University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory
Fall 1999

DENDRITES: Final Report

Jeffrey Wisgo
Email: Locksley@ufl.edu
December 12, 1999
TAs: Scott Jantz, Ivan Zapata
Instructor: A. A. Arroyo

Table of Contents

Abstract 3

Executive Summary 4

Introduction 6

Integrated System 7

Mobile Platform 9

Actuation 10

Sensors: Experimental Layout and Results:

- IR Sensor Detail and Tests 11
- IR Sensor Graphs 15
- CdS Sensor Detail and Tests 17
- CdS Sensor Graphs 19

Communication 20

Behaviors:

- Tag 22
- Simon Says 24
- Follow the Leader 25
- Follow the Light 26

Conclusion 27

Documentation 29

Appendix A: Program Code "multi10.c" 29

Appendix B: Program Code "commsy14.c" 56

Appendix C: Expenses 72

Abstract

Dendrites are four mobile robots which use very low-cost hardware to achieve obstacle avoidance, robot location and inter-robot communication. These actions have been integrated into complex game playing behaviors such as Tag, Simon Says, and Follow the leader. Problems arose in programming Tag with regards to designing a suitable communication protocol and a responsive bump-sensor.

Executive Summary

Each of the four Dendrites consists of the TJ™ chassis and the MTJPRO11 microcontroller board, designed by Mekatonix, Inc. The MTJPRO11 board contains the Motorola 68HC11 microprocessor and 32K of SRAM, as well as 8 analog input ports and 5 servo control outputs. Two Hitec HS-422 servos were used for movement, and a visible-light LED was used to display useful information regarding the current behavior. Each Dendrite was powered with six NiCad rechargeable batteries connected in series. Dendrite code could identify the Dendrite it was running on by the reading of the jumpers normally used for the bump switch.

Communication, robot location and obstacle avoidance were all done using five Sharp IR receivers (spread nearly equal around the robot) and five IR LEDs per robot (placed above receivers). The leader Dendrite also utilized a single collated CdS light cell and six contact switches. Experiments showed that IR LEDs have a

range of 5 feet and an approximate spread of +/- 25°. Sharp IR receivers were shown to have an approximate spread of +/- 40°. These results confirm that the decision to use five IR LEDs and receivers per robot only allows for small windows of non-reception ('dead zones'). It was experimentally shown that the CdS cell could effectively detect when a bright light was shining directly at it (or at an angle), but it could not determine the distance to the light source.

Dendrites were programmed for three main behaviors: Tag, Simon Says, and Follow the Leader (who follows the light). These behaviors all contain one or more sub-behaviors such as obstacle avoidance, robot communication, robot location, light following, and collision detection. In the following paragraphs, each behavior is outlined briefly. For more details see the Behavior section of this paper.

Tag works with only two dendrites, the leader and another Dendrite. It begins with the leader chasing another robot who is actively avoiding it and avoiding obstacles. When the robots collide (and this collision is communicated to the non-leader robot) both robots switch roles. This results in the robot who is *it* is now not *it* and vice versa. The robots now proceed to chase or avoid, based on their

current role.

Simon Says works with all four Dendrites and works in the following manner:

The leader randomly decides on an action to perform (spinning in a circle,

backing up, etc.) and then broadcasts that action to perform (using IR). The leader

then performs the chosen action. When the other Dendrites receive the IR

message, they re-broadcast the signal and perform the action. The leader waits a

short time and the process repeats.

Follow the leader is invoked by shining a bright light (i.e. flashlight) on the leader

who broadcasts a 'lets play follow the leader' message to other robots. The leader

then proceeds to follow the light itself, with the other robots following behind it. As

a result, all the robots follow the light with only robot actually having a CdS sensor.

The CdS sensor is virtually 'shared' by the Dendrites, and this sharing can be

exploited in future designs by adding different sensors such that each robot

contains only one (advanced) sensor, but has virtual access to the others.

Introduction

This project initially stemmed from my desire to observe robots interacting in some form. From the beginning, I knew that the robots would need to be extremely inexpensive and easy to build so I could focus on the programming of robot behaviors. I wanted the robot communication to be versatile as to allow communication in-motion or at a distance of several feet. After discovering that several IMDL students had done similar projects using sonar and IR for communication [Garcia-Feliu and Turner], where docking was used for IR communication, I decided to design a simpler system using only IR and allowing for communication without docking. Since IR transmitters and receivers are inexpensive, this fulfilled my budget requirements.

In order to demonstrate the robot abilities to communicate, find each other and avoid obstacles, I decided to design several 'robot games' which would be played. For this I chose the popular children's games Tag, Follow the Leader, and Simon Says. Although not strictly productive behaviors themselves, these behaviors showed that a simple and inexpensive robot community had the capability for complex and interesting activities.

This paper begins with a discussion of the general integrated system used in the

Dendrites. Next is a description of the mobile platform, actuation mechanisms,

and sensors detail and experiments. Behaviors, the cornerstone of the Dendrites,

are discussed in the following section. Last is the conclusion, followed by several

appendices relating to code and project expenses.

Integrated System

Since the Dendrites were designed with an emphasis on programming, the

integrated system chosen is quite simple. The heart of the system, the MTJPRO11

(see figure 1), contains a Motorola 68HC11 microprocessor and 32K of SRAM. All

inputs and outputs connect directly to the M68HC11 or another chip on the

MTJPRO11 (such as one which produces 40kHz).

The inputs to a Dendrite contain five Sharp IR receivers and are connected

directly to analog ports PE2 – PE4 and PE6 - PE7 (refer to appendix A for exact

mappings). The Dendrite leader also contains a CdS light sensor and a group of

contact sensors, connected in parallel. The former connects directly to PE5, and

latter to PE1. Additionally, a reset switch, power switch and download switch are

connected to the MTJPRO11 through headers not shown on diagram 1. Analog

port PE0, which was originally intended for the connection of several different

bump switches, has instead been used as a hardware robot identification selector.

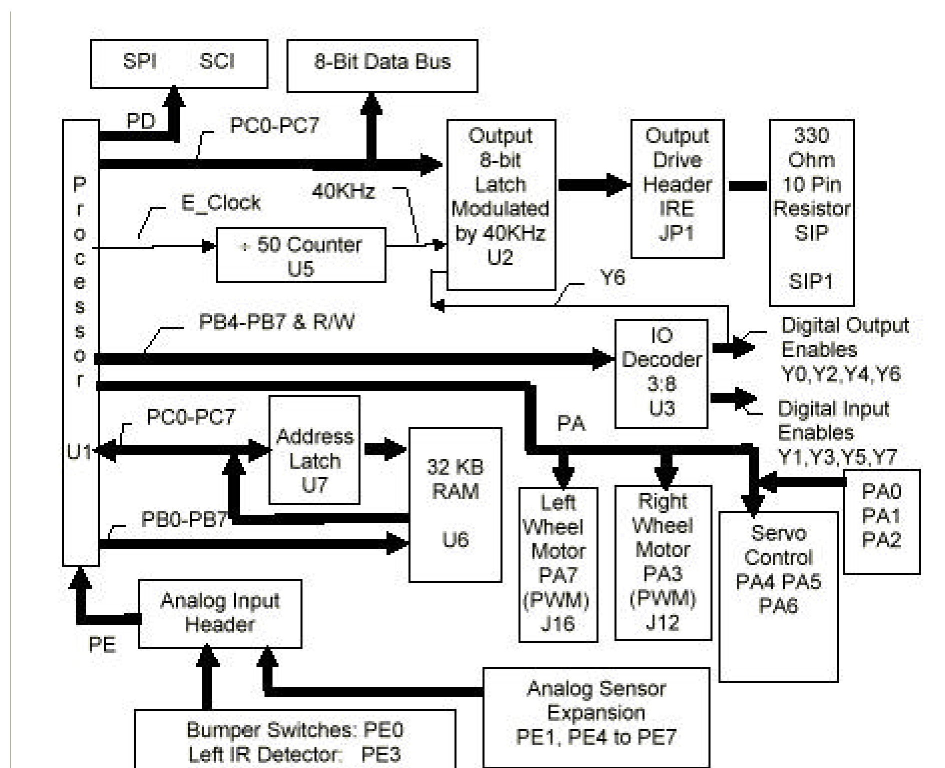
See appendix A for information about jumper settings.

The outputs from each Dendrite contain two servos directly connected to PA3 and

PA7. The five IR LEDs are connected to digital outputs Y0-Y4 and the visible light

LED is connected to digital output Y5. As we will see, this simple suite of sensors

and actuators allows the Dendrites to perform complex behaviors.



Mobile Platform

The chassis chosen for the Dendrites is the TJ™ body, designed by Mekatronix,

Inc. (see figure 2). The TJ™ body is a small and simple frame made of balsa wood

which was cut out of a T-Tech machine in IMDL lab. It supports two servos and a

6-pack of 1.2 Volt AA NiCad batteries (not shown). The servo placement design

allows pivoting about its own axis, a convenient feature for navigation.

Additionally, the frame supports topside mounts for LEDs and slots for underside

mounting of IR receivers. Each Dendrite uses three topside mounts and velcro for

attaching the five IR LEDs. The underside mounts were not used since they were

far from the disc's edge, a placement which allows a limited reception angle.

Instead, the IR receivers were attached to the underside using velcro. The CdS

cell(not shown) was attached on a raised wood platform directly in between the

two front LEDs.

The leader Dendrite uses a circular bump ring (made of wood) which fits around

the upper disc as seen in Figure 2. This ring, which triggers the contact switches

on the disc, caused considerable difficulty in regards to the Tag behavior. First, it

was too large and required extensive sanding to fit nicely. Second, the Dendrites

didn't exactly line up vertically, so the ring was made thicker by gluing additional

bump rings on both top and bottom of it. Nevertheless, the bump ring would *still*

fail to trigger the contact switches at least two-thirds of the time. To facilitate

proper contact switch operation, much change (pennies, nickels, etc) was glued

on top of a Dendrite. In practice, this improved the contact switch operation only

slightly.

Figure 2: TJ™ body

Actuation

As previously mentioned, the Dendrites use a very simple locomotion scheme:

two HiTec HS-422 servos and a plastic caster to provide balance. The servos have

been hacked to act like bi-directional motors by disconnecting the internal potentiometer from the gear mechanism. Since the servo control algorithm (implemented in the servo's internal hardware) moves the servo slower for a nearer angle and faster for a farther angle, this can be used to our advantage. All that is needed is to give the servo a certain positive or negative angle, and it will go forwards or backwards forever, with speed (non-linearly) proportional to the size of the angle. By giving the right and left servos certain angle combinations, the robot can move forward, backward, left, right, and pivot either direction.

One effect of using the hacked-servo design was each robot had to be individually calibrated. This was done by first setting the potentiometers (by hand) to a position such that the servo ceased moving. Since this method wasn't perfect, each completed Dendrite was run through a servo test which swept each servo through all available values. When the servo stopped moving, I recorded the servo value associated with that point in time. That value was used as an offset when giving that Dendrite's servo any motor command. See appendix A for more details.

