C. Andrew Davis

Final Report

SATYAGRAHA

December 10, 2002

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

Table of Contents

1. ABSTRACT

SATYAGRAHA is a multiple robot project inspired by the teachings of Mohandas Gandhi. He describes Satyagraha as civil resistance without any intentional injury to other parties. Gandhi himself called these actions “experiments in truth.” Project SATYAGRAHA consists of two belligerent robots engaged in a battle. A third robot, GANDHI, acts as a Satyagrahi (a participant in Satyagraha) and intervenes in the battle to peacefully end the violence between its robotic brethren. In a sense, this project is an experiment in truth.

2. EXECUTIVE SUMMARY

SATYAGRAHA consists of two types of robots—a peaceful robot (GANDHI) and two belligerent robots (WARRIOR's). All robots are equipped with IR collision avoidance and bump detection, but each type of robot serves a different purpose.

GANDHI roams around looking for WARRIOR's by tracing their 56.5 kHz IR signal through IR detectors. Once a WARRIOR is detected, GANDHI will drives towards the WARRIOR and bump into it. Next, GANDHI will continue seeking out other WARRIOR's.

The WARRIOR's are equipped with 56.5 kHz IR to broadcast their position. A WARRIOR will wander about and periodically stop. If another WARRIOR's IR beacon is detected, it will turn around to avoid facing it. Once found and bumped into by GANDHI, two parallel wires on the WARRIOR's bump skirt are shorted together, signaling that a hit by GANDHI has been made. This causes the WARRIOR to become pacified. The WARRIOR turns off its IR beacons and displays a "smile" on its seven segment LED's instead of a "frown". The WARRIOR will then proceed to wander around and periodically "dance" by rotating in place.

3. INTRODUCTION

SATYAGRAHA originated from my desire to study multiple-robot interaction and to create an artificial model of the natural world. SATYAGRAHA aims to serve as a simplistic model of human interactions pertaining to strife and conflict. The project also acts as an exercise in studying communication and behavior amongst a multitude of autonomous agents.

The system presented consists of three robots. Two of these robots are belligerent warrior robots. Each WARRIOR exhibits its anger via a face created on 7-segment LED's. The third robot, GANDHI, seeks out the warrior robots and then proceeds to pacify the WARRIOR by convincing it to stop fighting and become peaceful.

This paper outlines the entire SATYAGRAHA system by analyzing each subsystem. The paper covers the robots' platform design, actuation, sensor construction, behaviors, and experiments leading to an effective system.

4. INTEGRATED SYSTEM

Each agent in the SATYAGRAHA system uses a Progressive AT-Mega163 development board as a means of control. The figure below shows the behaviors that each robot will exhibit.

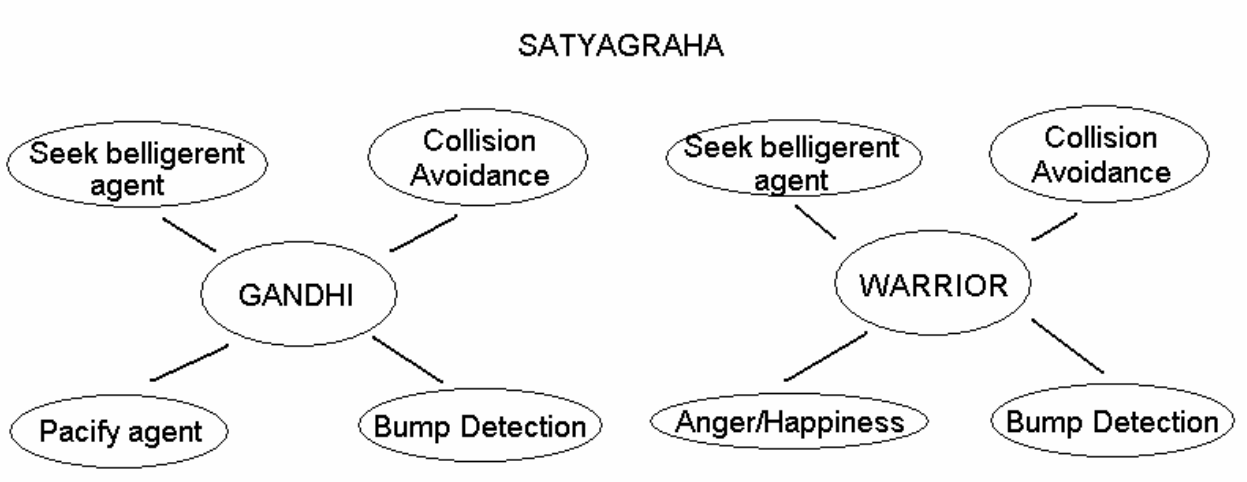

Figure 1

Collision Avoidance is implemented using 40 kHz modulated IR, operated though analog-to-digital (A/D) ports, and bump detection is operated through digital input ports. Belligerent agent seeking is accomplished through hacked 56.5 kHz modulated IR detectors that are read through A/D ports as well. Anger and happiness are displayed by WARRIOR's on 7-segment LED's controlled through digital output ports. Hacked servos are used for locomotion and are controlled using pulse width modulation (PWM) implemented through the AT-Mega board. 56.5 kHz modulated IR LED beacons allow belligerent agents to be tracked. The beacons are controlled through PWM outputs on the microcontroller.

5. MOBILE PLATFORM

The platforms are modified versions of the standard TJ design. The platforms are made of wood, designed in AutoCAD and cut using the T-Tech machine. The platform is designed to hold for the microcontroller, battery pack, protoboard, and two servos. The platform is customized to allow cables and sensors to be attached to the exterior. Below are Figures 2 and 3 of the platform.

