The “99% Robot” Report

Team Members:
Alecia Camillo
Jane Esterline
Caris Moses

Bin Number: 37

TA: Jeremiah Shiff

INTRODUCTION

Our goal for this project was to create a robot capable of pushing an opposing robot out of a ring in three minutes or less, or to disable our opponent. Given size, weight, and budget limits, among other guidelines, we came up with a design we call "the 99% Robot" which was reasonably successful in the competition, winning 4 matches total. This report details our motivation, strategy, design, and microprocessor code, as well as our thoughts on the competition itself.

OVERVIEW

Our design comprises of a flipping mechanism, tactical placement of circuitry, egg crate armor, sticky wheels, and strategic steel blocks. Using newly acquired knowledge from class, our main approach to our design and coding was simplicity and efficiency. "The 99% Robot" was cost efficient and in the end performed with success in the competition. Our flipping device, designed as an offensive strategy, proved to be a weakness that could be improved on in the future by using a rat trap. In the end our main strengths were the sticky tires and strategic blocks, one of our defensive strategies, and made us stronger and more powerful than some of our opponents.

MOTIVATION

Our design was motivated largely by the resources available to our team. As none of us are especially experienced in electrical engineering or coding in C, we opted to stick with and perfect the basics supplied to us through the course of Mechatronics. By using a simple H-bridge to control our motors and minimally complicated circuitry to control our sensors, we ensured that we would have the knowledge to code the microprocessor correctly and fix any errors that arose, both before and during the competition. Similarly, by starting from the provided sonar and QTI code, we were able to make the necessary alterations to mesh these with our basic motor code and get our robot working very well.

While this may not seem like the most ambitious way to approach this project, in the spirit of competition, we accepted that other teams were stronger than us in these areas and decided to focus instead on a purely mechanical flipping device to strengthen our robot. Again, as none of us are remarkably experienced with machining, we went for a simple, effective design. We opted for a mouse trap flipping device because mouse traps provide a cheap, easy-to-alter source of a large amount of torsion spring power. Flipping in general was chosen as the method of attack because we guessed that many opponents would not be guarding against being flipped in particular, and because, should the flipping device deploy and fail, pushing the opponent out of the ring remains a perfectly viable option for our robot.

We were motivated to make good use of what we had learned in the course to produce a working robot for the competition while using few outside resources other than to obtain materials. This inspired our robot's name – the 99% Robot – as we did not do anything that the other teams were not capable of doing. Simplicity seemed to work out well for our team. We were not slowed by complicated code or electrical errors in the days (or hours) leading up to the competition, as it seems many other teams were, and our stable, relatively powerful robot advanced easily through the first few rounds of the competition as a result.

DESIGN and STRATEGY

Our design used the provided chassis and wheels as the base of the robot. To make out robot as wide as possible to avoid tipping over, we moved our wheels out as far as possible while still remaining within the 20 cm size limit. To do this, we attached our motors to the outside of the chassis using metal standoffs. Also, the widened the wheel base to create stability and prevent other robots from flipping ours. We ordered special "sticky tires" online to improve our traction and pushing power, and these proved to be a great advantage during the competition and our most effective offensive strategy. We also attached two steel blocks to the underside of the chassis, a larger one in the front to prevent the flipping device from also flipping our robot, and a smaller one in the back to simply add weight and power. The additional steel blocks also served as a defensive strategy as a heavier robot would make it harder for our opponents to push us out of the ring. The final weight of our robot was around 0.92 kg, just under the 1 kg limit. Finally, we positioned our QTI sensors on standoffs under the chassis, two in the front to prevent our robot from leaving the ring and one in the middle to disable the robot upon leaving the ring, all as close to the ground as possible. These aspects of our design are shown in Figure 1.

