whether a machine can actually think, perceive or act with intent is beyond the scope of this manual.

Programming behaviors is what autonomous mobile robots is all about, or, at least a substantial part of what it is all about! Without being technical, a behavior is whatever the robot does. The emphasis is on action. From the engineering viewpoint, you want to program behaviors that produce useful results. Make a robot vacuum cleaner, or a robot valet. With an artistic eye, you want to program behaviors that esthetically please or excite. Why not make TJ PRO™ dance? Have it perform a ballet on wheels.

From the scientific perspective you can inquire about the scope of machine intelligence and test your theories on a real robot. Out of intellectual curiosity and the creation urge, you might want to develop physically embodied *animats*, or artificial animals. Develop your own ecology with predator robots that drain the prey robots' batteries and prey robots that hide and avoid predator robots and seek battery recharging stations as food sources. Or, you can tailor a robot to enter many of the robot contests around the world. Many of these contests require manipulation and sensors not supplied with TJ PRO™. But, with one or more Mekatronix MSCC11 single chip computers controlling the additional sensors and servo driven manipulation devices, a TJ PRO™ can often be expanded to meet contest requirements.

The text does not presume extensive experience in programming. If you have no experience at programming, you should read the *PROGO™ Language Reference Manual* first. Depending on your level of sophistication, you may need to have a capable programmer tutor you for an hour or so to get you started with the basics. Neither this manual nor the language reference manual actually teaches programming, an effort beyond the scope of both. Read and applied together, however, the two manuals should be sufficient to enable you to read and write elementary PROGO™ programs in a short time. You will pick up programming almost by "osmosis" just using your natural language skills.

As you progress through the manual you will find yourself in the following program development cycle:

1. Specify a problem
2. Determine the data and algorithms to solve the problem
3. Compose and edit program,
4. Compile program,

5. Load program,
6. Execute program and test: observe and note errors, possible corrections, and enhancements,
7. Repeat cycle, back to item 1 until satisfied.
8. Document program, before, during and after items 1 through 7!

To a hobbyist, documentation (item 8) may seem a drag, but even a modest amount of documentation can save considerable effort later. Most programmers forget coding details after a few months and often must reinvent a solution to solve a similar problem. A documented program will facilitate the reuse or modification of old code to new applications.

The next sections breifly describe how to implement the program development cycle using your TJ PRO™ robot and personal computer.

2.1 What Connects PROGO™ to C?

ICC11 is a C cross compiler that executes on a personal computer. The ICC11 preprocessor converts valid PROGO™ programs into valid *C* programs by combining the PROGO™ program with the `PROGO.c` file. The ICC11 compiler then translates the resulting C program into MC68HC11 object code, and more importantly, Motorola's "`.s19`" file format. All Motorola's boot loaders work on "`.s19`" files, so this is important. You can load "`.s19`" files directly into the robot computer via a Mekatronix MB2325 serial board and serial communications bus using the Mekatronix HSSDL11. The compiler also produces other handy outputs, listings, assembly code and object files to name the more important ones. If you are sophisticated enough to read those outputs, then you may be 1) teaching others PROGO™, 2) interested in learning PROGO™ as an easy to use robot language. Robot programming instructors may have to refer to these outputs to help debug the more subtle errors that can arise in programming. Careful observation of robot behavior, in most instances, provides the best insight into sources of error in your programs and usually leads to faster debugging.

Caution

ICC11 is not a quite a Standard ANSI C, so be careful of language limitations using ICC11. These differences are discussed in the ICC11 Users Manual.

3 TJ PRO™ EXPERIMENTAL SETUP

You will need a PC running Windows, ICC11 for DOS or Windows, a Mekatronix MB2325 serial communications board, a serial cable, a 6-wire serial cable, and a small box to hold TJ PRO™'s wheels off a desktop surface. Refer to the *TJ PRO™ Users Manual*[1] for instructions on how to configure these components into a working system. This manual assumes you have

1) Installed TJ PRO™ ICC11 distribution software,
2) Read and understood the *TJ PRO™ Users Manual* and have it available as a handy reference,
3) The *PROGO Language Reference Manual* readily available,

[1] You can download all Mekatronix manuals free from the web site: http://www.mekatronix.com

4) Connected the robot through the six wire serial cable, the MB2325 Communications board and a serial cable to your PC on COM1. (If you choose a different COM port be alert to slight changes, if any, required in the code and procedures used here.)

The TJ PRO™ robot does not move well on rugs or rough surfaces. You will also get longer battery life if you restrict the robot motion experiments to smooth flat surfaces.

Caution!

Do all robot moving experiments on a tile, wooden, or otherwise smooth flat surface!

3.1 About Batteries and Bench Testing Programs

Only use six AA nickel-cadmium rechargeable batteries to power the robot. Always keep the robot's batteries charged. While bench testing, keep the wheels off the bench and the charger plugged into the robot. This will insure many hours of testing and debugging without having to wait six hours for the batteries to charge or the hassle and expense of changing out the batteries. When leaving the bench for extended periods of time, turn the robot power off with the charger plugged into the robot. Leave the DOWNLOAD/RUN switch in RUN mode. The charger will keep the memory indefinitely, so if a program is in memory, it will be there when you power up again. This saves the hassle, minimal to be sure, of loading the program into memory every time you begin experimenting anew.

If the power light suddenly disappears anytime during experimentation with the robot, the most likely problem is that rapid, jerky motion of the robot has loosened the batteries in the battery-pack. Simply reseat the batteries firmly into the pack and power will usually come up. If reseating loose batteries does not fix the problem, check for a loose power connection to the computer board or for broken battery wires. If power problems persist, check the battery-pack voltage and the computer-board voltage with a multimeter to make sure power is actually available. You should get a nominal 6 to 7 volts from the battery-pack, depending upon level of charge, and the regulated computer voltage should be about 5 volts. A battery-pack reading of 5.4volts indicates the batteries are discharged and require recharging or exchanged out with fresh batteries.

When the robot looses power, for whatever reason, you will have to download the program again.

3.2 Reset of Robot through Jerky Motion

In rare cases, the robot may stop after making a series of rapid moves. This behavior usually indicates that the robot has reset because of high motor current surges. For example, if your program changes both wheel motors simultaneously from full speed forward to full speed backward, there might be a chance of reset when the batteries are low. These reset-type conditions can be avoided by more careful motor control. Change motor speeds, hence, currents, gradually. What is "gradual" to the motor, however, will hardly be observable to you if done in tens of milliseconds. When you program smooth motor control, the robot motion becomes

smooth and looks more "organic". One precaution you can easily take is to stop a motor before reversing its direction.

3.3 Installing PROGO™

Before you can install PROGO™ you must first install ICC11. Follow the directions given with the ICC11 distribution software and the TJ PRO™ distribution software. The PROGO™ distribution disk that comes with this manual, `progo11,` contains one directory, `PROGOtjp`. That directory holds the source code `PROGO.c`, the batch file `comprogo.bat`, and PROGO™ examples, applications, and solutions to problems in this manual. All PROGO™ programs have a "`.prg`" extension to identify them. When you compose PROGO™ programs be sure to suffix the "`.prg`" extension to the program's file name.

Install the `PROGOtjp` directory into `ICCTJP`, the root directory of your ICC11 installation. Copy `comprogo.bat` into the directory `ICCTJP\Bin` and you are finished with the installation of PROGO™!

3.4 Serial Communication with the TJ PRO™

In order to communicate with the robot, you must establish a serial link. Typically, this is done with Mekatronix's serial cable and MB2325 board. Future enhancements will include IR and radio links.

To do the downloading you will need the robot, a 6-wire Mekatronix serial cable and a serial cable to connect your personal computer to the Mekatronix D25 connector on the MB2325 communications board. Any of these items can be obtained through a Mekatronix distributor. For distributors check (http://www.mekatronix.com/distributors).

To establish the standard Mekatronix serial link, perform the following.

Connect a C2325, 6-wire serial communications cable, to the Mekatronix MB2325 serial communications board at one end and to the SCI male header on the MTJPRO11 at the other end. The MB2325 is connected via a 25-pin D-connector and cable to your Personal Computer. This setup establishes a serial link from TJ PRO™ to your PC. Check to make sure diode D1 on the MB2325 lights.

IMPORTANT NOTE:

If you execute a TJ PRO™ program that uses PROGO™ input and output statements and you loaded the "`.s19`" file with PCBUG11 in a WIN95 environment, then you must close the DOS Window in WIN95 before executing the program, because PCBUG11 locks up the COM port. HyperTerminal will not be able to communicate through the COM port with the robot. For an archaic reason PCBUG11 in DOS hogs the COM port and does not release it unless you kill the DOS window.

SOLUTION:

Eliminate PCBUG11 use in a WIN95 environment. Buy the (inexpensive and well worth it!!!) Mekatronix high-speed serial downloader (HSSDL11) for the MC68HC11 and avoid the pain of closing DOS and opening and setting up the ICC11 parameters every time you reopen the DOS window!

When using the HSSDL11 with HyperTerminal (HT)) on the same COM port, say COM1, you must disconnect HT ("ALT-C d" or select the HT telephone icon with receiver off the hook) to download. After download, connect HT ("ALT-C c" "or select HT telephone icon with receiver on the hook). You do not have to do anything with HSSDL11 since it does not hog COM1, an extremely useful convenience because it retains its state and lets you quickly download a program you repeatedly compile and change in DOS. Just switch back and forth between the DOS window and the HSSDL11 window...for convenience keep both open and side by side!

3.5 Compiling PROGO™ Programs

To compile PROGO™ programs in a WIN95 environment using the DOS version of the ICC11 compiler:

1. Open a DOS prompt.
2. Change the directory to `ICCTJP`.
3. Execute `tjpsetup`.
4. Type

```
comprogo <filename> <enter>
```

where `<filename>` is a PROGO™ program in the directory `PROGOtjp`. Do not put the "`.prg`" extension on the name. The batch file `comprogo` will do that for you. You can also provide a complete path to `<filename>` if it is not in the `PROGOtjp` directory.

The batch file `comprogo` generates two "`.s19`" files, `progo.s19` and `<filename>.s19` and places them into the directory `PROGOtjp`.

5. To compile other files during a session, repeat only Step 4.

If you are in an exclusively DOS environment, Step 1 is automatic, otherwise the procedure is the same.

3.6 Downloading Files

The procedure for downloading files:

1. Connect a C2325, 6-wire serial communications cable, to the Mekatronix MB2325 serial communications board at one end and to the SCI male header on the MTJPRO11 at the other end. The MB2325 is connected via a 25-pin D-connector and cable to your Personal

Computer. This setup establishes a link from TJ PRO™ to your PC. Check to make sure diode D1 of the MB2325 lights.
2. Switch DOWNLOAD/RUN Switch to DOWNLOAD.
3. Turn on TJ PRO™ power with POWER-ON Switch. LED D2 on the MB2325 will probably light.
4. Press RESET switch. D2 should go out. However, D2 lights when you hold RESET down. If it does not light with RESET held down, reverse the C2325 6-pin connector on the robot.

IF YOU DO NOT HAVE THE HSSDL11, THEN

5a. Move the "`.s19`" file you wish to download from `PROGOtjp` into the `tjpcode` directory. If there is a name conflict, you may wish to change the name of the"`.s19`" file before moving it.
6a. Go to a DOS window and prompt.
7a. If this is first time in this DOS window, perform the Compiler setup sequence in the previous section.
8a. Type

```
loadtjp <filename> <COM_PORT_NUMBER>
```

and then press

```
<enter>
```

For example, type

```
loadtjp avoidtjp 1
```

followed by `<enter>` and PCBUG11 will download `avoidtjp.s19` into TJ PRO™'s memory. The serial communications will be via COM1 (COM1 is the default if no COM-port is specified).