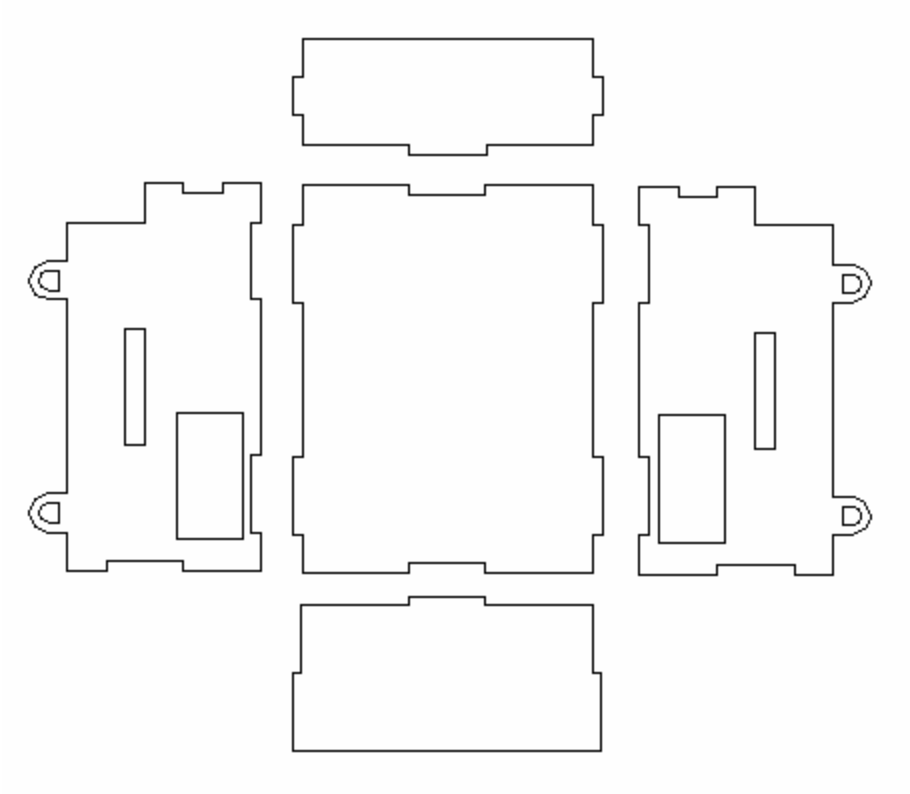

Figure 2: Platform base

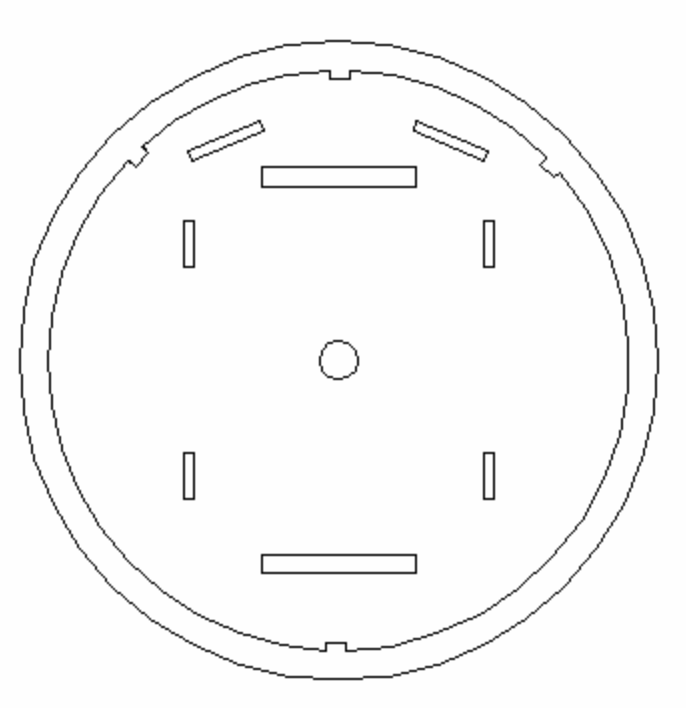

Figure 3: Platform top

Figure 4 shows the completed platform with hardware mounted

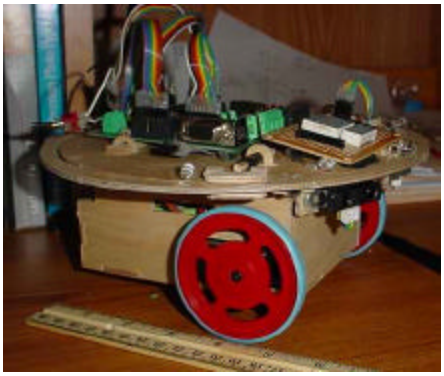

Figure 4: Complete platform

The original platform had a bump skirt that was too thin. It broke while the platform was being constructed. The servo ports had to be filed down extensively for the servos to fit. On successive platform cut-outs, the bump skirt was thickened and the radius of the platform top was increased to allow more room for attaching sensors. It was discovered that the bottom portion of the platform was too small to accommodate for all the wires and cables. The cables were often

smashed together against the protoboard. The result was that the microcontroller had to be placed on top of the platform. A caster for the platform to slide on consisted of a small disk designed for furniture, purchased for a nominal price. The platform was adhered together using standard wood glue and sensors were attached using screws, hot glue, vecro, or electrical tape.

6. ACTUATION

Each robot has dual hacked servos for locomotion. Servos are attached to the left and right sides of the platform and have plastic molded wheels attached to them. The servos are hacked to allow for continuous motion. This provides a simpler system since motor drivers are not needed. Pulse width modulated signals are produced by the microcontroller. MX-400 standard servos were chosen due to their cheap cost and sufficient speed and torque. This application does not required significant amounts of torque or speed.

7. SENSORS

Project SATYAGRAHI requires several dynamic sensor systems. All three robots include IR collision avoidance and bump sensors. Additionally, infrared is needed to allow GANDHI to seek out belligerent robots equipped with infrared emitters. A contact detection system is required to determine when GANDHI has made physical contact with another robot.

INFRARED SENSOR SYSTEMS

Infrared sensing is utilized in two separate systems in two completely different ways. This section discusses each sensor system separately.