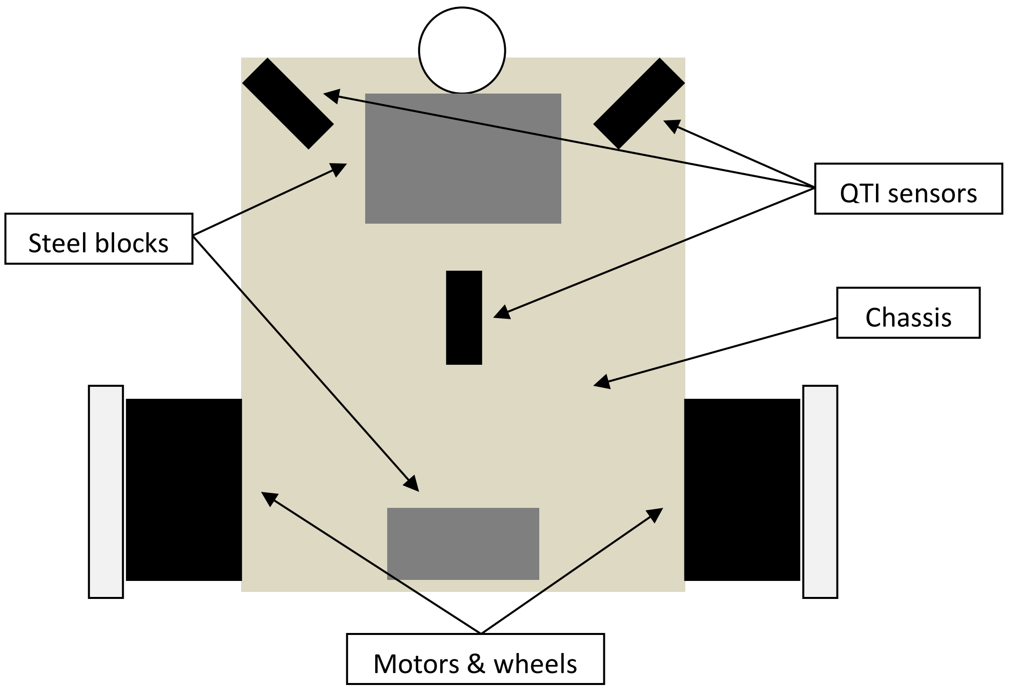

Figure 1: Underside of chassis

On top of our chassis, we had two levels of “storage” for our electronic components, the lower one housing our microprocessor in order to protect it and the upper one housing our H-bridge and battery. We covered the sides of our robot with egg-carton-like packaging material in an attempt to throw off our opponent’s SONAR and offer some protection from attacks, this being our main defensive strategy. We made this covering easy to remove to make changing batteries and making any necessary repairs during the competition faster and easier.

The innovative aspect of the design and main offensive strategy was the flipping device we created. The flipper was designed with the intention of flipping the opponent and in consequence disabling them. The flipping device was attached to the front of our robot with a small piece of sheet metal. A mouse trap, sawed in half to remove the pin that would normally be used to set, was attached to the upper part of this plate. Two pieces of threaded rod were attached to another piece of sheet metal which would serve as the “flipper”. These two pieces of threaded rod were then attached to the movable bar of the mouse trap so that, when the device was set, the flipper would be as close to the ground as possible, making it easier and more likely for opponent to drive on the “flipper” and set it off. Figure 2 shows the flipper in both the “set” position and the position after being deployed.

The flipper is set with varying sensitivity by a hook, shown in Figure 3. This hook was securely attached with wire to a standoff, which was then attached to the top of the center chassis, as close to the front as possible. When the flipping device is then fixed to the front of the chassis, the hook simply hangs freely through a hole drilled in the appropriate place on the sheet metal. When the bar of the mouse trap (and the attached flipper) is pulled down to be set, the hook is used to hold it in place. In this way, the flipper is deployed when enough pressure is put on the device to allow the hook to fall and hang freely again. The idea is that this pressure would be supplied either by another robot’s attempt to push ours or by our attempt to push another robot. This idea is hopefully more clearly explained in Figure 2. In order to increase the surface area over which the flipping device can be pushed in order to deploy, we fixed a steel bar across the two bars that connect to the flipper. This, as well as the position of the SONAR in our design, is shown in figure 4.