Sensors: Experimental Layout and Results

IR Sensor Detail and Tests

Infrared(IR) light has wavelength around 900nm, and is modulated at 40kHz to

improve signal-to-noise ratio. IR is emitted using small IR LEDs and captured with

Sharp GP1U58X receivers hacked to produce an analog output between 86 and

128, with higher numbers corresponding to more IR. On the MTJPRO-11 board,

two 330Ω SIPs were used in parallel to achieve brighter LED output. All

experiments were done using a ruler and protractor (when necessary). Some

experiments were done twice with a different IR receiver to assure validity.

Experiment 1: IR reception vs. distance.

The infrared receiver was station to be directly across from the transmitter at all

times

Distance(cm)	IR1	IR2
5	128	127
10	128	127
50	127	128
80	123	124
90	122	123
100	119	120
110	117	118
120	114	116
130	113	114
140	111	112
150	108	110
160	106	108
170	105	106
180	102	103

210	97	98
240	94	94
270	92	92

Experiment 2: IR reception vs. LED angle (w.r.t. receiver)

This experiment was performed with 3 feet between transmitter and receiver.

The first IR column was recorded by using a LED whose entire area was exposed,

and the other used a LED with only the very tip visible (due to a wood mount). The

fact that that the data is not (perfectly) symmetrical around 0 degrees is probably

due to inaccuracies in experiment setup.

If the IR threshold is defined as 100, the LED effective range is +/- 25°. The

exposed LED allowed a slightly larger range, while the enclosed LED yielded

slightly larger readings overall. This data supports the claim that the two types of

LED configurations can be used independently without any problems.

Angle (deg)	**-60**	**-50**	**-40**	**-30**	**-20**	**-10**	**0**	**10**	**20**	**30**	**40**	**50**	**60**
Irexp	88	90	91	96	105	116	121	120	112	96	92	88	87
Irenc	87	87	87	94	109	118	122	121	115	102	87	87	87

Experiment 3: IR reception vs. Receiver angle (w.r.t. LED)

This experiment was performed with 3 feet between transmitter and receiver.

ANGLE (DEG)	**-50**	**-40**	**-30**	**-20**	**-10**	**0**	**10**	**20**	**30**	**40**	**50**
IR1	99	115	118	120	121	121	122	121	121	119	110

This data illustrates that the IR cans can receive incoming IR light at a range +/-

40° with little signal loss.

Experiment 4: IR reception vs. time

This experiment was performed by sending a 300 ms pulse from an IR LED which

reflects from a wall 1 foot away and returns to a receiver near the LED. The

upward and downward slope times are due to capacitances within the IR receiver,

wires, and MCU board. These slopes are the primary bandwidth-limiting factor in

robot communication.

Time (ms)	0	5	11	20	28	36	45	53	61	70	78	86	95
IR Value	85	96	104	110	113	116	117	118	119	120	120	121	121

The IR value stays a constant 121 until 318 ms:

Time (ms)	318	328	337	346	356	365	374	384	393	402	412	421	431
IR value	120	119	117	116	114	113	112	111	109	108	107	105	104

Time (ms)	440	449	459	468	476	485	493	501	510	518	526	535	543
IR value	103	101	100	99	98	96	95	94	93	92	91	90	89

Time (ms)	551	559	568	576	584	593	601
IR value	89	88	88	87	87	87	86

The time from no signal to full signal (86 - 121) is around 80ms, while the time from

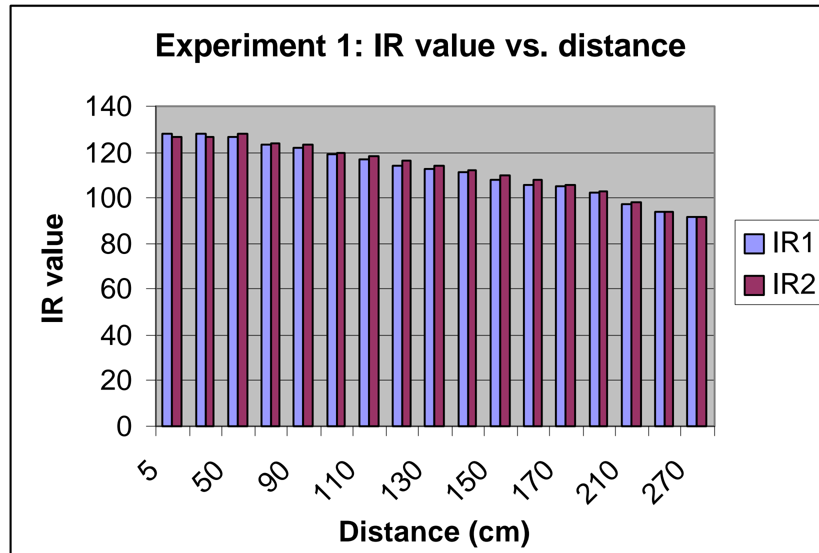
full signal to no signal (121 - 86) is around 300ms. If given a IR threshold of 100,

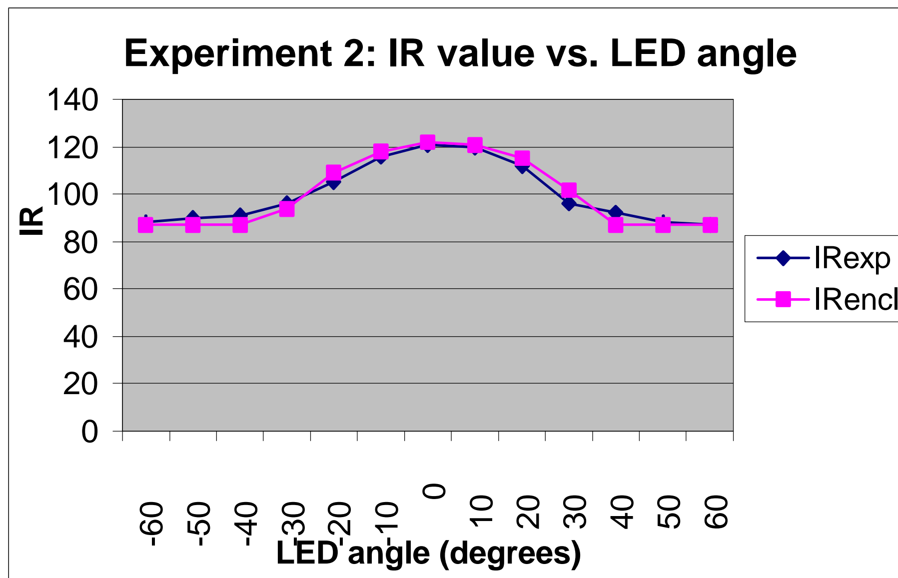
these values become approximately 10ms and 160ms, respectively.

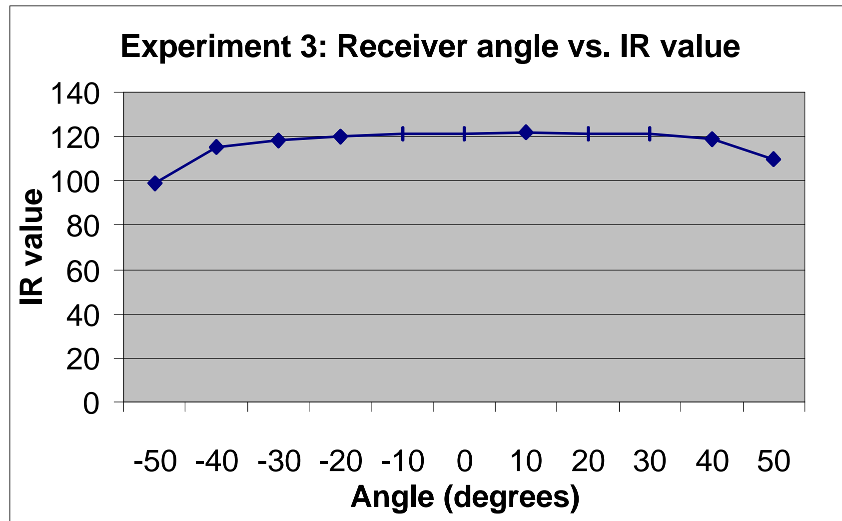
For more detail about how IR was used in the Dendrites, see the Communication

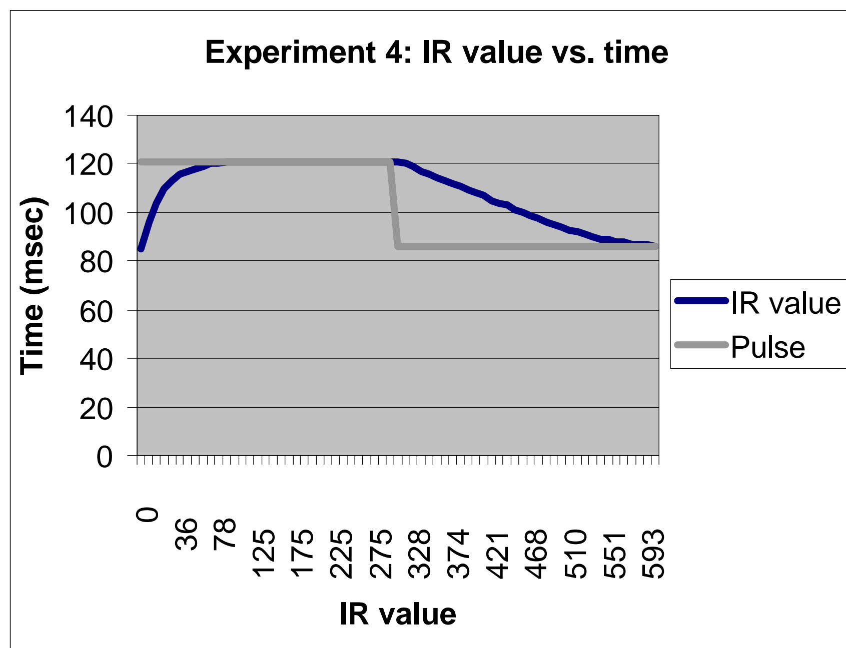
and Behaviors sections.

IR Sensor Graphs






Experiment 3: Receiver angle vs. IR value
IR value
140
120
100
80
60
40
20
0
-50 -40 -30 -20 -10 0 10 20 30 40 50
Angle (degrees)


Experiment 4: IR value vs. time
Time (msec)
140
120
100
80
60
40
20
0
0
36
78
125
175
225
275
328
374
421
468
510
551
593
IR value
IR value
Pulse

CdS Sensor Detail and Tests

The CdS cell is made of a material whose resistance decreases when light

contacts it. When put in series with a resistor and shunted across a voltage source,

it becomes a light-controlled voltage source. Trial and error was used to

determine a series resistance of 100kΩ for optimum sensitivity. The cell was made

more directional by collating it with 2cm of shrink tubing. A very bright flashlight

with a tight beam was used as a light source.

Experiment 1: CdS reading vs. distance

Distance (cm)	**0**	**20**	**30**	**40**	**50**	**90**	**120**	**150**	**180**	**210**

CdS val	1	1	4	4	4	6	7	10	15	15

This experiment illustrated that the collating of the CdS cell makes it directional within +/- 40° for a threshold of 25.

