Trebuchet

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

Steven Theriault

TA: Uriel Rodriguez
Jason Plew
Instructor: A. A. Arroyo

December 10, 2002

Table of Contents

Abstract

Trebuchet is a model of a 13th century, French-built trebuchet. It searches for a castle and attempts to throw projectiles at it.

Executive Summary

Trebuchet is a simulation of a 13th century, French-built trebuchet. The trebuchet was an advancement on the catapult. It utilizes a counterweight and a sling to throw projectiles much farther and more accurately than its predecessor.

My robot trebuchet will have two wheels to move around and attempt to locate a castle. It does this by using two infrared (IR) detectors to locate a beacon located on the top of the castle.

Once the trebuchet has found the castle it will move forwards or backwards to the exact distance the projectile will be thrown. It uses sonar to do this.

To avoid obstacles while doing its function, it uses IR and bump switches.

Introduction

Trebuchet is an autonomous robot that replicates a 13th century trebuchet. Its job is to locate a castle and throw projectiles at it. In this report the trebuchet's platform, electronics, and behaviors are discussed. The castle's platform and electronics are also discussed.

Mobile Robot

The mobile robot is the trebuchet. It contains the electronics to move around, locate and avoid, and launch the projectile.

Integrated System

Trebuchet is based around the Atmel ATmega163. The bump switches and IR devices use the analog-to-digital converters, the sonar uses the input capture, the solenoid uses a port, and the servos use the output compare.

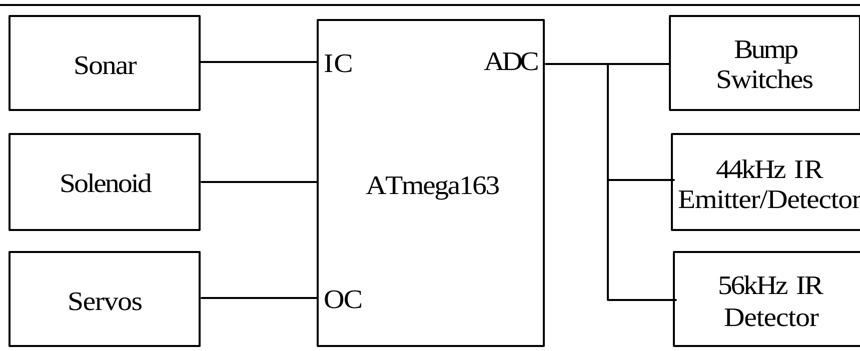

Figure 1: Block Diagram of the Trebuchet's Electronics

Development Board

A development board is used in the design to eliminate the need of creating a PCB for the robot. The development board was purchased from Progressive Resources and is the MegaAVR-Dev board.

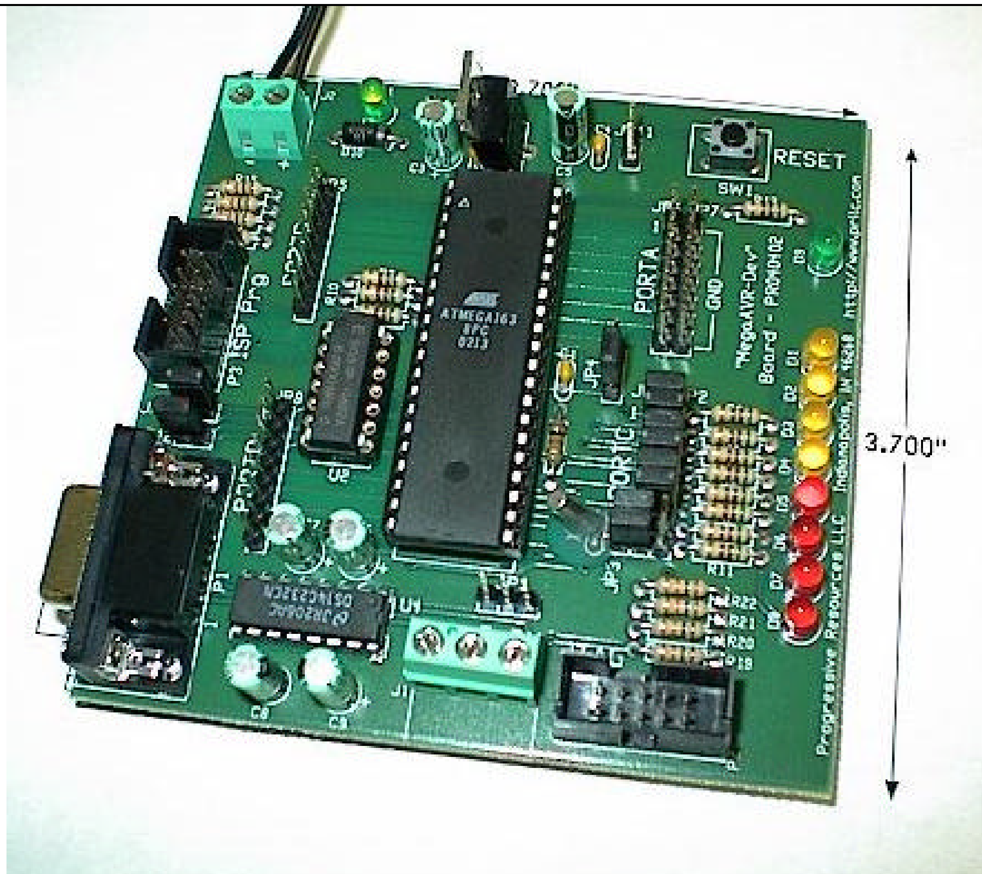

Figure 2: Progressive Resources MegaAVR-Dev Development Board

This board runs the microcontroller at 8Mhz. It has RS232 communication implemented on the board, so that was very beneficial in debugging. The LEDs seen on the right side of figure 2, were used for user feedback.

Platform

The trebuchet (Figure 3) stands about 9 ½ inches tall from the ground to the top of the A-Frames. If the pendulum is released then the trebuchet stands 16 inches.