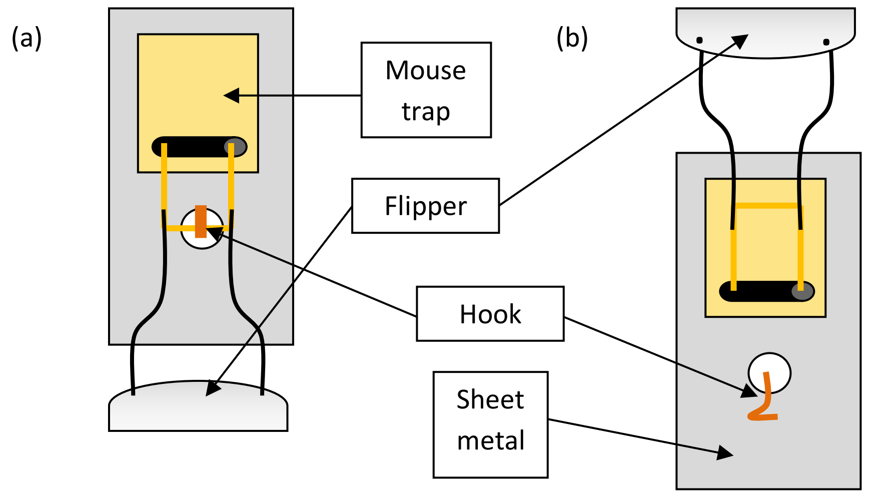

Figure 2: The flipper in (a) set and (b) deployed positions

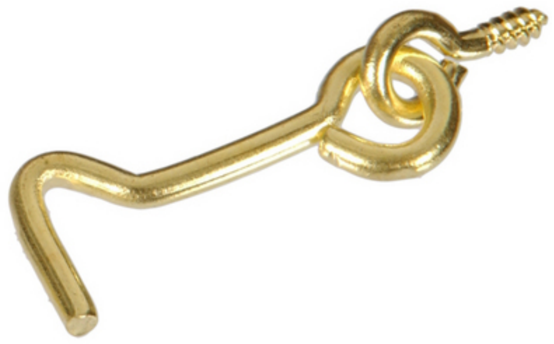

Figure 3: Hook used to set flipping device

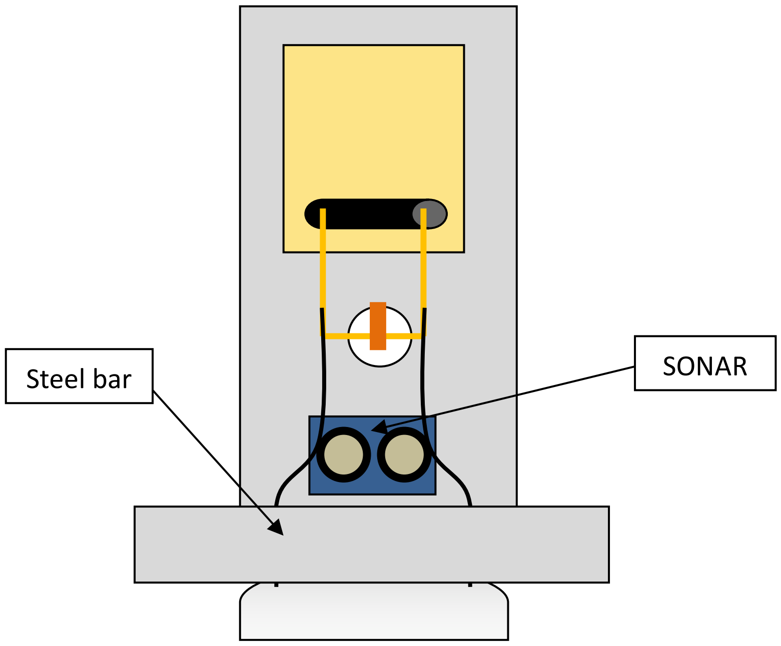

Figure 4: Close up of front of flipping device

The design of our H-bridge is extremely similar to the design described in the H-bridge lab. Other than that, our circuitry consisted purely of using the breadboard on the microprocessor to connect the power and signal wires of the QTI sensors with 10 kΩ resistors and connecting the sensors and motors to signal pins.

