

Shooter

EEL 5666
Intelligent Machines Design Laboratory
Written Report

Jason Metzenthin
April 15, 2003

Table of Contents

Abstract	3
Executive Summary	4
Introduction	5
Integrated System	6
Mobile Platform	7
Actuation	9
Sensors	12
IR beacon	13
IR beacon Detector	14
Sonar	14
Sharp IR	15
Bump	16
Behaviors	17
Experimental Layout and Results	18
Conclusion	19
Documentation	20
Appendices	21

Abstract

Shooter's purpose is to provide entertainment for spectators as they watch him attempt to shoot three balls successfully into his basket. One ball is loaded and then shoot at a time and it gets repeated. Afterwards Shooter wonders off disinterested avoiding bumping into anything.

Executive Summary

Shooter's intelligence is the ATEML mega 323 chip on a Progressive board. The programming was written in C++. With this intelligence Shooter locates his basketball hoops, lines up with it, then loads a ball, shoots the ball, loads a second ball, shoots, loads a third ball and then avoids obstacles while he is wondering around disinterested.

Shooter has two high torque servo-powered wheels that are both controlled by PWM channels. They are set close to the center of Shooter so that he would have a tight turn radius. Three rubber casters are placed on the bottom of the back-end that Shooter rests on.

The key sensors for guiding the movement of Shooter are the three IR sensors located in the right, center and left. They reliably give good results and near similar values from one to another.

The basketball hoop has an IR beacon consisting of a circuit that oscillates 3 IR LEDs near 56.8kHz. This signal is detected by a IR 56.8kHz detector. Shooter rotates in place till he sees the hoop then starts loading and shooting.

A set of extension springs is what propels the ball towards the basketball hoop. The springs are pulled back by a grouping of two servos. One that blocks the shooting arm and another that pulls back the shooting arm holding in back till the blocking lever is removed. A lock system is used to make sure that one and only one ball drop into the shooting arm's basket. The shooting arm is blocked at approximately a 60-degree angle to ensure the shoot ball has an arch to it. Shooter's typical range for making baskets is around 3 feet, varying due to a couple of factors. Spring wear and tear, servo wear and tear and amount of charge on the batteries.

Introduction

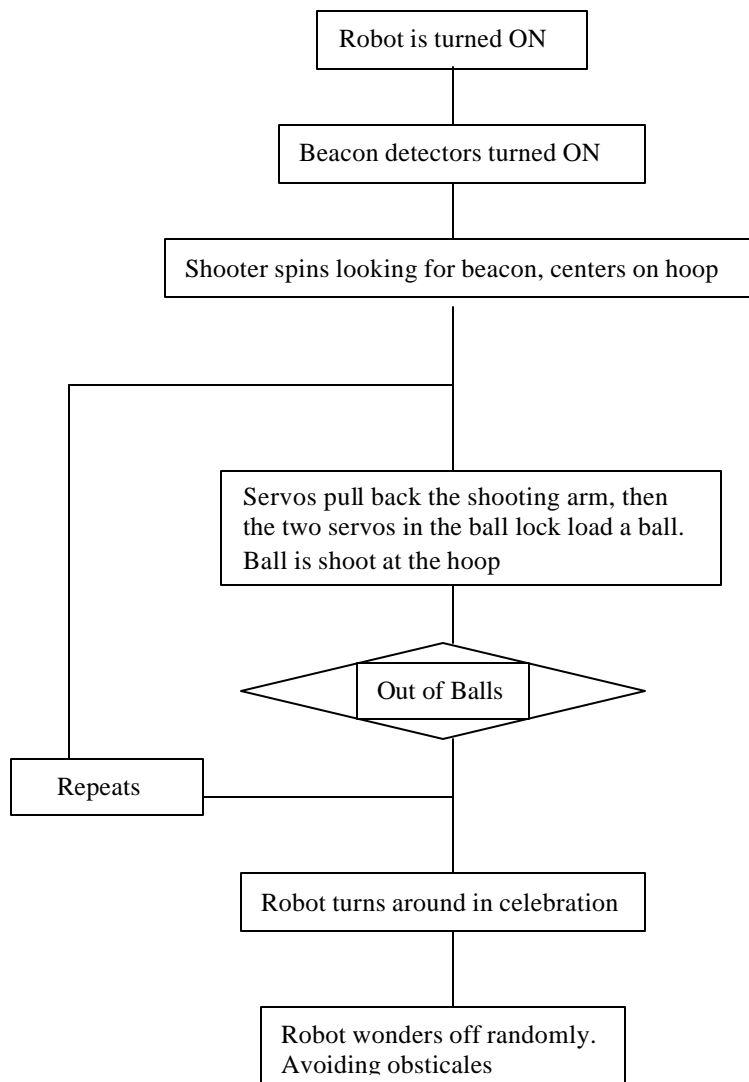
Shooter works in tandem with his basketball hoop. Shooter finds his hoop and then attempts to score. To accomplish this the robot uses six servos and four sensors. The robot's movement is done using its two wheels. Only using two wheels allows for a simplified design and tighter turn radius. Since the robot is using IR, it is important to limit the robot's exposure to IR so it is best to be used only indoors where it is sheltered from the sun's rays.

