University of Florida

Search, And, Destroy:
The Intimate Life of Three Robots…

Dr. A. Arroyo
EEL 5666
IMDL

Jose Garcia-Feliu

Table of Contents

Abstract

Executive Summary

Introduction

Integrated System

Mobile Platform

Actuation

Sensors

Sensor Experiments

Behaviors

Experimental Layout and Results

Problems

Conclusion

Documentation

Appendices

Figure 1: Location of bumper switches

Figure 2: Location of IR

Figure 3: Location for sonar and tower

Figure 4: Emitter Sonar Circuit

Figure 5: Detector Sonar Circuit

Figure 6: Flow chart of behaviors

Figure 7a: Picture of robots in playing field

Figure 7b: Picture of Search

Figure 7c: Picture of Robots from every angle

Figure 8a: Picture of Servos

Figure 8b: Picture of IR

Program Code (Interactive C)

Abstract

The objective of our design is to make robots that will simulate a friendly game of search and destroy. Three robots will enter the playing field and search the opponent with the intent to shoot sonar at it. Each contender can only be shot 8 times. Whenever the robot detects the opponent shooting at it, it will find and return fire to the attacker. If the robot has been "hit" too many times, it must return to its base for fixing and reloading. If the robot cannot find the base before it is hit its maximum, the robot will stop indefinitely.

Introduction

We will go through all of the steps required to design, assemble, program and activate these robots. We decided to perform a game platform, since I have already done a task-performing robot for my senior design. This format helps make the design more fun, interesting and at the same time more demanding for the robots to interact. This design is not as simple as it seems. We chose to design and program these robots the same, so that a multitude of robots could be built, if desired, and added to the game without any further modifications. The advantage is a more flexible assembly of the robots. However, disadvantage that this format brings are that because they have the same behaviors, very complex combinations can be achieved. Below, we will describe all the systems and behaviors encountered and expected from these robots. Furthermore, we will explain the relations between their behaviors and how the combination of sensors affects this complex set of actions.

Executive Summary

This project was very successful. We accomplished the goals of making robots that would have multiple behaviors and also to show them. The robots wandered until they saw signs of another running near them. Then shoot sonar waves at each other. We had a bit of difficulty with the sonar, because the same sonar detector cannot distinguish between its own and another robot. We dealt with this by making the robot that just shot to wait a bit and then to detect sonar. This way the sonar wave that bounced off the robot or wall would have passed them. The robots first start by rotating in a counter-clockwise manner. Doing this helps the robots calibrate their infrared. This is accomplished by reading from the environment any signal from the IR detector. If a lower value than a preset value is found, that lower value becomes the new lower bound for the range for that IR. This enables the robot to better approximate the types of changes when the robot encounters an object. The robot is also equipped with an LED bar graph and a piezo buzzer. The bar graph lets the spectators see at a glance the status of the game, while the buzzer lets them hear when a shot is being fired. At the beginning of the game, after the robots finished calibrating, the bar graph sequentially turns on and off the LEDs to show that everything is working. The buzzer will then play a funny tone. Once the robots are ready, they will start looking for each other. This is performed by detecting infrared from the environment. If they detect it, it must be that another robot is present. The robots are also equipped with a sonar detector tower consisting of three detectors. We used relays to switch from one to the next in a clockwise manner. The advantage of using relays is that one can physically connect and disconnect to each detector. The disadvantage is that because they are mechanical devices, they are slower than the rest of the circuit. To overcome this problem, we only switched once at every completion of a cycle of the main program. The main program has many simulations of behaviors. They have a total of 13 detectable behaviors. However, we found out that more complex behaviors surged from this combination. We noticed that the robots were searching in a very random manner. We suspect that this is the result of mixing the behaviors of collision avoidance, sonar detection, avoiding physical contact and infrared detection. Each behavior act by itself, but because the processor examines all of them many times a second, a new random behavior surfaced. Overall, the robot performed as expected.

Integrated System

Once all of the hardware is tested and put in the correct place, we will use the program to control the behaviors of the robots. The board is in charge of managing, 1 set of IR for wall-following, 1 set for base-finding, 1 sonar for shooting opponent, 3 sonars on a tower for receiving shots, 4 contact sensors, a status LED bar graph, and a piezo buzzer. Each behavior will be programmed separately until it works properly. Since many behaviors are incorporated, we will resolve any problem they may have with other sensors and their possible new behavior. Refining the behavior interaction can be done as the robots achieve a working format. The behavior interaction can be seen by the following flow-chart in Figure 6, in the appendix section at the end of the report. Here we see how all behaviors are related to each other and when does a particular behavior is allowed and when it is not. This combination of behaviors and reactions will permit each robot to perform the objectives described in the introduction. One of the most difficult and challenging problems will be with the sonar. Because of its long-range capabilities, incorporating the sonar to the design will increase the chances of, not only error, but also of flexibility requirements. Because each robot will have 20 sensors and 2 motors, coordinating all of them will be a real challenge. For this reason, we chose the TJ Pro board as our desired processor board. This board offers plenty of flexibility, and possesses many of the needed connections and pins as part of its layout. Being the purpose of the game for the robots to find each other, it is a requirement that the robots be agile enough but also well equipped with many sensors. The TJ body offers a simple structure with many levels for different sensors. Proper integration of all components, including correct behaviors, is vital to our success in this project.

Mobile Platform

The body of our robot is the familiar Talrik Junior or TJ for short. A picture of the TJ can be seen in Figure 7a in the appendix section. The TJ body offers the strength of a hard and simple frame, and a horizontal circular cap, which will later help to separate sensors for specific purposes. But most important of all, it offers savings on weight. We intend the robots to be agile and fast. The structure has enough space for two servos, which will drive the robot using 3-inch wheels. It also has a surrounding flexible bumper used to detect physical contact with the field's wall, obstacles or the other robots. Inside the structure, we can also see the board, which operates all of the sensors and gives the robot its commands and behaviors. Because all of the robots wiring and devices are hidden on the inside, we were able to spray paint the frames and make the robots presentable. Also, because the purpose of the game is for spectators to constantly watch them, the more uniform and smooth looking appearance makes them more friendly and pleasing to the eye. We cut out the layout of the TJ the first day in lab. Each sheet of wood has enough pieces for two TJs, therefore, we got two. We used an upside down sander to smooth out the corners and sides of the newly cut parts. We also used super glue to hold the pieces together. After a minute of drying, we applied more glue to the joints to strengthen them. An embarrassing mistake made was to miscount the number of pieces from one robot by gluing extra pieces on the other. To make up for the missing pieces, we looked around in the lab and found extra pieces of an old layout that apparently broke in half. Once the body was glued, we took the robots home and sprayed them each a different color. One important tip to remember when spray painting is to wear eye goggles or glasses when cleaning the spray while looking really closely at the tip. The robot named "Search" was painted white, "And" was sprayed metallic silver, while the robot named "Destroy" was painted red. This will help us to distinguish between them. To make them even more creative and colorful, we exchanged the circular section to a different color, so that each one would have two colors. Figure 7b in the Appendix shows the diversity of colors for each one.