Figure 5 shows a picture of our final design, the 99% Robot.

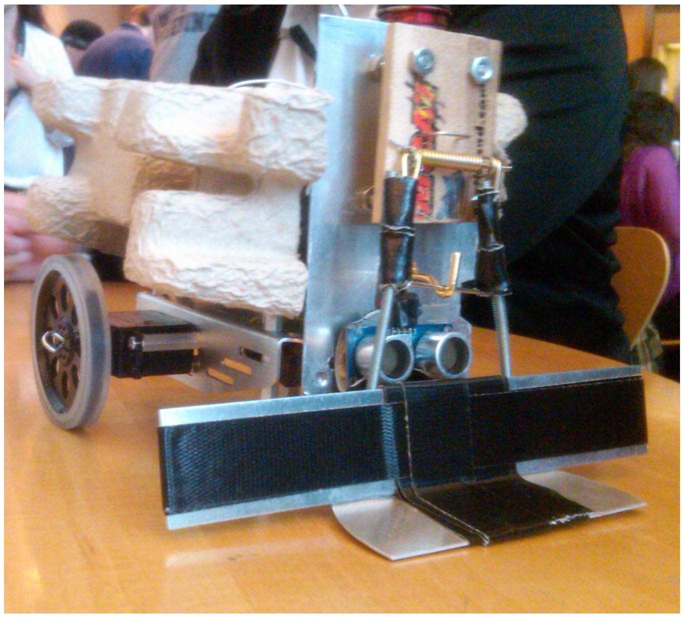

Figure 5: Final design

CODE

The main function of the code for the 99% Robot is to read the signals coming from the four sensors and convert them into mechanical actions. First, the robot directions are declared in a function *robotDirec*. In this function, a variable named *phase* is given four possible values, each representing a robot movement. Each phase variable and corresponding robot direction is shown below:

phase = 1 → spin
phase = 2 → forward
phase = 3 → stop
phase = 4 → backwards

Now, we will describe the placement of our sensors on the 99% robot. Figure 1 also gives a visual of our sensor placement. The robot has 3 QTIs and SONAR attached to it. One of the QTI sensors is attached to front right and another to the front left corner. When the falling edge is detected on either of them, the robot reverses and then returns to search mode, spinning until the SONAR is triggered. When the center QTI detects the falling edge, the “kill” signal is sent and the robot stops and turns on a red LED located at the top of the robot. The SONAR detects the opposing robot and moves towards it when it detects it. All of these actions are enabled by interrupts. Inside of the ISR (interrupt service routine) functions, a variable *running* is set to a different value (*running* will be discussed later). Each sensor has a corresponding ISR. Each ISR contains code that is executed when the interrupt is triggered. Below we will show the ISR for each sensor.

The following code executes when the front QTIs sense the falling edge:

```
ISR(SIG_INTERRUPT0){
TIMSK ^= _BV(OCIE0);

double a;
a=0;
while (a< 12000 ){
robotDirec(4);          // go backwards
a++;
}

running=2;              // search mode (look for opponent)
TIMSK ^= _BV(OCIE0);
}
```

The following code executes when the center QTI is triggered:

```
ISR(SIG_INTERRUPT1) {
running = 0;          //stop
}
```