MAIN BODY

Integrated System

The robot design is centered around the ATMELE micro-controller on the progressive board, which controls all the functionality of the robot. Below is a flowchart of the program of Shooter.

Figure 1 – Flowchart of Actions

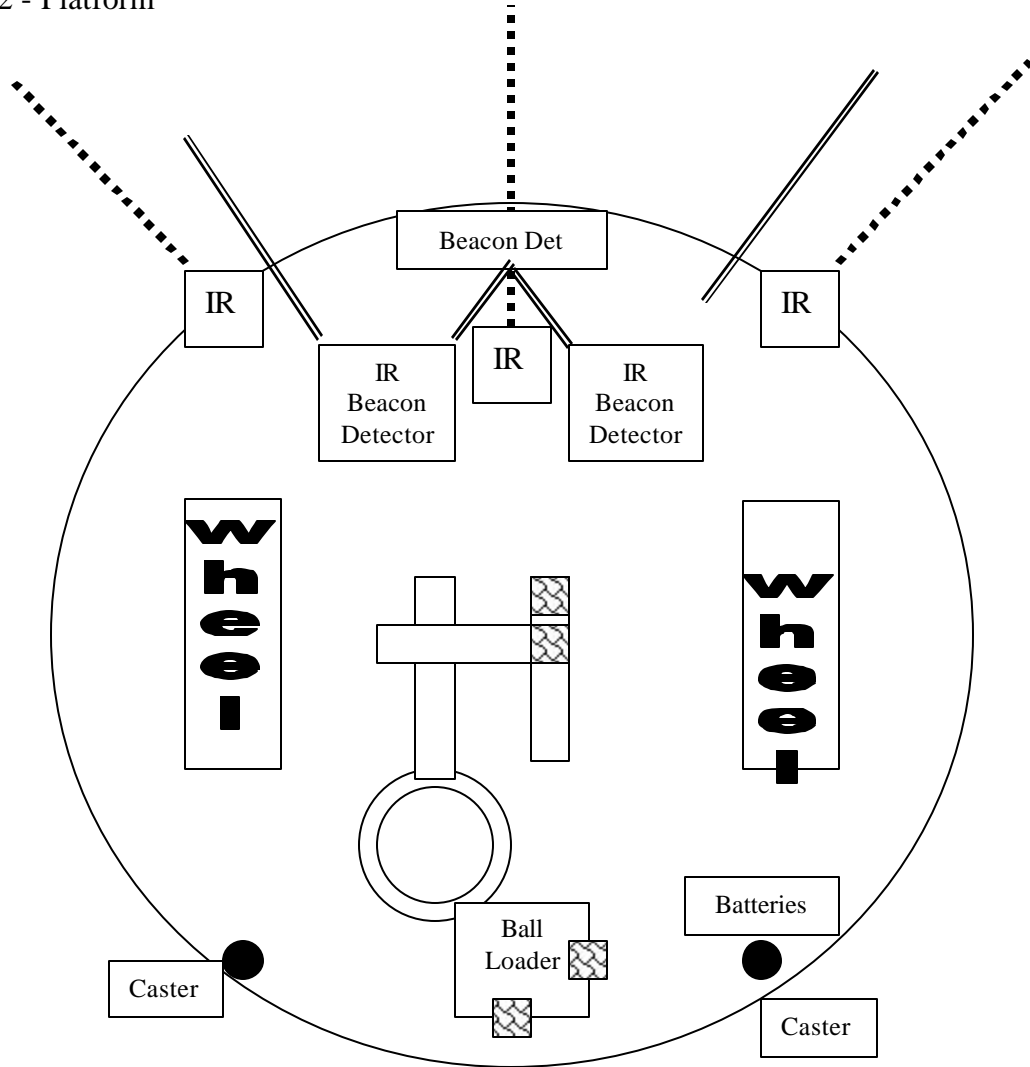



Mobile Platform

The objective of the platform was to be as lightweight as possible since it was going to be large but still powered by servos and not motors. To this end the platform is mainly composed of wood with the exception of the PVC pipe for the ball loader and the metal L-brackets used to reinforce the structure. The wooden levers that control the lock system are also made out of wood. The shooting arm was designed to be strong so that it would not bend or break when being pulled back. The spring holder that the spring was stretched from was designed in a manner that the whole platform would reinforce it. At the end of the shooting arm a wooden basket was constructed to hold the ball perfectly. A set of springs is attached to the shooting arm, which propels the shooting arm forward so that the ball is shoot at the basketball. Parts for platform were found at Injection Molded Wheels \$6.00 from Mark III Robot Store, Wood from lab and PVC pipe for \$4.98 from Lowe's Hardware Store.