Actuation

The robots use two servos for mobility. A picture of the servos can be found in Figure 8a, under the appendix section. The board, using a fairly simple program, will control these devices. The program will tell the servos what speed to rotate by sending it a number. We will use the command ***"motorp(a, b);"*** where a is the number for the left or right servo (0 or 1). In turn, b will be the relative speed of the motor. We hacked these servos to make them work as motors by physically breaking off the connection between the final gear and the position potentiometer. Each servo includes a controller that tells the current position and the desired one. The farther the degree difference the faster it will rotate. Therefore, the variable b used previously is used to tell the servo how fast to go. Just like the TJ body, these servos are also lightweight and very efficient. The servo motion will help the robot in the search-and-destroy behavior by turning the entire robot in the direction of the opponent. Once aligned with the enemy, the shooting sonar will be ready for use. Mounted onto the servos are two big 3-inch wheels. These will help the robot have a smooth ride and provide enough underside space for the base-detecting sensors. Because of its lightweight characteristic, each robot will be able to smoothly drive over objects without too much trouble. Another important aspect of the servos is that they are not entirely calibrated to zero as their stopping motion. Therefore, we had to make a calibration system so different clones of the robot can use the same program without the need of further changes. We approached this problem by first finding out the exact value for which each robot would be at zero. To our surprise, all six servos had about the same value. The value for the left one was 6, while the right one stopped moving at –13. We then proceeded to make global variables at the beginning of the program where these values could be entered. By adding these values to the desired values, we accomplished to send the servo controller boards the correct values. Most important of all, these variables can be changed in case the robot re-calibrates or if another robot having different values is used.

Sensors

The robot will be loaded with a number of sensors to help it move smoothly around the field and towards its target. It will use IR sensors to help it follow a wall while it is looking for the base. A sample IR is shown in Figure 8b in the appendix section. To detect the base, it will have an IR set underneath to sense the dark floor of the base. We will use a piece of flat black plastic. To locate its opponent, it will use sonar to sense the direction of shots, while also using infrared to detect close presence of the opponent. Many evasive maneuvers can be developed from this combination of sensors. Along with these sensors, each unit will have touch sensors (bumper), in case the robots make contact or an unexpected obstacle is present. This is necessary because the robots turn and move at great speed. To aid spectators view the status of the game, each robot contains a 10-LED status segment connected to an 8-bit flip-flop chip. From these 10 LEDs, we will use 8 of them. Each segment will light up every time the opponent robot accomplishes a hit onto the target. After 4 hits, if the robot successfully reaches the base, the segments will then clear. To show the creativity of the game, we also added a function that makes the bar graph sequentially turn on and off each LED in order back and forth. The robots will also be equipped with a piezo buzzer, which will announce to the audience that a shot has been fired. The same piece will also indicate when one of the robots has perished by giving a "flat-line" sound.

The sensors used by our robots will enable them to view the world around it in a general but limited way. The sensors will give information with respect to objects in the proximity and moving objects in front of it. With this data, the robot can satisfy its only purpose… to find the other robots and to shoot sonar in their direction. Because integrating so many sensors in one robot might overwhelm the processor, we had to rely on better programming and logic to device a program that will read sensors and using data only when it needs it. In this paper, we will see how we adapted each sensor to the robot and how we incorporated them together.

Bumper Sensors

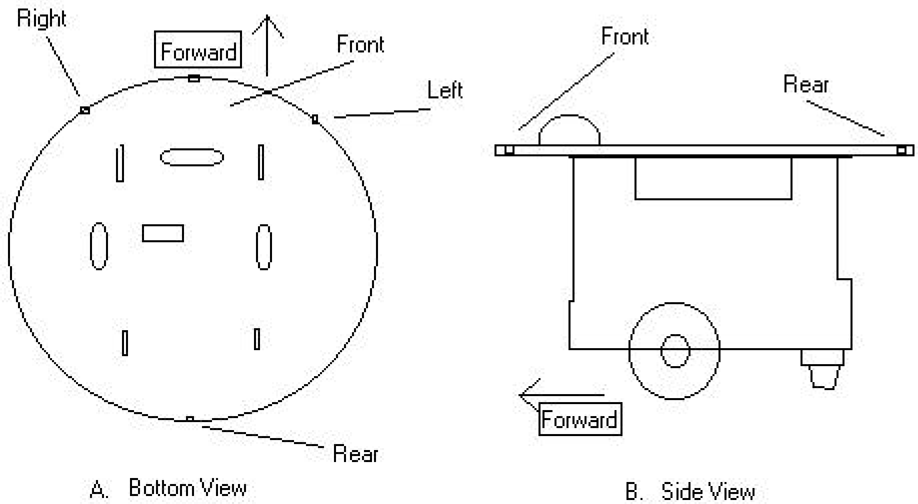

Figure 1: Bumper sensor location

The bumper sensor is a series of switches positioned around a movable bumper frame around the head of the robot. We can see their location in Figure 1. When the robot bumps into an object, the switch closes and the processor can then inspect the analog-to-digital converter PE0. The TJPRO board is setup so that four switches can be individually connected. When PE0 is checked, each switch is differentiated in steps. In our robots, we only used two switches, one in the rear and three parallel connected in the front. Because these switches are positioned on the top section of the TJ, objects smaller than that height will not make contact and therefore will not be discovered. Knowing this, the bumper sensors are a last resort in avoiding running into things. Overall, the bumper will not be used for the main purpose of the game, but becomes important, since one of the behaviors will be to further injure an already badly injured robot. Therefore, a robot that has been hit more than 4 times will further have an extra shot if bumped. These features are mostly to avoid collisions between fast moving robots.