Experiment 2: CdS reading vs. CdS angle (w.r.t flashlight)

Angle (deg)	**-50**	**-40**	**-30**	**-20**	**10**	**0**	**10**	**20**	**30**	**40**	**50**
CdS val	162	149	67	16	9	11	19	37	120	170	170

Other CdS Sensor details

It was experimentally verified that the CdS Sensor's value was partially based on

the amount of light hitting the backside of the cell. This can be easily remedied by

placing something black (such as more heat shrink tubing) over the cell's

backside. Additionally, the CdS sensor reacted very strongly to ambient light

conditions. For example, if the sensor was placed in a dark enclosed space, it

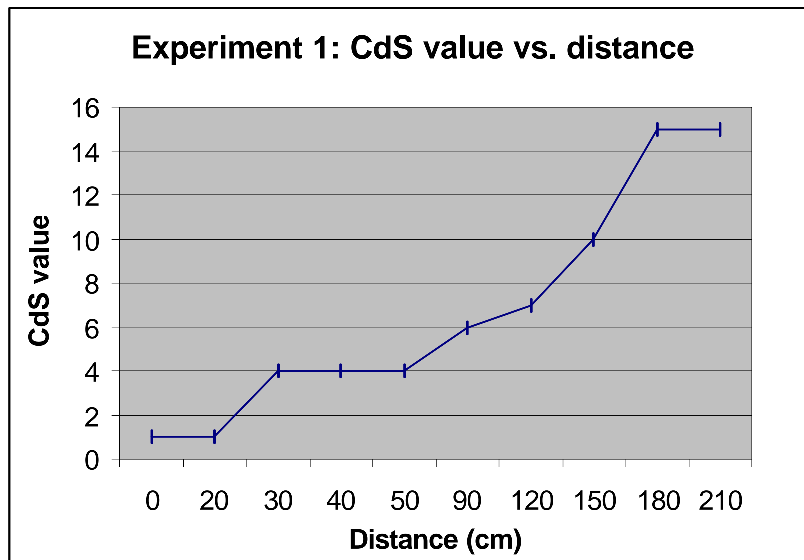
gave a reading of around 220. This ambient light sensitivity allows easy

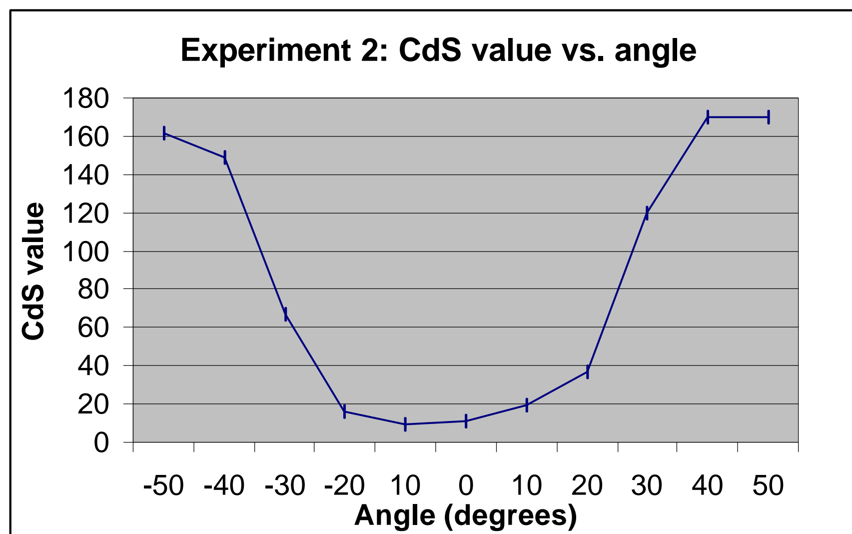
implementation of 'hiding in a dark place' or 'going into the open' behaviors.

For more detail about how the CdS was used in the Dendrites, see the Behaviors

section.

CdS Sensor Graphs





Communication

During the first few months of design and programming, I resigned to making a IR

communication protocol which would support up to four Dendrites at a time.

During each communication cycle, every robot would be informed of all the other

robot's locations, as well a as global 8-bit message sent by the leader. In special

circumstances, each non-leader robot could also send its own 8-bit message.

Although some synchronization was required to start the cycle running smoothly,

it would *in theory* work fine and provide a very versatile method of robot

communication and location.

I did manage to develop such a communication system (see Appendix B), with

several restrictions. First of all, it only supported two robots although I knew how

to increase this to four (with some difficulty). Secondly, the capacitance present in

the IR receivers required a communication cycle length of about 2 seconds.

Unfortunately, this was far too long for any real-time behaviors such as obstacle

avoidance and robot following.

As a result of this last consideration, I scraped this communication system for a

simpler system with a much higher bandwidth. In addition, I decided to design the

new communication system around my behaviors, instead of a making a general-

purpose communication system.

I decided to formulate a communication system which would work with my Tag

behavior. Tag required three things: obstacle avoidance, robot location and an

occasional message sent. It turns out the message is either a yes or no

(depending if the contact sensors were engaged), and this fact greatly simplified

the communication. My first problem was finding a way of distinguishing the IR

rays reflected off an object from those directly emitted from the other robot. I

decided to implement an alternating communication system that works as follows:

one robot waits, with its IR turned off, for a given amount of time to see if it

receives any IR pulses. If it does not receive a pulse, it waits a random time and

sends a pulse itself.

If the Dendrite does receive a pulse, it records the length of the pulse and then

immediately sends a pulse itself. Then the other robot hears this pulse and sends

its pulse again, repeating the cycle. If the leader's bump switch is engaged, it

sends a very long pulse to signal the other robot that it has been touched. Robot

following/avoiding was implemented easily by using the IR values during the

pulse, and obstacle avoidance was implemented by using the IR values the rest of

the time. This system worked well and accomplished all it needed too, without

being excessively complicated. The new communication cycle length was about

250 ms, an acceptable value for slow-moving real-time operations.

The communication system used in Simon Says is a watered-down version of the

above, with extra attention being paid to pulse length. The leader sends a pulse,

and the others record the pulse length and re-broadcast it themselves. Instead of

a long pulse/short pulse distinction, pulse lengths are quantized by 200 ms and

can in theory represent any number from 1 to infinity (requiring longer times for

larger numbers). The bandwidth of this scheme prohibits it from being used in

critical real-time behaviors, but it suits Simon Says just fine.

During Follow the Leader, no structured communication *per se* is in use. The

leader is constantly sending IR and the others are constantly following this IR. If

the leader stops sending IR, they stop following.

Behaviors

Programmed behaviors are the soul of the Dendrites – they combine their sensor's

and actuators into something much more than just eyes and feet; The Dendrites

become small creatures able to play children's games! The Dendrites have been

programmed for three games: Tag, Simon Says, and Follow the Leader(who

follows the light). Although stemming from actual children's games, several small

but important changes have been made to these behaviors to allow the Dendrites

to play them effectively. For example, Simon Says was changed into more of a Do-

What-I-Say game, since robots who fail to do the correct action do not 'loose' the

game.

Behavior: Tag

The Dendrite behavior of Tag is a close match to the commonly known children's

game of the same name, with a few small differences. This and other behaviors

are best represented by a list of steps, and those for Tag follow. ‘It’ stands for the robot who is doing the chasing.

(1) Initially, both robots are turned on, with the Dendrite leader as ‘it’ and the other robot as ‘not it’.

(2) At this point, ‘it’ begins to move towards ‘not it’, as ‘not it’ is attempting to move away from ‘it’ while avoiding obstacles. Both robots are running the alternating communication system referred to in the last section.

(3) When the robots make physical contact and the contact switches on the Dendrite leader engage, the leader’s next few pulses are of a very long length (~800 ms). Additionally, the leader moves quickly away from the other robot to assure his pulses are received correctly. The longer pulse length is received by the other robot who now becomes aware of the contact.

(4) At this point, both robots switch roles (the leader is now ‘not it’ and the other is ‘it’). The robot who is ‘it’ waits for a few seconds while ‘not it’ proceeds to run away from ‘it’.

(5) Once ‘it’ finishes waiting, it begins chasing ‘not it’ as in step (1), and the process repeats indefinitely.

In order to assure eventual contact (in a limited amount of time), the robot temporarily assigned to be 'it' was programmed to move much faster than the 'not it' robot could run away.

In addition to the preceding steps, two sub-behaviors have been added to improve the performance of Tag. First, each robot randomly backs up if it hasn't been tagged in awhile. This allows the robots to recover from unusual scenarios like getting stuck in a corner, or getting wedged on top of the another robot. Second, whenever a robot has not heard a pulse in a few seconds, it begins to turn slowly in a random direction. I call this 'floating', and floating effectively eliminates the IR reception dead zone between the IR receivers.

In the event that one of the Dendrites who was 'it' was near a wall, it would often head towards the wall – it was following the reflected IR from the other robot instead of the direct IR rays. This problem was ignored by keeping the Dendrites to relatively open areas. However, there is no known solution to this problem.

Behavior: Simon Says

While the Simon Says behavior is algorithmically simpler and took less

programming time than Tag, it demonstrates the Dendrites' ability to communicate more adequately than Tag does.

(1) Initially all robots are turned on and wait silently for the leader to act.

(2) The leader randomly chooses an action (move forward, spin, etc.) and broadcast an IR pulse corresponding to that action. It then performs the action.

(3) Upon receiving the pulse, the other robots re-broadcast the pulse to assure all robots have received the message. Then they perform the action themselves.

(4) Any robots not receiving the leader's pulse may receive the pulse sent in (3). If so, they respond as the robots did in (3). This step repeats until all robots (in range) have received the pulse and acted accordingly.

(5) After waiting for a short time, the leader repeats (1) and the cycle continues.

The re-broadcasting can have interesting cascading effects, especially when robots are set in a line. This scenario is similar to the children's game of 'Telephone' and shows how the robots can cooperate to achieve a wider range of communication.

Behavior: Follow the Leader

This behavior utilizes the leader's CdS light sensor and illustrates virtual sensor

sharing.

(1) During Tag, if the leader detects a strong light shining on its CdS sensor, it

sends a pulse length corresponding to 'lets play Follow the Leader' to the

other robots.

(2) After turning slightly and repeating this message to assure valid reception,

the leader begins following the light (see next section).

(3) Upon heard the message sent in (1), the other robots begin following the

leader while avoiding obstacles. As in Tag, they backup randomly to avoid

getting stuck.

(4) This behavior proceeds until the leader determines the light has gone at

which point all robots stand still.

Since the other robots don't have access to a CdS cell but are following the light

(indirectly), they are virtually sharing the sensor. In Tag, the bump sensor is

virtually shared by the other robot. In future projects, this idea may be used to

share multiple sensors between many robots.

Behavior: Follow the Light

Since following a light source with only a single sensor is not as simple as determining direction with five IR receivers, the algorithm follows:

(1) Record the CdS value. Move in a random direction (left or right) for 100 milliseconds.

(2) If the current CdS value is much higher than the recorded value, record the value, continue going in this direction, wait 100 milliseconds and repeat step (2).

(3) Otherwise, begin moving in the opposite direction. Wait 100 milliseconds and repeat step (2).

(4) Special case: the light source is very bright (above a given threshold). Move straight and re-evaluate the CdS reading 100 milliseconds later.

(5) Special case: the light source is very dim (below a given threshold). Begin spinning in a circle, stopping when a much brighter value is reached. If leader spins for awhile, decide that light has stopped and stop transmitting IR so other robots stop following.