I was worried about stress and strain on the structure of Shooter that I made the mistake to make the internal workings completely enclosed. This made getting to the reset button and connections a hassle that could have been easily avoided. Another near mistake I almost made was making my robot too heavy. Shooter no longer moves much on carpet, but still moves on tile.

Figure 2 - Platform



Servo = 

Actuation

Shooter has six servos that are used. There are three applications for these servos: movement via its wheels, the loading mechanism and lastly the shooting mechanism.

GWS servos were used and purchased from Mark III Robot Store. High torque (S03 TXF) for \$10.50 each and normal servos (S03N) for \$10.50 each were used.

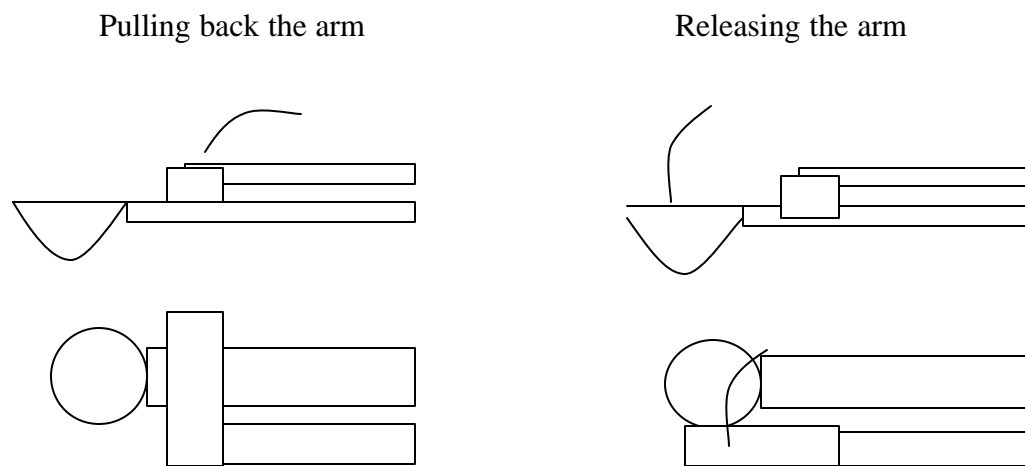
Item	Size (mm)	Weight (g/oz)	Speed (sec/60)	Torque (kg*cm/cz*in)
S03N	39.5x20.0x39.6	41/1.44	0.23	2.40/47
S03 TXF	39.5x20.0x39.6	46/1.62	0.21	5.00/69

The wheels are driven by two hacked TXF servos. This allows for a simplistic design in comparison to using a DC motor. This was also the suggested course of action for driving any IMDL robot's wheels at the start of the course. A PWM pin controls each of these hacked servos. Servos are easy to deal with and do not take too much effort to get working and programmed, which is why they were chosen.

The arming mechanism employs two servos. The first servo is a TXF and pulls back the shooting arm, which stretches the extension springs. This servo operates in a 90-degree range of motion, so a servo was ideal. Extension springs were employed since they provided the most force for launching balls. A compression spring has a narrower range of motion, is typically harder to compress than an extension is to pull and is more difficult to position for a ball shooting design. The second is a S03N servo that is used in releasing the shooting arm. It blocks and then slides away to let free the shooting arm. A single inexpensive servo would not be powerful enough to through a ball, so that design

concept was disregarded early on in the course. A single shoot robot could have used a string spool and/or gear system to pull back the shooting arm. Or an even more simple means would to have the arm manually loaded before the robot would be turned on and the robot should having to release the mechanism. A single timer controls these two and the two servos used in the ball loader. This code is based on what Michael Collins had written to run all the servos on his robot.