Infrared Sensors

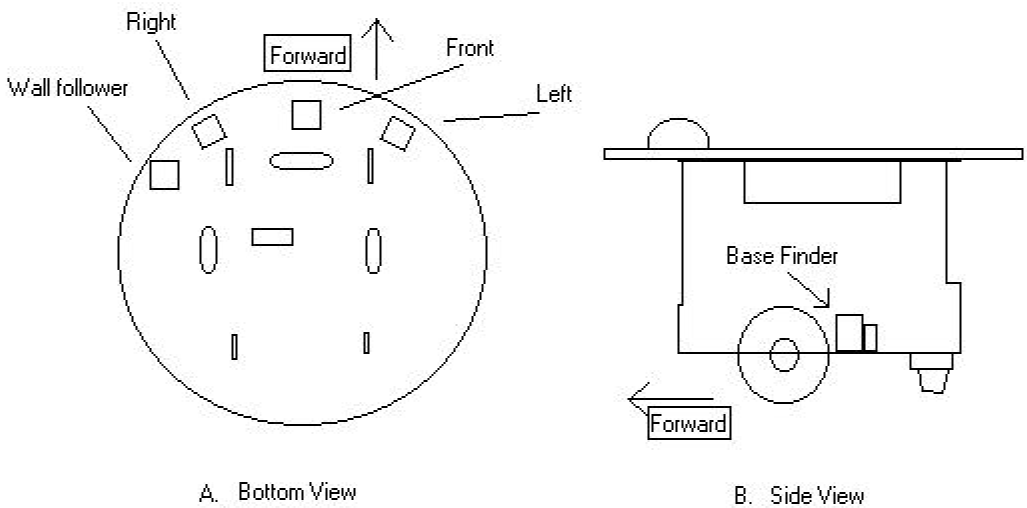

Figure 2: IR sensor location

We are using five infrared sensors and emitter pairs on each robot. The location of the IR sensors is drawn in Figure 2. The IR emitters are connected to pre-wired connections of 40KHz signals. These can individually be controlled by using the command poke(0x7000,0xff), where ff can be replaced by the actual combination of sensors desired to be on. We have three IR placed in front of the robot below each of the front bumper switches. Each connected to PE2, PE3, PE4, PE5 and PE7. Every detected value has a range from approximately 80 to 130. The A-to-D converters give a digital value to the processor about how far an object would be from the robot. At the beginning of the main program, we call a subroutine called calib_IR(). his routine makes the robot spin slowly and take measurements of the lowest value of the IR. This helps set the lower bound for each IR set. This self-calibration step is needed since each sensor does not necessarily possess the same characteristics as others might. We also positioned another IR on the right side of its body. This sensor aims perpendicular to its forward path and it is intended to measure its distance from the playing field wall. This IR is used when the robot has been hit more than half of the allowable number of times. When this occurs, the robot follows the wall until it finds the base. When this behavior is active, the robot will run away from battle and attempt to find the base, which we assume will be near the wall of the playing field. This way, the robot can give full "attention" to the wall following mechanism. At the base, the robot can then clear the number of hits and go back to the game. The last IR is used to aid in the search of the base. This "base-finder" IR works simply by detecting a lack of IR. This only occurs when either the robot is lifted or when a black, non-reflective surface is directly underneath the robot. We must assume that the base is found somewhere next to the wall. The other feature of the base-finder is that the robots stop when they are picked up. This becomes handy when dealing with multiple robots. This last IR sensor is always pointing down and therefore, cannot be calibrated. This is why it is used in a digital format (if there is IR, keep going; else, stop).

Sonar Sensors

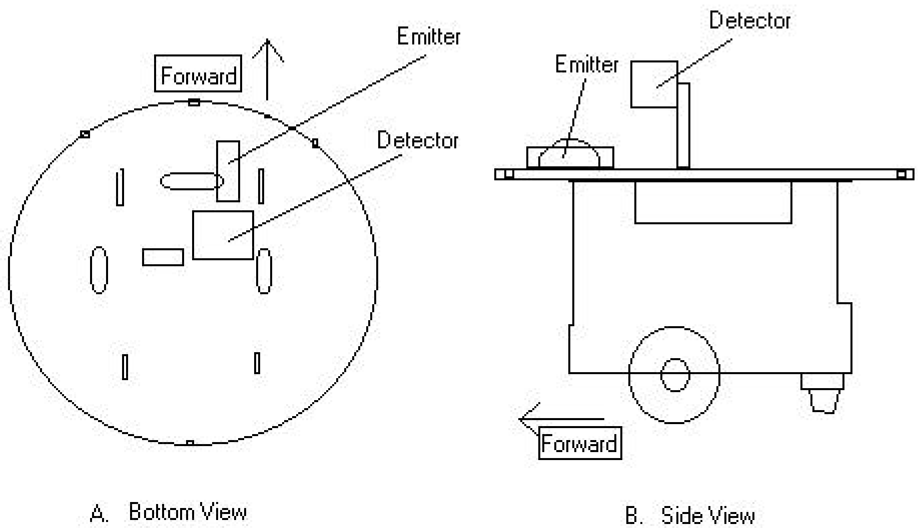

Figure 3: Sonar location

The final sensor used by the robots is the sonar. Their location can be seen in Figure 3. The sonar is a simple device that transmits ultrasonic waves. When the waves encounter an object, they bounce back and are detected. The sensor can then determine how long it took the wave to travel. We will make the robot stop before we use the sonar. Emitting these waves without moving the robot is essential to the accuracy of the sonar. To help avoid false detection, we will give the robot a certain degree of error. The signal processed by the robot will be a number from 0 to 255, just like the IR. But because we added a low pass filter to the output signal, we got a value from 1 to 60. The advantage of this extra step is to convert the value so that the higher the value the closer the sonar shot. The sonar emitters will be connected to the 40KHz pre-wired outputs in the TJPRO board. The emitter will only be turned on when the TJ is stopped and ready to shoot. However, the detector will always be detecting. The detector will be connected to PE6. The code will constantly check this for possible hits. If a hit is detected, then the robot will know another robot is in pursuit. We will have three sonar detectors per robot. These will be pointed to face in different directions on each. We will space each one about 120 degrees apart. This setup will maximize the sensitivity of the sonar. This pinpoint aid tells the robot from what direction is the opponent shooting from. At this point, the target robot would then turn towards the direction of the opponent and reply with a sonar shot. Then the target robot would move away and relocate the opponent for extra defense.