IR Short Range Collision Avoidance

Each robot is equipped with two Sharp GP2D12 Distance Measuring Sensors. The GP2D12's include an IR emitter and sensor in one package, modulating at a frequency of 40 kHz. They require a 5V power signal and ground and output consists of a analog voltage output within the range of 0V to 3V. The sensors are attached to the front of the robot body at a difference angle of 45 degrees. Figure 5 below shows the layout.

Figure 5: IR Collision Avoidance Layout

The analog output on each of the two Sharp sensors is fed into ADC ports PA0 and PA1, respectively, on the Mega-AVR microcontroller. Software then interprets the digital signal produced by the ADC for collision avoidance functions.

IR Beacon Emitter and Detector System

In order for GANDHI to exhibit the behavior of “finding” a warrior robot, an IR beacon is attached to each warrior robot. Three Detectors on GANDHI allow it to sense these beacons and move towards a warrior robot. As to not interfere with the IR in place for collision avoidance, 56.5 kHz modulated IR is used. A warrior robot is equipped with five IR LED’s producing the required 56.5 kHz signal. Figure 6 shows the circuit driving the LED’s.

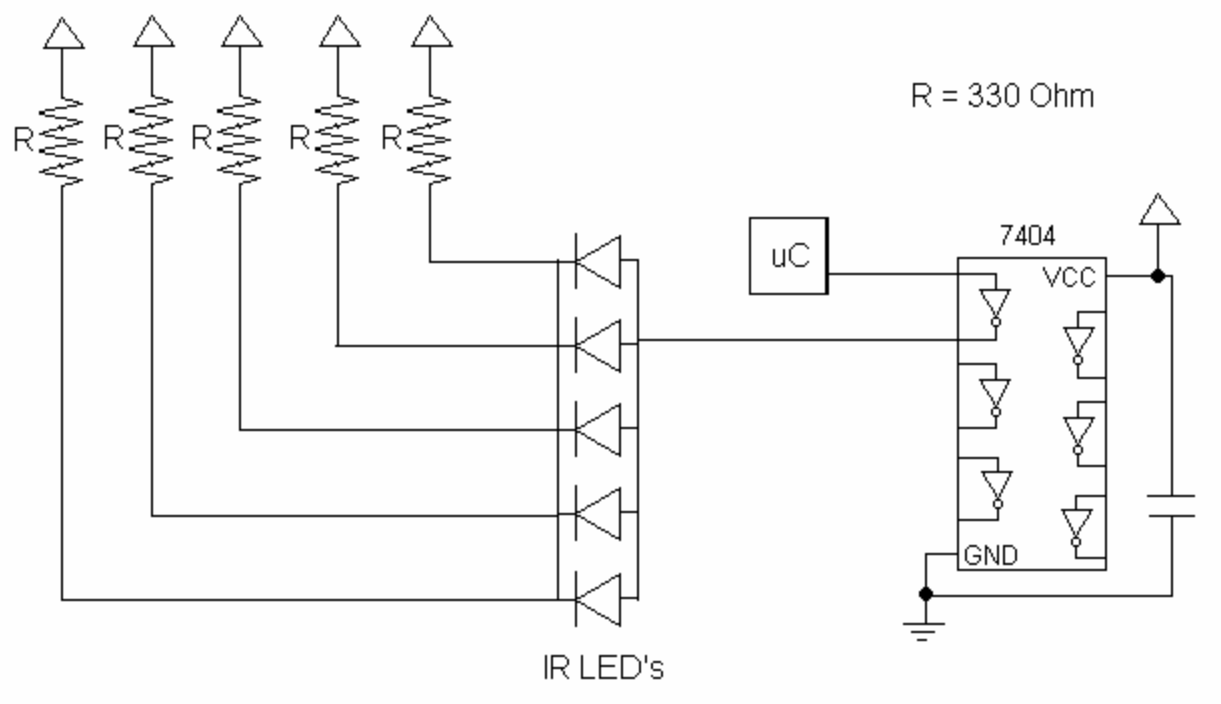

Figure 6: IR LED circuit

It was decided to place all of the IR LED’s towards the front rim of the robot. This will allow GANDHI to determine orientation and path information of warrior robots. Figure 7 shows the placement of the IR LED’s.

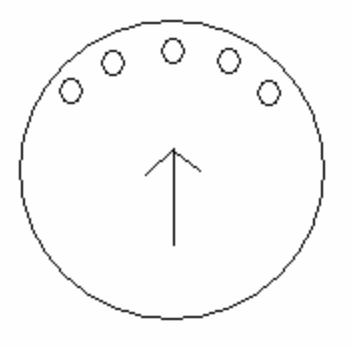

Figure 7: IR LED Layout

To read the IR beacons, GANDHI is equipped with hacked LiteOn digital IR sensors. The sensors were hacked following Michael Hattermann's instruction given in his Spring 2002 final report. Figure 8 is a copy of his instruction on how to hack the LiteOn sensor.