The Body that contains the electronics and holds up the pendulum is 5x7 inches. The A-Frames are 6 inches tall and the pendulum is 9 ½ inches long.

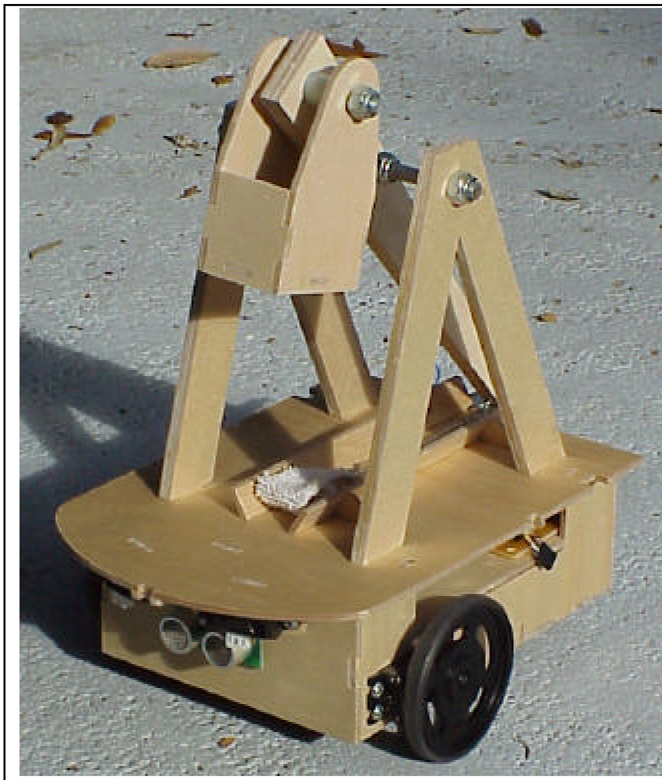

Figure 3: Picture of the Trebuchet

The bucket shown in Figure 3 is the counterweight for the trebuchet. When the pendulum is released, the counterweight pulls the sling, connected to the other end of the pendulum, and throws the projectile.

The sensor placement was well thought out. In the front, there are 2 IR detectors, 2 IR emitter/detectors, and the sonar (Figure 4).

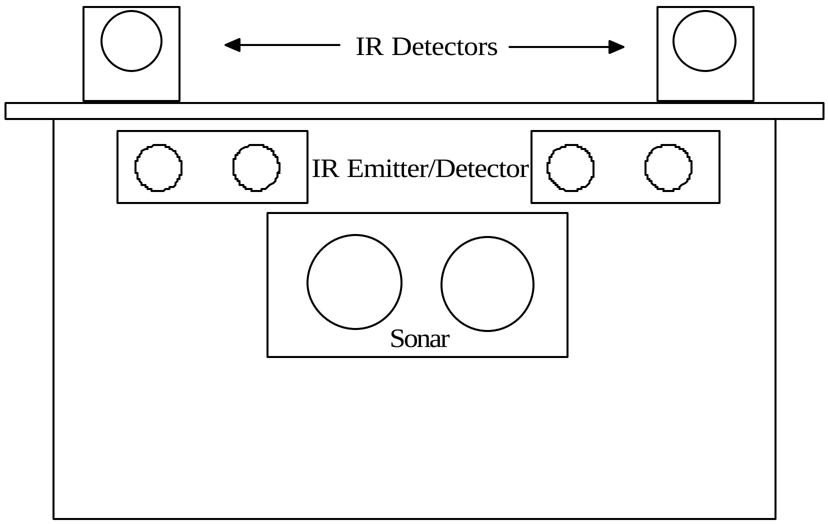

Figure 4: Front of Trebuchet

In the rear of the trebuchet are the solenoid and the third IR emitter/detector (Figure 5).

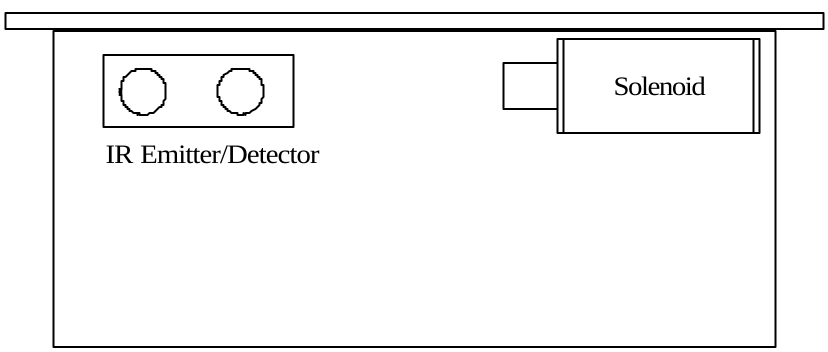

Figure 5: Rear of Trebuchet

Actuation

The Trebuchet uses two GWS servos for movement. They are hacked to allow continuous turning. The wheels where purchased from the Mark III robotics store, and fit well on the axle of the GWS servo. A furniture mover is used to hold the backend up.

Sensors

Bump Switches

The bump switches are used to avoid obstacles. They are triggered when the robot fails to detect the object with the IR modules and runs into the obstacle. The bump switches use a voltage divider network (Figure 6) to use the analog-to-digital converter on the microcontroller. Three resister values are used 10k, 22k, and 47k ohms.

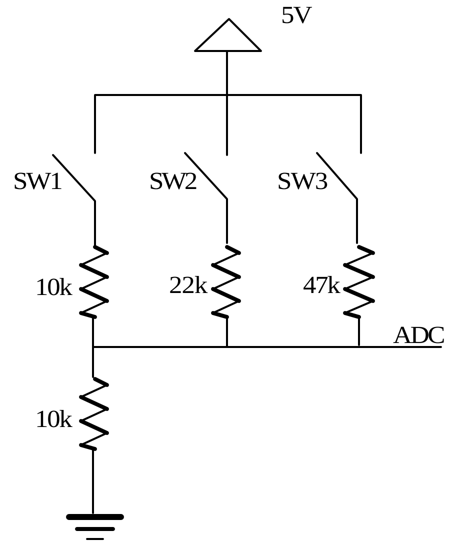

Figure 6: Bump Switch Resister Network

With SW1 closed, 2.5V is present on the ADC line, 3.4V with SW2 closed, and 4.1V with SW3 closed.