9a. If your program does not produce serial input or read serial input, steps 2 to 8a can be repeated unchanged after each compile. However, if your program does use serial IO and you operate in a DOS environment, you will have to startup Kermit or some other terminal program before executing your robot program. If you operate in a WIN95 environment, then you must terminate the DOS window before enabling HyperTerminal and executing the robot program. When finished with your experiment, you must repeat the four-step sequence COMPILE WITH ICC11 described in the previous paragraph in order to be able to compile again. You cannot simply execute step 4 in that sequence. This is a real inconvenience, so...

IF YOU USE THE HSSDL11

5b. Select the HSSDL11, press or select PROGRAM, browse until you find `PROGOtjp`, or whatever directory you are using, and open the "`.s19`" file of the program you wish to download, say, the one you just compiled. No name-length restrictions for the HSSDL11!

6b. Press d or select DOWNLOAD button and the green lights flash and the program is downloaded at about 115KBAUD!

3.7 Execute a Program

Before executing a program, you must, of course, download it to the robot. The procedure below assumes you have already downloaded an "`.s19`" file into the robot memory.

1. If the power is off, then turn the power on. If the switch has been off for more than six hours without a charger connected, you will probably have to recharge or switch-out the batteries and download the program again.
2. Flip DOWNLOAD/RUN switch to RUN.
3. Press RESET.
4. Place the robot on a smooth floor and press the back bumper switch closed. All PROGO™ programs wait for a back bumper switch closure before execution.
5. Watch TJ PRO™ GO!

Many times you will only want to test minor changes in a program, changes easily observed with the robot on the bench and wheels off the surface. In such cases you simply leave the charger and serial connections in place and simply flip the DOWNLOAD/RUN switch to RUN, press RESET and the back bumper. Observe the robot behavior and note corrections. After editing corrections in your favorite editor, recompile, download and execute.

3.8 Robot Time Variables

An interrupt service routine automatically controls the robot's clock. The global variables `msec, seconds, hours,` and `days` record the passage of their associated time units. The millisecond timer-counter variable `timertjp` counts-up each millisecond and rolls over to zero on the 65,536th count. You should treat this variable as a read-only free running counter. Programs can change the value of this variable, but if they do, they will probably wreck all kinds of unintended havoc!

The PROGO™ function

```
Wait <msec> ms
```

permits your program to generate precise delays in your code, up to the full count of `timertjp`, namely, 65,535 milliseconds.

4 TOUR DE ROBOT WITH PROGO™

The following exercises allow you to interact with your TJ PRO™ robot and learn how to control the motors and observe the sensory inputs. As you expand your TJ PRO™ with more sensors and actuators you will be able to do the similar tests with your own functions and drivers. An understanding of these basic functions will enable you to implement robot behaviors!

4.1 Start Command

After inputting motor and other parameters through a serial port, a TJ PRO™ program will jump into its execution phase automatically. If no provisions are made, you will not have time to disconnect the robot and place it on the floor before the wheels begin to move. A switch to prevent motion until you have placed the robot where you want it would be extremely useful. The *start statement*

```
Start  /* While BUMPER<120 perform end*/
```

provides just that functionality. A rear bumper switch closure will generate a value of about 126 on `BUMPER`. Until that happens, the program hangs up on the underlying `While` and prevents subsequent execution of the program.

PROGO™ programs always include a hidden Start statement. This means to execute a PROGO™ program you must close the rear bumper switch with a press.

You can use the `Start` statement elsewhere in your program, too. For example, you may program the robot to stop and await a back bumper switch closure whenever it detects a particular environmental or internal state.

4.2 Motor Control

A number of functions from the *TJ PRO™ Robot Kernel* enable you to control TJ PRO™'s motors. The direct motor control functions for the left and right wheel,

```
Left_wheel <number> percent
Right_wheel <number> percent
```

take a single parameter representing the percentage of full speed. For example,

```
Right_wheel 45 percent
```

commands the right wheel motor to run at 45% of full speed. Since the motor controllers are not precise, the percentages do not accurately correspond to true motor speeds, but do preserve relative speed sequence. For example 45% is always less than 50%, but not necessarily 5% less. The effects of these two commands persist even when the program continues. After a robot program executes the **`Right_wheel 45 percent`** command, for example, the right wheel will continue to spin at 45% full speed until some other command changes the setting.

Another motor control command permits you to specify the speed of both the left and right motors together,

```
Move <number> lws <number> rws
```

where **lws** stands for left-wheel-speed and **rws** for right-wheel-speed.
The command

```
Move 50 lws 80 rws
```

is equivalent to the two commands

```
Left_wheel 50 percent
Right_wheel 80 percent
```

and is, therefore, redundant. You may prefer the wheel commands for readability as opposed to the move command for brevity.

Here is a complete PROGO™ program to turn the left motor on 50 percent:

```
Program_begin
  Left_wheel 50 percent
Program_end
```

To move both wheels 100 percent simply program

```
Program_begin
  Left_wheel 100 percent
  Right_wheel 100 percent
Program_end
```

or

```
Program_begin
  Move 100 lws 100 rws
Program_end
```

or

```
Program_begin
  Go
Program_end
```

Robot Motor Exercises

1. *Write each of the sample programs above on your favorite editor. Compile, download, and execute them. Remember all PROGO™ statements begin with a capital letter and the compiler is case sensitive.*

2. *Describe in your own words what the robot will do when it executes the following three programs. After you do that compose, compile, download and execute each in turn. Where your predictions correct?*[2]

a)
```
Program_begin
  Move 100 lws 100 rws
Program_end
```

b)
```
Program_begin
  Move 100 lws 100 rws
  Stop
Program_end
```

c)
```
Program_begin
  Move 100 lws 100 rws
  Wait 5000 ms
  Stop
Program_end
```

3. *What is the difference between the* **Move** *and* **Go** *commands? Give an advantage for each command relative to the other.*

4.3 Interactive control of the TJ PRO™ Motors

Download the motor program, `xm2tjp.s19` into your robot. The program `xm2tjp.prg` appears in
Figure 1. The program serially communicates to the PC, which, for example, is executing HyperTerminal simulating a VT100 terminal with the communication protocol 9600baud, 8bits, no parity, 1 stop bit and no flow control.

Before executing the program in
Figure 1, open up HyperTerminal and set up the serial communications protocol as indicated (refer to Section 3.4).

Through the VT100 interface you can specify the speed and direction of both motors through serial input. With this control you can specify the robot to spin clockwise, counterclockwise, move in a straight or curved trajectory.

Exercise
Input speeds into `xm2tjp.s19` *to make the robot spin clockwise, counterclockwise, go forward, back up, and move forward but veering to the left.*

4.4 TJPRO™ PROGO™ Robot Motion Function Kernel

Type, compile, download and execute the following simple PROGO™ programs to increase your understanding of the PROGO™ motion commands on the TJ PRO™. Save your programs.

a)
```
Program_begin
  Spincw
```

b)
```
Program_begin
  Spinccw
```

c)
```
Program_begin
  Forward 20 inches
```

[2] a) Robot moves forward until something blocks its motion. b) The robot does not move! The Stop command executes immediately after the Move. This is to fast for the motors to respond. c) The robot moves forward for 5 seconds and stops.

```
Program_end
```

```
Program_end
```

```
Program_end
```

d)

```
Program_begin
  Turn_right 90 degrees
Program_end
```

e)

```
Program_begin
 Turn_left 45 degrees
Program_end
```

f)

```
Program_begin
  Backward 20 inches
Program_end
```

g)

```
Program_begin
  Reverse
Program_end
```

h)

```
Program_begin
  Pivot_left 90 degrees
Program_end
```

i)

```
Program_begin
  Pivot_right 45 degrees
Program_end
```

4.5 Introducing Waits

The next program `xm3tjp.prg` enhances `xm2tjp.prg` by permitting you to interactively specify the duration of the motion through serial IO, as well as the motor speeds. The `Wait` function times the motor on-duration with variable T <65,535 milliseconds. The motors run according to your input time specification T and then stop. Read `xm3tjp.prg` and develop an understanding for how it works. Now, download and run it to test your program reading skills. Were there any surprises for you?

4.6 Bumper Sensor

The TJ PRO™ bump sensors uses a slick trick. Each of the three bumper switch closures adds more current through a voltage divider circuit. This allows a program to actually determine where an object has made contact on the bumper. A robot program can even determine the precise bumper switch closures in simultaneous, multiple switch-closures. With this feature a robot can determine multiple, simultaneous collisions!

The bumper voltage is measured by the function `BUMPER`. Any combination of front bump switch closures, indicated symbolically by `FRONT_BUMP` is computed by the following logic expression which is always true (a tautologoy):

```
FRONT_BUMP equal_to (BUMPER>10)&&(BUMPER< 120)
```

A pure back bump satisfies the tautology

```
BACK_BUMP equal_to BUMPER>120
```

```
Program_begin

  Dictionary
    lspeed,
    rspeed
  ok

  Clear_screen
  Home_screen

  Display "Enter percent of left motor speed, a number between -100 and 100:   "
    on_screen
  Set lspeed to input_number ok

  Display "Enter percent of right motor speed, a number between -100 and 100:  "
    on_screen
  Set rspeed to input_number ok

  Display "Press back bumper again to start the motors" on_screen
  Start

  Move lspeed lws rspeed rws

Program_end
```

Figure 1 The motor control program `xm2tjp.prg` permits you to input the speeds for the left and right motors.

Neither test accounts for simultaneous front nor back bumps. To make this and finer bump discriminations requires a breakdown of the various bump values and the use of `BUMPER`.

Before exploring the bump values, examine the program `xbmp1tjp.prg` (Figure 2). Load the "`.s19`" files and execute it. This program blindly drives the robot forward until the robot detects a front bump. A front bump causes the robot to stop, backup for a short time, turn a "random" angle and then go forward once again. The function `turn_random()` generates the "random" angle turn based on the TCNT register, the free running microprocessor timer-counter (To obtain the names of all the MC68HC11 registers, refer to the include file `Hc11.h` provided in the ICC11 distribution software directory `Include`).

Caution!
In the bump experiments TJ PRO™ will bump into objects in order to respond to them, so take care that TJ PRO™ does not bump into anything that will harm TJ PRO™ or the object. The small size of TJ PRO™ does not usually invoke cause of concern, but the world is complex and a word of caution will remind you to evaluate TJ PRO™'s environmental situation before setting it free.

The "random" nature of the turn in `turn_random()`depends on the "random" time of the collisions. Each time the robot detects a front-bump, the variable `rand` records the value of the TCNT. An even value generates a right turn and an odd value a left turn:

```
If rand bit_and 0x0001 equal_to 1
 then
   Spinccw
 end
 or_else
   Spincw
 end
```

The value of the most rapidly changing bit, the least significant bit, determines the direction of the turn. The robot begins to spin. How long? The ten least significant, most rapidly changing bits determine the duration of the turn:

```
Set delay to rand modulo 1024 ok
```

as long as the duration exceeds 250ms, otherwise the duration is set to 250ms. The 250 millisecond bias eliminates small, ineffectual turning angles.

4.7 Bumper Values

In anticipation of later experiments, the table at the right records the values produced by individual switch closures on the bumper.