Sensor Experiments

Bumper Experiment

To test the correct working order of the bumper sensors, we wrote a simple program which would call the subroutine bumper(). Table 1 shows the results obtained from this experiment. This routine would check the bumpers for any sign of contact. If any of the switches were detected to be close, then the robot would move in reverse and rotate in the opposite direction. This code checks for any of the bumpers to be closed. If any of them has contact, the routine would play out its instructions. Otherwise, it would do nothing. The bumpers are located at a height of 3.25 inches from the ground. If we assume the ground to be level, any contact between robots or any objects within the playing field should, at worst case, contact at that height.

SENSOR	LOCATION	ANALOG READING
Bumper	Front	45
Bumper	Front-Left	45
Bumper	Front Right	45
Bumper	Rear	250

Table 1: Bumper results

Infrared Experiments

The infrared sensors were tested similarly. To protect the sensitive IR, we enclosed them underneath the upper body. The base finder was hidden within the center of the main body. This would ensure that any bumps would not directly affect the sensors, or the calibration. We wrote a simple program routine for obstacle avoidance. This code is shown in Appendix B. This code shows how as each sensor senses an obstacle, the motors are arranged so that the robot swerves around it. The base finder sensor was tested when we picked up the robot and the robot stopped moving. Below, we show Table 2 of the IR sensors and what the robot does when an obstacle is directly in front. The IR sensors were located slightly below the bumper switches at a height of 3 inches. Anything much higher than 3 inches would not, otherwise, be detected by the sensors. The base finder is located at a distance of 0.5 inches from the ground. Our tests showed that if the robots were raised over a height of 18 inches, the robot would stop. This is equivalent to the robot passing a dark surface. This is very important, since they cannot get confused when searching for the base. The forward and side looking IR reacted at a distance of 4 inches. This relatively small distance was intentionally chosen to maximize the space within the playing field, as well as, to give the robots a really closed feel when detecting each other.

SENSOR	ACTION	A/D	FUNCTION
Front-Left	Avoid left	PE3	Detect obstacles on left
Front	Avoid front	PE7	Detect obstacles head on
Front-Right	Avoid right	PE2	Detect obstacles on right
Wall Follower	Follow wall	PE5	Detect wall
Base finder	Detect black	PE4	Detect base when needed

Table 2: IR sensor table

Sonar experiment

The sonar was used to detect shots. We performed a series of simple tests. The results are shown in Table 3. We put the robot in front of another robot shooting at intervals and programmed it to sense sonar measurements. At the end of the measurements, the robot either would light up the bars from 1 to 8 if there was sonar detected. The first would be lit if the robot detected a signal with a value from 10 to 19, the second if the value was from 20 to 29, etc. The results from the sonar test showed that the sonar would work at a distance of over 3 feet, which was more than enough to give each robot enough capability. Because the sonar is a wave, the calibration routine did not need to adjust the sonar. To distinct between it's own shot and the opponents' shooting, we made the program wait a bit long before it measured any readings from the detector. Otherwise, if the robot detects a sonar signal not from itself, it would be considered a hit. We positioned the detectors on top of the robot and made them aim outward 120 degrees apart. This will give the robot the maximum range of detection. The circuit design for the sonar emitter is shown in Figure 4 in the Appendix and the circuit for the detector in Figure 5.

SENSOR	**EMITTER**	**DISTANCE(ft)**	**RESULT(# LEDs)**
Sonar	off	any	0
Sonar	on	0.5	6
Sonar	on	1.0	5
Sonar	on	1.5	4
Sonar	on	2.0	3
Sonar	on	2.5	2
Sonar	on	3.0	1
Sonar	on	3.5	0
Sonar	on	4.0	0
Sonar	on	4.5	0

Table 3: Sonar sensor table

We can also see in Figure 4, the layout of the schematics used for the sonar emitter.

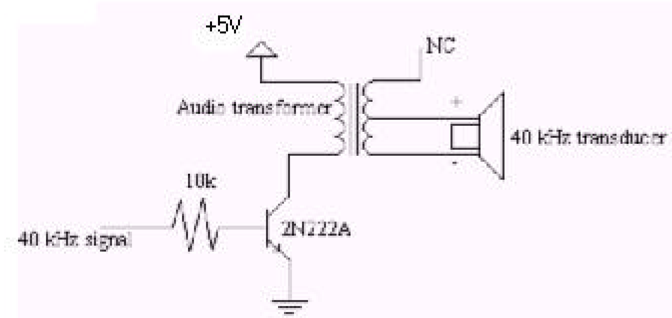

Figure 4: Sonar Transmitter

In Figure 5, we also show the circuit layout of the sonar detector.