Sharp GP2D12

IR emitter/detectors (Figure 7) were used to locate obstacles that the robot may run into. Three Sharp GP2D12 modules were used, two on the front and one on the back. They are modulated at 44kHz and have a viewing angle of about 15 degrees from center. They give an analog signal back which is proportional to the distance to the object.

Figure 7: Sharp GP2D12

LITEON IR Detectors

Two LITEON IR detector modules are used to locate an IR beacon located on top of the castle and line up with it. They are modulated at 56kHz so that they do not interfere with the Sharp IR modules. They have been hacked to output an analog voltage rather than a digital voltage. The hack was provided by Michael Hattermann in Spring of 2002's IMDL.

The two IR modules are collimated with about 2 inches of black heat shrink tubing. This allows the trebuchet to line up very straight with the castle. It does this by turning towards the IR detector with the higher voltage which relates to where the IR beacon is located.

Devantech SRF04 Sonar

To find the correct distance from the castle, the robot will use a sonar system. The SRF04 ultrasonic range finder (Figure 8) from Devantech will be the sonar used. The specifications say that it is able to detect a 3 cm pole at 2 meters, with a range from 3 cm to 3 m.

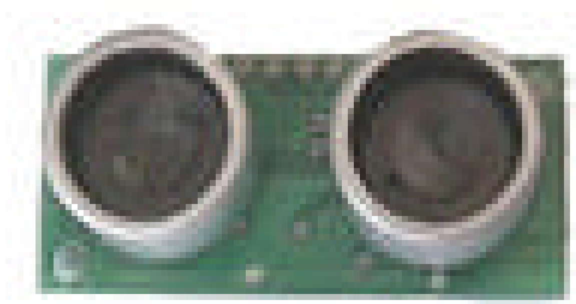

Figure 8: SRF04 Ultrasonic Range Finder

The sonar works by sending out a sound pulse at a frequency above the human hearing range. When the sound hits an object, the pulse is sent back to the sonar where is can be heard. The sonar measures the time it took for the sound to return to the sonar and can be used to calculate the distance to the object.

The SRF04 has four connections to use the sonar. Two of them are power and ground. There is an input line for initiating a sonar ping, and an output line to receive a pulse. The width of the pulse is determined length of time it takes for the sound to return to the sonar.

To initiate a sonar ping, bring the input line high for 10µs and then back low. This will send a sonic burst out (see Figure 9). After 100µs the output will go high and will remain high for 100µs to 18ms. If an echo is not detected, the sonar will time out after 18ms and the output line will go low. To initiate another pulse, the controller must wait 10ms.

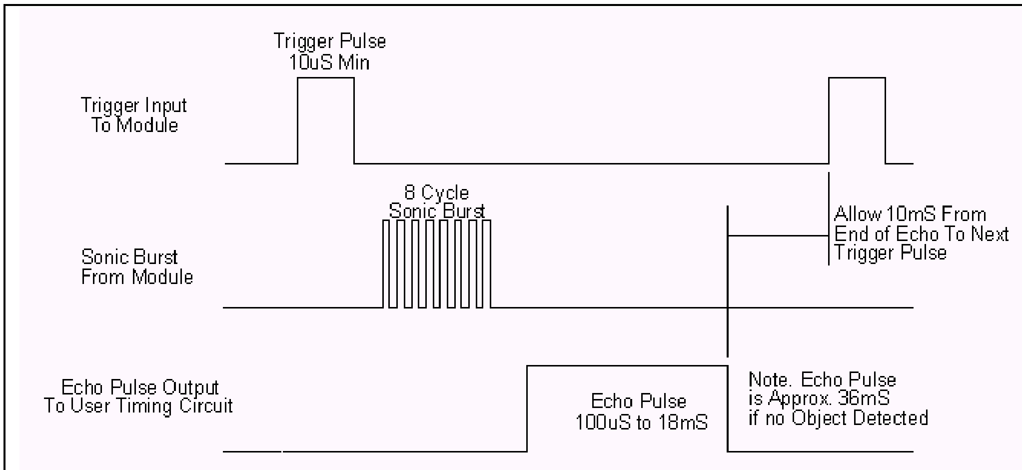

Figure 9: SRF04 Timing Diagram

The sonar has a resolution of about 2.5 inches and a useful range of 3 inches to 7 feet. After 7 feet the output is not steady, and an accurate reading cannot be obtained. Shown below (Figure 10) is a graph of the distance vs. the delay of the pulse.

Figure 10: Distance vs. Delay

Solenoid

A solenoid is used to release the pendulum thus releasing the projectile. The circuit for controlling the solenoid is optoisolated to eliminate noise near the microcontroller and other ICs. Below is the circuit used (Figure 11).