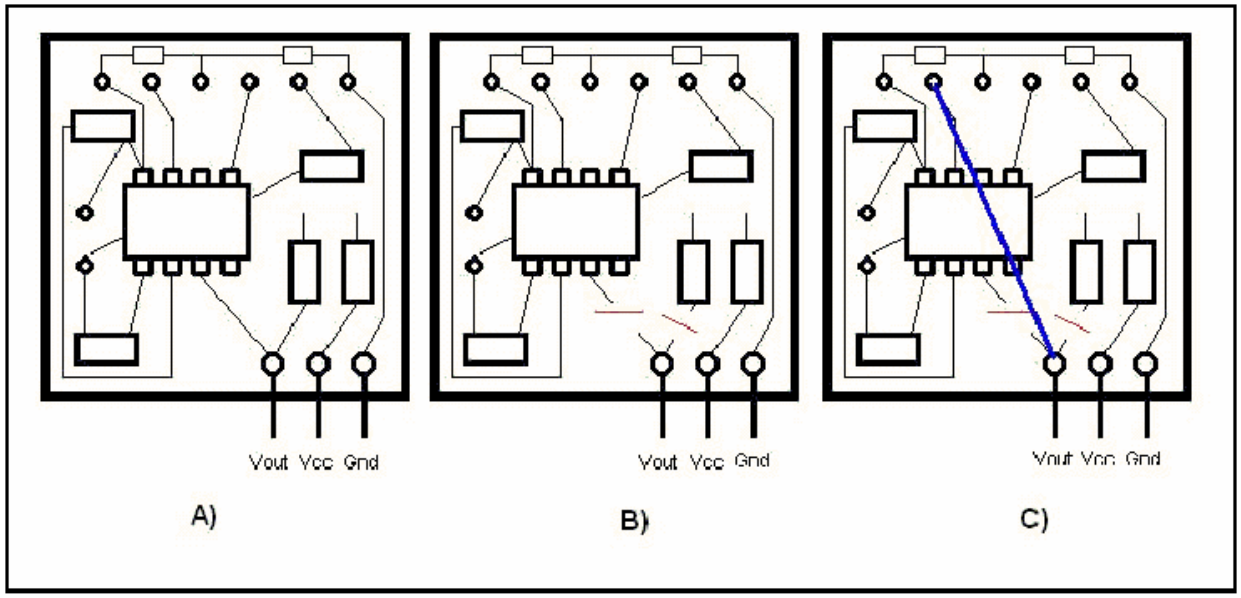

Figure 8: Michael Hatterman's hack

The hacked LiteOn sensors provide analog signals of the measured intensity of 56.5 kHz modulated IR detected. Three of these sensors were place on GANDHI, underneath the top platform. All three are along the front rim and are aimed forward. One sensor was placed on each WARRIOR so that it can sense the general direction of other WARRIORS. Originally, Sharp

GP1UM267XK sensors were purchased but it was realized that there is no feasible way to hack them.

BUMP SENSOR SYSTEM

All SATYAGRAHI robots are equipped with four bump switches for detecting collisions. When open, the switch leads are at a very high impedance. When closed, the switch is close to a short circuit. The switches are wired to individual digital input ports on Port B of the AT-Mega microcontroller. The switches are attached to Port B, pin 0 through 3. Three of the bump switches are aligned along the front of the platform while the fourth is on the rear. Figure 9 shows the layout of the bump sensors.

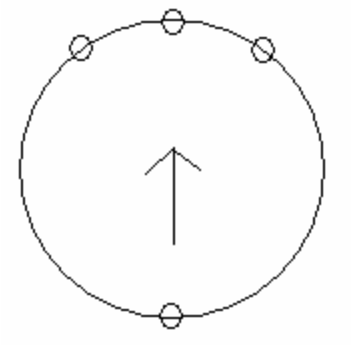

Figure 9: Bump Sensor Layout

CONTACT DETECTION SYSTEM

The contact detection system is used when GANDHI collides with a warrior robot. It is necessary to distinguish a bump with another robot from a bump with any other object. The sensor consists of parallel, stripped, wirewrap gauge wires affixed to the circumference of the WARRIOR's bump skirt. GANDHI's bump skirt is covered with a foil tape. WARRIOR robots monitor the voltage on their bump skirts through a digital port to detect deviations. When two

WARRIOR robots collide, the bump skirt wires are not shorted and the robots react as if they have hit any ordinary object. When a WARRIOR collides with GANDHI, the bump skirt wires are shorted together. This brings the input pin down to ground and ultimately leads to behavior changes in the WARRIOR robot.

This is modified from a similar sensor developed by Jason Plew. His original design used a voltage divider circuit to detect robot collisions. A wire pulled up to 5 V would be attached to GANDHI and a wire pulled down to ground would be attached to a WARRIOR. In theory, a collision results in a voltage divider leaving a voltage of approximately 2.5 V. In actuality, this does not work without a common ground wire attached to each robot. Since GANDHI does not require feedback when a robot collision is made, the modification I have presented is more suitable for my design. Figure 10 shows the sensor as it appears on GANDHI (top) and a WARRIOR (bottom).

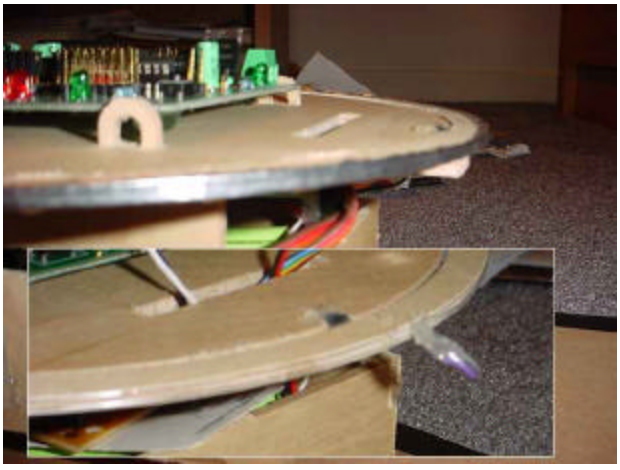

Figure 110: Robot Contact Sensor System

CONCLUSION

Infrared sensors modulated at 40 kHz allow the robots to avoid collisions. Once objects are within 18 inches, suitable action can be taken to evade obstacles. Additionally, infrared sensors

modulated at 56.5 kHz allow GANDHI to track warrior robots. The intensity of the IR detected allows GANDHI to determine a general direction and proximity of a WARRIOR. The bump sensors send discrete voltages to the microcontroller to signify where a robot has collided with something. Software can then interpret this voltage to take corrective measures. The contact sensor works in conjunction with the bump sensors to determine when a collision occurs between GANDHI and a warrior. This important distinction allows for changes in robot behavior.

PARTS LIST

Infrared

Emitter/Detector
Sharp GP2D12 Distance Measuring Sensor
Mark III Robot Store
http://www.junun.org/MarkIII/Store.jsp
$8.25/unit (6)

Emitter
IR LED
IMDL Lab
$0/unit (10)

Detector
LiteON 56.5kHz Digital IR Detector
Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002
1-800-536-4316
Part # 176541
$1.95/unit (10)

Bump Switch

Switch
IMDL Lab
$0/unit (5)

Robot Contact System

Wirewrap wire
Wirewrap wire stripper

Each robot includes a similar sensor package and all sensors are implemented via a Progressive

Mega-AVR development board.