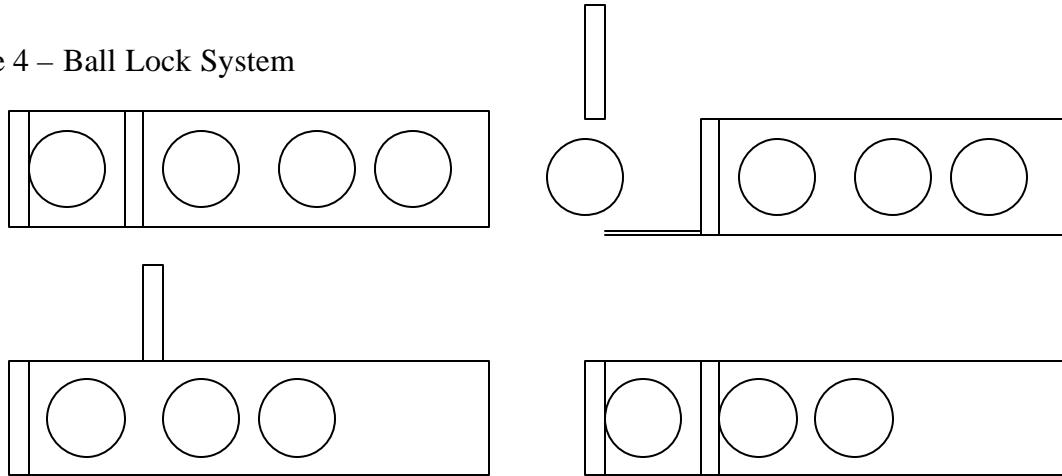
Figure 3 – Shooting arm



The last two servos are used in the loading process. They create a lock system making sure that only one ball gets loaded at a time. A more clever design that only needed one servo could have been constructed, but would have had other issues incorporated with it. Most of the concepts I came up with would have required more space and weight than I desired to place on my robot. How the lock system works is that both levers are closed. Then the bottom servo opens up the bottom lever allowing a ball to drop. It then closes. After that is done the top lever is opened and another ball falls through. After waiting for the balls to refill the top lever closes pushing up any extra

balls so that only one ball remains below it. The bottom lever is powered by a TXF servo and the top lever is powered by a S03N servo.

Figure 4 – Ball Lock System



When I initially ordered my servos I ordered two extra ones figuring something might well go wrong. Early on I burned out one servo when my power supply stopped producing 5 V and instead produced 12 V. Another servo I wore out by straining it too much in the tasks I gave it to operate.

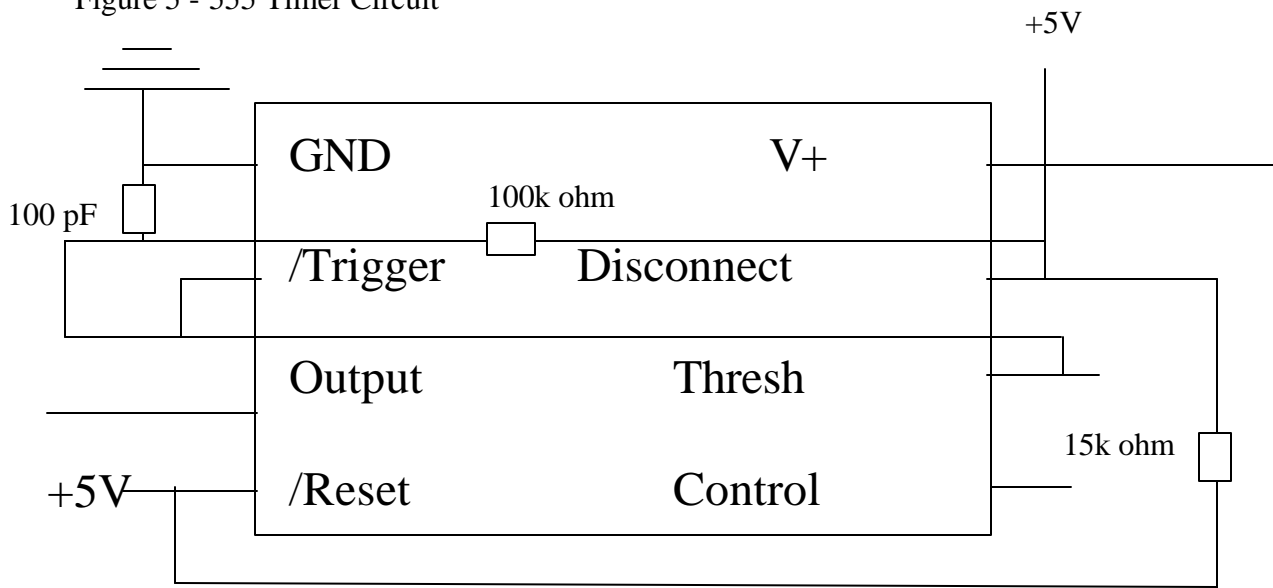
Sensors

There are electronics on both on the basketball hoop and on the robot that will work together to accomplish the overall goal of the robot's design. There is an IR beacon on the basketball hoop that generates an IR signal at a frequency of 55kHz allowing Shooter to find it.

IR beacon (basketball hoop)

The IR beacon oscillates near 56k so as not to interfere with the obstacle avoidance system. A 555 timer has been used to generate the frequency. The output was then hooked up to a Darlington transistor and then to three IR LEDs with shrink-wrap around them to focus their signal, which allows for only one detector to be needed. All the parts were from the lab except the Darlington transistor that was purchased at Radioshack for a few dollars. Below is the circuit the timer circuit I used, which was figured out with the help from Kyle Tripician and Steven Vanderploeg.