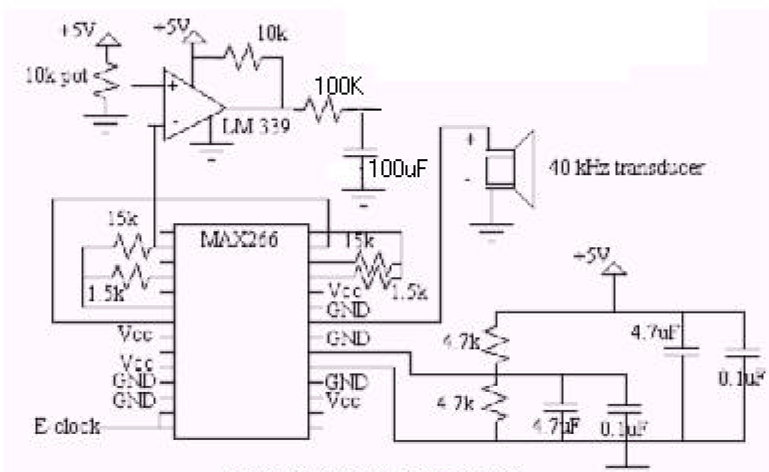

Figure 5: Sonar Detector Circuit

Behaviors

The robots will have quite a few behaviors, which we hope to combine to make more complex behaviors. The first behavior is to avoid physical contact. It accomplishes this by using the four bumper switches. A basic behavior for the robots will be collision avoidance. The three IR sets located in front of the robot will aid in this task. Another behavior is to find the base by using the IR pointing down. This behavior will be used primarily in conjunction with wall following. The wall following behavior will use side-positioned IR sets to compare the distance to the field wall and to follow that wall until the base is found. Positive reinforcement is given when the robot reaches the base while injured. At the base the robots can get "repaired" and "reloaded", they need to find the base and stay there for at 10 seconds. A behavior characteristic of this game will be to find the other robot. Shooting and detecting shots is a vital behavior for this game, using sonar as the sensor of choice. It also simulates injury by slowing down its speed. At the same time, it shows a preference of flight (not fight) behavior when seriously injured. When in this situation, the robot will eagerly look for the base and not shoot the opponent in hopes of not attracting attention to itself. It can also detect the direction of the shot using the detector tower. Once the direction is specified, the robot can proceed to reply to that section of the area with a shot. A very important behavior occurs at the beginning of the game. The robots rotate at a slow speed and calibrate their IR. This accomplished by taking readings of the environment and comparing the lowest possible value to a predetermined variable. If the value is lower, then that variable is updated with that value. In essence, this calibration helps determine the changes in infrared ranges. Here, the robot under attacked, will start to move and shoot back in the direction of the shot. It will then search and try to locate the enemy for a more precise hit. The last behavior these robots will perform is to simulate the end of existence by stopping indefinitely. The sequential light shows this on the bar graph and the “flat-line” sound from the buzzer. Below we show a summary of the behaviors in Table 4.

Behaviors		**Preferred Sensor**
1.	Avoid Physical Contact	(Bumpers)
2.	Collision Avoidance	(IR)
3.	Wall Following	(IR)
4.	Base Finding	(IR)
5.	Self-Calibration	(of IR)
6.	If “stuck” in corner, gets out	(Random stop and go)
7.	Shoots and detects shots	(Sonar)
8.	Senses direction of Shot	(Sonar Tower)
9.	Simulates injury	(Change in speed)
10.	Negative reinforcement	(Shot if too close to wall)
11.	Recuperates after injury	(If it makes it to base)
12.	Flight (not fight) behavior	(If shot more than half)
13.	Simulates end of existence	(Gives a flat line and light)

Table 4: Behaviors at a glance

Experimental Layout and Results

We will be testing the MTJPRO11 boards using simple programs to test the servos. A simple board test will also be used to check the correct working condition of all of its elements. We need to perform these tests to assure the working order of the robots, since these boards were assembled in the lab. The next test will check for correct direction of the servo rotation to assure correct response from each servo. The memory test will check for correct space handling from the 32K bytes of RAM in the boards. Along with the servo test, we need to check if the new 3-inch wheels will fit and not interfere with any other component from inside the TJ.

Problems

Below is a table describing the problems we had while completing this project. These are found in Table 5.

- Sonar contradiction: Hard to distinct between own shot and opponent's shot.
  (SOLUTION: wait long enough, then detect opponent's.)
- Relay delay: Mechanical limitations on relay speed.
  (SOLUTION: change relays about once every complete cycle of the main program.)
- 3 sonars and 1 circuit: Needs constant attention of detector.
  (SOLUTION: use relays to physically connect to each one.)
- Modularity of components: Every module must be exchangeable between robots.
  (SOLUTION: be patient and hope for the best.)
- 74HC574's only have 8 Flip-flops, not 10: The bar gragphs have 10 LEDs, what to do with extra two.
  (SOLUTION: change to 8 maximum shots, not 10.)
- Beware of Murphy's Law: The day before the final demo all robots had sudden problems.

(SOLUTION: do not panic, pray…)

Conclusion

The robot construction was a definite success. We built these robots using a modular approach. Each part of its total 12 sections was built identically. This helped us to later test them by simply interchanging them with robots loaded with a testing program. We design the program using the same approach. The software has many modules or subroutines. Each one can easily be tested individually. This format helped us later in the programming by providing an easy to understand and compose a sequence of behaviors.   We met our original goal: to make a friendly game of search and destroy. Of also designing a platform that can be copied exactly and replicated infinite times and expected to perform the same way. We also accomplished other goals of building modular hardware identically enough that it can be exchanged between robots. Also to use software to help solve conflicts in differentiating sonar pulses from different sources. And last, to develop a modular software flexible enough that it can adapt to any of the robots. Other future improvements that can be added are: a self-diagnostic program (which could detect if any of the components are not working), add multiple robots (to see the behavior change in having more robots), adding behaviors to remember the location of the base, (maybe even moving the base to the middle of the playing field).

Documentation

The following is a list of our sources for information, specifications, and design:

Fred Martin, The 6.270 Robot Builder's Guide, MIT Media Lab, Cambridge, MA, 1992.

Intelligent Machines Design Laboratory Web page:
http://www.mil.ufl.edu/
http://www.mil.ufl.edu/imdl