Conclusions

Overall, the project was a moderate success. It was shown that a group of robots

with simple and inexpensive sensors could interact with one another and have

basic abilities such as obstacle avoidance, robot location, and inter-robot

communication. The Tag Behavior demonstrated that a simple communication

system could allow a robot to distinguish between its IR signals and that of

another. The inability of the bump ring to trigger the contact switches seriously

hampered this behavior, and a redesigned bumper/contact switch suite could

make the behavior much more dependable and impressive.

Simon Says illustrated that hacked Sharp IR receivers can support robot

communication, albeit at a slow bandwidth due to capacitance problems. Using a

few unhacked IR receivers for communication would increase the bandwidth and

improve Simon Says response time, but this is not easy due to the lack of

remaining analog input ports on the M68HC11. Follow the Leader/Follow the Light

was a reasonable success, in that it showed how following a light source can be

done with a single sensor and that simple robots can virtually share a sensor.

The main design flaw which wasted many hours of programming was the lack of

foresight when designing the original, general purpose, communication system. I

had realized that the large capacitance in the IR receivers would make a

communication cycle of around one – two seconds for a system with 8-bits per

cycle, but I failed to acknowledge that this bandwidth would be unacceptable for

any real-time application.

In future work I would like add more behaviors which make use of more robots

and provide more complex interaction (like manhunt). Another idea I would have

liked to try is the implementation of parallel processing by sharing complex

computations between many robots. Finally, I would have liked to implement a

organic hive simulation where each robot has a certain (variable) amount of life

and it must gather food and bring the food home to queen. Learning algorithms

could also be tested in this scenario.

Documentation

The 10 Stooges: The Intimate Life of a Colony, Jose Garcia-Feliu, UF IMDL,

Summer 1999

Group Behavior in Multiple Autonomous Agents, Bruce Turner, UF IMDL,
Summer

1999.

Mekatronix, Inc. http://www.mekatronix.com/

(I'd also like to Thank Scott Jantz and Ivan Zapata for all their useful advice)