Switch	Value
Front	46
Front right	24
Rear	129
Front left	68

In future experiments you will press and hold down combination of bumper switches to obtain unique values. For example, with the front and back switches down BUMPER returns a value around 140. A press of the front and front-left bumper switches yields about 61. A value near 61, therefore, indicates that the bumper struck an object on the right side between the front and right-front bumper switches.

For now, let us continue the quick tour of the robot and discuss the infrared sensors.

4.8 Infrared Proximity Sensors

The two IR LEDs on top of the TJ PRO™'s plate and the one underneath the plate in the back emit 40KHz modulated infrared light when enabled. The MC68HC11 processor uses memory-mapped input and outputs, so the programmer controls all output devices by writing to memory. The TJ PRO™ *Robot Kernel* commands `IRE_on` and `IRE_off` allow you to control the IR emitters as a group. `IRE_on` turns on all three IR LEDs, two in front and one in the back, and `IRE_off` turns them off.

About 240 milliseconds after the IR emitters turn on, the outputs of the right and left IR detector circuits will reach their final values determined by the amount of reflected IR light they receive.

The IR detector maximum reading ranges from about 125 to 128. The detector's no-light level output typically lies between 83 to 86. The IR detectors decay from a maximum reading to the minimum in under 200 milliseconds after the emitters stop shinning. The difference in maximum and minimum IR readings, say, 128 – 84 = 42, yields slightly more than 5 bits precision ($2^5 = 32 < 42$).

```
Function turn_random
/*
  Will turn in a random direction for a "random" amount of time, dictated
  by the fast changing lower bits in TCTN register.
*/

Function_begin
 Dictionary
   delay,
   rand
 ok

/*
  Compute “random” time duration of the turn.
*/

 Set rand to TCNT ok

 Set delay to rand modulo 1024 ok

 If delay less_than 250
 then
   Set delay to 250 ok
 end

/*
  Compute direction of turn and do it.
*/

 If rand bit_and 0x0001 equal_to 1
 then
   Spinccw
 end
 or_else
   Spincw
 end

 Wait delay ms

Function_end
```

Figure 2. The direction and amount of turn produced by the `turn_random()` function depends upon the value of the rapidly changing TCNT register at the time of collision.

```
Program_begin
  Dictionary
    i, IR_delay, irdl1, irdl2, irdr1, irdr2, temp
  ok

  IRE_off
  Clear_screen
  Home_screen
  Display "Title        xir1tjp.c"      on_screen CRLF
  Display "Programmer   Keith L. Doty"  on_screen CRLF
  Display "Date         Oct 28, 1998"   on_screen CRLF
  Display "Version      1"              on_screen CRLF
  CRLF

  Do_forever
    Display  "Enter  millisecond  time  delay  called  IR_delay  (1...65,535):  "
on_screen
    Set IR_delay to input_number ok
    CRLF CRLF

    IRE_on
    Wait IR_delay ms   /* Give IR detectors time to respond */

  /* Read the two front IR detectors  */
    Set irdl1 to LEFT_IR ok
    Set irdr1 to RIGHT_IR ok

    IRE_off

    Wait IR_delay ms   /* Give IR detectors time to discharge */

    Set irdl2 to LEFT_IR ok
    Set irdr2 to RIGHT_IR ok

  /* Display IR values */
    Display "IR values after IREs are ON for " on_screen
    Write IR_delay on_screen
    Display "milliseconds." on_screen CRLF

    Display "Left IR        Right IR" on_screen CRLF
    Write irdl1 on_screen
    Display "               " on_screen
    Write irdr1 on_screen CRLF
    CRLF

    Display "Decayed IR values after IREs are OFF for " on_screen
    Write IR_delay on_screen
    Display "milliseconds." on_screen CRLF

    Display "Left IR        Right IR" on_screen CRLF
    Write irdl2 on_screen
    Display "               " on_screen
    Write irdr2 on_screen CRLF
    CRLF
  end
Program_end
```

Figure 3 Description of a program that allows you to interactively determine the response characteristics of the IR emitters.

Experiment with *xir1tjp.prg*

Download `xir1tjp.s19` into your robot. Set up HyperTerminal for serial IO and then execute the program. The program will ask you to enter a number with time units of milliseconds.

1) Keep the front of the robot clear of obstacles out to two feet. Enter 500 (milliseconds). Record the response of the IR detectors 500ms after the emitters come on and 500ms after the emitters shut off. *The values should be in the mid-eighties for both measurements.*

2) Place a small box with a flat surface directly in front of the robot, about 4inches away. Take IR readings for the time delay entries of 1, 50, 100, 150, 200, 250, 300, 350 and 400ms.

Questions (Refer to footnotes for selected answers)

a. At 1ms did the robot "see" anything?[3]
b. What delay in your experiment permitted the IR detectors to reach their final values after the IR emitters were turned on? [4]
c. What delay in your experiment permitted the IR detectors to reach their final values after the IR emitters were turned off?[5]
d. Where the two delays in parts b. and c. the same?[6]
e. Repeat the experiment several times. Do you notice any differences in the readings? By how much?[7]
f. Make two graphs on the same sheet, one showing the rise-time and the other the fall-time of the IR detector output vs. time. Be sure to keep the configuration and environment constant during all your measurements.

After performing the above experiments, you will recognize that the IR detectors have a dynamic response that can affect how you program robot behaviors. Being aware that the IR detectors take as long as 100ms to reach a final value may change your approach to programming the sensor. Since changes in detector output occur much more quickly than 100ms, programs designed to detect differential IR changes usually perform faster and better than those designed to detect absolute values of IR readings. The presence of noise indicates that differential changes of one unit may be spurious. A change of two or more units typically indicates a true event. The longer IR discharge times can affect how you program certain behaviors as well. In stealth mode, a robot may turn off its IR emitters and try to detect other robot IR emissions. To be certain, the robot must wait for its own IR detectors to reach final values.

[3] No, the time was to short for the IR detectors to register any light levels.
[4] For my robot, using a light colored facial tissue box, the IR detectors reached the final value of 110 and 117 in 100ms, left-right, respectively. The actual values you get will probably be different, but the objective of the experiment will not be affected.
[5] After 250ms IR emitter off time, the IR detectors reached their final off values of 83 and 84, left-right, respectively.
[6] No. The IR detector readings take longer to decay then to rise.
[7] You will occasionally see differences of one unit, sometimes two, rarely three, if at all. Noise in the electronics probably accounts for these readings.

We have completed the tour of TJ PRO™'s basic capabilities. I predict you will be amazed (I was!) when you discover the sophisticated robot behaviors you can program with just these simple capabilities.

5 SOME POSSIBLE BEHAVIORS

TJ PRO™ library programs provide the basic hardware interrupt and device driver routines for the robot. These allow the user to access the sensor readings and drive the motors. With these routines, the user can program an unlimited number of behaviors using PROGO™. A representative set, but, by no means, an exhaustive set, of primitive set of behaviors, from which more complex ones can be developed, are listed in Table 1. Some of these behaviors, like line following, require installation of auxiliary sensor kits.

Table 1 Possible Primitive Behaviors

Collision avoidance	Collision detection	Line following	Light following
IR light avoidance	Pushing	Collision detection	Shy behavior
Aggressive behavior	Exploring behavior	Wall following	IR beacon tracking
Attraction to motion	Floor drawing	Fixed motion patterns	Dance motions

6 PROGRAMMING BEHAVIOURS

The exercises in this manual will help you write simple TJ PRO™ behavior programs to become more familiar with the robot and its features while, at the same time, building your confidence in TJ PRO™ program development. Understanding how to program the various robot features and simple behaviors will then permit you to piece them together to create complex behaviors. You will certainly be surprised along the way, especially when you think you have programmed the robot to do one thing and its actual behavior is quite different. Many times the robot's behaviors can only be understood after post-experimental analysis.

7 ADVICE ON DEVELOPING BEHAVIORS

As a general principle, try simple programs first. Develop your understanding of the robot and its features. Become aware of the complex interactions between the robot and its environment. Build more complex programs incrementally, based on your increased understanding and awareness. Use previously debugged code and build on it.

This advice, as well as that to follow, is based on several years experience teaching engineering students to program autonomous robot behaviors.

7.1 Vulcan Mind Meld

To effectively program a behavior for TJ PRO™, or, quite possibly, any autonomous robot, and to gain insight into the problems encountered by your robot, you should play Vulcan to the robot and imagine performing a *Vulcan Mind Meld* with it. All you Trekkie fans know what this means. But, to be specific, try to perceive the universe as the robot does with its limited sense capabilities. This is harder to do then you might think. Imagine yourself one with the robot. Play out different sensations and responses. Help yourself by actually recording robot sense data and

examine typical responses, or responses to special environmental conditions of interest in the behavior you are developing. The mind meld will help prevent the common error of asking the robot to respond to environmental conditions it cannot detect with its sensors! While this statement is so totally obvious, it is a difficult self-discipline to psychologically enforce. Why? Humans typically interact with each other, or intelligent animals, expecting and perceiving sophisticated behavior and sensory performance. These expectations seem to subconsciously creep into our agenda when working with autonomous machines, often with disappointing results! Autonomous robots have nowhere near the sensory and behavioral capabilities of an insect, let alone higher animals.

7.2 Relative calibration of sensors of the same type

Manufacturing tolerances, circuit tolerances, and mounting variations make it possible for two instances of the same type of sensor to respond differently to the same stimulus. Behaviors, therefore, should not be programmed to depend upon two sensors of the same type producing identical responses to the same stimulus. Instead, write programs to calibrate sensors of the same type in some fixed environment. For example, place a cardboard box in front of, and parallel to, the wheel axis of a TJ PRO™. Measure the response of the two front IR detectors using `xir1tjp.prg`. Note the differences in the readings. If there are none, that's great! In general, however, they will differ somewhat due to electrical noise or manufacturing tolerances.

Another approach for making your robot behave more reliably is to program robot behaviors that respond to relative sense stimuli, not absolute sense measurements. This will make the robot behave more organically and robustly to uncertain, dynamic environments.

7.3 Adjusting to Ambient Conditions

A programmed behavior will often be *brittle*, i.e., not flexible or adaptive, if that behavior depends upon specific magnitudes of robot sensor readings. Brittle behaviors fail when the environment changes from the environment in which the behavior was developed. The smaller the change that causes the failure, the more brittle that behavior is. For example, the IR detectors on TJ PRO™ will detect white objects at larger distances than dark objects. Suppose a collision avoidance algorithm sets a threshold value of the IR as an indication of an impending collision. If this threshold is determined experimentally with light colored obstacles, then dark colored obstacles will not be detected and the robot will bump into them. On the other hand, if the threshold is set for dark colored obstacles, the robot will end up spinning in circles in a light colored environment because it detects threats everywhere. The solution is not to pick an average color threshold, but rather, program the robot to adjust its threshold downward if it has not detected a collision for some specified time, or, to adjust the threshold upward if it is colliding too frequently. The difficulty, of course, is determining exactly what the "specified time" between collision should be or what "colliding too frequently" means! The easy, but difficult to implement, answer is to let the robot learn these parameters based upon some performance criteria. An example solution to this problem is provided in `avcaltjp.prg` in the directory `PROGO11_APPLICATIONS`.

Robot behaviors and sensors, therefore, should adjust to ambient conditions. Biological organisms perform this function fantastically well. The human eye adjusts to bright sunlight or a darkened cathedral with dimly lit candles. This goal is easier to state then execute, but serves as a general principle.