Figure 5 - 555 Timer Circuit



To achieve the above circuit took a large amount of time and effort. I looked at various websites and tried to use a couple of formulas. My suggestion to any student reading this is to reuse an already completed circuit since they are possibilities in past reports already that are known to work.

IR detector for the beacon (robot)

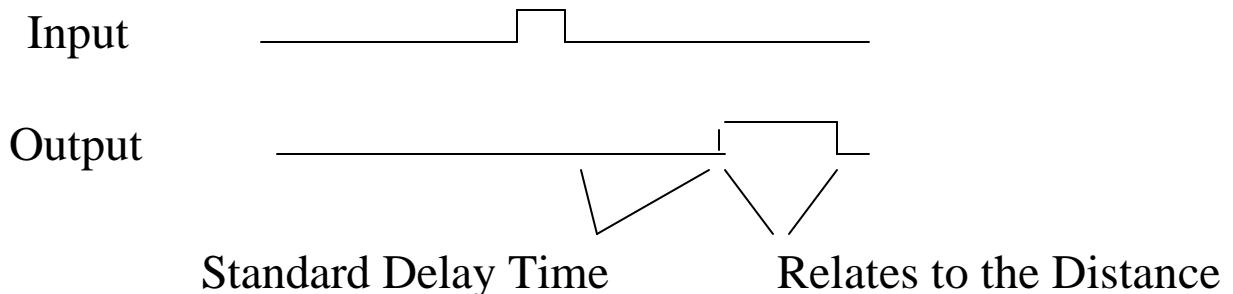
Shooter uses a hacked IR detector to find and then center on the basketball hoop. It was hacked following the instructions given by Michael Hattermann http://www.mil.ufl.edu/imdl/papers/IMDL_Report_Spring_02/michael_hatterman/hacked_ir.pdf. Shooter knows that he is centered on the hoop, because the IR's are in a directed cone that keeps the signal to a small area, so it is a matter of finding the analog voltage spike. The detector was purchased from Jameco for \$1.95 its Mfg Ref is #LITEONLTM9034-2. Hacking and using these was simple enough, but I did run into problems because of my Analog to Digital conversion code.

Sonar receiver/transmitter combination (robot)

The sonar receiver/transmitter combination was to be used for finding the range to the basketball hoop. This would have been done by a sonar signal being bounced of the large base of the basketball hoop and finding the time it takes to return back. From this distance the position of the shooting arm is going to be determined. The sonar I tried to get working with my robot was purchased from Acroname for \$33. I was unable to achieve a desired result from the sonar sensor and moved on to other alternatives, especially since the sonar was a redundant system.

Beam detection is roughly a 30 degree cone
Minimal Detection Range = 30 cm
Maximal Detection Range = 3 m

Figure 6 – Sonar



IR range detectors for obstacle avoidance (robot)

Three IR detectors and emitters are used to avoid bumping into obstacles in the front or side of the robot. The IR signal sent out will be detected on its return and from that a distance will be determined. When an obstacle gets too close the robot will turn to avoid it. Purchased from digikey for \$10.58 each.

Figure 7 – Sharp IR

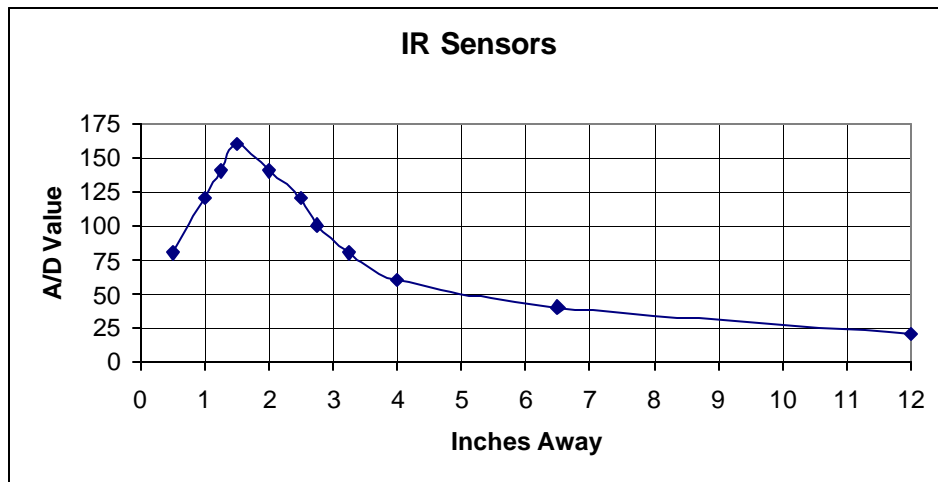
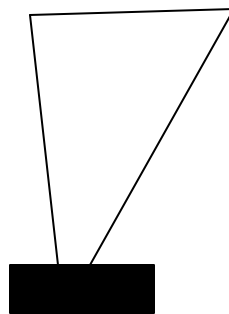