8. BEHAVIORS

All of the robots must be able to avoid obstacles and retreat when it collides with an obstacle. IR sensing and bump detection are used to assume these behaviors. Additionally, all robots must be capable of detecting a belligerent robot.

BEHAVIORS: GANDHI

GANDHI continuously searches for 56.5 kHz modulate IR signals. Once GANDHI detects one, it will track the emitter and collide with the target emitter. Once this is done, GANDHI will back off and continue searching. GANDHI is equipped with two 7-segment LED's that display a smile face to exhibit his peacefulness.

BEHAVIORS: WARRIOR

WARRIOR's exhibit belligerence by displaying a frown face on three 7-segment LED's. A WARRIOR's motion is random roaming with intermittent periods of rest. This is to allow GANDHI to catch up to a WARRIOR it is chasing. If an enemy robot's IR is detected while at rest, the WARRIOR will turn to face away from the enemy. The WARRIOR becomes pacified when GANDHI runs into it. This causes the frown face on the 7-segment LED's to change to a smile face. The WARRIOR's IR beacons are turned off, and it spins in circles. The WARRIOR will randomly roam and periodically begin "dancing" by spinning in place.

EXPERIMENTAL LAYOUT AND RESULTS

Test 1: Sharp GP2D12

To test functionality, a Sharp GP2D12 sensor was connected to a breadboard, powered, and a mulitmeter attached to the output. A white piece of cardboard was placed at varying distances from the sensor and the voltage output was recorded. Table 1 shows the data collected.

Distance (in.)	Voltage		Distance (in.)	Voltage
1	0.94		16	0.70
2	1.85		17	0.67
3	2.67		18	0.64
3.5	**2.71**		19	0.61
4	2.46		20	0.60
5	2.05		21	0.60
6	1.70		22	0.56
7	1.42		23	0.52
8	1.25		24	0.50
9	1.12		26	0.48
10	1.03		28	0.45
11	0.96		30	0.41
12	0.88		32	0.20
13	0.83		34	0.02
14	0.77		36	0.02

Table 1: GP2D12 Signal Characteristics

It was determined that 3.5 inches is the critical distance at which the sensor fails to provide consistent data. It was observed that as the object approaches the sensor, the output voltage increases. Once within 3.5 inches however, this voltage begins to decrease. The GP2D12 sensor is most suitable for measuring distances up to 18 inches away. Figure 11 is a graph of the experimental data collected. Code appears in the appendix detailing the testing procedures for the GP2D12 when interfaced with the Mega-AVR microcontroller.

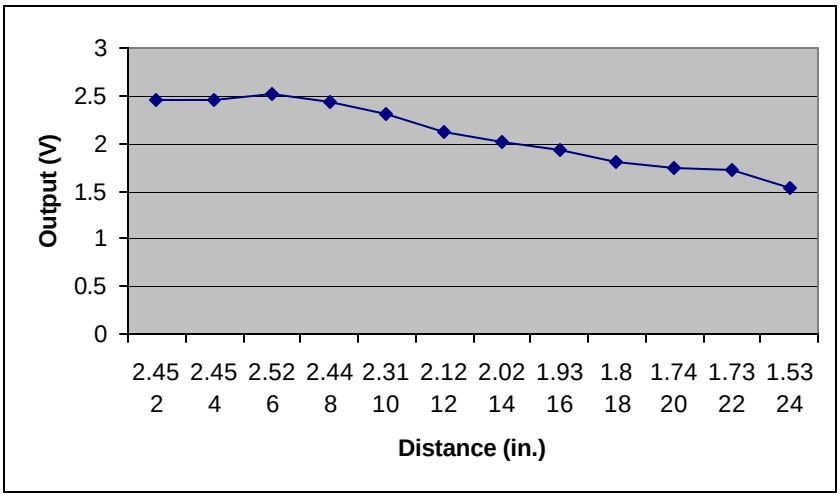

Figure 11: Sharp GP2D12 Data Plot

Test 2: Hacked LiteOn 56.5 kHz IR detector

To test functionality, a hacked LiteOn sensor was connected to a breadboard, powered, and a mulitmeter attached to the output. A IR LED modulated at 56 kHz was placed at varying distances from the sensor and the voltage output on the detector was recorded. Table 2 shows the data collected.

Distance (in.)	Output (V)		Distance (in.)	Output (V)
2	2.45		14	2.02
4	2.45		16	1.93
6	2.52		18	1.8
8	2.44		20	1.74
10	2.31		22	1.73
12	2.12		24	1.53

Table 2: LiteOn Sensor Data

It was determined that 8 inches is the critical distance at which the sensor fails to provide consistent data. It was observed that as the object approaches the sensor, the output voltage increases. Once within 8 inches however, this voltage begins to decrease. The LiteOn sensor is most suitable for measuring distances up to 30 inches away. Figure 12 is a graph of the

experimental data collected. Code appears in the appendix detailing the testing procedures for the LiteOn IR detector when interfaced with the Mega-AVR microcontroller.