7.4 Create simple behaviors

The beginning robot practitioner usually formulates behaviors too complicated to implement directly. With experience, the virtue of simple, direct behaviors becomes apparent. Complex behaviors should be broken down into sequences of simple, primitive behaviors. If this can be done, the chances of successful implementation are high. There is great difficulty in implementing such behaviors directly.

7.5 Build on simple behaviors

As the user accumulates a repertoire of primitive behaviors, complex behaviors open up. Perhaps the easiest way to generate complex behaviors is simply to sequence a collection of primitive behaviors. For example, wall following might be decomposed as follows: 1) detect a “large” object, 2) approach the object until “near”, 3) turn until the robot front-to-rear axis aligns “parallel” with the “surface” of the obstacle, 4) move “parallel” to the obstacle surface. At each instant of time a particular behavior in the sequence is invoked based on the current state of the robot and its sensory inputs. Of course, the programmer will have to relate to the robot’s perception the meaning of such terms as “large”, “near”, “surface”, and “parallel”. Remember to Vulcan Mind Meld!

7.6 Integrating Behaviors

More complex behaviors may require the combination of primitive behaviors in a way not well understood. Artificial neural network activation, opinion guided reaction, non-linear dynamics, and fuzzy logic all offer techniques for integrating behaviors. Each technique offers specific advantages and specific difficulties. Discussion of such issues is beyond the scope of this manual. The reader’s attention is brought to this matter to encourage investigation into these possibilities.

8 TJ PRO™ EXPERIMENTS

The numbered items below suggest a sequence of ever more complex PROGO™ programs and experiments you can perform to familiarize yourself with TJ PRO™’s capabilities. These experiments will open up to you the rich variety of behaviors you can program. After each successful experiment, save your program and do not change it. Use copies of it to begin other programs, but do not write over and destroy your only copy of a successful program. You will never know when you might want to use it again, either as is, or as a basis for another program.

Program solutions to the following experiments are on the `progo11` diskette. Since a computer language provides a rich structure for developing procedural solutions to problems, you should realize that the solutions supplied here represent only one of large number of possible ways of solving the problems stated. Also, ambiguity of language will lead to different interpretations of what the problem states and, thus, give rise to other solutions. In fact, one of the great difficulties

in the discipline of computer programming is to translate an imprecise natural language specification of a problem into a precise computer language specification and then determine if the program does what you specified. In fact, once the program performs to the end-user's satisfaction, the program *becomes* the precise statement of the solution and *defines* what problem it solves. Often, the resulting program solution does not exactly match the original, ambiguous specification, but provides what the end-user finds acceptable!

8.1 Robot Connections During Program Development

Do not forget the recommended operating procedure for robot program development.

1) Keep the robot on its charger during program development. In this way the robot will almost always have fresh batteries to perform floor experiments when you reach that point in your design.
2) Maintain serial connection between the robot and your PC.
3) For convenience, keep the TJ PRO™ mounted on a stand with the wheels suspended in the air. This way you can perform most experiments and tests during early program development without placing the robot on the floor, which requires connecting and disconnecting the serial cable and the battery charger each time.

8.2 Measure Wheel Angular Speed

In this section, you will measure the wheel motor angular speed, the number of wheel revolutions/second, at various speed settings.

Writing a program for each choice of motor speeds can get tedious and messy. To characterize the angular speed versus motor percentage more efficiently, you can input the various motor speeds to generate different turning rates with just one program, `xm3tjp.s19`.

Download `xm3tjp.s19` into the robot and set up HyperTerminal.

Interactively through the serial port, you will set the wheel speeds and duration of the experiment. If you want the experiment to last "indefinitely", you can specify up to 65 seconds with 65,535 for the duration in milliseconds. This time is much longer than any of the following experiments require.

Wheel Angular Speed Experiment 1

1) Put a narrow strip of white tape on each wheel to make it easy for you to identify a complete rotation.
2) Execute the program. Enter 100% for each motor and 40000 for the duration of the experiment.
3) Use a stopwatch and measure how long it takes the wheel to turn 10 times.
4) Compute the angular rotation rate of the wheel in revolutions/second (rps).
5) Change the speed of the motors to 75%, 50%, 25%, 15%, 10%, 5%, successively, and measure the angular rotation at each speed.

6) Plot angular rotation of each wheel vs. the speed.

Questions

a) What conclusions can you draw from the curve?[8]
b) Do the left and right motors have the same response to each percentage specification?
c) What does this tell you about precise control of the robot motion?[9]

Wheel Angular Speed Experiment 2

Repeat **Angular Speed Experiment 1** using negative percentages.

Question

Compare the graphed results of this experiment with the graphs plotted in **Wheel Angular Speed Experiment 1.** *Do the graphs appear to be close?*

The idea behind the second experiment is to illustrate that the motors probably perform about the same when turning in the same relative direction,[10] but have significant differences in performance when their relative angular velocities are oppositely directed. Consequently, rotations using a negative speed percentage on one wheel and a positive speed percentage on the other will, in general, be more precise than motion resulting from same-sign speed percentages on both wheels.

Wheel Angular Speed Exercise

1) Measure the diameter of each wheel. Are they the same? How much error do you estimate in your measurement?
2) From the measurement of the right wheel diameter convert one of the two graphs generated for the right motor angular speed to linear speed of the wheel contact on the floor versus the various percentages. Recall that the wheel contact linear speed magnitude equals the radius of the wheel times the angular speed.

8.3 Robot Translation

The motion of a rigid body breaks down into two essential motions, translation and rotation. Translation means the body moves in a direction without changing its orientation. Translation does not necessarily mean that the body moves in a straight line. However, since TJ PRO™ must change its orientation to change its direction, TJ PRO™ can only translate in a straight line. If the robot's trajectory curves, the wheels must be rotating at different speeds and the orientation of the

[8] One Possible Conclusion: The curve is non-linear, meaning that a % change in the motor software speed parameter does not give you the same % change in speed of the actual motor.
[9] The two motors appear to have quite different responses. This asymmetry may result from the mechanical structure of the brush mechanism, which performs differently when the servo motor rotates in one direction versus the other direction. Each motor is actually turning in a direction opposite to the other when the robot goes forward or backward , hence, the different motion characteristics for the same speed % . You need to keep this in mind when programming motion control.
[10] The motors are turning in the same relative direction if one wheel tend to push the robot forward and the other backward. In other words, when the robot rolls forward, one motor is turning in the opposite direction of the other.

robot must be changing. In this section you examine how to control the motors to translate TJ PRO™ (well, sort-of!).

Translation Experiment 1: Programming A General Motion Control Program

Objective

Measure the robot's deviation from straight-line motion when it is supposed to be going straight.

Specification

Turn on both motors at 100% forward for 10 seconds and stop the robot. Use the back bumper switch to start the motion.

A Solution

```
Program_begin
  Go
  Wait 10000 ms
  Stop
Program_end
```

To systematically execute this program for different motor speeds and times, use the program `xm3tjp.prg` in `PROGO11_Experiments` directory.

Perform this experiment in a wide-open space at least 4 feet wide and 12 feet long. After entering 100, 100, 10000 for the left and right motor speeds and the duration of the experiment in milliseconds, disconnect the robot and place it on the floor. To start the robot tap the rear bumper to close the bumper switch. The robot will move forward for 10 seconds.

Questions

a. *After 10 seconds, how many inches has the robot moved forward along its initial direction?*
b. *How many inches has the robot veered to the left or right from the line of its initial direction?*

To enable answering questions b and c, you can perform the experiment on a tile floor. Line up the left wheel on a long, straight tile-line and determine how far the wheel contact point has deviated from the axis of that tile-line after the robot stops. Repeat the experiment several times and compare the results.

c. *Now, type, compile, load and execute*

```
Program_begin
  Forward 120 inches
Program_end
```

d. *Measure the actual distance of travel in part c. Was it 120inches = 10feet?*

e. With a stopwatch, measure how long the robot takes to move the distance you measured. What is the average speed of the robot over that distance?[11]

f. What is the accuracy (magnitude of % error) of the PROGO™ command `Forward` *?*[12]

Translation Experiment 2

Objective

Move the robot at various forward speeds and determine how straight it goes.

Specification

Turn on both motors in the forward direction for equal percentages for several different values.

You can use the programs `xm2tjp.prg` or `xm3tjp.prg` to perform this experiment. A really simple PROGO™ solution for 100% motor speeds is simply

```
Program_begin
  Go
Program_end
```

Questions

a. Does the robot go straight? Which way does it prefer to turn, to the left or to the right?

b. Does this make sense with respect to you previous motor graphs? Explain.

Translation Experiment 3

Objective

Move the robot in reverse for various percentages and determine if it goes straight backward.

Specification

Turn on both motors in the reverse direction at the same percentage. Experiment with different percentages.

For 100 percent you can use the simple PROGO™ program you saw earlier:

```
Program_begin
  Reverse
Program_end
```

For other percentages, you can use `xm2tjp.prg` or `xm3tjp.prg` to input various negative percentages for the wheel motors. For "straight" motion, of course, you will specify, at least initially, the same percentages. As before, you will discover the motors behave differently running in opposite directions and set at the same percentages.

Questions

[11] Take the distance measured and divide by the time measured.

[12] Magnitude of % error = 100*|1-(measured_distance)/(actual_distance)|

a. *Does the robot go straight backwards? Which way does it prefer to turn, to the left or to the right?*
b. *Which motor, left or right, appears to be turning faster for the 100% reverse command? Are your results consistent with the previous experiments? Explain.*

c. *Now, type, compile, load and execute*

```
Program_begin
  Backward 120 inches
Program_end
```

d. *Measure the actual distance of travel in part c. Was it 120inches = 10feet?*
e. *With a stopwatch, measure how long the robot takes to move the distance you measured. What is the average speed of the robot over that distance?*
f. *What is the accuracy (magnitude of % error) of the* PROGO™ *command* `Backward` *?*
g. *Compare the accuracies of the* PROGO™ *command* `Forward` *and* `Backward`. *Are they the same? Explain any differences.*

Further Experiments:

1. Try to compensate for the robot veering from forward straight-line motion by slowing down the faster motor .
2. Always keep the slow motor at 100%. You could use a systematic, binary search to find the best percentage for the fast motor. First, try 50% for the fast motor. If 50% slows the fast motor too much and the robot veers in the direction of the fast motor, try 75%, half way between 50 % and 100%. If 50% does not slow the fast motor enough, i.e., the robot continues to veer in the direction of the slow motor, then try 25%, half way between 0% and 50% (this is not likely!). Continue dividing the resulting range in half for each experiment, increasing your estimate for the fast motor if the robot veers in the direction of the fast motor and decreasing your estimate for the fast motor if the robot veers in the direction of the slow motor. Repeat this process until you cannot divide the percentage range further and get an integer quotient. This *binary search process* will require, at most, 6 experiments (*Why?*) before you obtain the best *open-loop* straight-line motion possible. *Open-loop* means no feedback (no sensory measurement) is used to provide control information that would help the robot compensate for error in its motion.

   Your program should drive the slow motor at 100% forward and the fast motor at X% for 10 seconds and stop the robot. X% will be assigned a specific value by you for each experiment. A good initial guess at X% will save you time by reducing the number of searches.