Appendix A: "multi10.c"

```
// Dendrite program code - final version
// Written by Jeffrey D. Wisgo
// Date Completed: 12/2/1999
//
//
```

```
// Note: If you use a significant portion of my code for anything
//              I'd appreciate if you told me about it. Thanks.
// Email: Locksley@ufl.edu


#include <tjpbase.h>
#include <stdio.h>
#include <math.h>
#include <mil.h>


/* DEFINES */

#define CHASE_SPEED     30
#define RUN_SPEED  11

#define TURNTHRS 23

#define mode *(unsigned char *)(0xF000)

#define CENTER 180
#define DONTMOVE 0

// sync pulse must start with a 1 (i.e. > 128)
#define SYNC_CATCHUP 40
#define SYNC_PULSE 154
#define PING_THRS 100
#define COMM_THRS 100

#define COMM_DELAY 150


#define AVTHRS  110
//#define AVOID_THRS AVOID_THRESHOLD

#define NO_SIGNAL 500

#define BACK_IR     analog(7)
#define BACK_LEFT_IR analog(6)
#define BACK_RIGHT_IR analog(4)

#define LIGHT analog(5)

#define BUMPSW (analog(1) > 30)

#define MOVELEAD 5
```

```
#define SELF -1

// was 2
#define RESYNCTHRS 3

// was 3
#define SIGTIMEOUT 4

#define PULSECOUNT 3

#define CHDLYTME 6
/* was 5 */


/* These are just used for convenience */
#define ACT             1
#define WAIT        0

#define BACK            1
#define RIGHT           2
#define LEFT        3
#define BACKLEFT    4
#define BACKRIGHT       5

#define WANDER          1
#defineSPIN         2
#defineSIT              3

#define STOP        10
#define TURNLEFT    11
#defineTURNRIGHT        12
#define BACKUP          13
#define TURNRAND 14
#define FORWARD         15
#define RIGHTFORWARD 16
#define LEFTFORWARD 17
#define IRE_ON2 irlatch |= 0x1F;  *(unsigned char *)(0x7000) = irlatch
#define IRE_OFF2 irlatch &= !(0x1F); *(unsigned char *)(0x7000) = irlatch

#define LED1_ON irlatch |= 0x80; *(unsigned char *)(0x7000) = irlatch
#define LED1_OFF irlatch &= !(0x80); *(unsigned char *)(0x7000) = irlatch

#define LED2_ON irlatch |= 0x40; *(unsigned char *)(0x7000) = irlatch
#define LED2_OFF irlatch &= !(0x40); *(unsigned char *)(0x7000) = irlatch
```

```
/* NEW DATA TYPES */
struct robot_type {
  char exists;  // if exists = SELF, then I am that robot
                // exists = 1 implies that we are communicating
                // with that robot

  int angle; // robot's last known location
             // specified in angle, 0 = left, 90 = in front,
             // 180 = right, 270 = behind
  int dist;              // (very) approximate distance to the dendrite

  unsigned char msg;    // the most recent message from this robot

};




/* GLOBAL VARIABLES */

char msgtosend; // message I'll send to other robots
unsigned char irdr, irdl, irdb, irdbr, irdbl;
char robotid;
struct robot_type robot[6];
const unsigned char bitmask[] = {1, 2, 4, 8, 16, 32, 64, 128};
unsigned char irlatch = 0x00;
unsigned char signal_locked = 0; // used with ir_highest_now()
char lastdir = LEFT;  // used in ir_highest_now();
unsigned int runtimer;
int simonvar;
int mt; /* used as a speed multiplier to compensate for slower servos on some robots */

/* servo calibration global variables */
int rcal, lcal;

int amblight;
int bumpbuffer;

/* PROTOTYPES */
unsigned char bitof(unsigned char b, unsigned char x);
unsigned char ir_highest_now(void);
unsigned char ir_highest(void);
void get_ir(void);
int abs(int x);
```

```
void server();
void client();
void locate_robot(char id);
void calibrate_servo(int servo);
void motor_control(unsigned char motor, char val);
void motor_vector(char mag, int angle);
int receive_pulse(int timeout);
void check_hardware();
int away(int angle);
void blink(char times);
void simon(void);
void player(void);
void action(int, char);
void light_follow(void);
int receive_simon(void);
void follow_leader(void);
char signal_exists(void);
void error(void);

#define PULSESIZE 70  // was 60
#define FALLTIME 135  // was 140

void client() {
  int plen, it, gotpulse, nopulse = 0, movelock = 0;
  char retry,movewait;
  int extralen, chasedelay;
  char timeout, bwait;
  int turntime;

  extralen = 0;
  movewait = 0;
  bwait = 0;
  timeout = 1;
  chasedelay = 0;
  turntime = 0;


  it = 0; /* at start, client is 'it' */

  printf("Server Dendrite active - Playing Tag\n");

  while(1) {
    timertjp = 0;
    printf("+");
    LED1_ON;
    IRE_ON2;
```

```
while(timertjp < PULSESIZE  + extralen);

        get_ir();
IRE_OFF2;
while(timertjp < PULSESIZE + FALLTIME);
LED1_OFF;
        printf ("irdr: %d, irdl: %d\n", irdr, irdl);

        if (it) {
          if (irdr > AVTHRS) { motor_control(TURNLEFT,15); movelock = 1; }
          else if (irdl > AVTHRS) { motor_control(TURNRIGHT,15);movelock=1; }
          else {
            if (movelock) {
              motor_control(STOP,0);
              //printf("Obstacle out of way, stopping.\n");
             }
            movelock = 0;
           }
        }


        //printf("timeout: %d, ", timeout);
        if ((--timeout) == 0) {
          timeout = 1;
          //motor_control(STOP,0);
          //printf("RANDTURN!\n");
          motor_control(TURNRAND, 5);
        }

        turntime++;
        if (turntime == TURNTHRS) {
          motor_control(BACKUP, 45);
          wait(200);
          turntime = turntime - TURNTHRS + ((TCNT % 10) - 5);
        }

// if a valid pulse, this fills the global ir variables
plen = receive_pulse(PULSESIZE + FALLTIME);
if (plen < 15) {  // if no pulse found
  nopulse ++;
}
        else { /* VALID PULSE */
  if (bwait) bwait--;
  if (plen > 200) { // if we got 'special message'
            if (bwait == 0) { // make sure we didn't just get a message
                      turntime = 0; // reset turn counter
```

```
            bwait = 6;
          if (it) {
            it = 0;
            chasedelay = CHDLYTME;
          }
          else it = 1; // switch it status
          movelock = 0;
          printf("Hit received - switching 'it' status.\n");
        }

    }


    nopulse = 0;
    printf("[%d]",plen);

        wait(FALLTIME - 20); // wait for sender cool down
        locate_robot(0);

        if (chasedelay > 0) {
          motor_control(STOP,0);
          chasedelay--;

        }
        else if (!movelock) {
          if (it) motor_vector(RUN_SPEED, away(robot[0].angle));
              else motor_vector(CHASE_SPEED, robot[0].angle);
      timeout = SIGTIMEOUT;
    }
   }
   if (nopulse > RESYNCTHRS) {
    printf("Attempting to re-sync\n");
    wait(TCNT % 200);
    nopulse = 0;
   }
  }

}




void server() {
  int plen, gotpulse, nopulse=0, extralen, cnt, diff, movelock =0;
  int it, chasedelay, bumplock;
  char retry, movewait;
```

```
unsigned int lasttime;
char timeout;
char bwait;
int turntime;

int lencnt;
lencnt = 0;
chasedelay = 0;
bumplock = 0;
movelock = 0;
turntime = 0;

//bwait = 0;


printf("Server Dendrite active - playing Tag\n");

cnt = 0;
extralen = 0;
it = 1; // at beginning, client is 'it'
movewait = 0;
timeout = 1;

while(1) {

  timertjp = 0;

  LED1_ON;
  IRE_ON2;
  while(timertjp < PULSESIZE + extralen);

        get_ir();
  IRE_OFF2;
  while(timertjp < PULSESIZE + FALLTIME);

  LED1_OFF;

        if (it) { // obstacle avoidance
          if (irdr > AVTHRS) { motor_control(TURNLEFT,15); movelock = 1; }
          else if (irdl > AVTHRS) { motor_control(TURNRIGHT,15);movelock=1; }
          else {
            if (movelock) motor_control(STOP,0);
            movelock = 0;
          }
        }
```

```
        turntime++;
        if (turntime == TURNTHRS) {
          motor_control(BACKUP, 45);
          wait(200);
          turntime = turntime - TURNTHRS + ((TCNT % 10) - 5);
        }

if (((BUMPSW || bumpbuffer)) && (!bumplock)) {
  printf("We hit something! \n");

          //if (it) motor_control(FORWARD,40);
          //else motor_control(BACKUP,40);

          if ((robot[0].angle > 90) && (robot[0].angle < 270))  // if in front
                motor_control(BACKUP,40);
          else motor_control(FORWARD,40);

  if (it) {
    it = 0;
    chasedelay = CHDLYTME;
  }
  else {
    it = 1; /* switch 'it' status */
  }

           turntime = 0; /* reset turn counter */
  bumplock = 5;
  extralen = 800; /* tell client that we've been touched */
  movelock = 0;
  bumpbuffer = 0;
  continue; /* send message now */
        }

        if ((--timeout) == 0) {   /* if no signal for awhile */
          timeout = 1;
          //motor_control(STOP,0);
          motor_control(TURNRAND, 5);
        }


if (extralen) {
  if (!it) chasedelay = CHDLYTME; // so he doesnt get a 'head start'
          motor_control(TURNRAND, 5);
  extralen = extralen - 1;
  if (extralen == 800 - PULSECOUNT) extralen = 0;
}
```

```
    // if a valid pulse, this fills the global ir variables
    plen = receive_pulse(PULSESIZE + FALLTIME);
    if (plen < 20) {  // if no pulse found
      nopulse ++;
    }

    if ((BUMPSW) && (!extralen)) bumpbuffer = 1; /* queue up bumping incase we loose
contact quickly */

    else { /* VALID PULSE */

      nopulse = 0;
    //printf("[%d]",plen);

            wait(FALLTIME - 20); // wait for sender cool down (Was 20)
      locate_robot(0);

            if (chasedelay > 0) {
              motor_control(TURNRAND, 5);
              chasedelay--;
      }
            else if (!movelock) {
              if (it) motor_vector(RUN_SPEED, away(robot[0].angle));
                    else motor_vector(CHASE_SPEED, robot[0].angle);
                    timeout = SIGTIMEOUT;
      }

      if (bumplock) bumplock--;
    }
    if (nopulse > RESYNCTHRS) {
      printf("Attempting to re-sync\n");
      wait(TCNT % 200);