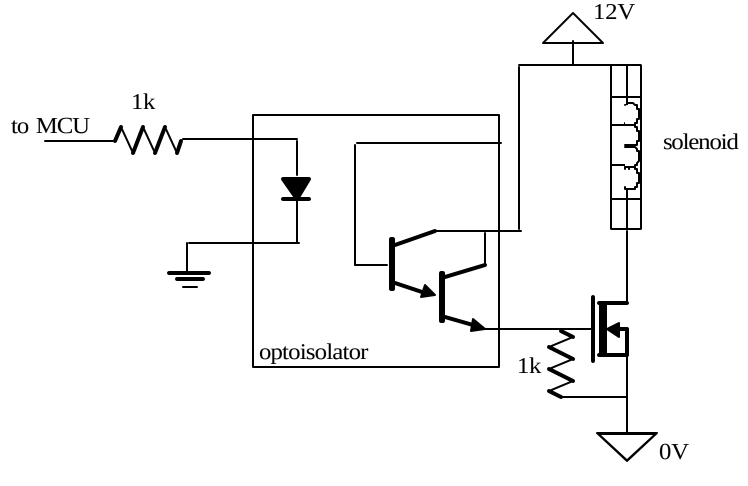

Figure 11: Circuit for Isolating the Solenoid

Stationary Castle

Platform

The castle is 3 ½ inches tall and 7x7 inches long. An IR beacon is located in the center to signal the trebuchet. The top of the castle is a bump switch to detect when a projectile has hit the castle.

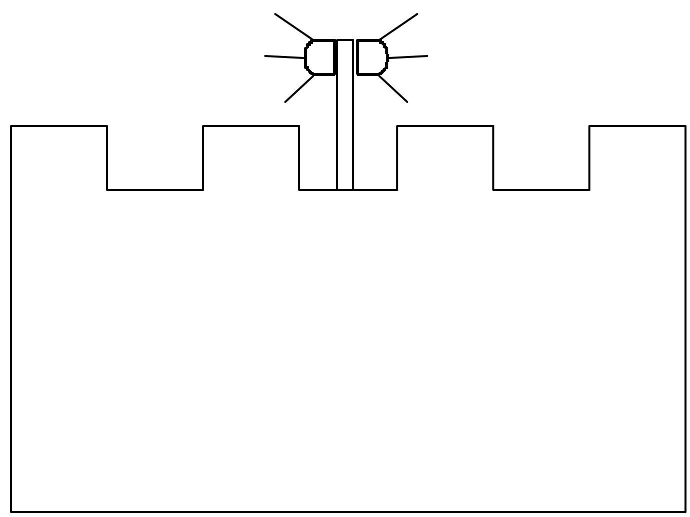

Figure 12: Sketch of the Castle

Electronics

The electronics on the castle is an IR beacon that is able to be turned on and off by switches. The top plate of the castle is a large bump switch (actually four bump switches), so when the top of the castle is hit, the beacon turns off.

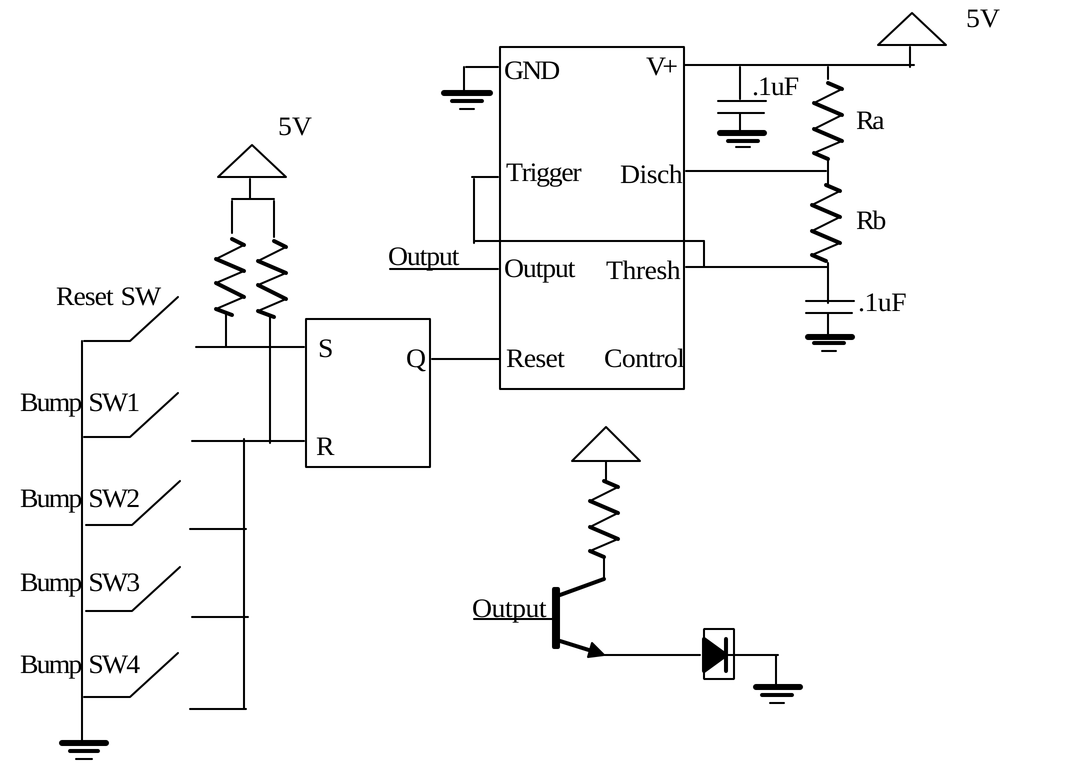

Figure 13: Schematic for Castle Electronics

In Figure 13, you can see how a 555 timer and an SR latched are used in combination to turn on and off the IR beacon. Ra and Rb are two leads from a potentiometer, so that it works as a variable resister. This allows me to use an oscilloscope to get exactly 56kHz. When the reset switch is pressed, the beacon is on, and when any of the bump switches are pressed, the beacon turns off.

Up to 8 IR LEDs can be used in parallel with my design, but for the most accuracy only one will be used.

Behaviors

The main goal of the trebuchet it to hit a castle with a projectile, but a lot is involved in doing so. Below is the software flow of the trebuchet (Figure 14).