Questions

a. *What percentage did you get for the fast motor that yielded the best straight-line motion?*
b. *Does the robot still veer? If so, which way, towards the fast motor or the slow motor? Does the robot veer consistently to the same side over a set of 5 to 10 experiments? Explain your results and discuss any puzzling features you discovered.*

8.4 Changing the Distance Factor

You have investigated the accuracy of the PROGO™ commands `Forward` and `Backward`. Can you improve their accuracy? Knowledgeable *C* programmers can go to the `PROGO.c` code and change the *distance-factor* coefficients `DFINCH` or `DFCM`. For example, if you want distances to be in centimeters replace `DFINCH` with `DFCM` and end the `Forward` and `Backward` commands with `cm` instead of `inches`. A warning about changing the values of `DFINCH` or `DFCM` : the optimal values change with battery level, so any adjustment might only improve performance temporarily.

Again knowledgeable *C* programmers could make those parameters floating point global variables and then adjust them according to an add-on battery level sensor. In this way the robot calibrates its coefficients according to the battery voltage level.

8.5 Forward and Back Motion

Write a new program using *`xm3tjp.prg`* as the starting point. The new program makes the robot go forward for *n* seconds (an input parameter), stops the robot for a full second, and then reverse for *n* seconds, after the back bumper switch is pressed. Edit, compile, download, execute and test your program.

A solution to this exercise is `xm4tjp.prg` in `PROGO11_Experiments` directory.

Questions

a. *Does the robot return exactly to the same spot it left?*
b. *Repeat the experiment 5-10 times and measure the x and y position errors from a fixed reference point.*

If you fix the time at 10 seconds and the motor speeds at 100 percent, you can run the simple PROGO™ program:

```
Program_begin

  Move 100 lws 100 rws

  Wait 10000 ms

  Stop

  Wait 1000 ms

  Move -100 lws -100 rws

  Wait 10000 ms

  Stop

Program_end
```

The above program illustrates the central idea behind the back-and-forth experiments.

8.6 Robot Spin

In this set of experiments you observe and measure the characteristics of robot spin, the second type of rigid body motion.

As a first step edit, compile and load the simple programs

a)
```
Program_begin
  Spincw
Program_end
```

b)
```
Program_begin
  Spinccw
Program_end
```

You have seen these programs earlier and should already have them on file. Remember the comment about saving your programs? If you did not save them last time, do not worry. Your loss of energy, time and effort for such short programs is not significant. If you did not save them the first time, save them now. No telling when you might want to run them again.

Execute each program in turn. Use a stopwatch to measure ten complete revolutions for each program. Do this at least five to ten times and take the average. This will give you the maximum angular speed of the robot about its center axis in the clockwise and the counterclockwise directions. Are they the same?[13]

Suppose you want to measure the angular speed of the robot at various motor speeds not just 100 percent. To perform such experiments you could write a program for each speed you wish to test. For example, the program

```
Program_begin
  Move 50 lws –50 rws
Program_end
```

will spin the robot clockwise with the motors driven at 50%. Due to the non-linear behavior of the motors, however, the angular speed does not equal half of the previous rate.[14]

Writing a program for each choice of motor speeds can get tedious and messy. To characterize the angular speed versus motor percentage more efficiently, you can input the various motor speeds to generate different spin rates with just one program, `xm3tjp.s19`. Furthermore, while the spins performed above where about the robots central axis, one could spin the robot about the contact point of either wheel. One wheel acts like the pivot foot of a basketball player.

[13] I got about 13.6 seconds for 10 revolutions as an average for both directions, so the angular speed is 1/1.36 = 0.735 revolutions / second. If there is a difference in angular speed at 100% motor speeds, its no more than 2 or 3 %.

[14] I get about 16.5 seconds for 10 revolutions. This computes to an angular speed of 1/1.65 = 0.606 revolutions / second. This is definitely not half of 0.735 computed at 100%.

Spin Experiments with *xm3tjp.c*

1. Make the robot spin clockwise (pivot) about the right wheel at maximum angular speed. The left wheel does not move. Make the time duration of the motion a large number in order to give you time to watch the robot behavior.

   a. Does the right wheel stay in one place or does it drift away from its initial spot?
   b. Measure the angular rotation rate in revolutions/second. Suggestion: pick a feature on the TJ PRO™'s top plate and use a stopwatch to measure how long it takes that feature to rotate 10 times.
   c. Measure the drift displacement. (Hint: Mark a water erasable footprint of a wheel on a tile floor or mark a footprint on a large piece of blank brown paper or blank paper from a newspaper roll. Place the wheel on the footprint. Enter the approximate time for the robot to make 10 turns. Run the robot for that time and mark a new footprint. Measure the direction and distance from a point on the first footprint to the corresponding point of the second.)
   d. If you double or triple the time for spinning, does the displacement distance double or triple?

2. Make the robot spin counterclockwise about the right wheel at maximum angular speed.
   Apply the questions in the previous experiment to this experiment.

3. Make the robot spin clockwise about the left wheel at maximum angular speed.
   Apply the questions in the first experiment to this experiment.

4. Make the robot spin counterclockwise about the left wheel at maximum angular speed.
   Apply the questions in the first experiment to this experiment.

5. Make the robot spin clockwise at maximum angular speed about its center axis by setting the left and right motor speed equal in magnitude, but opposite in sign.
   Apply the questions in the first experiment to this experiment. Which Robot Kernel Command does this realize?

6. Make the robot spin counterclockwise at maximum angular speed about its center axis by setting the left and right motor speed equal in magnitude, but opposite in sign.
   Apply the questions in the first experiment to this experiment. Which Robot Kernel Command does this realize?

Data Analysis

a. Compare the drift displacements for Experiments 3 and 4.
b. Compare the drift displacements for Experiments 5 and 6.
c. Compare the angular rotation rates of Experiments 3 and 4.
d. Compare the angular rotation rates of Experiments 5 and 6.

e. If r_i is the angular rotation rate in Experiment-i, then compute the ratios r_1/r_5, r_2/r_6, r_3/r_5, r_4/r_6. Do you see a relationship between the angular rates? Explain or justify your claims.

8.7 Turns and Pivots

The spin experiments examined the behavior of the robot as it spun continuously. Turns and pivots turn the robot a specified number of degrees and stops. You have examined such commands in the TJ PRO™ *Robot Kernel* at the beginning of Section 4.4. In this section you will examine the accuracy of those turns and spins. Knowledgeable *C* programmers can go to the `PROGO.c` code and change the *angle-factor* coefficients `AF` and `AF2 = 2*AF`. A warning about changing the values of `AF` and `AF2`: the optimal values change with battery level, so any adjustment might only improve performance temporarily.

Again knowledgeable *C* programmers could make those parameters floating point global variables and then adjust them according to an add-on battery level sensor. In this way the robot calibrates its coefficients according to the battery voltage level.

1. Experiment with Turns

Objective

Command the robot a user specified angle and measure the error.

Specification

Write a PROGO™ program that allows user input of the turn direction and angle.

Solution: Before looking up the solution try your own version. Use `xm3tjp.prg` as a starting point. You will find a coded solution to this exercise, `xm5tjp.prg` , in `progo11_Experiments`.

After you get your program running, or you have read and understood `xm5tjp.prg` continue with the following experiments. One might call this experiment, *Applied Geometry* or *Geometry Made Fun*! Ah! The beauty of intelligent autonomous mobile robot experiments.

Experimental Setup

Tape blank newsprint or a piece of white paper on a flat smooth floor. Secure an erasable-ink felt pen to the robot's front plate so it does not wobble. Set the robot onto the paper and execute the various turns. After each turn, label the drawn arcs sequentially, indicate the commanded angle on the arc (so you won't forget!), and mark their end points.

When finished use a compass and straight edge to find the turning center of each arc. How? Recall the perpendicular bisector of a cord must pass through the center of the circle. Make two cords and their perpendicular bisectors. The intersection of the two bisectors is the center (why?). Next, if you have a protractor, measure the angle of each arc and compare with the angle you commanded. If you do not have a protractor, you can create your own by first bisecting the first quadrant between two intersecting, mutually perpendicular lines. Repeat, bisecting the resulting two angles at least twice more. The smallest angle on your protractor is now 90/8 = 11.25 degrees. If you want more accuracy, bisect one more time to get 5.625 degrees per angle. Two more bisections will produce a theoretical accuracy of 1.40625 degrees.

Questions

a) *Specify the robot to turn right 11, 22, 33, 45, 56, 67, 78, and 90 degrees. Measure the errors in the angle.*
b) *Specify the robot to turn right 11, 22, 33, 45, 56, 67, 78, and 90 degrees. Measure the errors in the angle.*
c) *Perform the experiments in parts a) and b) about 5 to 10 times for each angle. Plot the average magnitude of the percent error:*

*100*Average(|actual angle – commanded angle| / |actual angle|)*

versus command angle for a right turn and average magnitude percent error versus command angle for a left turn.
d) *Interpret the plots in part c). For example, does the average magnitude percent error stay the same? Decrease with angle? Increase with angle? Try to explain what you get based on the noisy, non-linear behavior of the motors.*

The next experiment is a variation of the first one. The robot pivots about a wheel contact point instead of its center.

2. Experiment with Pivots

Objective

Command the robot a user specified angle and measure the error.

Specification

Write a PROGO™ program that allows user input of the pivot direction and angle.

Solution: Before looking up the program solution try your own version. Use `xm5tjp.prg` *as a starting point. You will find a coded solution to this exercise,* `xm6tjp.prg`*, in* `progo11_Experiments`.

Questions

Answer the questions in the turning experiments, except now assume those questions refer to pivoting right or left through an angle.

8.8 Bumper Experiments

To this point you have investigated the actuation capabilities of the TJ PRO™ robot and you have seen how to access sensory information. In this section, and the next, you will explore the interaction between TJ PRO™'s perceptual and motor capabilities to realize robot behaviors.

TJ PRO™'s bumpers provide the robot with a sense of touch. TJ PRO™ can differentiate a bump contact at six different points: at each of the four bumper switches, between the front-middle and front-right, and between the front-middle and front-left switches. Three bumper switches in front and one in back make TJ PRO™ more sensitive to frontal contact than back contact. The design presumption is that TJ PRO™ mostly moves forward and needs to be more touch-capable in the forward direction.

You can actually program the TJ PRO™ to detect any multiple switch closures. This capability should become more apparent as you perform the bumper experiment below.

Experiment to Measure the Bumper Switch Values
Run the program `sensetjp.prg` by downloading `sensetjp.s19` in the `progo11_Experiments` directory. This program uses serial IO to output the current values of the IR detectors and the bumper switches and the command

```
Move_cursor "<row_number>" row "<column_number>" column
```

which controls the cursor position of the screen of the VT100 terminal simulator run by HyperTerminal. The **`Move_cursor`** command moves the cursor to the next print position on the screen. The numbers `"<row_number>"` and `"<column_number>"` specify the row and column position, respectively. The double quotes must be present.

Open and connect HyperTerminal to COM1, after downloading the program into your robot. Put the robot into RUN and press RESET. The screen should immediately display the outputs of both IR detectors and the bumper.

1) Press each bumper switch in turn and record the numerical output for each.
2) Press any two bumper switches simultaneously and record the outputs.
   Be careful to press exactly the two switches you intend!
3) Press any three switches simultaneously and record their outputs.
   Be careful to press exactly the three switches you intend!
4) Press all four switches down and measure the output.

Questions
a) The outputs probably fluctuate. What percentage fluctuation do you observe for each switch closure measurement?
b) How should the knowledge in part a) help you to write better robot behavior programs that use bump sensing?
c) Did you obtain unique numerical output ranges for all combinations?[15]

Turning_Away Behavior Based on Bumper Contact
Earlier you may have executed `xbmp1tjp.prg` to watch the robot bump around in its environment. Each time the robot bumped an object, it would turn a random angle and proceed. In this exercise, the robot turns a fixed direction based upon where it was struck.