      nopulse = 0;
    }
  }

}

int receive_pulse(int timeout) {
  int start = timertjp;
  int time2;
  int max,val;

  max = ir_highest_now(); // used as min here
```

```
  while(1) {
    if (ir_highest_now() > max + 4) break;  // find UPWARD SLOPE
    if (timertjp > timeout + start) return 0; // if timed out
    if (ir_highest_now() > 115) break; // if a strong signal!
  }
  time2 = timertjp;
  get_ir();
  signal_locked = 1;
  wait(10);
  while(1) {
    val = ir_highest_now();
    if (BUMPSW) bumpbuffer = 1; /* queue up bumping incase we loose contact quickly
(TIMING?!?!? */
    if (max < val) max = val;  // get max
    if (val < max - 4) break; // find DOWNARD SLOPE
    if (val < 95) break; // if a weak signal!! (TWEAK THIS)
    /* if we receive a long ping, we are sending a message */
    //if (timertjp - time2 > 200) motor_control(STOP,0);
    /* if too long, must be an error */
    if (timertjp - time2 > 2000) {
      signal_locked = 0;
      return(1);
    }
  }

  time2 = timertjp - time2;
  signal_locked = 0;

  return(time2);
}



// This function sets a robot's location based on the values
// in the globals irdr, irdl, etc.
// So you must call get_ir() before calling this function!


void locate_robot(char id) { // uses IR to find direction of robot
  int loc;  // do we need a float? I hope not.
  int sum;

  switch (lastdir) {
     case BACKLEFT:
       robot[id].angle = 290;
       break;
```

```
    case BACKRIGHT:
      robot[id].angle = 70;
      break;
    case BACK:
      robot[id].angle = 0;
      break;
    case RIGHT:
      if (abs(irdr - irdl) < 5) robot[id].angle = 180;
      else robot[id].angle = 150;
      break;
    case LEFT:
      if (abs(irdr - irdl) < 5) robot[id].angle = 180;
      else robot[id].angle = 210;
      break;

  }

/*  this routine gives skewed results, above is easier
  printf(".");

  irdr -= 80; irdl -= 80; irdbl -= 80; irdbr -=80;
  irdb -= 80;
  robot[id].exists = 1;
  //printf(" %d, %d, %d, %d, %d  \n", irdl, irdr, irdbr, irdb, irdbl);
  loc = 72 * irdbr + 144 * irdr + 216 * irdl + 288 * irdbl;
  if (irdbl > irdbr) loc += 360 * irdb; // compensate for wrap-around
  sum = (irdb +  irdbl + irdbr + irdl + irdr);
  if (!sum) printf ("Division by zero!\n");
  robot[id]. angle = loc / sum;
  //printf("%d, %d: %d\n", loc, sum, robot[id].angle);
*/
  robot[id].dist = ir_highest_now(); // relative distance, test this!
                 //    ^^^ is this safe??
  switch(lastdir) {
    case LEFT: robot[id].dist = irdl; return;
    case RIGHT: robot[id].dist = irdr; return;
    case BACK: robot[id].dist = irdb; return;
    case BACKLEFT: robot[id].dist = irdbl; return;
    case BACKRIGHT: robot[id].dist = irdbr; return;
  }

}



int abs(int x) {
```

```
  if (x > 0) return (x);
  else return (-1*x);
}

void get_ir(void) {  // read ir values in now
  irdr = RIGHT_IR;
  irdl = LEFT_IR;
  irdb = BACK_IR;
  irdbl = BACK_LEFT_IR;
  irdbr = BACK_RIGHT_IR;
}


void get_robotid() {
  int mult;

  if (BUMPER < 5) {
    robotid = 1;
    rcal = -35;  /* tested for D1 */
    lcal = 33;
  }
  else if (BUMPER < 30) {
    robotid = 3;
    rcal = -4;     /* tested for D3 */
    lcal = 3;
  }
  else if (BUMPER < 80) {
    robotid = 2;
    rcal = 18;     /* tested for D2 */
    lcal = -23;
  }
  else {
    robotid = 4;
    rcal = 3;      /* tested for D4 */
    lcal = -6;
  }

  mult = ((robotid == 3) || (robotid == 4));
  mt = (1 + (1)*mult); /* used in motor_control() */

  robot[robotid].exists = SELF;  // set self tag
  return;
}

void display_info() {
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* clear screen */
```

```
  char place[]= "\x1b[1;1H";              /* Home cursor */

  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/

  printf("--== DENDRITE ==--\n");
  printf("Robot ID: %d\n", robotid);
  printf("Programmed by: Jeffrey D. Wisgo\n");
  printf("Code revision: 9.0\n");
  return;
}
// returns bit b of char x
unsigned char bitof(unsigned char b, unsigned char x) {
  unsigned char t;
  t = x & (bitmask[b]);
  return t;
}




void init_data() {
  int i;
  for (i=0;i<6;i++) {
    robot[i].exists = 0;
    robot[i].angle = 0;
    robot[i].dist = 0;
    robot[i].msg = 0;
  }
  msgtosend = 34;
}


// This function finds the middle point of each servo by
// assuming that there is some differential IR readings
// coming from an object (near) in front of it.
void calibrate_servo(int servo) {
  int diff, i;
  int val, min_right_val, min_left_val;
  int diff_right[31], diff_left[31];
  int min_right_read, min_left_read, max_right_read;

  IRE_ON2;
  printf("Calibrating servo.\n");
  min_right_read = 1000; min_right_val = 665;
  max_right_read = 0;
```

```
  for (val = -40; val < 40; val++) { // go through range
    if (!val) continue;
    motorp(servo, val);
    diff_right[val] = 0;
    diff = ir_highest_now();
    //diff = ir_highest();
    for (i=0;i < 60;i++) {  // spend some time at each spot

      wait(5);
      // add change in ir readings to total change
              diff_right[val] += abs(ir_highest_now() - diff);
      diff = ir_highest_now();
              //diff_right[val] += abs(ir_highest() - diff);
      //diff = ir_highest();

    }
    if (diff_right[val] < min_right_read) {
      min_right_read = diff_right[val];
      min_right_val = val;
      printf("Min found, reading %d, val %d\n", diff_right[val], val);
    }
    if (diff_right[val] > max_right_read)
      max_right_read = diff_right[val];

    printf("Val: %d, total change: %d\n", val, diff_right[val]);
  }
  diff = 0;
  for (val = -20; val < 20; val++) {
    diff += val * (max_right_read - diff_right[val]) / max_right_read;
  }
  printf ("** weighted average -> val %d\n", diff);

  printf("** Min reading %d on val %d\n", min_right_read, min_right_val);
  motorp(servo,0);



}


/* I could make this a more general function,
   but this is easier and gives better performance
   for now */

void motor_vector(char mag, int angle) {
  int val;
```

```
  if (DONTMOVE) return;

  //printf("*** motor_vector(%d, %d)\n", mag, angle);

  if (angle == 180) {
    motorp(LEFT_MOTOR, (mag + lcal)*mt);
    motorp(RIGHT_MOTOR, (mag + rcal)*mt);
  }
  else if (angle == 0) { /* just spin */
    motorp(LEFT_MOTOR, (-mag + lcal)*mt);
    motorp(RIGHT_MOTOR,(mag + rcal)*mt);
  }
  else if (angle == 210) {
    motorp(LEFT_MOTOR, (mag*2/3 + lcal)*mt);
    motorp(RIGHT_MOTOR, (mag + rcal)*mt);
  }
  else if (angle == 290) {
    motorp(LEFT_MOTOR, (0 + lcal)*mt);
    motorp(RIGHT_MOTOR, (mag + rcal)*mt);
  }
  else if (angle == 150) {
    motorp(LEFT_MOTOR, (mag + lcal)*mt);
    motorp(RIGHT_MOTOR, (mag*2/3 + rcal)*mt);
  }
  else if (angle == 70) {
    motorp(LEFT_MOTOR, (mag + lcal)*mt);
    motorp(RIGHT_MOTOR, (0 + rcal)*mt);
  }

}


void motor_control(unsigned char motor, char val) {

  if (DONTMOVE) return;

  switch(motor) {
    case STOP :
      motorp(LEFT_MOTOR, 0);
      motorp(RIGHT_MOTOR, 0);
      return;
    case TURNLEFT:
      motorp(LEFT_MOTOR, (-val + lcal)*mt);
      motorp(RIGHT_MOTOR,( val + rcal)*mt);
      return;
```

```
  case TURNRIGHT:
    motorp(LEFT_MOTOR, (val + lcal)*mt);
    motorp(RIGHT_MOTOR, (-val + rcal)*mt);
    return;
        case LEFTFORWARD:
    motorp(LEFT_MOTOR, (-val*1/4 + lcal)*mt);
    motorp(RIGHT_MOTOR,( val + rcal)*mt);
    return;
        case RIGHTFORWARD:
    motorp(LEFT_MOTOR, (val + lcal)*mt);
    motorp(RIGHT_MOTOR, (-val*1/4 + rcal)*mt);
    return;
  case TURNRAND:
          if ((TCNT % 100) < 50) {
      motorp(LEFT_MOTOR, (-val + lcal)*mt); // turn left
      motorp(RIGHT_MOTOR,( val + rcal)*mt);
    }
    else {
      motorp(LEFT_MOTOR, (val + lcal)*mt); // or turn right
      motorp(RIGHT_MOTOR, (-val + rcal)*mt);
    }
    return;

        case BACKUP:
          motorp(LEFT_MOTOR, (-val + lcal)*mt);
          motorp(RIGHT_MOTOR, (-val + rcal)*mt);
          return;
        case FORWARD:
          motorp(LEFT_MOTOR, (val + lcal)*mt);
          motorp(RIGHT_MOTOR, (val + rcal)*mt);
          return;

  default:
    motorp(motor, val*mt);
 }
}



void main(void)
{
 IRE_OFF2;
 LED1_OFF;


 init_analog();
```

```
  init_motortjp();
  init_clocktjp();
  init_data();
  get_robotid();

  display_info();
  check_hardware();

  //librate_servo(LEFT_MOTOR);
  //librate_servo(RIGHT_MOTOR);

  switch (mode) {
          default:
    case 5: mode = 6;   // tag mode
                        simonvar = 0;
                        blink(1);
                if (robotid == 1) server();
                        else client();
            break;

          case 6: mode = 5;       // simon says mode (and light following)
                        simonvar = 1;
                        blink(2);
                        if (robotid == 1) simon();
                        else player();
                        break;
  }

}

unsigned char ir_highest_now(void) {
  static unsigned char irdr, irdl, irdb, irdbl, irdbr;
  // static char lastdir = LEFT;  -- uses global now --
    irdr = RIGHT_IR;
    irdl = LEFT_IR;
    irdb = BACK_IR;
    irdbl = BACK_LEFT_IR;
    irdbr = BACK_RIGHT_IR;

    if (signal_locked) { // if we are getting a signal, dont change direction
      if (lastdir == RIGHT) return irdr;
      if (lastdir == LEFT)        return irdl;
      if (lastdir == BACK)        return irdb;
      if (lastdir == BACKLEFT) return irdbl;
      if (lastdir == BACKRIGHT) return irdbr;
      return 0;
```

```
   }
   if ( (irdr > irdl) && (irdr > irdb) && (irdr > irdbl)
     &&(irdr > irdbr) ) {