Mekatronix home page:
http://www.mekatronix.com/

Appendices

Figure 7a: picture of the robots in playing field

Figure 7b: picture of one “Search” from top

Figure 7c: picture of the robots from every angle

Figure 8a: picture of servo

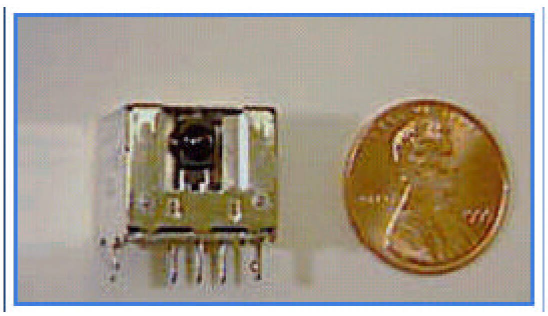

Figure 8b: picture of infrared

ROBOT PROGRAM

```
/**************************************************************\
****************************************************************
**                                                            **
**      Title:      TJPRO.C                                   **
**                                                            **
**      Programmer: Jose Garcia-Feliu                         **
**                                                            **
**      Description:   Program to control TJ's for game.      **
**                                                            **
****************************************************************
\**************************************************************/
int i = 0;                          /* Variable for calibration */
int IR2=99,IR3=99,IR4=99;           /* Variables for calibration */
int IR5=99,IR7=99;                  /* Variables for calibration */
int y = 0, h = 0, j = 0, m = 0;     /* Variable for sonar loop */
int counter, counter_b, motion;     /* Variable for Sonar */
int shots = 0;                      /* Number of shots */
int sonar_counter = 0;              /* Variable for sonar detection */
int m0 = -13, m1 = 6;               /* Calibration for motors at stop */
/************************ Program ********************************/
void main() {                       /* Start of Main */

/************************ Setup **********************************/
     poke(0x7000,0x00);             /* Turn IR off */
     poke(0x6000,0x00);             /* Turn Bars off */
     poke(0x1008,0x00);             /* Turns beeps off */
     sleep(.1);                     /* Relax for a bit */
     servo_on();                    /* Turn servos on */
/************************ Calibration ****************************/
     calib_IR();                    /* Calls calibration routine */
     sounds();                      /* Makes funny sounds */
     display_seq();                 /* Uses display before program */
     display_0();                   /* Clears display */
/************************ Infinite Loop **************************/
while (1) {                         /* Infinite Loop */
   if (shots < 4) {bumpers();       /* Avoids physical contact */
          IR();                     /* Controls IR */
          detect_sonar();           /* Detects sonar */
          if (y > 100){y = 0;       /* Reset variable, every so often */
                       sonar(); }   /* Uses sonar to find opponent */
   }
   if (shots >= 8) {game_over(); }  /* Robot is disabled */
   if (shots >= 4) {bumpers();      /* Avoid physical contact */
                 wall();            /* If close to dying */
                 detect_sonar(); }  /* Detects sonar */
           y = y + 1;               /* Increase variable */

 }                                  /* End while - infinite */
}                                   /* End Main */
/************************ End of Program *************************/


/******************************************************************/
/*                      Calib_IR                                  */
/******************************************************************/
```

```
void calib_IR() {                  /* Calibrates the minimum values for
IR */
                 motorp(1,12+m1);  /* Goes around slowly */
                 motorp(0,-12+m0);
while (i < 20) {                   /* Loop to calibrate all IR */
                 poke(0x7000,0x04);
             if (analog(2) < IR2) {IR2 = analog(2);}
                 poke(0x7000,0x01);
             if (analog(3) < IR3) {IR3 = analog(3);}
                 poke(0x7000,0x80);
             if (analog(4) < IR4) {IR4 = analog(4);}
                 poke(0x7000,0x10);
             if (analog(5) < IR5) {IR5 = analog(5);}
                 poke(0x7000,0x02);
             if (analog(7) < IR7) {IR7 = analog(7);}
             i = i + 1;
             sleep(.1); }
}                                        /* End of subroutine */

/*********************************************************************/
/*                      Sounds                                       */
/*********************************************************************/
void sounds() {                          /* Beeps speaker */
           poke(0x1008,0x10);
           sleep(.45);
           poke(0x1008,0x00);
           sleep(.2);
           poke(0x1008,0x10);
           sleep(.2);
           poke(0x1008,0x00);
           sleep(.1);
           poke(0x1008,0x10);
           sleep(.2);
           poke(0x1008,0x00);
           sleep(.1);
           poke(0x1008,0x10);
           sleep(.1);
           poke(0x1008,0x00);
             sleep(.1);
           poke(0x1008,0x10);
           sleep(.1);
           poke(0x1008,0x00);
             sleep(.2);
           poke(0x1008,0x10);
             sleep(.3);
           poke(0x1008,0x00);
           sleep(.4);
           poke(0x1008,0x10);
           sleep(.1);
           poke(0x1008,0x00);
           sleep(.1);
           poke(0x1008,0x10);
           sleep(.1);
           poke(0x1008,0x00);
}                                        /* End of subroutine */

/*********************************************************************/
```

```
/*                    Display_seq                                  */
/*****************************************************************/
void display_seq() {                      /* Displays the bar graph */
          while ( h < 2 ) {               /* Loop for bars */
           poke(0x6000,0x01);
           sleep(.1);
           poke(0x6000,0x02);
           sleep(.1);
           poke(0x6000,0x04);
           sleep(.1);
           poke(0x6000,0x08);
           sleep(.1);
           poke(0x6000,0x10);
           sleep(.1);
           poke(0x6000,0x20);
           sleep(.1);
           poke(0x6000,0x40);
           sleep(.1);
           poke(0x6000,0x80);
           sleep(.1);
           poke(0x6000,0x40);
           sleep(.1);
           poke(0x6000,0x20);
           sleep(.1);
           poke(0x6000,0x10);
           sleep(.1);
           poke(0x6000,0x08);
           sleep(.1);
           poke(0x6000,0x04);
           sleep(.1);
           poke(0x6000,0x02);
           sleep(.1);
           poke(0x6000,0x01);
           sleep(.1);
           h = h + 1; }                 /* End of While */
           h = 0;                       /* Reset value of H */
}                                         /* End of subroutine */

/*****************************************************************/
/*                    Display_0                                    */
/*****************************************************************/
void display_0() {                        /* Displays the bar graph */
           poke(0x6000,0x00);           /* Display '0' bars */
}                                         /* End of subroutine */

/*****************************************************************/
/*                    Display_1                                    */
/*****************************************************************/
void display_1() {                        /* Displays the bar graph */
           poke(0x6000,0x01);           /* Display '0' bars */
}                                         /* End of subroutine */

/*****************************************************************/
/*                    Display_2                                    */
/*****************************************************************/
void display_2() {                        /* Displays the bar graph */
           poke(0x6000,0x03);           /* Display '0' bars */
```

```
}                                         /* End of subroutine */

/**********************************************************************/
/*                      Display_3                                      */
/**********************************************************************/
void display_3() {                        /* Displays the bar graph */
           poke(0x6000,0x07);           /* Display '0' bars */
}                                         /* End of subroutine */

/**********************************************************************/
/*                      Display_4                                      */
/**********************************************************************/
void display_4() {                        /* Displays the bar graph */