[15] All the combinations should yield unique values.

Program TJ PRO™ to turn away from a bumper contact as described in the Table 2 below. Use the *Bump_Around* behavior, program `xbmp1tjp.prg` , as the starting point for your new program. You can change the value of T experimentally to see what other values do. I find `T=350` gives good performance, but you may find another value that suits your purposes better.

Table 2 Action taken for different bump sensations

Switch Closure(s)	Range	Action
Front-Middle	40-50	Back up 350 ms[1]. Turn right 500 ms.
Front-Right	20-30	Back up 350 ms. Turn left 250 ms.
Front-Left	67-70	Back up 350 ms. Turn right 250 ms.
Front-Middle:Front-Right	60-63	Back up 350 ms. Turn left 125 ms.
Front-Middle:Front-Left	90-100	Back up 350 ms. Turn right 125 ms.
Back	120-135	Forward 175 ms. Turn left 500 ms.

[1]ms = milliseconds = 0.001 seconds

Redefine the `turn_random` function to `turn_bump` which

1) Uses the value of the global variable `bump`, the current reading of the bumper switches, to compute both the direction and duration of the turn and then
2) Executes that turn.

A solution to this exercise is `xbmp2tjp.prg` *in* `progo11_Experiments` *directory.*

Questions

a) Qualitatively compare this fixed turning away behavior with the previous one with "random" turns. Which motion appears more sophisticated? More organic, animal like? Which program has fewer lines of code? Which behavior appeals more to your sense of esthetics, or which motion gives you more satisfaction? Which behavior appeared more interesting to you?

(There is no one "right" answer to each of the above questions, but can you defend your answers with reasoned arguments?)

b) Can you find a "behavioral" weakness for the robot that executes computed turns? (Hint: Is it possible to make the robot oscillate back and forth in a corner when the robot executes `xbmp2tjp.pr`*? Can you make* `xbmp1tjp.prg` *do the same? Justify "no" answers and demonstrate "yes" answers experimentally.)*

Application of Sensors in Multiple Behavioral Contexts

One of the neat aspects of an autonomous robotic agent is the multiplicity of uses to which we can put its sensory and actuation capabilities. We have already demonstrated this with the bumper switches. The rear bumper switch, for example, has been used not only to detect rear collisions, but also to serve as a "Startup" switch. It gets even better. A *bump-program* can use the back bumper switch both ways, depending on context. After the robot is told to "Startup" by pressing the back bumper, the switch resumes its role as a collision detection sensor. The contextual

meaning of a sensory input looms as an important concept in autonomous robotics. As you develop your own programs you can take advantage of this powerful idea and use the sensor suite in multiple ways.

8.9 Infrared Experiments

The IR emitters, the blue eyes on top of the plate, emit 940 nanometer electromagnetic radiation (infrared) modulated at 40KHz. This light expands out in cone, strikes an object and some reflects back. Underneath the plate, on the right and left, the IR detectors (shinny metal cubes) receive the reflected light and output a voltage proportional to the received intensity. We will use this sensory capability in the following experiments.

For the IR experiments, use the program `sensortjp.prg` in the directory `progo11_Experiments`. Do not forget to set up HyperTerminal before executing this code.

Experiments with IR Detectors
Observe the sensor outputs printed on your terminal screen.

Place an object or your hand to the right front of the robot. Observe the change in the IR sensor readings. If the object is close enough, you will have certainly changed the right front IR detector reading. Depending on your placement of the object, you may or may not change the left front IR detector reading. Repeat with the object at various distances.

Explore the robot's "visual" limits. Try objects of various sizes and placements. These visual limits will help you understand why TJ PRO™ fails to detect objects sometimes and bumps into them.

Questions

a. *Determine the minimum and maximum readings of the two IR detectors. Are they the same for both IR detectors? What impact would any differences have on collision avoidance?*
b. *Can TJ PRO™ see the point of a small shoe on the floor in front of it? Can it see an object suspended several inches above its plate?*
c. *Determine the minimum height of a white wall that the robot can see.*
d. *Determine how low an overhang can be without the robot seeing it.*
e. *What is the smallest rectangular wall surface the robot can see?*
f. *What is the smallest right circular cylinder (chair leg!) the robot can see?*
g. *Explain why TJ PRO™ IR "vision" has "blind" spots.*

In the next experiment you will measure IR detector intensity as a function of the distance from a reflecting wall surface.

Procedure

Place a flat, white cardboard "wall" in front of the right IR detector. I recommend taping a white sheet of paper on the side of a cardboard box. Arrange the front surface of the right IR detector with respect to your "wall" to give a maximum reading. To do this, place TJ PRO™'s front about 8 inches from the wall and take an IR reading. In place, manually rotate TJ PRO™ a small amount and take another reading. Be careful not to move the robot closer or further away from the wall! If the IR reading got bigger, rotate a bit more and take another reading. If it got smaller rotate back the other way and take another reading. Keep rotating in the direction that causes increases in the IR readings. The IR readings will then level off and not change for small rotations. Eventually the readings start to decrease to one side and then the other of the flat high region you found. Pick the middle of that region. That point is the geometric arrangement you want between the wall and the IR detector. The robot sensor does not exactly square off with the wall, but does give the greatest measurement sensitivity. Can you explain why?

A key to the success of this experiment is to keep this alignment as you move the wall various distances from the robot. For example, you can slide the cardboard box along straight tile-lines on a tile floor.

Questions

a. *After you arrange the robot with respect to the wall using the above technique, the robot wheel axis is not really parallel to the wall. Why?*
b. *As the distance to the wall changes, does your arrangement continue to yield a maximum reading for the IR with respect to other orientations at the same position?*
c. *With the proper orientation of the robot and the wall. Measure the IR intensity with the wall touching the robot and in increments of 1 inch all the way out to where the reading no longer is affected. Plots this data on a graph. The IR readings on the y-axis and the distance in inches on the x-axis.*

Data Analysis (Advanced)

a. *Did you get a strange value for the wall up against the robot? Explain.*
b. *At what distance did the reflected IR light from the wall have no effect on the IR reading?*
c. *Consider only the part of the plot where the data changes each successive measurement. Solve for the coefficients (a,b,c) of the quadratic* $I = ar^2 + br + c$ *from three simultaneous linear equations. Obtained the three equations by finding I, on the curved part of your plot, for 3 widely separated values of known distance r. For more sophisticated users, use a minimum least-squares fit to a quadratic equation.*
d. *Plot the resulting quadratic with the coefficients you found, together with the measured curve. Compare at several points. Does the quadratic curve seem to fit the points in between the ones you actually selected to compute it?*
e. *What is the maximum error between the quadratic and the measured plot for the curved part?*

f. From your data, describe two ways to determine the distance to an object given the reflected IR intensity? Will your answer be correct if the object is not the same material and color as the one used to get the data? Explain.

9 APPLICATIONS

The directory `progo11_Applications` contains several TJ PRO™ "toy applications". You can approach these applications in several ways. One fun technique, perhaps the hardest, load a program you know nothing about (don't cheat and read the descriptions below, or the code!) and run it and try to guess what it does. This, in general, is not easy! Next, program the behavior you observed and compare the robot's behavior executing your program with its behavior executing the original. Does the robot really behave the same for the two programs? This question is not trivial. Comparing behaviors of robots is not yet a science. Now, compare your code with the original code. The two programs probably differ greatly in detail, if you didn't look-ahead!

Another approach is for you to write a behavior program from behavior specifications. Here, too, you can compare the robot's behavior running your program and running the given solution. This approach is taken for the program `faceoff.prg` in the paragraphs below.

A third approach to these programs: read the code and descriptions. Load the code and observe the robot in action. Was it what you expected? Does the robot do strange things in strange circumstances? Think of ways to improve the behavior program or expand the behavior capabilities using one of the given programs as a base.

Finally, of course, make copies of these programs, modify them to change the behaviors by a little or a lot, and test your changes. As you become more experienced you will write applications of your own, applications limited only by your imagination.

9.1 Programming a Behavior from a Specification

In this section you will program a behavior for the robot from a description of that behavior. I identify and name some key variables to provide a common basis for discussion and help you get started. You do not have to use those variable names, or even develop the algorithm in the direction that I layout, of course. But, the learning game requires you to meet specs, or have fun, or diverge in your own direction with an off-the- wall "non-solution"…as long as you get something out of the game!

.

Program a Face_Off Behavior

Write an PROGO™ program `faceoff.prg` for TJ PRO™ that turns the robot to face parallel to a wall and stops.

Detailed Description of Face_Off Behavior

If the IR detectors and the motor control were perfect, equality of high IR values would indicate the robot has squared-off or faced-off against a wall. Because perfection of components and control is unreasonable to expect in practice, the program will have to compensate for uncertainties in both.

To face-off, TJ PRO™ slowly rotates counterclockwise (`speedl=10, speedr=-10`) about its center until its front faces parallel to a wall, i.e., parallel to the wheel axis. The robot detects the *parallel-to-wall* state when the left IR equals the right IR detector value. To detect this state, the robot rotates the default `move_time = 25` milliseconds and then stops for the default `ir_measure_time = 350` milliseconds to give the IR detectors time to stabilize on their new values, which it measures. This process repeats until both IRs values differ by less than *n* (you pick *n = 0, 1, 2, or 3*) and are above some threshold, `irdr_threshold = 97` (default). In order to try various parameter values, write your program so that you can input different values for the program parameters `speedl, speedr, move_time, ir_measure_time,` and `irdr_threshold` using HyperTerminal on COM1.

I place the robot at an angle with the left IR sensor further from the wall than the right IR sensor. The right IR sensor is about 6 inches from the wall. As the robot rotates clockwise, the left IR values slowly increase, so when it matches the right IR, the motion is complete. You can actually place the robot at any orientation near the wall. It moves so slowly, however, that you may not have the patience to wait for it to rotate almost 360 degrees!

The value of the IRs must indicate the presence of an object. Since the baseline IR values hover around the low eighties, I selected `irr_threshold = 97`, meaning the right IR must read at least that amount or no wall is assumed to be nearby. By symmetry, if the two IRs are identical and the surface is flat with uniform reflectance, the robot's wheel-axis diameter should be almost parallel to the surface when the IRs read the same value. Because the dc motors on TJ PRO™ cannot be precisely controlled, it may not be possible to get the robot wheel axis exactly parallel, even for a really good surface. Additionally, the IR sensors may not be matched. One may read slightly higher than the other for a given distance. Finally, noise on the IR will make the "equal-value" determination uncertain.

Comment: Although you may have calibrated the servo motors, they still drift, so low speed control requires experimentation.

A solution to this exercise is `faceoff.prg` *in the* `progo11_Applications` *directory.*

Questions

a. Arrange a box with a white sheet of paper taped to it landscape style. Place the robot 4 inches from the side with the white paper. As accurately as you can, face-off the robot. (On my desk I have a rectangular notepaper holder which works perfectly as a fixture. I place one straight edge of the holder against the side of the box and the other against the front bottom

plate of the TJ PRO™ robot. I gingerly slip the holder laterally away so it will not affect the experiment.) What is the difference in the IR detector readings?[16]

b. *Use the difference in part a) to adjust the "equality" test of the IR detector outputs. Execute your program. When the robot stops, carefully, without moving the robot, download* `sensetjp.s19` *into the robot and measure the values of the IR detectors. What were the values you obtained?*
c. *Measure the angle the front plate of the robot makes with the wall. This angle will be small (at least I hope so!) so you will have to do some clever measuring. Perform the experiment 10 times and measure the angle 10 times. What is the mean value of the angle?*

Program Exercise
Combine parts of `sensetjp.prg` and `faceoff.prg` so you can see the IR detector values displayed as the robot faces off.