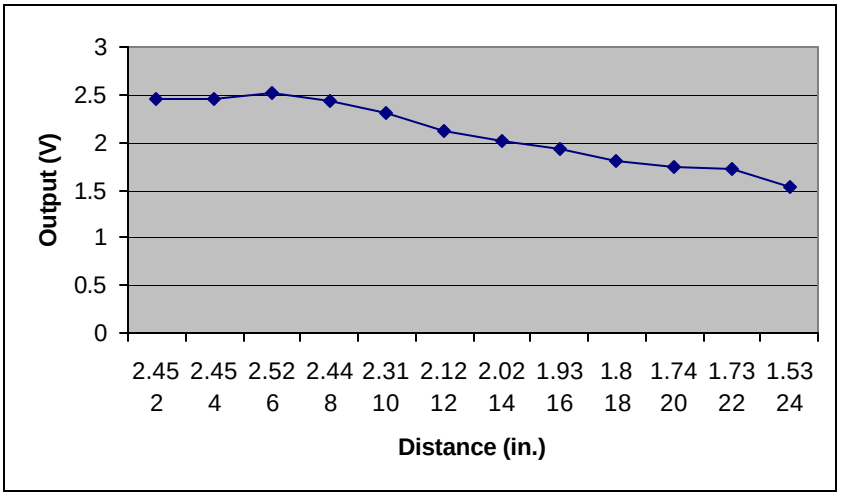

Figure 12: LiteOn Sensor Data Plot

10. CONCLUSION

Overall, I accomplished most of the goals set forth in my proposal. A multiple robot system exhibiting human behaviors was the intention and the result.

My original proposal included plans to have the WARRIOR robots fight by firing projectiles at each other. I decided against this idea based on the TA's advice. Solenoids are costly and greatly increase weight of each robot due to the added battery power.

I originally specified that I would make three to five robots. I met this goal by completing three robots. Five robots would make for a more attractive demonstration but the increased costs and time made this non-feasible. Future work may include building additional WARRIOR's.

No major problems occurred while designing and testing the robots. The delay in getting my original microcontroller of choice (manufactured by Futurlec) put me behind schedule towards the beginning of the semester. I finally decided to use the Progressive AT-Mega boards and I am very satisfied with them. The documentation and support software are excellent.

A good rule of thumb when designing robots is to always have plenty of male and female headers. I constantly found myself running short on female headers for making cables. Additionally, it is good to check that all of your batteries are charged if a robot begins acting strangely. I noticed my servos would act in unpredictable ways but I could not determine the origin. Finally, I tested the batteries and realized the servos were not getting a high enough voltage.

11. Documentation

Thanks to Jason Plew for his theoretical robot collision detection system

Source for LiteOn 56.5 kHz IR detector hack: Michael Hattermann (IMDL Spring 2002)
- Schematic located at:
  http://mil.ufl.edu/imdl/papers/IMDL_Report_Spring_02/michael_hatterman/hacked_ir.pdf