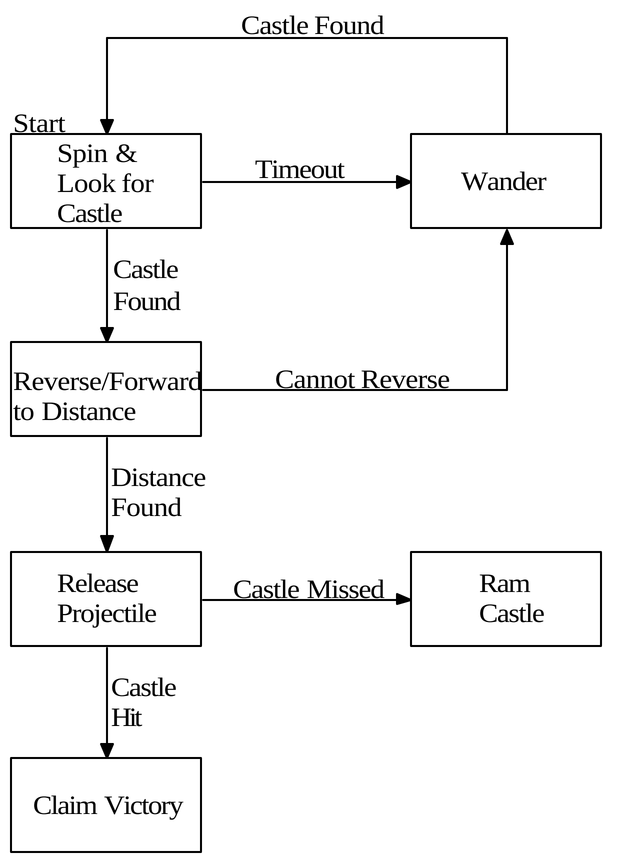

Figure 14: Software Flowchart

When the trebuchet starts, it waits for a user to press any of the bump switches on the trebuchet. Once a bump switch is pressed it begins by spinning and searching for the castle with the IR detectors. If it cannot find the castle and times out, it begins to wander. Wandering is just moving along the floor with random turns while looking for a signal from the castle.

Once it gets a signal, it again spins and tries to line up with the castle. When it is lined up with the castle, the trebuchet goes either forward or reverse to a correct distance to hit the castle and launches the projectile. It then looks for the beacon, if the beacon is still present, the trebuchet knows that it missed the castle and attempts to ram it. If the beacon is no longer running, it retreats.

Experiment Layouts and Results

All of the code was written in Atmel AVR assembly language, and the final codes takes a little more than 1kbyte of space.

Every sensor was tested as soon as it arrived at my house. I wrote assembly code for each, and then wrote other programs to integrate several components together. This made it very easy when it came to the end, because all of the parts were written and just needed to be put into one file.

I experimented with different distances to throw the projectile and different thresholds for certain values. It turns out that if I do not have enough weight in the counterweight bucket, the solenoid will not release the pendulum reliably, and having too much weight means that it will throw the projectile very far, but not very accurately.

At first the trebuchet did a terrible job lining up with the castle. At times it would be up to 90 degrees off. After much playing with the code, it turns out to be very accurate 90% of the time.

The most common problem with the trebuchet is throwing the projectile too long or too short. I expect that these difference are caused by the way I set up the pendulum and the sling each time I fire the trebuchet.

One thing that I have learned is that change is bad. Two days before the trebuchet was due, I decided to rebuild the castle, moving it from a bread board to a wooden castle and a perforated board. It turns out that even thought I created the castle the exact same way as it was on the bread board, it did not produce the same results. I spent a few nights not getting sleep trying to get my robot working again. Change that late in the game is just not good.

Documentation

- The LITEON hack was provided by Michael Hattermann's robot in the Spring of 2002.

- I would like to thank the T.A.s for their help in the lab.

- Thanks to Radio Shack, where the employees there now know my name.