Table 1 – Sharp IR Range Finders

Inches	IR value
0.50	80
1.00	120
1.25	140
1.50	160
2.00	140
2.50	120
2.75	100
3.25	80
4.00	60
6.50	40
12.00	20



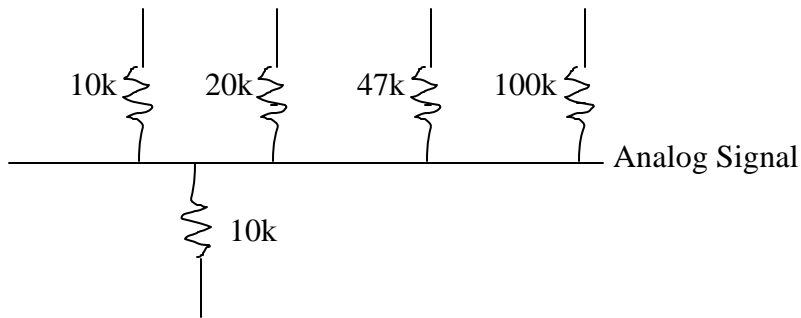
Covers about a 30 degree arc

The sharp detectors is linear to about 1.5 inches, then it is no longer linear. Either region can be used to detect obstacles. Also the detector does not have a very wide detection cone.

Bump switches for obstacle avoidance (robot)

Four bump switches were used, one on each side of the robot. The purpose of these sensors is to act like a backup encase the IR does not see an obstacle. Depending on the voltage inputted the robot will determine which switch(s) are pressed. I ran out of time to reattach my bump switches, but they work similar to all the other A/D sensors.

Figure 8 – Bump Switches



	10k (Left)	20k (Center)	47k (Right)	100k (Back)
10k (Left)	115	135-140	125-130	115-120
20k (Center)	135-140	76	95-100	90-95
47k (Right)	125-130	95-100	40	50-55
100k (Back)	115-120	90-95	50-55	20

Behaviors

The robot will have the following behaviors:

- The first behavior is that the basketball hoop is located. The robot spins around till it is centered on the basketball hoop.
- Shooter pulls back his shooting arm.
- Shooter drops a single ball into the basket.
- Shooter then releases the shooting arm, which flings the ball at the hoop.
- This is repeated a set number of times
- Afterwards Shooter wonders off in a random direction
- While wondering the robot will avoid obstacles.

The success of most robots depend on how well the A/D code is written since there are so many analog inputs from various sensors. Building a sub behavior, then building another sub behavior, then combining the two and repeating this process worked really well. It allowed me to quickly find bugs and problems, along with giving me a place to restart if something went wrong later on.

Experimental Layout and Results

Range of Basketball Shooting

Unworn Servo & freshly charged batteries	Approximately 4 feet
Worn Servo & used batteries	Approximately 3 feet

Beacon Detector

Distance	Voltage Out
infinite	1.57
Next to	2.63
1.5 inches	2.61
2.0 inches	2.55

Sharp IR Range Finders

<i>Inches</i>	<i>IR value</i>
0.50	80
1.00	120
1.25	140
1.50	160
2.00	140
2.50	120
2.75	100
3.25	80
4.00	60
6.50	40
12.00	20

Bump Switches

	10k (Left)	20k (Center)	47k (Right)	100k (Back)
10k (Left)	<i>115</i>	135-140	125-130	115-120
20k (Center)	135-140	<i>76</i>	95-100	90-95
47k (Right)	125-130	95-100	<i>40</i>	50-55
100k (Back)	115-120	90-95	50-55	<i>20</i>

CLOSING

Conclusion

Due to Shooter's various revisions during the semester I am not surprised that I did not need to have all the sensors I originally planned on. I was surprised at how fast the semester went after Obstacle avoidance was due, even with all the warnings given beforehand that this would be the case. To that end I was unable to accomplish exactly what I desired to early on. Getting the ball loading and shooting mechanism working turned out to be a lot more difficult than I expect so I am very pleased with it being functioning. Adding in the ability to locate the hoop and then avoid obstacles means that I do have an intelligent autonomous robot, which means all the work did pay off while teaching me tons of things.

The cheapness of the servos was the biggest constraint on my robot. First it prevented me from using stronger springs which would have allowed me to launch the paddleball easily over 10 feet. The lack of strength also means my robot now has problems moving on carpet due to Shooter's weight. Second they do not seem to be able to take a lot of wear and tire since it appears I need to replace one that has been only used in the last two weeks.