           poke(0x6000,0x0f);           /* Display '0' bars */
}                                         /* End of subroutine */

/**********************************************************************/
/*                      Display_5                                      */
/**********************************************************************/
void display_5() {                        /* Displays the bar graph */
           poke(0x6000,0x1f);           /* Display '0' bars */
}                                         /* End of subroutine */

/**********************************************************************/
/*                      Display_6                                      */
/**********************************************************************/
void display_6() {                        /* Displays the bar graph */
           poke(0x6000,0x3f);           /* Display '0' bars */
}                                         /* End of subroutine */

/**********************************************************************/
/*                      Display_7                                      */
/**********************************************************************/
void display_7() {                        /* Displays the bar graph */
           poke(0x6000,0x7f);           /* Display '0' bars */
}                                         /* End of subroutine */

/**********************************************************************/
/*                      Display_8                                      */
/**********************************************************************/
void display_8() {                        /* Displays the bar graph */
           poke(0x6000,0xff);           /* Display '0' bars */
}                                         /* End of subroutine */

/**********************************************************************/
/*                       Bumper                                        */
/**********************************************************************/
void bumpers() {                          /* Avoid physical contact */
  if((analog(0)>10)&&(analog(0)<70)) {    /* If bumped from front */
                  motorp(1,-20+m1);       /* Backs up */
                  motorp(0,-20+m0);
              sleep(1.0);               /* Then turn around */
                  sonar_emitter();        /* Sounds shot */
                  shot_sound();           /* Gets angry if bumped */
                  motorp(1,-50+m1);       /* Go elsewhere */
                  motorp(0,50+m0);
     if (shots >= 5) {shots = shots + 1;}/* If bumped when weak */
```

```
               sleep(1.0); }
  else if (analog(0) > 100) {              /* If bumped from behind */
                   motorp(1,20+m1);        /* Move forward for a bit */
                   motorp(0,20+m0);
               sleep(1.0);             /* Then turn around */
                   motorp(1,-50+m1);
                   motorp(0,50+m0);
                   sleep(.85);
               motorp(1,0);            /* Stops */
               motorp(0,0);
                   shots = shots + 1;      /* When hit from behind */
               sonar_emitter();        /* Sends sonar */
               shot_sound(); }         /* And make sounds */
}                                          /* End of subroutine */

/*********************************************************************/
/*                     IR                                             */
/*********************************************************************/
void IR() {                                /* Controls IR sensors */

  if ((setIR_2())&&(analog(2) > (IR2+(130-IR2)/2))) {
                   detect_ir();            /* Detects other robots */
                   motorp(1,-50+m1);       /* If obstacle on right */
                   motorp(0,50+m0); }
  else if ((setIR_3())&&(analog(3) > (IR3+(130-IR3)/2))) {
                   detect_ir();            /* Detects other robots */
                   motorp(1,50+m1);        /* If obstacle on left */
                   motorp(0,-50+m0); }
  else if ((setIR_7())&&(analog(7) > (IR7+(130-IR7)/2))) {
                   detect_ir();            /* Detects other robots */
                   motorp(1,-50+m1);       /* If obstacle in front */
                   motorp(0,50+m0); }
  else {                                   /* If no obstacles on IR */
                   motorp(1,50+m1);
                   motorp(0,50+m0); }
}                                          /* End of Subroutine */

/*********************************************************************/
/*                  wall following                                    */
/*********************************************************************/
void wall() {                              /* Follows the contour of wall
*/
  if (setIR_4()&&(analog(4) < (IR4+2*(130-IR4)/3))) {
               motorp(1,0);            /* When base is found */
               motorp(0,0);            /* Gets fixed and reloaded */
               sounds();               /* Let know the base is found */
               sleep(10.0);            /* Wait a little bit */
               shots = 0;              /* Resets shots */
               display_0(); }          /* Clears display */
  else if (setIR_2()&&(analog(2)>(IR2+2*(130-IR2)/3))) {
                   motorp(1,-20+m1);
                   motorp(0,20+m0); }
  else if (setIR_7()&&(analog(7)>(IR7+2*(130-IR7)/3))) {
                   motorp(1,-20+m1);
                   motorp(0,20+m0); }
  else if (setIR_5()&&(analog(5)>(IR5+5*(130-IR5)/7))) {
                   motorp(1,15+m1);
```

```
                motorp(0,25+m0); }
  else if (setIR_5()&&(analog(5)>(IR5+(130-IR5)/4))) {
                motorp(1,25+m1);
                motorp(0,15+m0); }
  else {                              /* If no wall, go straight */
                motorp(1,20+m1);
                motorp(0,20+m0); }
}                                     /* End of Subroutine */

/**********************************************************************/
/*                   setIR_0                                        */
/**********************************************************************/
void setIR_0() {                     /* Turns all IRs off */
              poke(0x7000,0x00);
              sleep(.05); }          /* End of Subroutine */

/**********************************************************************/
/*                   setIR_2                                        */
/**********************************************************************/
int setIR_2() {                      /* Turns corresponding IR */
              poke(0x7000,0x04);
              sleep(.05);
              return 1; }            /* End of Subroutine */

/**********************************************************************/
/*                   setIR_3                                        */
/**********************************************************************/
int setIR_3() {                      /* Turns corresponding IR */
              poke(0x7000,0x01);
              sleep(.05);
              return 1; }            /* End of Subroutine */

/**********************************************************************/
/*                   setIR_4                                        */
/**********************************************************************/
int setIR_4() {                      /* Turns corresponding IR */
              poke(0x7000,0x80);
              sleep(.05);
              return 1; }            /* End of Subroutine */

/**********************************************************************/
/*                   setIR_5                                        */
/**********************************************************************/
int setIR_5() {                      /* Turns corresponding IR */
              poke(0x7000,0x10);
              sleep(.05);
              return 1; }            /* End of Subroutine */

/**********************************************************************/
/*                   setIR_7                                        */
/**********************************************************************/
int setIR_7() {                      /* Turns corresponding IR */
              poke(0x7000,0x02);
              sleep(.05);
              return 1; }            /* End of Subroutine */

/**********************************************************************/
```

```
/*                    setIR_73                                    */
/******************************************************************/
int setIR_73() {                        /* Turns corresponding IR */
                poke(0x7000,0x03);
                sleep(.05);