Data Sheets

Devantech SRF04 sonar and Sharp GP2D12

http://www.junun.org/MarkIII/Store.jsp

LMC7805 and LMC555

http://www.national.com

Apendix A – Code for Trebuchet

```
;Trebuchet.asm
;code used to implement the behavior of Trebuchet
;
;Written by: Steven Theriault
;
;December 2002
;

.nolist
.include "C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\m163def.inc"

;register definitions
.def capturel=r1
.def captureh=r2
.def bump=r3
.def leftIR=r4
.def rightIR=r5
.def rearIR=r6
.def IRleft=r7
.def IRright=r8
.def timeout=r9

.def temp=r16
.def temp1=r17
.def temp2=r18
.def temp3=r19
.def subret=r20
.def subarg0=r21
.def rwsl=r22
.def rwsh=r23
.def lwsl=r24
.def lwsh=r25

;equates
.equ lwfh=$06
.equ lwfl=$27
.equ rwfh=$02
.equ rwfl=$A3

.equ lwrh=$02
.equ lwrl=$A3
.equ rwrh=$06
```

```
.equ rwrl=$27

.equ lwph=$04
.equ lwpl=$6f
.equ rwph=$04
.equ rwpl=$6f

;macros
.listmac
.macro outi @0,@1
        ldi temp,@1
        out @0,temp
                        .endmacro

.list


;interrupt vectors
.org 0
        jmp     main
.org $00c
        jmp T1OCA_ISR
.org $00e
        jmp T1OCB_ISR
.org $012
        jmp T0_OVR_ISR




;main routine
.org $24
main:
        outi SPH,high(ramend)           ; Stack Pointer Setup
        outi SPL,low(ramend)

;set data direction
        outi ddrc,$ff
        outi ddrb,$05
        outi DDRD,0b10110000            ;pins7,5,4 as outputs
        cbi portb,2                     ;clear portb pin 2


;init T1
        outi TCCR1B,0b10000010          ;prescaler clk/8
```

```
                                                ;enable input noise filter
                                                ;falling edge input capture
        outi TIMSK,0b00011000           ;enable interrupt for T1OCA and T1OCB

;set Timer 0 prescaler to 1024
        outi TCCR0,5

;init ADC
        sbi admux,adlar                 ;left adjust result

        sbi adcsr,aden          ;enable ADC

;wait for user to start by pressing any bump switch
waitbump:
        outi admux,$20
        sbi     adcsr,adsc              ;start A2D conversion
wait4bump:
        sbis adcsr,adif         ;wait until A2D conversion complete
        jmp wait4bump
        in temp,adch            ;move adc data to temp
        andi temp,$E0
        breq waitbump                   ;if zero then loop

        sei









spinInit:
;init values for wheels forward
        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwrh
        ldi lwsl,lwrl

;start wheel and enable interrupts
        outi TCCR1A,0b01010000          ;set to toggle pins T1OCA and T1OCB

;set T0 OVR interrupt
        outi TIMSK,0b00011001           ;set Timer 0 Overflow interrupt enable

        clr timeout
```

```
        ;user feedback
        ldi temp,$1
        com temp
        out portc,temp


spin:

        call wait100ms

;get a2d for IRdetLeft
        outi admux,$24
        sbi     adcsr,adsc                      ;start A2D conversion
waitADC4:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC4
        in IRleft,adch                          ;move adc data to bump

;get a2d for IRdetRight
        outi admux,$25
        sbi     adcsr,adsc                      ;start A2D conversion
waitADC5:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC5
        in IRright,adch                                 ;move adc data to bump


;if timeout goto wanderInit
        ldi temp,$F0
        cp timeout,temp
        brlo testIR

        outi TIMSK,0b00011000                   ;disable T0_OVR
        jmp wanderInit

testIR:
;if IRright && IRleft < $70, jmp mainloop
        ldi temp,$70
        cp IRleft,temp
        brsh greaterThanSixty

        cp IRright,temp
        brlo spin

greaterThanSixty:
```

```
;if IRleft == IRright, stop turning
        cp IRleft,IRright
        breq stop

;if IRright-6 < IRleft < IRright+6, stop turning
        ldi temp,$6
        sub IRright,temp                ;IRright - 6
        cp IRright,IRleft
        brsh alterDirection

        add IRright,temp                ;IRright-6+6+6=IRright+6
        add IRright,temp
        cp IRleft,IRright
        brsh alterDirection

;stop wheels and wait .5 sec and jump to toDistance
stop:
        ldi rwsh,rwph
        ldi rwsl,rwpl
        ldi lwsh,lwph
        ldi lwsl,lwpl
        call wait500ms                          ;wait 500ms
        outi TCCR1A,0b00000000          ;stop wheels
        jmp toDistance

;turn towards the higher IR reading
alterDirection:
        cp IRright,IRleft
        brsh turnRight

;turn left
        outi TCCR1A,0b01010000          ;start wheels
        ldi rwsh,rwfh+$100
        ldi rwsl,rwfl+$100
        ldi lwsh,lwrh+$100
        ldi lwsl,lwrl+$100
        jmp spin

;turn right
turnRight:
        outi TCCR1A,0b01010000          ;start wheels
        ldi rwsh,rwrh-$100
        ldi rwsl,rwrl-$100
        ldi lwsh,lwfh-$100
        ldi lwsl,lwfl-$100
```

```
	jmp spin

toDistance:
	;user feedback
	ldi temp,$3
	com temp
	out portc,temp

	outi TIMSK,0b00011000		;disable T0_OVR
	sbi PORTB,0			;initialize sonar ping

	ldi temp,20			;wait 10us
wait10us:
	dec temp
	brpl wait10us

	ldi temp,0b00100100		;clear overflow flag 1
	out TIFR,temp		;clear IC flag


	in capturel,TCNT1L		;store timer1 value
	in captureh,TCNT1H

	cbi PORTB,0			;start sonar ping

;while sonar pinging, check for room behind robot
;get a2d for rearIR
	outi admux,$23
	sbi	adcsr,adsc			;start A2D conversion
waitADC3:
	sbis adcsr,adif		;wait until A2D conversion complete
	jmp waitADC3
	in rearIR,adch			;move adc data to bump

;wait IC flag
chk_ic_flg:
	in temp,TIFR
	sbrs temp,ICF1			;wait for IC flag
	rjmp chk_ic_flg
```

```
        in temp,ICR1l               ;load capture registers
        in temp1,ICR1h

        cp temp,capturel            ;test original time with new capture time
        cpc temp1,captureh

        brsh captSub                ;if new>original branch

;$FFFF - original time + new time
        ldi temp2,$ff               ;use temp3:temp2 for accumulator
        ldi temp3,$ff

        sub temp2,capturel          ;$FFFF - original time
        sbc temp3,captureh

        add temp2,temp                      ;     + new time
        adc temp3,temp1