Limiting myself to a robot that could have all the parts made on the T-tech machine meant that I ran out of space on the top of the robot where I had the vast majority of sensors and actuation. More space would have allowed me to have a longer shooting arm allowing for more force to through the ball at the hoop. The additional space would also allow me to drop the ball loader down and to the side instead of being high up in the air.

The ability for one IR beacon detector to be needed was unexpected. Shrink-wrap does allow for a very focus beam. My expectations for the shooting and loading were surpassed, because I was never sure if it would all come together or not.

If I were to start this project over I would have started before finals week last semester. Gotten my final design approved first day of class and start building then. I would aim for a less mechanical intensive design, since I am an electrical engineer. I would also make sure to have a more open platform so that I could get to my wiring and board a lot easier then the closed design I have now. There are better ways to support and reinforce your robot then using all four of your sides in a box type design.

I would stick to a circular design, since the ability to turn and not have to worry about corners is wonderful. The major change I would make would be to go with motors instead of servos. The added strength would allow for a more impressive demonstration. Probably over the summer I will work on adding in the sonar and tweaking the design, after spending so much time on Shooter there is no way I am going to just leave it alone.

Documentation

<http://www.mil.ufl.edu/imdl/>

http://www.atmel.com/dyn/resources/prod_documents/DOC1457.PDF

<http://www.avrfreaks.com/>

<http://www.prllc.com/>

<http://dkc3.digikey.com/pdf/T032/1085.pdf>

http://www.sharp.co.jp/products/device/ctlg/jsite22_10/table/pdf/osd/optical_sd/gp2d120_j.pdf

<http://www.jameco.com/jameco/Products/ProdDS/176541.pdf>

<http://www.google.com/>

Appendices

```
//Demo2.c by Jason Metzenthin
//Shooter turns till he finds the IR beacon
//Then loads and shoots three shoots
//Then goes into the obsticale avoidance

//Launching program uses Timer0
//Movement uses Timer1 as PWM channels

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

typedef unsigned char u08;
u08 channel;

u08 avgSpeed;

u08 centerIR;
u08 leftIR;
u08 rightIR;

u08 cIRarray[9]={0,0,0,0,0,0,0,0,0};
u08 rIRarray[9]={0,0,0,0,0,0,0,0,0};
u08 lIRarray[9]={0,0,0,0,0,0,0,0,0};

u08 rSpeed[9]={0,0,0,0,0,0,0,0,0};
u08 lSpeed[9]={0,0,0,0,0,0,0,0,0};

u08 bArray[9]={0,0,0,0,0,0,0,0,0};

u08 centerIRtotal;
u08 rightIRtotal;
u08 leftIRtotal;

u08 avgBump;

int ic=1;
int ir=1;
int il=1;
int ib=1;
int j=1;
int k=1;
```

```

#define phaseLength 0x74 //116*1024/6MHz = 20ms
#define Servo_Min 0x03
#define Servo_Max 0x0c
#define Servo_DutyOn (((Servo_Max-
Servo_Min)*ServoDutyCycle/Servo_Max)+Servo_Min)
#define Servo_DutyOff (phaseLength-Servo_DutyOn)

// #define drivers
#define numServos          0x05

#define DrivePort 0x00          //Pulling back arm
#define DriveStar 0x01          //Relasing mechanism
#define LiftServo 0x02          //Bottom gate in loader
#define TiltServo 0x03          //Top gate in loader
#define GrabServo 0x04

#define DriveAheadFull          Servo_Max
#define DriveReverseFull        Servo_Min
#define DriveAheadHalf          0x09
#define DriveReverseHalf        0x06
#define DriveStop                0x00

typedef unsigned short u16;
typedef unsigned long  u32;
// Globals for servo control
volatile u08 Phase= 0; //where am I in a Servo phase
volatile u08 segment = 0;
volatile u08 ServoDutyCycle[numServos]; //Servo_Min 0x03 to Servo_Max 0x0c
volatile u08 comeBackIn;
volatile u08 resetAll = 0;
volatile u08 currSpeed[numServos];
volatile u08 servoState,beenServiced[numServos],timeUsed;
u08 testcount;

u08 Det;

u08 detArray[9]={0,0,0,0,0,0,0,0,0};

int di=1;

volatile int flag;

SIGNAL(SIG_ADC)
{

```

```

if (channel==0x20)
{
    channel=0x21;
    detArray[di]=inp(ADCH);
    di++;
    if(di>9)
    {
        di=1;
    }
    Det = avgArray(detArray);
    if (Det>120)
    {
        flag=1;
    }
}
else if (channel==0x21)
{ //centerIR
    channel=0x22;
    cIRarray[ic]=inp(ADCH);
    ic++;
    if(ic>9)
    {
        ic=1;
    }
    centerIR = avgArray(cIRarray);
}
else if (channel==0x22)
{ //rightIR
    channel=0x23;
    rIRarray[ir]=inp(ADCH);
    ir++;
    if(ir>9)
    {
        ir=1;
    }
    rightIR = avgArray(rIRarray);
}
else if (channel==0x23)
{ //leftIR
    channel=0x20;
    lIRarray[il]=inp(ADCH);
    il++;
    if(il>9)
    {
        il=1;
    }
}