The following code is for the SONAR

```
ISR(SIG_INPUT_CAPTURE1) {

 if(InPulse==1) {       // First state of machine (captured
                        //rising edge, start of pulse in)
 TCNT1 = 0;             // Start counting
 TCCR1B &= ~_BV(ICES1); // Sets capture pin to respond to
                        // falling edge
 InPulse = 2;
 }
 else if(InPulse==2) {    // Next state of machine
                          //(captured falling edge, end of
                          //pulse in)
 SonarReturnTime = ICR1;  // Reads from ICR1, which was
                          //equal to TCNT1 when interrupt
                          //is thrown
if(SonarReturnTime < 250){
running=1;      //go forward is opponent is in range
 }
 if(SonarReturnTime >= 250){
running =2;     //search (spin) – opponent not is range
```

```
 }
 InPulse = 0;
 }
}
```

In the main function, the variable *running* is changed every time a sensor is triggered. At the very beginning of the code it is set to 2 (search mode). The meaning of all values of *running* is shown below.

running = 0 → middle QTI has detected falling edge, stop
running = 1 → opponent in range, move forward
running = 2 → search for opponent (spin)

BUDGET

In addition to the materials we were given for this project, we purchased sticky tires and a mouse trap. All other materials used were scrap pieces found in various places. All additional parts totaled…

Sticky Tires	$6.50
Mouse Trap	$0.60
Others (scrap metal, screws, tape, etc.)	$1.00
TOTAL	**$8.10**

STRENGTHS AND WEAKNESSES

The 99% Robot had many strengths. Due to the “sticky tires” and weighted blocks, our robot had very high traction and was better suited to push other robots out of the arena. Also, our egg crate armor kept opponents away from our wheels. Our widened wheel base made it harder for wedge robots to flip us over. The QTIs towards the front also helped steer near the edge of the arena. Our mouse trap flipping device was capable of lifting opponents, decreasing their traction and giving us a better chance of pushing them out of the ring. In addition to mechanical strengths, our code and electrical design were simple and easy to understand. As a result, there was less room for coding and electrical error.

We did have a few weaknesses as well. Our mousetrap was not strong enough to flip our opponents as planned when triggered, and it was not triggered as easily as we had hoped. One possible improvement would be to a use a rat trap, which demonstrated to be more powerful and successful in flipping robots. Also, we only used one SONAR, so when the opponent was out of range, we had to turn a full 360° until it was in range again (as opposed to two SONARs which would only require a 180° turn). To solve this problem, we could have placed two SONARs on opposing sides of our robot to increase our detection range at all times.

SUGGESTIONS FOR IMPROVING COMPETITION

There are several possible modifications possible to improve the competition. One rule that could be implemented is to force all battles to begin with the robots placed back to back. Unless having two sonars, this would give a fair start to both teams if the robots spin in alternate directions. Another improvement would be that as soon as the battle starts everyone surrounding the ring, such as referees and team mates, immediately step at least three steps back. This would make a difference because in several of this year's battles robots were picking up the people surrounding them with the SONAR and causing their robots to disqualify on their own. Also the rule about not modifying your chassis and limited machining should be more enforced as several teams made modifications to reinforce their robots.

CONCLUSION

Our robot was not eliminated until Round 7 of the competition. It performed very well and was only defeated when it was flipped over by a wedge robot and when it was pushed out of the arena by a pretty evenly matched opponent. If given another chance we would have added another SONAR and used a stronger trap to flip the opponent with. We also could have used 4 motors instead of 2. This would have given us more power to push with.