9.2 Application Program Descriptions

You can use any C-Language feature to extend the PROGO™ language. The following PROGO™ programs take advantage of that capability and define named constants using the `#define` feature. For example, `attractjp.prg` employs

```
#define LOW_IR 95
#define MED_IR 99
#define DELTA_IR 2
```

to define three constants. Everywhere the named constant `LOW_IR` appears the C-preprocessor will replace it with 95. Similarly for the other named constants.

The syntax for `#define` is simple:

```
#define <Name_of_constant> <string_of_characters>
```

where by tradition `<Name_of_constant>` consists of all capital letters. The semantics are equally simple. Wherever `<Name>` appears in the program it is replaced by `<string_of_characters>` before compilation.

Brief Description of `attractjp.prg`

TJ PRO™ will be attracted by anything that moves close to it, or it moves close to! If TJ PRO™ bumps into anything it becomes shy and backs off and then pouts 3 seconds. If an object faces the front of the robot after the pouting period, the robot will move towards the object.

[16] In my experiment, the leading front edge of the bumper was 5.125 inches from the box. The left IR detector read 115-118 and the right one 119-121. So, the IR detectors were not matched and their outputs varied with noise by as much as two or three units. If I take 117 and 120 as the nominal readings, then "equality" of the two sensors translates to "`| irdr - irdl | < 3`". The program `faceoff.prg` does a better job using this test rather than a purely equal test of the two IR detector readings.

Playtime with `attractjp.prg`

Put the robot in the middle of a room with lots of free space around it. Reset the robot and touch the back bumper switch to start the robot. Nothing happens. Wave your hand in front of the robot. It's alive! Now, slowly move your hand about. TJ PRO™ will follow your hand motion unless it locks onto your leg and bumps you. Miffed, TJ PRO™ backs off and turns some random angle and pouts. After pouting for three seconds, the robot will head directly for any close-by object, even the object it bumped, provided it still faces the object. Or, if the robot is faced away from the bumped object, it will wait until an object moves close to its front IR detectors.

Questions

a) Roughly, how close does your hand or object have to be to activate attraction*?*
b) What program parameter will change the answer to a)?[17]
c) Roughly determine how close your hand has to be for the robot to track it?
d) What program parameter will change the answer to c)?[18]
e) Make a copy of this program and make changes to the parameters and observe the behavior changes. Try to predict what will happen beforehand. How often are you surprised at the robot behavior resulting from your changes?

Brief Description of `avoidtjp.prg`

This program realizes a simple collision avoidance program. TJ PRO™ will read each IR detector, and turn away from any obstacles in its path. Also, if something hits TJ PRO™'s bumper, it will backup, turn, and go forward again.

Playtime with `avoidtjp.prg`

Put the robot anywhere in a room. Reset the robot and touch the back bumper switch to start the robot. The robot will begin to move about avoiding bumping into obstacles. Sometimes the robot backs up and jams into an obstacle without responding until the backup action timeout occurs. The program doesn't account for that situation. In the section on multitasking we will see how to correct for this behavior.

The robot may get trapped between two obstacles or caught in a corner and oscillate indefinitely. I call this behavior the Braitenburg Trap, in honor of Valentino Braitenberg whose first vehicle in synthetic psychology will invariably oscillate in a corner until it dies of exhaustion.[19]

Caution!

[17] LOW_IR 95
[18] MED_IR 99
[19] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, 1984.

Do not let TJ PRO™ oscillate too long as it can overheat and damage the motor control electronics. Oscillations also increase the wear on the motors and gearboxes, decreasing their lifetimes.

Questions

a) What obstacles in your room does TJ PRO™ get stuck on/under/against? Explain why the robot got stuck for those obstacles.

b) Explain the Braitenberg trap. Why does the robot oscillate?

c) With the existing program, TJ PRO™ will not see certain types of black surfaces and bump into them. What program parameter will change the sensitivity of TJ PRO™ so that it might possibly see the black surfaces?[20] If you program the robot to "see" the black surface, what happens when the robot approaches light colored surfaces?[21]

d) Make a copy of this program and make changes to the above parameter and observe the behavior changes. Try to predict what will happen beforehand. How often are you surprised at the robot behavior resulting from your changes?

Evaluation of `avoidtjp.prg`

The robot will not perform well in both a light colored environment and a dark colored environment using `avoidtjp.prg`*. In environments that have both light and dark obstacles, the overall performance can be marginal. How can this be corrected or, at least, be improved? If the robot could somehow learn to adjust IR thresholds according to environmental circumstances, surely its performance would improve. This brings us to the next program.*

Description of `avcaltjp.prg`

This collision avoidance program that learns to adjust its IR sensitivity based on environmental factors. The self-calibration is based on bumper activity and average translational speed (not velocity). TJ PRO™ will read each IR detector, and turn away from any obstacles in its path that reads greater than the learned threshold. Also, if TJ PRO™'s front bumper hits something, it will backup, turn "randomly", and go forward. If something hits TJ PRO™'s back bumper it will go forward only for half the time a front bump causes the robot to backup, turn "randomly", and continue forward.

The novel feature of this program is its ability to adjust its behavior based on environmental conditions. The form of learning employed, while simple, is quite powerful. Whenever the robot bumps an object from the front, the control algorithm "presumes"[22] that the IR threshold value (`avoid_threshold`) must be too high and the objects being bumped do not reflect enough light to exceed the threshold before the robot bumps into it. The `avcaltjp.prg`

[20] AVOID_THRESHOLD 100

[21] The robot becomes "shy" and will not approach as close to obstacles as it did before.

[22] To be more precise, the programmer who wrote the algorithm, namely me, presumes! I will tend not to make such fine, semantic distinctions, but the reader should always be aware of their underlying philosophic importance.

algorithm corrects for the undesired bumps by increasing the sensitivity of the robot motion control to lower IR readings. The process continues until there are no further bumps. For low or overhanging objects that TJ PRO™ cannot see, lowering the IR threshold is counter productive as the bump has nothing to do with the threshold being too high. Such situations "mislead" the algorithm, but only temporarily, and insignificantly, if such situations are statistically rare.

A new problem now arises. The robot can avoid "painful" bumps by simply not moving or spinning in place! I discovered this in my first attempt to get it to automatically calibrate the IR threshold that invokes aversion or evasive moves by the robot. The robot would "panic" after a number of "painful" bumps in close sequence and simply spin in the middle of the floor! To avoid these behavior *attractors,* there must be a counterbalancing driving force "urging" the robot onward. The program does this by positively reinforcing average forward motion during a fixed-sized, periodic time window. I experimentally determined 3 seconds for mixing light and dark environments: my study (light environment with brown stained, wooden baseboards and scattered cardboard boxes) and my kitchen (black baseboards all around). I found 5 seconds too long and 1 second too short. With this countervailing force, the robot might "panic" for a while, but, the continued urging of forward motion by steadily decreasing `avoid_threshold` would get it moving forward again.

I limited the dynamic range of IR threshold adjustment between 90 and 120. These constants can be changed through the define constants

```
#define THRESHOLD_HIGH 120
#define THRESHOLD_LOW   90
```

The IRs typically do not read below 84 (everything is an obstacle) and above 127 (nothing is an obstacle), so working thresholds should not be too "close" (experimentally determined) to those numbers.

Since this program produces extremely interesting robot behavior, you will likely want to play with this program extensively. I recommend that you convert the above constant parameters, as well as the others discussed here, into terminal input variables so you can conveniently change them from your PC. Otherwise, you have to change the program, unload the old version and download the new version. If you take this approach, do not forget to initialize your variables appropriately.

The most difficult part of the program appears in Figure 4. The instantaneous translational speed of the robot equals `(speedr + speedl)/2`. By taking a running average of the robot translational speed (lines 30),

$$average\_speed_i = average\_speed_{i-1} + (speedr + speedl - 2*average\_speed_{i-1}) /(2*n);$$

or

$average\_speed_i = (n-1)*average\_speed_{i-1}/n + (speedr + speedl)/2*n;$

the program nominally can determine if the robot is moving about. The average_speed being large, close to 100%, does not guarantee the robot is moving about since its wheels could be spinning full forward with the robot hung up. In such situations you have to rescue the robot anyway, so a "hung up robot" has little effect on the correct functioning of the program under normal circumstances.

The variable `mark_cal_time` records the instant of time the running average of the speed begins to compute. After `AVERAGING_TIME` seconds (line 13), the program resets computing `average_speed`, checks to determine whether the robot has performed better than `AVERAGE_SPEED_MIN` (line 16), and, if not increase the `avoid_threshold` by 3 (line 20). Next, the code initializes the parameters for computing `average_speed` once again (lines 22, 23, 24).

```
1  Function average_speed_calculation
2  /*
3  Computes a running average of the robot forward speed. This average must
4  be above a minimum value or the avoid_threshold is increased to make the
5  robot less sensitive to obstacles.
6  */
7  Function_begin

8  /* Set new start time for average calculation time window?*/
9  If mark_cal_time equal_to 0
10 then
11 Set mark_cal_time to timertjp ok
12 end

13 If abs(timertjp-mark_cal_time) greater_than AVERAGING_TIME
14 then
15 /*Adjust IR threshold if average speed not up to snuff*/
16 If (average_speed less_than AVERAGE_SPEED_MIN)
17                     and (avoid_threshold less_than THRESHOLD_HIGH )
18 then
19 /*Not going forward. Get less sensitive*/
20 Set avoid_threshold to avoid_threshold + 3 ok

21 /*Reset calculation window for running average of robot speed */
22 Set n to 0 ok
23 Set average_speed to 0 ok
24 Set mark_cal_time to 0 ok
25 end
26 end
27 or_else
28 /*Calculate the average speed during a three-second time window*/
29 Set n to n+1 ok
30 Set average_speed to average_speed +
                        (speedr + speedl - 2*average_speed)/(2*n)ok
31 end

32 Function_end /*end average_speed_calculation*/
```

Figure 4. This code urges the robot to move forward.

The functions `backup` and `frontup` in Figure 5 adjusts the threshold whenever the robot bumps into anything. Front-bumps invoke `backup`, which lowers the IR threshold, making the robot more sensitive in avoiding obstacles. Back bumps invoke `frontup`, which increases the IR threshold, making the robot less sensitive to obstacles. The latter may seem counter intuitive or just simply wrong. Without it, however, the robot can get in such a "panicked" state that it backs up full speed and jams into a wall or other object and strains and whines as its wheels spin on the floor, not a pleasant sight or sound! The average speed adjustment will eventually correct this behavior, but it takes 3 seconds per adjustment of 3 threshold units, so the robot might be in "pain" for 12 seconds or more!

```
Function backup
Function_begin
  Reverse
/*If front bump, get more sensitive*/
  If avoid_threshold greater_than THRESHOLD_LOW
  then
    Set avoid_threshold to avoid_threshold - 3 ok
  end

/*Time out back reaction motion*/
  Wait 600 ms
  turn_random call

Function_end /*end backup*/
/*************************************************************************/

Function frontup
Function_begin
  Go
/*A back bump, get less sensitive*/
  If avoid_threshold less_than THRESHOLD_HIGH
  then
    Set avoid_threshold to avoid_threshold + 3 ok
  end

/* Time out forward reaction motion*/
  Wait 300 ms

  turn_random call

Function_end /*end frontup*/
/*************************************************************************/
```

Figure 5. This code adjusts the IR threshold when bumper contact occurs.