                return 1; }             /* End of Subroutine */

/******************************************************************/
/*                    setIR_75                                    */
/******************************************************************/
int setIR_75() {                        /* Turns corresponding IR */
                poke(0x7000,0x12);
                sleep(.05);
                return 1; }             /* End of Subroutine */

/******************************************************************/
/*                    sonar                                       */
/******************************************************************/
void sonar() {                          /* Sends sonar signal */

        motorp(1,0);                  /* Robot stops */
        motorp(0,0);
while(m < 10) {
          detect_ir();
          detect_sonar();
          m = m + 1; }                  /* Increases variable */
          m = 0;                        /* Resets variable */
}                                       /* End of Subroutine */

/******************************************************************/
/*                    sonar_emitter                               */
/******************************************************************/
void sonar_emitter() {                  /* Sends sonar signal */

          poke(0x7000,0x40);            /* Send sonar pulse */
          msleep(1L);                   /* Waits a little bit */
          poke(0x7000,0x00);            /* Turns sonar emitter off */
          msleep(1L);                   /* Waits a little bit */
          poke(0x7000,0x40);            /* Send sonar pulse */
          msleep(2L);                   /* Waits a little bit */
          poke(0x7000,0x00);            /* Turns sonar emitter off */
          msleep(1L);                   /* Waits a little bit */
          poke(0x7000,0x40);            /* Send sonar pulse */
          msleep(3L);                   /* Waits a little bit */
          poke(0x7000,0x00);            /* Turns sonar emitter off */
          msleep(1L);                   /* Waits a little bit */
          poke(0x7000,0x40);            /* Send sonar pulse */
          msleep(4L);                   /* Waits a little bit */
          poke(0x7000,0x00);            /* Turns sonar emitter off */
          msleep(1L);                   /* Waits a little bit */
          poke(0x7000,0x40);            /* Send sonar pulse */
          msleep(5L);                   /* Waits a little bit */
          poke(0x7000,0x00);            /* Turns sonar emitter off */
          msleep(1L);                   /* Waits a little bit */
          poke(0x7000,0x40);            /* Send sonar pulse */
          msleep(6L);                   /* Waits a little bit */
          poke(0x7000,0x00);            /* Turns sonar emitter off */
```

```
        msleep(1L);                     /* Waits a little bit */
        poke(0x7000,0x40);              /* Send sonar pulse */
        msleep(7L);                     /* Waits a little bit */
        poke(0x7000,0x00);              /* Turns sonar emitter off */
          sonar_counter = 0;              /* Resets variable */

}                                       /* End of Subroutine */

/********************************************************************/
/*                        Detect_sonar                              */
/********************************************************************/
void detect_sonar() {                   /* Detects shots */

if (sonar_counter > 0) {
  if (1)     {poke(0x1008,0x00);}  /* Looks at front sonar */
  else if (j == 2){poke(0x1008,0x04);}  /* Looks at rear left sonar */
  else if (j == 4){poke(0x1008,0x0c);    /* Looks at rear right sonar
*/
                   j = 0; }

  if (analog(1) > 10) {                 /* If shot */
                shots = shots + 1;      /* Then increase variable */
     if (shots==1){display_1();}        /* Update bar graph */
     else if (shots==2){display_2();}   /* Update bar graph */
     else if (shots==3){display_3();}   /* Update bar graph */
     else if (shots==4){display_4();}   /* Update bar graph */
     else if (shots==5){display_5();}   /* Update bar graph */
     else if (shots==6){display_6();}   /* Update bar graph */
     else if (shots==7){display_7();}   /* Update bar graph */
     else if (shots==8){display_8();}   /* Update bar graph */
     if      (0) {shot_sound();
                   sonar_emitter(); }   /* Shoots if fired from front
*/
     else if (0){motorp(1,-10+m1); /* Rotate to direction of shot */
                   motorp(0,10+m0);
                   sleep(.5);
                   shot_sound();        /* And shoot */
                   sonar_emitter(); }
     else if (0){motorp(1,10+m1);  /* Rotate to direction of shot */
                   motorp(0,-10+m0);
                   sleep(.5);
                   shot_sound();        /* And shoot */
                   sonar_emitter();
                   j = 0; } }           /* Resets variable */
  j = j + 1;                            /* Increases variable */
  sonar_counter = 0; }                  /* Reset variable */
  sonar_counter = sonar_counter + 1;    /* Increases value */

}                                       /* End of Subroutine */

/********************************************************************/
/*                        Detect_ir                                 */
/********************************************************************/
void detect_ir() {                      /* Detects robots */

      setIR_0();                        /* Turns all IRs off */
      if (analog(2) > IR2) {sonar_emitter();
```

```
                              shot_sound();}
      else if (analog(3) > IR3) {sonar_emitter();
                              shot_sound();}
      else if (analog(4) > IR4) {sonar_emitter();
                              shot_sound();}
      else if (analog(5) > IR5) {sonar_emitter();
                              shot_sound();}
      else if (analog(7) > IR7) {sonar_emitter();
                              shot_sound();}

while (0) {                             /* TESTING */
  if (j == 0) {poke(0x1008,0x00);}      /* Looks at front sonar */
  else if (j == 1) {poke(0x1008,0x0c);} /* Looks at rear left sonar */
  else if (j == 2) {poke(0x1008,0x08);} /* Looks at rear right sonar */
  poke(0x7000,0x00);
  sleep(.5);
  shot_sound();
  sonar_emitter();
  if (analog(1) > 10) {                 /* If shot */
                shots = shots + 1;      /* Then increase variable */
     if (shots==1)     {display_1();}   /* Update bar graph */
     else if (shots==2){display_2();}   /* Update bar graph */
     else if (shots==3){display_3();}   /* Update bar graph */
     else if (shots==4){display_4();}   /* Update bar graph */
     else if (shots==5){display_5();}   /* Update bar graph */
     else if (shots==6){display_6();}   /* Update bar graph */
     else if (shots==7){display_7();}   /* Update bar graph */
     else if (shots==8){display_8();}   /* Update bar graph */
     else             {display_0();} }  /* In case of error, reset */
  j = j + 1; }                          /* Increases variable */

}                                       /* End of Subroutine */

/**********************************************************************/
/*                    Shot_sound                                      */
/**********************************************************************/

void shot_sound() {                     /* Beeps speaker */

          poke(0x1008,0x10);          /* Makes shooting sound */
            sleep(.15);
          poke(0x1008,0x00);          /* Turns beep off */
}                                       /* End of Subroutine */

/**********************************************************************/
/*                        Game_over                                   */
/**********************************************************************/
void game_over() {                     /* Robot is disabled */

       while(1) {
             poke(0x7000,0x00);         /* Stops emitting IR and sonar
*/
             motorp(1,0);               /* Stops motors indefinetely */
             motorp(0,0);
             display_seq();             /* Displays lights */
             poke(0x1008,0x10);         /* Beeps forever */
       }                                /* End of infinite loop */
```

```
}                                        /* End of Subroutine */

/**********************************************************************/
/*                            t                                       */
/**********************************************************************/
```