APPENDIX

Full Code

```
// 99% Robot Code

// files to include
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>

// define variables
volatile int running=2;       //initially set robot to search mode
#define true 1 // defined b/c Atmega32 doesn't like
#define false 0 // booleans even though the compiler
uint16_t volatile SonarReturnTime; //Stores the current time reading
from the sonar
uint8_t volatile InPulse; //State machine variable
```

```
// initialize interrupts for QTIs
void initializeINT1();
void initializeINT0();

// ISR for front QTIs
ISR(SIG_INTERRUPT0){
TIMSK ^= _BV(OCIE0);
double a;
a=0;
while (a< 12000 ){
robotDirec(4);           // go backwards
a++;
}

running=2;               // search mode
TIMSK ^= _BV(OCIE0);
}

// ISR for middle QTI
ISR(SIG_INTERRUPT1){ /* the interrupt function*/
running = 0;           // stop
}

// declare robot directiong
void robotDirec(char phase){
if (phase==1){
              PORTB = 0b00001001; //robot spins
}
else if (phase==2){
              PORTB = 0b00000101; //robot moves forward
              }

else if (phase == 3){
              PORTB = 0b00000000; //robot stops
                          }

else if (phase == 4){
              PORTB = 0b00001010;     //robot moves backwards
}

}

//  Function that activates the sonar sensor.
void PulseSonar(void) {
TCCR1B|=_BV(CS12); // Prescalar is clock/256 (16 us per count)
cli(); // Clears interrupts temporarily
```

```
TIMSK&=~_BV(TICIE1);     // Disable input capture pin interrupt (ICP)
DDRD |=0x40; // Change D6 (7th pin) to output mode
PORTD |=0x40; // Turn on D6
_delay_us(5); // Enable Port D as input for 5us
PORTD &=~0x40; // Turn off D6
DDRD &=~0x40; // Change D6 to input mode
//Re-enable ICP (PD6 as input)
TIMSK|=_BV(TICIE1);  // Renable ICP
TCCR1B |=_BV(ICES1);     // Responds to rising edge initially
InPulse=1;       // Variable for state machine
sei();
}

// ISR for SONAR
ISR(SIG_INPUT_CAPTURE1) {
if(InPulse==1) {  // First state of machine (captured rising
 // edge, start of pulse in)
TCNT1 = 0;  // Start counting
TCCR1B &= ~_BV(ICES1); // Sets capture pin to respond to
// falling edge
InPulse = 2;
}
else if(InPulse==2) { // Next state of machine (captured falling
// edge, end of pulse in)
SonarReturnTime = ICR1;  // Reads from ICR1, which was equal
  // to TCNT1 when interrupt is thrown
if(SonarReturnTime < 250){

running=1;                // go forward in opponent in range

}
if(SonarReturnTime >= 250){

running =2;             // search is opponent not in range
}
InPulse = 0;
}
}

uint8_t volatile InPulse=0; // Declare variable from external
int volatile update=0; //Counter variable used in Timer0's ISR

// ISR for the timer which gets a new sonar reading.
ISR(SIG_OUTPUT_COMPARE0) {
update++;
 if(update==250) {
```

```
 PulseSonar();   // Read the sonar
 update=0;
}
}

// Initializes a timer to read the sonar sensor at a given frequency.
void init_ticker(void) {
TCNT0=0;
OCR0=250;
TCCR0=_BV(WGM01)|_BV(CS01)|_BV(CS00); // CTC on OCR0, clock/64
TIMSK|=_BV(OCIE0);
}

// MAIN FUNCTION
int main(void) {

DDRC = 0x00; // set port C as inputs

DDRA = 0xFF; // set port A as outputs

initializeINT1(); /* our initialize interrupt function*/
initializeINT0();

sei(); /* enable all interrupt functions*/

DDRD|=0x80;
PORTD|=0x20;
init_ticker();
sei();
PulseSonar();
DDRB = 0xFF;

int a;
a=0;
robotDirec(3);    // wait
while (a<=200 ){
_delay_ms(10);
a++;
}

while(running>0) {

if (running==1){
robotDirec(2);     //opponent not in range, search mode
}
else if (running == 2){
```

```
robotDirec(1);      // opponent in range, go forward
}
else if (running == 3){
robotDirec(4);     // front QTI triggered, move backwards
}
}
robotDirec(3);    // running set to 0, stop

PORTA = 0x00;
PORTA = 0x01;  //turn on LED
}

void initializeINT0(){
GICR |= _BV(INT0);
MCUCR |=_BV(ISC11);
MCUCSR = 0;
}

void initializeINT1(){ /* function to enable the interrupt*/
GICR |= _BV(INT1); /* Enable INT1*/
MCUCR |= _BV(ISC11); /* Trigger off falling edge (black to white)*/
MCUCSR = 0;
}
```