12. Appendix A: Sources for Parts

Microcontroller: Progressive AT-Mega163
http://www.prllc.com

Servos: MX-400 – Acroname
http://www.acroname.com

Plastic Molded Wheels: Mark III
http://www.junun.org/MarkIII/Store.jsp

40 kHz modulated IR detectors: Sharp GP2D12 – Mark III

56 kHz modulated IR detectors: LiteOn LTM-9034 – Jameco
http://www.jameco.com

Batteries: Ni-MH Rechargeable – Hosfelt Electronics
http://www.hosfelt.com

13.Appendix B: Programming Code

```
// IR Test
// Reads IR data in from PA0. The ADC converts the voltage
// into a digial value that is then displayed on the
// LED array.

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

#define AVR_MEGA 0

typedef unsigned char  u08;

u08 analog(u08 channel)
{
        u08 sample_L, sample_H;
        outp(channel, ADMUX);
        sbi(ADCSR, ADSC); /* begin conversion */
        loop_until_bit_is_set(ADCSR, ADIF);
        sample_L = inp(ADCL); /* get low 8 bits */
        sample_H = inp(ADCH);/* get high bits */
        sbi(ADCSR, ADIF); /* clear ADC interrupt flag */
        return sample_H;
}

int main(void)
{

        outp(0xCE,ADCSR);
        outp(0x20,ADMUX);
        outp(0xff,DDRC);

    for (;;) {
        register u08 led = analog(0x20);
        outp(led,PORTC);

    }
}
```

```
//Servo Testing Code

#include <io.h>
#include <sig-avr.h>

#define AVR_MEGA 0

int main(void)
{
	outp(0xFF,DDRD);
	outp(0xA1,TCCR1A);
	outp(0x04,TCCR1B);
	outp(0x00,TCNT1H);
	outp(0x00,TCNT1L);
	outp(0x00,OCR1AH);
	outp(0x14,OCR1AL);
	outp(0x00,OCR1BH);
	outp(0x14,OCR1BL);
	for (;;) {}
}
```

```
//Collision Avoidance Testing Code

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

#define AVR_MEGA 0

#define close 0x30
#define very_close 0x4B
#define stop 0x12
#define error 0x96
#define l_max 0x14
#define l_half 0x13
#define l_maxr 0x10
#define l_halfr 0x11
#define r_max 0x10
#define r_half 0x11
#define r_maxr 0x14
#define r_halfr 0x13
#define l_motor 0x00
#define r_motor 0x01
#define zero 0x00

typedef unsigned char  u08;

u08 ir_left=0;
u08 ir_right=0;

u08 analog(u08 channel)
{
        u08 sample_val_L, sample_val_H;
        outp(channel, ADMUX);
        sbi(ADCSR, ADSC); /* begin conversion */
        loop_until_bit_is_set(ADCSR, ADIF);
        sample_val_L = inp(ADCL); /* get low 8 bits */
        sample_val_H = inp(ADCH);/* high 2 bits, not used */
        sbi(ADCSR, ADIF); /* clear ADC interrupt flag */
        return sample_val_H;
}
```

```
void Wait_opt(int time)
{
	volatile int a, b, c, d;
	for (a = 0; a < time; ++a) {
		for (b = 0; b < 10; ++b) {
		for (c = 0; c < 66; ++c) {
			d = a + 1;
		}
		}
	}
	return;
}

int main(void)
{
// Set up servos
	outp(0xFF,DDRD);
	outp(0xA1,TCCR1A);
	outp(0x04,TCCR1B);
	outp(0x00,TCNT1H);
	outp(0x00,TCNT1L);
	outp(0x00,OCR1AH);
	outp(0x12,OCR1AL);
	outp(0x00,OCR1BH);
	outp(0x12,OCR1BL);
// Set up A/D
	outp(0xCE,ADCSR);
	outp(0x20,ADMUX);
	outp(0xff,DDRC);

	for (;;) {
		ir_left=analog(0x20);
		ir_right=analog(0x21);
		Wait_opt(50);
		if (ir_left>error || ir_right>error) {
			Wait_opt(10);
		}
		else if (ir_left>very_close) {
			outp(l_max,OCR1AL);
```

```
                outp(r_maxr,OCR1BL);
                Wait_opt(100);
            }
            else if (ir_left<very_close && ir_right>very_close) {
                outp(l_maxr,OCR1AL);
                outp(r_max,OCR1BL);
            }
            else if (ir_left>close && ir_right>close) {
                outp(l_max,OCR1AL);
                outp(stop,OCR1BL);
            }
            else if (ir_left>close && ir_right<close) {
                outp(l_max,OCR1AL);
                outp(r_half,OCR1BL);
            }
            else if (ir_left<close && ir_right>close) {
                outp(l_half,OCR1AL);
                outp(r_max,OCR1BL);
            }
            else if (ir_left<close && ir_right<close) {
                outp(l_max,OCR1AL);
                outp(r_max,OCR1BL);
            }
        }
}
```

```
// Main program code for GANDHI

#include <io.h>
#include <sig-avr.h>

#define AVR_MEGA 0

#define close 0x30
#define very_close 0x4B
#define stop 0x12
#define error 0x96
#define l_max 0x14
#define l_half 0x13
#define l_maxr 0x10
#define l_halfr 0x11
#define r_max 0x10
#define r_half 0x11
#define r_maxr 0x14
#define r_halfr 0x13
#define l_motor 0x00
#define r_motor 0x01
#define zero 0x00
#define thresh 0x57
#define svery_close

typedef unsigned char  u08;

u08 ir_left=0;
u08 ir_right=0;
u08 seek_left=0;
u08 seek_center=0;
u08 seek_right=0;
u08 tracking=0;

u08 analog(u08 channel)
{
        u08 sample_val_L, sample_val_H;
        outp(channel, ADMUX);
        sbi(ADCSR, ADSC); /* begin conversion */
        loop_until_bit_is_set(ADCSR, ADIF);
```

```
	sample_val_L = inp(ADCL); /* get low 8 bits */
	sample_val_H = inp(ADCH);/* high 2 bits, not used */
	sbi(ADCSR, ADIF); /* clear ADC interrupt flag */
	return sample_val_H;
}

void Wait_opt(int time)
{
	volatile int a, b, c, d;
	for (a = 0; a < time; ++a) {
		for (b = 0; b < 10; ++b) {
		for (c = 0; c < 66; ++c) {
			d = a + 1;
		}
		}
	}
	return;
}

int main(void)
{
//Turn on beacons
	outp(0xFF,DDRD);
	outp(0x19,TCCR2);
	outp(0x35,OCR2);

// Set up servos
	outp(0xFF,DDRD);
	outp(0xA1,TCCR1A);
	outp(0x04,TCCR1B);
	outp(0x00,TCNT1H);
	outp(0x00,TCNT1L);
	outp(0x00,OCR1AH);
	outp(0x12,OCR1AL);
	outp(0x00,OCR1BH);
	outp(0x12,OCR1BL);
// Set up A/D
	outp(0xCE,ADCSR);
	outp(0x20,ADMUX);
	outp(0xff,DDRC);
```

```
// Set up Port B
        outp(0x00,DDRB);
        outp(0x1F,PORTB);

// Turn on LED's
        outp(0xFF,DDRC);

// Main program code
        for (;;) {

                register u08 LED = inp(PINB);
                register u08 bump = inp(PINB)&0x1F;
                register u08 b_back = bump&0x01;
                register u08 b_left = bump&0x02;
                register u08 b_center = bump&0x04;
                register u08 b_right = bump&0x08;
                outp(LED,PORTC);

                if (bump<0x1F) { // Bump detection
                        outp(stop,OCR1AL);
                        outp(stop,OCR1BL);
                        Wait_opt(100);
                        if (b_back==0) {
                                outp(l_max,OCR1AL);
                                outp(r_max,OCR1BL);
                        }
                        else if (b_left==0 || b_center==0) {
                                outp(l_halfr,OCR1AL);
                                outp(r_maxr,OCR1BL);
                        }
                        else if (b_right==0) {
                                outp(l_maxr,OCR1AL);
                                outp(r_halfr,OCR1BL);
                        }
                        Wait_opt(500);
                }

// IR Beacon Detection
                seek_left = analog(0x22);
```

```
		seek_center = analog(0x23);
		seek_right = analog(0x24);

		if (seek_left>error || seek_center>error || seek_right>error) {
			Wait_opt(5);
		}
		else if (seek_left>thresh || seek_center>thresh || seek_right>thresh) {