     lastdir = RIGHT;
     return(irdr);
   }
   else if ( (irdl > irdr) && (irdl > irdb) && (irdl > irdbl)
     &&(irdl > irdbr) ) {
     lastdir = LEFT;
           return(irdl);
   }
   else if ( (irdb > irdr) && (irdb > irdbr) && (irdb > irdbl)
     &&(irdb > irdl) ) {
     lastdir = BACK;
           return(irdb);
   }
   else if ( (irdbr > irdr) && (irdbr > irdl) && (irdbr > irdbl)
     &&(irdbr > irdb) ) {
     lastdir = BACKRIGHT;
           return(irdbr);
   }

   else if ( (irdbl > irdr) && (irdbl > irdl) && (irdbl > irdbr)
     &&(irdbl > irdb) ) {
     lastdir = BACKLEFT;
           return(irdbl);
   }
   return(80);
}

// This function takes the average of ir_highest_now() over time
unsigned char ir_highest(void) {
  static unsigned char h[6] = {80, 80, 80, 80, 80, 80};
  static int i; // static should save time ??
  static int sum;
  static unsigned char size = 3;
  static unsigned char t;
  //static int lasttime = 0 ;

  //return(ir_highest_now()); //asdfasdf
  t = ir_highest_now(); // get value right away

  for (i=size-1;i--;i != 0) {
    h[i] = h[i-1];
  }
  h[0] = t;
```

```
  sum = 0;
  for (i=0;i<size;i++) {
    sum += h[i];
  }
  sum = sum / size;
  //printf("Now: %d, Avg: %d\n", h[0],sum);

  //if (timertjp - lasttime > 100) { // if its been awhile since last reading
  //  lasttime = timertjp;
  // return(h[0]);
  //}
  //lasttime = timertjp;
  return(sum);
}

void check_hardware() {
  char ok;
  int i;
  int val;

  ok = 1;
  printf("Checking hardware: \n");
  printf("  IR: ");
  wait(50);
  get_ir();
  wait(50);
  get_ir();

  if ((irdr < 80) || (irdr > 130)) { ok = 0; printf("Right IR error! "); error();}
  if ((irdl < 80) || (irdl > 130)) { ok = 0; printf("Left IR error! "); error();}
  if ((irdb < 80) || (irdb > 130)) { ok = 0; printf("Back IR error! "); error();}
  if ((irdbr < 80) || (irdbr > 130)) { ok = 0; printf("Back Right IR error! "); error();}
  if ((irdbl < 80) || (irdbl > 130)) { ok = 0; printf("Back Left error! "); error();}
  if (ok) printf("IR working properly\n");
  else printf("\n");
  if (robotid == 1) { // if server, should have bump switch
    printf("  Bump switch: ");
    if (!BUMPSW) printf("open.\n");
    else {
      printf("closed.\n");
      error();
    }
/*
    printf("  Light cell ambient value: ");
    val = 0;
```

```
    amblight = 150; // CHANGE THIS?!?!?!?
    return;

            motor_control(TURNLEFT,10);
    for (i=0;i<5;i++) {
      val = val + LIGHT;
      wait(50);
    }
            motor_control(TURNRIGHT,10);
    for (i=0;i<5;i++) {
      val = val + LIGHT;
      wait(50);
    }
    motor_control(STOP,0);
    amblight = val / 10;
    printf("%d\n", amblight);
*/
  }

}

// Gives the appropriate angle away from the direction given
// note: this doesn't give 180 degrees, since only certain
// angles are hardcoded.
int away(int angle) {
  switch (angle){
    case 0:   return (180);
    case 180: return(0);
    case 210: return(70);
    case 70:  return(210);
    case 150: return(290);
    case 290: return(150);
    default: return(180);
  }
}

/* blink the LED so many times, used for startup mode notification */

void blink(char times) {
  int i;
  for (i=0;i<times;i++) {
    LED1_ON;
            IRE_ON2;
    wait(400);
    LED1_OFF;
    IRE_OFF2;
```

```
    wait(300);
   }
  wait(300);
}

void simon(void) {
  int ch = 0;
  int i;
  printf("Dendrite playing Simon Says (leader)\n");
  while(1) {

    //ch = ch + 1; if (ch == 7) ch = 1;

    ch = (TCNT % 7) + 1;

    if (LIGHT < 20) ch = 8; // if a light present (1st time)



    for (i=0;i<1;i++) {
      timertjp = 0;
      IRE_ON2;
      LED1_ON;
      while(timertjp < 100+200*ch + 100);

      IRE_OFF2;
      while(timertjp < 100+200*ch + 250);
      LED1_OFF;
    }

    if (ch == 8) {
      motor_control(TURNLEFT, 15);
            wait(300);
             motor_control(STOP,0);

      timertjp = 0;  /* resend signal to make sure they get it!! */
      IRE_ON2;
      LED1_ON;
      while(timertjp < 100+200*ch + 100);

      IRE_OFF2;
      while(timertjp < 100+200*ch + 250);
      LED1_OFF;

      light_follow();
    }
```

```
        else action(ch, ACT);

    action(ch,WAIT);  /* wait until players finish movement */
    wait(500);




  }
}

void player(void) {
  int plen;
  int ch;
  printf("Dendrite playing Simon Says (follower)\n");
  while(1) {
    plen = receive_simon();
    if (plen > 150) {
      printf("Pulse received, size: %d\n", plen);
      ch = (plen - 100) / 200;
      printf("Choice: %d\n", ch);

          // echo it
      timertjp = 0;

      LED1_ON;
      IRE_ON2;
      while(timertjp < 100+200*ch + 100);

      IRE_OFF2;
      while(timertjp < 100+200*ch + 350);
      LED1_OFF;
      action(ch, ACT);
    }
  }

}

void action(int ch, char opt) {
  switch(ch) {
  case 1: if (opt == ACT) motor_control(FORWARD,30);
              wait(700);
              break;
  case 2: if (opt == ACT) motor_control(BACKUP, 20);
              wait(600);
              break;
```

```
  case 3: if (opt == ACT) motor_control(TURNLEFT, 20);
                        wait(900);
                        break;
  case 4: if (opt == ACT) motor_control(TURNRIGHT, 20);
                        wait(900);
                        break;
  case 5: if (opt == ACT) motor_control(TURNLEFT, 20);
                        wait(200);
          if (opt == ACT) motor_control(TURNRIGHT, 20);
                        wait(200);
          if (opt == ACT) motor_control(TURNLEFT, 20);
                        wait(200);
          if (opt == ACT) motor_control(TURNRIGHT, 20);
                        wait(200);
                        break;
  case 6: if (opt == ACT) motor_control(TURNRIGHT, 25);
                        wait(1400);
                        break;
  case 7: // do nothing here
                        break;
  case 8: // follow friend!
                        follow_leader();
                        break;

  default:break;
  }
  if (opt == ACT) motor_control(STOP,0);
}

void light_follow(void) {
  int ldir, oldlight;
  int count;
  int rcount;
  rcount = 15 + (TCNT % 15);

  IRE_ON2; // light up so we can be followed
  LED1_ON;
  ldir = LEFT;
  while(1) {
    oldlight = LIGHT;

    if (--rcount == 0) {
      motor_control(STOP,0);
      wait(50);
      motor_control(BACKUP,15);
      wait(400);
```

```
   motor_control(STOP,0);
   rcount = 15 + (TCNT % 15);
  }

      get_ir();

      if (irdr > AVTHRS + 5) {
        motor_control(TURNLEFT,15);
        printf("OBSTACLE.\n");
        wait(300);
        motor_control(STOP,0);
        wait(100);
      }
      else if (irdl > AVTHRS + 5) {
        motor_control(TURNRIGHT,15);
        printf("OBSTACLE.\n");
        wait(300);
        motor_control(STOP,0);
        wait(100);
      }


/*
       if (LIGHT > amblight*3/5) { // if real dim, search for brighter NOW
        count = 20;
   IRE_OFF2; // so we arent followed during this
        while(LIGHT > amblight*3/5 - 10) {
          motor_control(TURNLEFT,15);
          wait(100);
          if (--count == 0) { // if no bright light, were done!
            IRE_OFF2;
            LED1_OFF;
            wait(3000);
            return;
          }
        }
   IRE_ON2;
      }
*/

  if (LIGHT < 30) { // if real bright, move forward!!
   motor_control(FORWARD,20);
  }
  else if (ldir == LEFT) {
   ldir = RIGHT;
   motor_control(RIGHTFORWARD,18);
```

```
    }
        else {
          ldir = LEFT;
          motor_control(LEFTFORWARD,18);
        }
        wait(300);
        if (LIGHT < oldlight - 8) {   // if its brighter here
          if (ldir == LEFT) ldir = RIGHT; else ldir = LEFT;
        }
    printf("LSens: %d\n", LIGHT);
  }
}

// modified version of receive_pulse for use with Simon Says only

int receive_simon(void) {
  int start = timertjp;
  int time2;
  int max,val;

  max = ir_highest_now(); // used as min here
  while(1) {
    if (ir_highest_now() > max + 3) break;  // find UPWARD SLOPE
    if (ir_highest_now() > 110) break; // if a strong signal!
  }
  time2 = timertjp;
  signal_locked = 1;
  wait(10);
  while(1) {
    val = ir_highest_now();
    if (max < val) max = val;  // get max
    if (val < max - 3) break; // find DOWNARD SLOPE
    if (val < 95) break; // if a weak signal!! (TWEAK THIS)
    /* if we receive a long ping, we are sending a message */
    //if (timertjp - time2 > 200) motor_control(STOP,0);
    /* if too long, must be an error */
    if (timertjp - time2 > 2000) {
      signal_locked = 0;
      return(0);
    }
  }

  time2 = timertjp - time2;
  signal_locked = 0;

  return(time2);
```

```
}


void follow_leader(void) {
  char count;
  char rcount;
  count = 0; rcount = 15 + (TCNT % 15);

  LED1_ON;
  while(1) {
    get_ir();
    if (--rcount == 0) {
      motor_control(STOP,0);
      wait(50);
      motor_control(BACKUP,15);
      wait(400);
      motor_control(STOP,0);
      rcount = 15 + (TCNT % 15);
    }
    if (!signal_exists) {
      motor_control(STOP,0);
      count++;
      if (count > 20) return;
    }
    else count = 0;

    locate_robot(0);
	motor_vector(20, robot[0].angle);
	wait(100);
  }
  LED1_OFF;
}

char signal_exists(void) {  // used with follow_leader()
  static int thr = 110;
  if (irdr  > 110) return 1;
  if (irdl  > 110) return 1;
  if (irdb  > 110) return 1;
  if (irdbl > 110) return 1;
  if (irdbr > 110) return 1;
  return 0;
}


/* this function just alerts the user that hardware isn't working correctly */
void error(void) {
```

```
  LED1_ON;
  while(1);

}
```