The two lines of the main program shown below insure that the code does not generate values of the IR threshold outside the specified range limits.

```
   /*Don't get too insensitive!*/
      if(avoid_threshold >THRESHOLD_HIGH ) avoid_threshold=THRESHOLD_HIGH;
```

```
/*Don't get too sensitive!*/
   if(avoid_threshold<THRESHOLD_LOW) avoid_threshold=THRESHOLD_LOW;
```

The rest of `the main program` in `avcaltjp.prg` more or less duplicates `avoidtjp.prg.`

Playtime with `avcaltjp.prg`

Put the robot anywhere in a room. Reset the robot and touch the back bumper switch to start the robot. The robot may begin to move about avoiding obstacles. Since it starts at the high IR threshold range, it will probably avoid most obstacles, even in a cluttered environment.

If the environment is too cluttered, the robot oscillates with "indecision". There are actually two distinct causes for indecision:

1) Everything appears too close to the robot and it rapidly turns here and there, or spins, trying to find a free path, but getting nowhere.
2) When the indecision manifests as oscillations back and forth from left to right, the cause most likely is the Braitenburg Trap. You probably saw this behavior when TJ PRO™ ran `avoidtjp.prg`.

If there is only marginally enough room for the robot to pass between two obstacles with an IR reading of 120, the robot will oscillate a great deal before passing through, although the process is quite painful to watch. If the opening is too marginal, the oscillations will take too much time (>15 seconds). In such situations, rescue the robot to prevent possible motor burnout.

Watch the performance of the robot. Expose the robot to a variety of rooms and situations. If the environment changes radically from light to dark obstacles, for example, the robot will start bumping into obstacles for a while until it adjusts its threshold to accommodate the new situation. If the switch is from dark to light obstacles, the robot may back up in "fear" or spin indecisively until it eventually compensates.

In mixed environments the robot will forever be adjusting its threshold, occasionally bumping into things, occasionally being "shy" or "fearful". Depending on the detailed nature of the environment, the robot may have extended periods of nominal behavior as well. Watching the robot in these situations is quite fascinating!

Questions

a. Do you consider that the TJ PRO™ executing `avcaltjp.prg` *actually "learns"? Carefully defend your position with a written response and file it for future reading.*

b. Ask people who knows nothing about robots to observe TJ PRO™ in action. Do they detect any "learning"? (Their answer will certainly depend upon their observational capabilities). Help them (bias them?) to see how the robot changes its behavior. How do they respond now?

c. Change the following to serial input parameters and write the necessary serial IO statements. Be sure to declare them appropriately.

Parameters:

```
#define AVERAGE_SPEED_MIN  50    (Range: 0 to 100)
#define THRESHOLD_HIGH 120        (Range: 84 to 127)
#define THRESHOLD_LOW   90          (Range: 84 to 127)
#define AVERAGING_TIME  3000L     (Range: 0 to infinity)
```

Vary these parameters and observe robot behavior changes. Can you quantify what you see (To perform quantitative analysis of intelligent machine behavior, in general, is a research question!)? In particular, note what happens for extreme values of the various parameters. The upper bound on the parameter `AVERAGING_TIME`*, in principle, does not exist. For practical matters, after a certain value (10000? Less?) the robot decision time frame becomes so large that, were it an insect, it would surely get eaten before it could "decide" about any behavior changes!*

IR Remote control of the TJ PRO™

You can program your TJ PRO™ to be attracted by 40KHz modulated IR light, say from your TV remote. If the robot bumps into anything, make it shy and have it backs off and stop. Point your TV remote at it and press any key. Have the robot start up again and follow the light from the remote. Make TJ PRO™ "heel" as a pet dog using your TV remote!

If you modify `attractjp.prg` with the addition of one line, IRE_off, you obtain an IR remote controlled TJ PRO™ of sorts! The robot does not look for reflected light generated by its own IR emitters, rather, it looks for an external source. Any 40KHz modulated IR will serve as a source, and your TV remote generates such a signal. Now you have an IR remote controlled robot. To stop the robot, just let it bump into something!

Modifying `attractjp.prg` this way has a number of shortcomings…try `racetjp.prg` for yourself and see. I have written three similar programs racextjp.prg, raceytjp.prg, `raceztjp.prg` that modify `racetjp.prg` in various ways to improve the performance from my viewpoint. I think the "z" version is the best. What do you think?

The term "race" in these program names suggests you can line up several TJ PRO™ robots and race them. Of course, the 'PRO™ cannot tell who is its owner controller, so it can make for some fun games with people trying to sabotage each other's control!

Question

Observe the robot as it executes the four different race programs. Describe the behavioral differences you observed and contrast that with the almost minor variations in the program code.

Games with the Race Programs

You can device an incredible (Ok, permit me a little hyperbole!) number of games with TJ PRO™ executing these programs. I will list two examples.

Single TJ PRO™

1. Develop an obstacle course with boxes in an enclosed arena, say 10 feet by 10 feet square. The arena can be any shape or size, but a light colored, six-inch wall should enclose it. Competitors cannot enter the arena. Set up a home position at one corner and a goal position at the diagonally opposite corner. See which competitor can take TJ PRO™ from home to goal the fastest using a TV remote.
2. Forget about the arena. Just make an obstacle course. My bedroom is a natural obstacle course, so this part does not take any effort! Mark off a home and a goal position. Race against time again. A problem with this version of the game is controlling the controller! Where can the controllers legally position themselves during the course of a run?

Multiple TJ PROs™ : ROBORACES™[23]

Make TJ PROs™ into *ROBORACERS*™[24]. Put racing stripes and numbers on them. Layout a 20 feet long, straight, racecourse with START, FINISH and several LANE lines. You can, at your own risk, use colored tape or washable colored ink to make the lines.

Be careful!
Make sure whatever you use to mark lines does not stain your floor!

Line up the robots on the START line and the competitors on the FINISH line, with remotes in hand and THUMBS-UP! The competitors cannot move from their posts during the contest, no matter how badly they want to help their robot or do something unspeakable to someone else's robot. Assign a *Racetrack Announcer* to call the race. Being the announcer can be a most entertaining assignment! When the race track announcer commands THUMBS-DOWN, the competitors press the TV remote buttons and the race is on. The first TJ PRO™ to cross the finish line is the winner (regardless of who got it across!). Do not tell the competitors that they can control their opponents 'PRO. They will discover that soon enough amidst the laughter and heat of competition!

10 BEHAVIOR INTEGRATION

The first principle in behavior integration is to define specific, independent functions for each primitive behavior. In a multitasking environment, one can assign a process to each behavior or function. Adding behaviors amounts to adding processes or functions. Subtle coupling between processes makes this task more difficult than it reads. Fortunately, low level intelligent control often does not require multitasking. You can implement many complex behaviors through context switching of behavior functions. Let me explain.

[23] ROBORACES™ is a registered trademark of Mekatronix.
[24] ROBORACERS™ is a registered trademark of Mekatronix.

You write a repertoire of simple, basic robot behavior functions. Based upon the robots' sensory information and current state, these define the *context*, the main program selects the next function (behavior) to execute. Execution time for a behavior may be context sensitive or timed or use a combination of both. The *animat* program in this section takes the behavior context-switching approach.

The *animat* program will integrate the *avoid* and *attract* behaviors along with the IR threshold auto-adjustment processes. The resulting program, `animatjp.prg`, appears in the directory `progo11_Applications`.

Program Operation

Function `arbitrate` invokes `spin` for a fixed amount of time. Process `spin` checks for external 40KHz modulated IR radiation. If `spin` finds IR or spin-time-out occurs, `spin` will stop turning the robot and return to `arbitrate`. After executing `spin`, `arbitrate` invokes `attract` if `spin` detected IR light. As long as the IR reception is above a certain minimum, the robot continues to execute `attract`. If `attract` does not detect any light during a three second time-out, it quits and returns to `arbitrate`.

If `spin` does not detect any IR light, then `arbitrate` invokes `avoid` for a specified number of seconds. During execution, `avoid` invokes `average_speed_calculation` to help decide about IR sensitivity calibration.

Playing with `animatjp.prg`

With TJ PRO™ on a stand, download the program, press reset and push the back bumper switch closed. Monitor `avoid_threshold` while the program executes. Notice that this variable decreases by 3 each time you press the front bumper and increases by 3 each time you press the back bumper. A front-bump is "punishment" for the robot being too insensitive to its IR readings and bumping into too many objects. By decreasing `avoid_threshold`, the robot becomes more aware of objects further away. A back-bump is "punishment" for backing up and bumping into things, presumably because the robot is so "shy" (too sensitive to IR). Increasing the `avoid_threshold` decreases the range at which the robot can detect objects.

Monitor the robot behavior. Watch how it changes with time. After you understand the characteristic response pattern, take any TV remote and hold it in front of the robot with any button pressed, say a channel-digit. Keep the button pressed and slowly wave the TV remote from side-to-side. The wheels should change their speed to effectively turn the robot towards the remote.

Since the TV remote's IR signal is so strong, the robot IR detectors may both saturate, even when you point directly only at one of them. In such cases, the remote appears to be straight ahead and the robot turns both wheels equally. You might have to point 45 degrees away from center to generate different IR detector responses, hence, an IR controlled turning motion. To

gain better IR control, you can partially block the remote's output LED with black electricians tape or mount a one inch black pen tube over the IR LED to make the IR signal more collimated, hence, directional.

Place TJ PRO™ into an environment with different colored walls. Watch the robot's response as it moves about.

Questions

a. *When the robot falls into a Braitenberg trap does it remain stuck there? Explain what happens.*[25]
b. *How long can the program stay in the* `attract` *process?*[26] *Hypothesize an answer based on experiments. What happens if you keep a TV remote button pushed down while pointing the remote at the front of the robot? You can also read the code for* `arbitrate` and `attract` *and find out.*
c. *Run the robot in a uniformly light-colored environment and then switch to a uniformly dark-colored environment. What happens initially?*[27] *Switch environments again and observe the initial behavior. If the light-colored environment is cluttered and the obstacles are separated by just a few robot diameters, what do you expect the initial behavior to be?*[28]
d. *If you run the robot in an environment with light and dark wall and obstacles, what behavior do you predict?*[29]

11 FURTHER EXPLORATION

This introductory manual only hints at the tremendous experimental possibilities available to you with your TJ PRO™ robot. The basic understanding of TJ PRO™'s capabilities that you have received from studying this manual and carrying out its experiments will help you to develop truly sophisticated robot behaviors.

Good luck and enjoy☺

[25] Typically, `spin` will break the trap.
[26] The program stays in `attract()` as long as external IR is detected.
[27] The robot frequently bumps into walls, and then not at all as it "learns" the new environment.
[28] TJ PRO™ should "shy" away from the light obstacles at a larger distance than from dark obstacles. If the obstacles are several diameters apart from each other, the robot might tend to spin ineffectively until the average speed requirement kicks-in and makes the robot less sensitive to IR.
[29] The robot will "seek" an IR threshold sensitivity that will prevent bumping into obstacles and at the same time produce a net 50% average speed. This goal may or may not be consistent with reality.