			tracking = 1;
			if (seek_center>seek_left && seek_center>seek_right) {
				outp(l_max,OCR1AL);
				outp(r_max,OCR1BL);
			}
			else if (seek_left>seek_center && seek_left>seek_right) {
				outp(l_half,OCR1AL);
				outp(r_max,OCR1BL);
			}
			else if (seek_right>seek_left && seek_right>seek_center) {
				outp(l_max,OCR1AL);
				outp(r_half,OCR1BL);
			}
		}
		else {
			tracking=0;
		}

// Collision Avoidance
		ir_left=analog(0x20);
		ir_right=analog(0x21);

		if (ir_left>error || ir_right>error) {
			Wait_opt(5);
		}
		else if (tracking == 1) {
			Wait_opt(5);
		}
		else if (ir_left>very_close) {
			outp(l_max,OCR1AL);
			outp(r_maxr,OCR1BL);
			Wait_opt(100);
		}
```

```
            else if (ir_left<very_close && ir_right>very_close) {
                outp(l_maxr,OCR1AL);
                outp(r_max,OCR1BL);
            }
            else if (ir_left>close && ir_right>close) {
                outp(l_max,OCR1AL);
                outp(stop,OCR1BL);
            }
            else if (ir_left>close && ir_right<close) {
                outp(l_max,OCR1AL);
                outp(r_half,OCR1BL);
            }
            else if (ir_left<close && ir_right>close) {
                outp(l_half,OCR1AL);
                outp(r_max,OCR1BL);
            }
            else if (ir_left<close && ir_right<close) {
                outp(l_max,OCR1AL);
                outp(r_max,OCR1BL);
            }
        }
}
```

```
// Main program code for WARRIOR
#include <io.h>
#include <sig-avr.h>

#define AVR_MEGA 0

#define close 0x30
#define very_close 0x4B
#define stop 0x12
#define error 0x96
#define l_max 0x14
#define l_half 0x13
#define l_maxr 0x10
#define l_halfr 0x11
#define r_max 0x10
#define r_half 0x11
#define r_maxr 0x14
#define r_halfr 0x13
#define l_motor 0x00
#define r_motor 0x01
#define zero 0x00
#define awhile 20000
#define thresh 0x57

typedef unsigned char  u08;

u08 ir_left=0;
u08 ir_right=0;
int timeout=awhile;
u08 rest=0;
u08 peace=0;
u08 enemy=0;

u08 analog(u08 channel)
{
        u08 sample_val_L, sample_val_H;
        outp(channel, ADMUX);
        sbi(ADCSR, ADSC); /* begin conversion */
        loop_until_bit_is_set(ADCSR, ADIF);
        sample_val_L = inp(ADCL); /* get low 8 bits */
```

```
	sample_val_H = inp(ADCH);/* high 2 bits, not used */
	sbi(ADCSR, ADIF); /* clear ADC interrupt flag */
	return sample_val_H;
}

void Wait_opt(int time)
{
	volatile int a, b, c, d;
	for (a = 0; a < time; ++a) {
		for (b = 0; b < 10; ++b) {
		for (c = 0; c < 66; ++c) {
			d = a + 1;
		}
		}
	}
	return;
}

int main(void)
{
//Turn on beacons
	outp(0xFF,DDRD);
	outp(0x19,TCCR2);
	outp(0x35,OCR2);

// Set up servos
	outp(0xFF,DDRD);
	outp(0xA1,TCCR1A);
	outp(0x04,TCCR1B);
	outp(0x00,TCNT1H);
	outp(0x00,TCNT1L);
	outp(0x00,OCR1AH);
	outp(0x12,OCR1AL);
	outp(0x00,OCR1BH);
	outp(0x12,OCR1BL);

// Set up A/D
	outp(0xCE,ADCSR);
	outp(0x20,ADMUX);
	outp(0xff,DDRC);
```

```
// Set up Port B
        outp(0x80,DDRB);
        outp(0x9F,PORTB);

// Turn on LED's
        outp(0xFF,DDRC);

// Main program code
        for (;;) {
                timeout--;
                if (timeout==0 ) {
                        timeout=awhile;
                        if (rest==0) {
                                rest=1;
                                if (peace==1) {
                                        outp(l_max,OCR1AL);
                                        outp(r_maxr,OCR1BL);
                                }
                                else if (peace==0) {
                                        outp(stop,OCR1AL);
                                        outp(stop,OCR1BL);
                                }
                        }
                        else {
                                rest = 0;
                                outp(l_max,OCR1AL);
                                outp(r_max,OCR1BL);
                        }
                }

// Collision Avoidance
                ir_left=analog(0x20);
                ir_right=analog(0x21);
                if (ir_left<error && ir_right<error && rest==0) {
                        if (ir_left>very_close) {
                                outp(l_max,OCR1AL);
                                outp(r_maxr,OCR1BL);
                                Wait_opt(150);
                        }
```

```
                else if (ir_left<very_close && ir_right>very_close) {
                        outp(l_maxr,OCR1AL);
                        outp(r_max,OCR1BL);
                }
                else if (ir_left>close && ir_right>close) {
                        outp(l_max,OCR1AL);
                        outp(stop,OCR1BL);
                }
                else if (ir_left>close && ir_right<close) {
                        outp(l_max,OCR1AL);
                        outp(r_half,OCR1BL);
                }
                else if (ir_left<close && ir_right>close) {
                        outp(l_half,OCR1AL);
                        outp(r_max,OCR1BL);
                }
                else if (ir_left<close && ir_right<close) {
                        outp(l_max,OCR1AL);
                        outp(r_max,OCR1BL);
                }
        }

// Bump Detection
        register u08 LED = inp(PINB);
        register u08 bump = inp(PINB)&0x1F;
        register u08 b_back = bump&0x01;
        register u08 b_left = bump&0x02;
        register u08 b_center = bump&0x04;
        register u08 b_right = bump&0x08;
        register u08 b_gandhi = bump&0x10;
        outp(LED,PORTC);
        if (b_gandhi==0 || (bump<0x1F && rest==1)) {
                outp(0x00,TCCR2);
                peace=1;
                outp(0x1F,PORTB);
                outp(stop,OCR1AL);
                outp(stop,OCR1BL);
                Wait_opt(100);
                timeout=1;
                rest=0;
```

```
                }
                if (bump<0x1F) {
                        outp(stop,OCR1AL);
                        outp(stop,OCR1BL);
                        Wait_opt(50);
                        if (b_back==0) {
                                outp(l_max,OCR1AL);
                                outp(r_max,OCR1BL);
                        }
                        else if (b_left==0 || b_center==0) {
                                outp(l_halfr,OCR1AL);
                                outp(r_maxr,OCR1BL);
                        }
                        else if (b_right==0) {
                                outp(l_maxr,OCR1AL);
                                outp(r_halfr,OCR1BL);
                        }
                        Wait_opt(200);
                }

// Enemy IR Detection
                enemy=analog(0x22);
                while ((enemy>thresh) && (rest==1) && (peace==0)) {
                        outp(l_max,OCR1AL);
                        outp(r_maxr,OCR1BL);
                        Wait_opt(10);
                        outp(stop,OCR1AL);
                        outp(stop,OCR1BL);
                }
        }
}
```