Appendix B: "commsy14.c"

```
// Dendrite experimental communication code
// Written by Jeffrey D. Wisgo
// Late date coded: 11/06/1999
//
// -- Most of this code isn't used in the final Dendrites.
//
// Note: If you use a significant portion of my code for anything
//                    I'd appreciate if you told me about it. Thanks.
// Email: Locksley@ufl.edu
//
// IMPORTANT: this code isn't guaranteed to work as-is,
//           it is a communications system not fully completed.


#define dprintf //

#include <tjpbase.h>
#include <stdio.h>
#include <math.h>
#include <mil.h>


/* DEFINES */
```

```
// sync pulse must start with a 1 (i.e. > 128)
#define SYNC_CATCHUP 40
#define SYNC_PULSE 154
#define PING_THRS 100
#define COMM_THRS 100

#define COMM_DELAY 150

#define AVOID_THRESHOLD 100
#define AVOID_THRS AVOID_THRESHOLD

#define NO_SIGNAL 500

#define BACK_IR      analog(7)
#define BACK_LEFT_IR analog(6)
#define BACK_RIGHT_IR analog(4)

#define SELF -1

/* These are just used for convenience */
#define BACK             1
#define RIGHT            2
#define LEFT       3
#define BACKLEFT   4
#define BACKRIGHT        5

//#define IRE_ON2 *(unsigned char *)(0x7000) = 0x1F
#define IRE_ON2 irlatch |= 0x1F;  *(unsigned char *)(0x7000) = irlatch
#define IRE_OFF2 irlatch &= !(0x1F); *(unsigned char *)(0x7000) = irlatch

#define LED1_ON irlatch |= 0xF0; *(unsigned char *)(0x7000) = irlatch
#define LED1_OFF irlatch &= !(0xF0); *(unsigned char *)(0x7000) = irlatch

/* New clock handler defines */

#pragma interrupt_handler TOC1_isr2
#define CLOCK_TICK 2000
void init_clocktjp2(void);
void TOC1_isr2(void); /*Interrupt service routine to generate clock ticks*/
void wait2(int);   /* Wait for int milliseconds */


/* NEW DATA TYPES */
struct robot_type {
  char exists;  // if exists = SELF, then I am that robot
```

```
                                  // exists = 1 implies that we are communicating
                                  // with that robot
  int location; // robot's last known location
                                   // specified in angle, 0 = left, 90 = in front,
                                   // 180 = right, 270 = behind

  unsigned char msg;    // the most recent message from this robot

};




/* GLOBAL VARIABLES */

char msgtosend; // message I'll send to other robots
unsigned char irdr, irdl, irdb, irdbr, irdbl;
char robotid;
struct robot_type robot[6];
const unsigned char bitmask[] = {1, 2, 4, 8, 16, 32, 64, 128};
unsigned char irlatch = 0x00;
unsigned char signal_locked = 0; // used with ir_highest_now()


/* PROTOTYPES */
unsigned char bitof(unsigned char b, unsigned char x);
void noping(int dly);
unsigned char ir_highest_now(void);
unsigned char ir_highest(void);
unsigned int abs(unsigned int x);
void server();
void server_cycle();
void client();
void transmit(unsigned char msg);
unsigned char receive(void);
unsigned int receive_ping(char id, unsigned int timeout, char usetime);
void locate_robot(char id);
unsigned char client_cycle();
void init_COP(void);
void COP_VERIFY(void);

void server() {

  printf("Communication Server active\n");

  server_cycle();
```

```
}

void server_cycle() {
  int len;

  timertjp = 0;

  while(1) {

  IRE_ON2; // SYNC ping starts here (300 msec total)
  LED1_ON;
  wait(100);
  // do obstacle avoidance here

  // printf("\x1b[18;1H    \b\b\b\b\b%d",LEFT_IR);
  //  printf("\x1b[18;13H   \b\b\b\b%d",RIGHT_IR);
  //  printf("\x1b[18;25H   \b\b\b\b%d",BACK_IR);
  //  printf("\x1b[18;37H   \b\b\b\b%d",BACK_LEFT_IR);
  //  printf("\x1b[18;49H   \b\b\b\b%d",BACK_RIGHT_IR);


  while(timertjp < 200);
  IRE_OFF2;
  LED1_OFF;

  //printf("\x1b[10;0H");
  //while(timertjp<460)
  //  printf("%u,%u \n", timertjp, ir_highest());


  while(timertjp < 300);

  len=receive_ping(2,200,1); // its time for second robot to ping



  // do behaviors here!!

  //printf("\x1b[6;1H    Robot %d: %d",1,robot[1].exists);
  printf("\x1b[7;1H    Robot %d: %d",2,robot[2].exists);
  //printf("\x1b[8;1H    Robot %d: %d",3,robot[3].exists);
  //printf("\x1b[9;1H    Robzxot %d: %d",4,robot[4].exists);
  //printf("\x1b[10;1H    Robot %d: %d",5,robot[5].exists);

  while(timertjp < 1000); // wait 100 msec extra
```

```
  if (robot[2].exists) transmit(7); // takes 1350 msec (w/o end delay)

  while(timertjp < 2600);  // kill last 150 msec or so

  timertjp=0;

  }

}


void client() {
  int len,slen;
  char retry;
  printf("Communications Client active\n");
  while(1) { // if in this loop, client is NOT SYNCHRONIZED
    //COP_VERIFY(); // everything is running smoothly
        //while(1) {
        //  printf("%d ", timertjp);
        //}
    IRE_ON2;
    wait(100);
    // do obstacle avoidance here

    //printf("\x1b[18;1H    \b\b\b\b\b%d",LEFT_IR);
    //printf("\x1b[18;13H   \b\b\b\b%d",RIGHT_IR);
    //printf("\x1b[18;25H   \b\b\b\b%d",BACK_IR);
    //printf("\x1b[18;37H   \b\b\b\b%d",BACK_LEFT_IR);
    //printf("\x1b[18;49H   \b\b\b\b%d",BACK_RIGHT_IR);

    IRE_OFF2;
    wait(200);
    len = receive_ping(0,700,0);

    if (len > 90) {
      printf("\n%d->\n",len);
      retry = 8;
      while(retry--) {
        len = receive_ping(0,350,0);
        printf("%d,", len);
        if ((len > 170) && (len < 280)) { // if we've found a server SYNC ping
          //COP_VERIFY(); // everything is running smoothly
          printf("\nOK\n");
          robot[1].exists = robot[0].exists;
          robot[1].location = robot[0].location;
          slen = client_cycle();
```

```
                retry = 0;
            wait (TCNT % 500);
        continue;

       }
      }
      // if were here, SYNC ping not found after retrying
      printf("\n SYNC ping not found!\n");

    }
    wait (TCNT % 300);

  }
}

// This function is called after a sucessful SYNC PING has
// been found
unsigned char client_cycle() {
  static int len;  // static saves time entering sub?
  static unsigned char msg = 0;
  static unsigned char slen = 0;
  static unsigned int temp = 0;
  static char first_time; // used to adjust sync first run
  static char bad_sync = 0;
  //printf("We've been unSYNCed for %d seconds.\n",seconds);
  //seconds = 0; // ERASE THIS LATER!!
  //temp = 0;
  bad_sync = 0;
  slen = 0;
  first_time=1;
  timertjp = 300; // were starting after first ping accepted
  while(1) {
    //while(1)
    //  printf("tt: %u, irh: %u\n", timertjp, ir_highest());
    slen++;
    IRE_ON2;
    LED1_ON;
    while(timertjp < 480); // HARDCODED FOR ROBOT 2!!!
    IRE_OFF2;
    LED1_OFF;
    while(timertjp < 500);

    while(timertjp < 1000);  // wait 100 msec extra
          msg = receive();
    if (msg) printf("[%u], ", msg);
```

```
        //if (temp > 2600) printf("^%u^, ", temp);


    if (first_time == 1) {
      first_time = 0;
      while(timertjp < 2600); // adjust for offset first time around
    }

//  else if (len < 200) { // if ping too small last time, resync this time
//      while(timertjp != 2600);  // wait until cycle end
//      len = receive_ping(0,300,0);
//      first_time = 1;
//      timertjp = 300;
//      timertjp = 300;
//      continue;
//    }
    else {
      //if (len < 200)
        while(timertjp < (2600 + (250 - len)/2)); // adjust slightly
      //else
      //  while(timertjp < 2600);  // wait until cycle end
    }
        //temp = timertjp;
        timertjp = 0;

    len = receive_ping(0,300,1);
    printf("%d ", len);

    if (len < 170) { // if we didnt find (enough of) SYNC ping
     break;
      // if (++bad_sync == 2) break;
    }
    //else bad_sync = 0;


  } // end of while
        //if (temp > 2600) printf("^%u^, ", temp);
        temp = 0;

    printf("Sucessfully SYNCed for %d cycles\n",slen);
    //seconds = 0; // ERASE THIS LATER
    return(slen);
}

// returns length of ping, and updates robot location info
// if usetime = 0, then routine will return only if
```

```
//                                  a signal is never found, or a signal ends
// if usetime = 1, then routine will return if a signal is never
//                                              found or time runs out
unsigned int receive_ping(char id, unsigned int timeout, char usetime) {
  unsigned int start = timertjp;
  unsigned int pstart = 0;
  int i;
  unsigned char fallamt;
  unsigned char highest, irnow;

  fallamt = 0;
  //highest = ir_highest();
  highest = ir_highest();
  highest = ir_highest(); // multicall to correct averaging
  highest = ir_highest();

  while (timertjp < start + timeout)  {
  irnow = ir_highest();
    if ((irnow > highest + 5) && (irnow > 100)) { // RISING EDGE
              // NOTE: the irnow > 100 makes closer dendrites give a smaller sync pulse size
      signal_locked = 1; // keep receiving from same dir
      locate_robot(id); // find dendrite location
      pstart = timertjp;
      highest = ir_highest();

      while((!usetime) || (timertjp < start + timeout)) {

        if (ir_highest() < highest) fallamt++;
        highest = ir_highest();
        if (fallamt == 3) {  // FALLING EDGE
          pstart = timertjp - pstart; // get length
          if (usetime)  // use up all available time!!
            while( (timertjp != start + timeout)
                                  && (timertjp != start + timeout + 1));
                        signal_locked = 0;
          return(pstart); // return length
        }

        for (i=0;i<150;i++); // delay a bit
      }
    // if were here, then pulse is too long
      signal_locked = 0;
      return(timertjp-pstart);
    }
  }
  robot[id].exists = 0;
```

```
  signal_locked = 0;
  return (0); // no signal detected
}

void ping(int timeon) {
  IRE_ON2;
  wait(timeon);
  IRE_OFF2;
}

void noping(int dly) {
  IRE_OFF2;
  wait(dly);
}

void locate_robot(char id) { // uses IR to find direction of robot
  robot[id].exists = 1;
}


#define BITLEN        150
#define BITEND  50

void transmit(unsigned char msg) {
  static int i;
  static unsigned int start;
  start = timertjp;
  //printf("Transmit start: %u\n", start);
  //ping(COMM_DELAY);  // send header
  IRE_ON2;
  while (timertjp < start + BITLEN - BITEND);
  IRE_OFF2;
  while (timertjp < start + BITLEN);
  for (i=0;i<4;i++) {
    if (bitof(i,msg)) {
      IRE_ON2;
      //printf("ON: %u\n", timertjp - start);
    }
    else {
      IRE_OFF2;
    }
    while(timertjp < ((start + BITLEN) + ((i+1) * BITLEN) - BITEND));
     //printf("OFF: %u\n", timertjp - start);
    IRE_OFF2;
    while(timertjp < ((start + BITLEN) + ((i+1) * BITLEN) ));
  }
```

```
}

unsigned char receive(void) {
  int i;
  int msg;
  unsigned char b[8];
  unsigned int last_ir;
  unsigned int start;
  unsigned int temp;

  dprintf("6");

  msg = 0;
  temp = receive_ping(0,COMM_DELAY + 100, 0);
  start = timertjp;

  if ( temp < 40 ) return 0;


  dprintf("7");

  signal_locked = 1;

  for(i=0;i<4;i++) {

    if ( receive_ping(0,BITLEN,1) > 30) b[i]=1;
    else b[i] = 0;
   }
  for(i=0;i<4;i++) {  // do this separately for timing purposes
          if (b[i]) SET_BIT(msg,bitmask[i]);
   }
  signal_locked = 0;

  // assume we got this message from the server
  // we should already know the server exists at this point
  robot[1].msg = msg;

  return(msg);
}

unsigned int abs(unsigned int x) {
  if (x > 0) return (x);
  else return (-1*x);
}

void get_ir() {  // read ir values in now
```

```
  irdr = RIGHT_IR;
  irdl = LEFT_IR;
  irdb = BACK_IR;
}


void get_robotid() {
  if (BUMPER < 5) {
    robotid = 1;
  }
  else if (BUMPER < 125) {
    robotid = 2;
  }
  robot[robotid].exists = SELF;  // set self tag
  return;
}

void display_info() {
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* clear screen */
  char place[]= "\x1b[1;1H";               /* Home cursor */

  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/

  printf("--== DENDRITE ==--\n");
  printf("Robot ID: %d\n", robotid);
  printf("Programmed by: Jeffrey D. Wisgo\n");
  printf("Code revision: 4.0\n");
  return;
}
// returns bit b of char x
unsigned char bitof(unsigned char b, unsigned char x) {
  unsigned char t;
  t = x & (bitmask[b]);
  return t;
}


int ir_avg() {
  int avg;
  avg = (RIGHT_IR + LEFT_IR + BACK_IR ) / 3;
  //printf("%d\n", avg);
  return avg;
}
```

```
void init_data() {
  int i;
  for (i=0;i<6;i++) {
    robot[i].exists = 0;
    robot[i].location = 0;
    robot[i].msg = 0;
  }
  msgtosend = 8;
}


void main(void)
{
  //init_COP();  /* this forces a reset the first time it is run */
  IRE_OFF2;
  LED1_OFF;
  init_analog();
  init_motortjp();
  init_clocktjp2();
  init_data();
  get_robotid();

  display_info();

  if (robotid == 1) {
    server();
  }
  else {
    client();
  }

}

unsigned char ir_highest_now(void) {
  static unsigned char irdr, irdl, irdb, irdbl, irdbr;
  static char lastdir = LEFT;
    irdr = RIGHT_IR;
    irdl = LEFT_IR;
    irdb = BACK_IR;
    irdbl = BACK_LEFT_IR;
    irdbr = BACK_RIGHT_IR;

    if (signal_locked) { // if we are getting a signal, dont change direction
      if (lastdir == RIGHT) return irdr;
      if (lastdir == LEFT)        return irdl;
      if (lastdir == BACK)        return irdb;
```

```
    if (lastdir == BACKLEFT) return irdbl;
    if (lastdir == BACKRIGHT) return irdbr;
    return 0;
  }
  if ( (irdr > irdl) && (irdr > irdb) && (irdr > irdbl)
     &&(irdr > irdbr) ) {
     lastdir = RIGHT;
     return(irdr);
  }
  else if ( (irdl > irdr) && (irdl > irdb) && (irdl > irdbl)
     &&(irdl > irdbr) ) {
     lastdir = LEFT;
          return(irdl);
  }
  else if ( (irdb > irdr) && (irdb > irdbr) && (irdb > irdbl)
     &&(irdb > irdl) ) {
     lastdir = BACK;
          return(irdb);
  }
  else if ( (irdbr > irdr) && (irdbr > irdl) && (irdbr > irdbl)
     &&(irdbr > irdb) ) {
     lastdir = BACKRIGHT;
          return(irdbr);
  }

  else if ( (irdbl > irdr) && (irdbl > irdl) && (irdbl > irdbr)
     &&(irdbl > irdb) ) {
     lastdir = BACKLEFT;
          return(irdbl);
  }
  return(80);
}

// This function takes the average of ir_highest_now() over time
unsigned char ir_highest(void) {
  static unsigned char h[6] = {80, 80, 80, 80, 80, 80};
  static int i; // static should save time ??
  static int sum;
  static unsigned char size = 3;
  static unsigned char t;
  //static int lasttime = 0 ;

  //return(ir_highest_now()); //asdfasdf
  t = ir_highest_now(); // get value right away

  for (i=size-1;i--;i != 0) {
```

```
    h[i] = h[i-1];
  }
  h[0] = t;
  sum = 0;
  for (i=0;i<size;i++) {
    sum += h[i];
  }
  sum = sum / size;
  //printf("Now: %d, Avg: %d\n", h[0],sum);

  //if (timertjp - lasttime > 400) { // if its been awhile since last reading
  //  lasttime = timertjp;
  // return(h[0]);
  //}
  //lasttime = timertjp;
  return(sum);
}



void init_COP(void) {
  if (CONFIG & 0X04) { // if COP currently disabled
    CLEAR_BIT(CONFIG,0x04);  /* ENABLE COP */

        SET_BIT(OPTION, 0X08); /* software reset */
        asm("clra");
        asm("tap");
        asm("stop");

    printf("we shouldnt get here!!!!!!!!!!!!\n");
  }
  SET_BIT(OPTION, 0X03); /* set COP for 1 sec timeout (fastest) */
  COP_VERIFY();
}

void COP_VERIFY(void) {
  COPRST = 0x55;                /* reset COP counter */
  COPRST = 0XAA;
}


/* MY REVISION OF THE FILES FOUND IN CLOCKTJP.C ARE BELOW */
```

```
void init_clocktjp2(void)
/***********************************************************************
*
 * Function: Initializes all necessary variables for the timer.        *
 * Returns: None                                          *
 *                                                  *
 * Inputs                                            *
 *   Parameters: None                                    *
 *   Globals:    msec, seconds, minutes, hours, days                  *
 * Outputs                                           *
 *   Parameters: None                                    *
 *   Globals:    msec, seconds, minutes, hours, days                  *
 *                                                  *
 * Registers: PACTL, TOC5,TMSK1, TCTL1                          *
 * Functions called: None                               *

***********************************************************************
/
{

  INTR_OFF();
  SET_BIT(HPRIO,0x0b);       /* Promote OC1 interrupt to highest */
  CLEAR_BIT(HPRIO,0x04);

  msec=timertjp=0;
  TOC1 = 0;
  SET_BIT(TMSK1,0x80);        /* Enable OC1 interrupt */
  INTR_ON();
}

/**********************End of init_clocktjp ****************************/


void wait2(int nmsec)  // for now, i just use the old wait()
/*********************************************************************
 * Function Description: Consumes nmsec milliseconds of time.        *
 * Returns: None                                          *
 *                                                  *
 * Inputs                                            *
 *   Parameters: nmsec, the number of milliseconds to pause.         *
 *   Globals:    timertj, millisecond counter                  *
 * Outputs                                           *
 *   None                                            *
 * Functions called: None                               *
 * Notes: None                                       *
 *********************************************************************/
```

```
{
unsigned int mark;
  mark = timertjp+nmsec;
  if(mark >= nmsec) /* Check for unsigned integer overflow */
    while (timertjp < mark); /* Wait for timer to count nmsec. */
  else /* Compensate for overflow */
    { while (!(0 == timertjp)); while (timertjp < mark); }

}
/*********************End Function wait **************************/



void TOC1_isr2(void)
/*********************************************************************
*
 * Function:                                           *
 *   Interrupt service routine to generate clock time variables. This   *
 *   isr executes every millisecond and increments the global variables *
 *   msec, seconds, minutes, hours, days appropriately.              *
 *                                                         *
 * Returns: None                                          *
 *                                                   *
 * Inputs                                              *
 *   Parameters: None                                       *
 *   Globals:    msec, seconds, minutes, hours, days              *
 * Outputs                                             *
 *   Parameters: None                                       *
 *   Globals:    msec, seconds, minutes, hours, days              *
 *                                                   *
 * Registers:  TOC1, TFLG1                                   *
 * Functions called: None                                 *
 * Notes: None                                          *

*********************************************************************
/
{
   TOC1 += CLOCK_TICK;     /* Set the timer to interrupt again in 1msec. */

   ++timertjp;          /* This is TJ's millisecond timer counter. */

  //if (timertjp > 3000) { // if we have a problem, just reset the system!
  //       main(); // this wont work unless we have a way of enabling interrupts.
  //}
  CLEAR_FLAG(TFLG1,0x80);         /* Clear OC1I flag */
}
```

Appendix C: Expenses

As the TJ™ PRO robot is a very handy robot base for any programming project,

expenses have been included in this report as a handy reference to those

deciding to build TJ™ or TJ™ PRO robots in future projects. The following

expenses leave out the cost of wood, switches, visible LED, wheel washers,

cables, and MTJPRO11 circuit components since these were provided by the

IMDL lab. No costs shown here are guaranteed in any way, shape, or form by

any company.

Product	Amount	Cost per unit	Total Cost
MTJPRO11 board (not populated)	4	$20	$80
Wheels	8	$1	$8

HiTec HS-422 Servos (or compat.) (incl. 1 extra)	9	$10	$90
IR LEDs	20	$0.75	$15
Sharp IR receivers (incl 2. extra)	22	$3	$66
Batteries (groups of 6) (2 extra groups incl.)	6	$6	$36
Total Cost			**$295**