        mov capturel,temp2                  ;move acculmulator back to
captureh/capturel
        mov captureh,temp3

        jmp wait10m

captSub:
        sub temp,capturel           ;subtract 16bit time
        sbc temp1,captureh

        mov capturel,temp                   ;move acculmulator back to
captureh/capturel
        mov captureh,temp1


wait10m:
        ldi temp,$4F                ;wait 10ms
wait10ms:
        ldi temp1,$FF
wait10msa:
        dec temp1
        brne wait10msa
        dec temp
        brne wait10ms

;test if captureh < $1D && >1B
        mov temp,captureh
```

```
        cpi temp,$1C
        brlo reverse

        cpi temp,$1D
        brsh forward

;stop wheels and shoot
        ldi rwsh,rwph
        ldi rwsl,rwpl
        ldi lwsh,lwph
        ldi lwsl,lwpl

        call wait100ms

        outi TCCR1A,0b00000000          ;stop wheels

        sbi portb,2                     ;set portb pin 2

;user feedback release
        ldi temp,$4
        com temp
        out portc,temp


wait1000ms:
        ldi temp,$10
wait1000msa:
        ldi temp1,$FF
wait1000msb:
        ldi temp2,$FF
wait1000msc:
        dec temp2
        brne wait1000msc
        dec temp1
        brne wait1000msb
        dec temp
        brne wait1000msa

        cbi portb,2                     ;clear portb pin 2

        jmp detectInit

reverse:
        outi TCCR1A,0b01010000          ;start wheels
        ldi rwsh,rwrh
        ldi rwsl,rwrl
```

```
        ldi lwsh,lwrh
        ldi lwsl,lwrl

;if no more room in rear goto wanderInit
        mov temp,rearIR
        cpi temp,$70
        brsh wanderInit

        jmp toDistance

forward:
        outi TCCR1A,0b01010000          ;start wheels
        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwfh
        ldi lwsl,lwfl
        jmp toDistance

wanderInit:
;go forward
        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwfh
        ldi lwsl,lwfl

        outi TCCR1A,0b01010000          ;set to toggle pins T1OCA and T1OCB

;user feedback
        ldi temp,$2
        com temp
        out portc,temp

wander:
;get a2d for bump
        outi admux,$20
        sbi     adcsr,adsc              ;start A2D conversion
waitADC0:
        sbis adcsr,adif         ;wait until A2D conversion complete
        jmp waitADC0
```

```
        in bump,adch                        ;move adc data to bump

;get a2d for left IR
        outi admux,$21
        sbi     adcsr,adsc                  ;start A2D conversion
waitADC1:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC1
        in leftIR,adch                      ;move adc data to leftIR

;get a2d for right IR
        outi admux,$22
        sbi     adcsr,adsc                  ;start A2D conversion
waitADC2:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC2
        in rightIR,adch                             ;move adc data to rightIR

;get a2d for IRdetLeft
        outi admux,$24
        sbi     adcsr,adsc                  ;start A2D conversion
waitADC4a:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC4a
        in IRleft,adch                      ;move adc data to bump

;get a2d for IRdetRight
        outi admux,$25
        sbi     adcsr,adsc                  ;start A2D conversion
waitADC5a:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC5a
        in IRright,adch                             ;move adc data to bump


;if castle found, goto spinInit
        ldi temp,$70
        cp IRleft,temp
        brsh jumpSpin

        cp IRright,temp
        brlo testbump

jumpSpin:
        jmp spinInit
```

```
;test sensors to determine obstance avoidance
testbump:
;if $40 < bump > $60, then reverse and turn randomly
        mov temp,bump
        cpi temp,$40            ;is bump < $40?
        brlo testLeftAndRightIR
        cpi temp,$60            ;is bump > $60
        brsh testLeftAndRightIR

        call reverseAndRandomTurn
        jmp wander

testLeftAndRightIR:
;if LeftIR > $60 and RightIR > $60, then reverse and turn randomly
        mov temp,LeftIR
        cpi temp,$60            ;is LeftIR < $60
        brlo testRightIR
        mov temp,RightIR
        cpi temp,$60            ;is RightIR < $60
        brlo testRightIR

        call reverseAndRandomTurn
        jmp wander

testRightIR:
;if RightIR > $60, then turn left
        mov temp,RightIR
        cpi temp,$60            ;is RightIR < $60
        brlo testLeftIR

        call turnLeftMethod
        jmp wander

testLeftIR:
;if LeftIR > $60, then turn right
        mov temp,LeftIR
        cpi temp,$60            ;is LeftIR < $60
        brlo default

        call turnRightMethod
        jmp wander

default:
;no sensor readings, wait ~100ms
        call wait100ms
```

```
	jmp wander

DetectInit:
;set T0 OVR interrupt
	outi TIMSK,0b00011001		;set Timer 0 Overflow interrupt enable

	clr timeout

	;user feedback
	ldi temp,$1
	com temp
	out portc,temp

Detect:
	call wait100ms

;if timeout goto ram
	ldi temp,$50
	cp timeout,temp
	brlo againDetect

	outi TIMSK,0b00011000		;disable T0_OVR
	jmp ram

againDetect:
	;get a2d for IRdetLeft
	outi admux,$24
	sbi	adcsr,adsc		;start A2D conversion
waitADC4b:
	sbis adcsr,adif		;wait until A2D conversion complete
	jmp waitADC4b
	in IRleft,adch			;move adc data to bump

;get a2d for IRdetRight
	outi admux,$25
```

```
	sbi	adcsr,adsc			;start A2D conversion
waitADC5b:
	sbis adcsr,adif			;wait until A2D conversion complete
	jmp waitADC5b
	in IRright,adch				;move adc data to bump

;if IRright or IRleft > $58, retreat
	ldi temp,$58
	cp IRright,temp
	brlo retreat

	cp IRleft,temp
	brlo retreat

	jmp detect

retreat:
;init values for wheels reverse
	ldi rwsh,rwrh
	ldi rwsl,rwrl
	ldi lwsh,lwrh
	ldi lwsl,lwrl

;start wheel and enable interrupts
	outi TCCR1A,0b01010000		;set to toggle pins T1OCA and T1OCB

retreata:
	call wait100ms


;get a2d for rearIR
	outi admux,$23
	sbi	adcsr,adsc			;start A2D conversion
waitADC3a:
	sbis adcsr,adif			;wait until A2D conversion complete
	jmp waitADC3a
	in rearIR,adch				;move adc data to bump
```

```
;if rearIR > 60, stop
        ldi temp,$60
        cp rearIR,temp
        brlo retreata

        jmp fullspeed
```

```
ram:
;init values for wheels reverse
        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwfh
        ldi lwsl,lwfl

;start wheel and enable interrupts
        outi TCCR1A,0b01010000          ;set to toggle pins T1OCA and T1OCB

rama:
        call wait100ms


;get a2d for left IR
        outi admux,$21
        sbi     adcsr,adsc              ;start A2D conversion
waitADC1a:
        sbis adcsr,adif             ;wait until A2D conversion complete
        jmp waitADC1a
        in leftIR,adch                  ;move adc data to leftIR

;get a2d for right IR
        outi admux,$22
        sbi     adcsr,adsc              ;start A2D conversion
```

```
waitADC2a:
        sbis adcsr,adif                 ;wait until A2D conversion complete
        jmp waitADC2a
        in rightIR,adch                                 ;move adc data to rightIR

;if rightIR or leftIR > $60, goto fullspeed
        ldi temp,$60
        cp rightIR,temp
        brsh fullspeed

        cp leftIR,temp
        brlo rama

fullspeed:
        call wait100ms


        outi TCCR1A,0b00000000          ;set to toggle pins T1OCA and T1OCB

        jmp main








reverseAndRandomTurn:
        ldi rwsh,rwrh
        ldi rwsl,rwrl
        ldi lwsh,lwrh
        ldi lwsl,lwrl

        ldi temp,$20
wait2sa:
        ldi temp1,$FF
wait2sb:
        ldi temp2,$FF
wait2sc:
        dec temp2
        brne wait2sc
        dec temp1
        brne wait2sb
```

```
        dec temp
        brne wait2sa

        in temp,TCNT1L
        andi temp,$01
        brne goRight
        call turnLeftMethod
        call turnLeftMethod
        ret
goRight:
        call turnRightMethod
        call turnRightMethod
        ret



turnLeftMethod:
        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwrh
        ldi lwsl,lwrl

        ldi temp,$19
wait1sa:
        ldi temp1,$FF
wait1sb:
        ldi temp2,$FF
wait1sc:
        dec temp2
        brne wait1sc
        dec temp1
        brne wait1sb
        dec temp
        brne wait1sa

        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwfh
        ldi lwsl,lwfl
        ret



turnRightMethod:
```

```
        ldi rwsh,rwrh
        ldi rwsl,rwrl
        ldi lwsh,lwfh
        ldi lwsl,lwfl

        ldi temp,$19
wait1sd:
        ldi temp1,$FF
wait1se:
        ldi temp2,$FF
wait1sf:
        dec temp2
        brne wait1sf
        dec temp1
        brne wait1se
        dec temp
        brne wait1sd

        ldi rwsh,rwfh
        ldi rwsl,rwfl
        ldi lwsh,lwfh
        ldi lwsl,lwfl
        ret
```

```
;method
;wait 100ms
wait100ms:
        ldi temp,$03
wait100msa:
        ldi temp1,$FF
wait100msb:
        ldi temp2,$FF
wait100msc:
        dec temp2
        brne wait100msc
        dec temp1
        brne wait100msb
        dec temp
        brne wait100msa
```

```
	ret

;method
;wait 500ms
wait500ms:
	ldi temp,$38
wait500msa:
	ldi temp1,$FF
wait500msb:
	ldi temp2,$FF
wait500msc:
	dec temp2
	brne wait500msc
	dec temp1
	brne wait500msb
	dec temp
	brne wait500msa

	ret



;Interrupt Service Routine to place a correct frequency on the port D pin5
;the period is %3A98 cycles at clk/8
;The duty cycle is determined by rwsh and rwsl
;high time is currentTime+rwsh:rwsl
;low time is currectTime-$3a98+rwsh:rwsl
T1OCA_ISR:
	push temp				;push temp to save contents
	in temp,SREG			;push SREG to save contents
	push temp
	push temp1
	push temp2

	in temp,PIND			;check value of toggle pin
	andi temp,0b00100000
	breq pd5low				;branch if port D pin 5 is low

	in temp,OCR1AL				;OCR1A = OCR1A + right wheel speed
	in temp1,OCR1AH
	add temp,rwsl
	adc temp1,rwsh
	out OCR1AH,temp1			;must write high byte first
	out OCR1AL,temp
	jmp END_T1OCA_ISR
```

```
pd5low:                                         ;pin 5 is low
                                                ;OCR1A = OCR1A + 15000 - right
wheel speed
        in temp,OCR1AL
        in temp1,OCR1AH

        ldi temp2,$98
        add temp,temp2
        ldi temp2,$3A
        adc temp1,temp2

        sub temp,rwsl
        sbc temp1,rwsh

        out OCR1AH,temp1                ;must write high byte first
        out OCR1AL,temp


END_T1OCA_ISR:
        pop temp2
        pop temp1
        pop temp                        ;restore register values
        out SREG,temp
        pop temp

        reti                            ;return from interrupt



;Interrupt Service Routine to place a correct frequency on the port D pin4
;the period is %3A98 cycles at clk/8
;The duty cycle is determined by lwsh and lwsl
;high time is currentTime+lwsh:lwsl
;low time is currectTime-$3a98+lwsh:lwsl
T1OCB_ISR:
        push temp                       ;push temp to save contents
        in temp,SREG                ;push SREG to save contents
        push temp
        push temp1
        push temp2

        in temp,PIND                ;check value of toggle pin
        andi temp,0b00010000
        breq pd4low                     ;branch if port D pin 5 is low
```

```
        in temp,OCR1BL                  ;OCR1A = OCR1A + right wheel speed
        in temp1,OCR1BH
        add temp,lwsl
        adc temp1,lwsh
        out OCR1BH,temp1                ;must write high byte first
        out OCR1BL,temp
        jmp END_T1OCB_ISR

pd4low:                                         ;pin 5 is low
                                                ;OCR1A = OCR1A + 15000 - right
wheel speed
        in temp,OCR1BL
        in temp1,OCR1BH

        ldi temp2,$98
        add temp,temp2
        ldi temp2,$3A
        adc temp1,temp2

        sub temp,lwsl
        sbc temp1,lwsh

        out OCR1BH,temp1                ;must write high byte first
        out OCR1BL,temp


END_T1OCB_ISR:
        pop temp2                       ;restore register values
        pop temp1
        pop temp
        out SREG,temp
        pop temp

        reti                            ;return from interrupt



;interrupt service routine to overflow every so often
T0_OVR_ISR:
        push temp                       ;push temp to save contents
        in temp,SREG            ;push SREG to save contents
        push temp

        inc timeout

        pop temp                        ;restore register values
```

```
out SREG,temp
pop temp

reti                                  ;return from interrupt
```