```

```

    leftIR = avgArray(IIRarray);
}

else
{
    //Bump Sensors
    channel=0x20;
    bArray[ib]=inp(ADCH);
    ib++;
    if(ib>9)
    {
        ib=1;
    }
    avgBump = avgArray(bArray);
}
outp(channel,ADMUX);
}

void init_motors(u08 num)
{ // make sure that you power with full six volts or this code is crap
    u08 cnt;
    //Init timer 0
    outp(0xFF,DDRB); //set portb as output
    outp(0x02,TIMSK); //set COIE bit=1 for interrupt enable at output compare
    outp(0x0d,TCCR0); //set CTC0=1 to clear at compare and CSO2:1:0=001
    prescale at ck speed/1024
    outp(0x75,OCR0); //set value in OCR0 reg to 117=0x75 servo period

    //Initialise the motors
    Phase= 0;
    segment = 0;
    timeUsed = 0;
    comeBackIn = phaseLength;
    resetAll = 0;
    servoState = 0x00;
    outp(servoState,PORTB); // set low
    for (cnt=0;cnt<=num;cnt++)
    {
        ServoDutyCycle[cnt]=0;
        currSpeed[cnt]=0;
        beenServiced[cnt] = 0;
    }
}

void off_motors()

```



```

//Turn off the motors for shooting
  outp(0x00,TIMSK);
  outp(0x00,TCCR0);
  outp(0x00,OCR0);
}

SIGNAL(SIG_OUTPUT_COMPARE0){ // This timer controls all the servos used in
Toby
  u08 pin = 0;

  if (resetAll == 1) {
    resetAll = 0;
    Phase = 0;
    outp(1,OCR0); //set value in OCR0 reg to duty pulses to count
  } else if (Phase > 0 && Phase < phaseLength) {
    segment = comeBackIn;
    timeUsed = Phase;
    for (pin = 0; pin < numServos; pin++){
      if( (beenServiced[pin] != 1) && (Phase == currSpeed[pin]) ){
        cbi(servoState,pin);
        beenServiced[pin] = 1;
      }
    }
    comeBackIn = phaseLength - timeUsed;
    for (pin = 0; pin < numServos; pin++){
      if ( ( comeBackIn > (currSpeed[pin] - timeUsed) ) &&
(beenServiced[pin] != 1) ){
        comeBackIn = (currSpeed[pin] - timeUsed);
      }
    }
    outp(comeBackIn,OCR0); //set value in OCR0 reg to duty pulses to
count
    Phase += comeBackIn;
    if (Phase == phaseLength) resetAll = 1;
  } else if (Phase > phaseLength) {
    resetAll = 0;
    Phase = 0;
  }
  if (Phase == 0){
    segment = 0;
    timeUsed = 0;
    for (pin = 0; pin < numServos; pin++){
      currSpeed[pin] = ServoDutyCycle[pin];
      if(currSpeed[pin]>0) {
        sbi(servoState,pin);
        beenServiced[pin] = 0;
      }
    }
  }
}

```

```

        } else {
            cbi(servoState,pin);
            beenServiced[pin] = 1;
        }
    }
    comeBackIn = phaseLength;
    for (pin = 0; pin < numServos; pin++){
        if ( ( comeBackIn > (currSpeed[pin] - timeUsed) ) &&
        (beenServiced[pin] != 1) ){
            comeBackIn = (currSpeed[pin] - timeUsed);
        }
    }
    outp(comeBackIn,OCR0); //set value in OCR0 reg to duty pulses to
count
    Phase += comeBackIn;
    if (Phase == phaseLength) resetAll = 1;
}
outp(servoState,PORTB);
return;
}

```

```

int main(void)
{

    u08 led;
    u08 test = 5;
    flag=0; //Beacon found 0 = NO
    int x;

    //A to D setup
    outp(0xff,DDRC);
    channel=0x20;
    outp(0x20,ADMUX);
    outp((1<<ADEN)|(1<<ADSC)|(1<<ADFR)|(1<<ADIE),ADCSR);
    outp(0xff,PORTC);

    sei();

    //PWM setup for wheels
    outp(0xFF,DDRD);
    outp(0xA1,TCCR1A);
    outp(0x04,TCCR1B);
    outp(0x00,TCNT1H);
    outp(0x00,TCNT1L);

```

```

sei();

while(flag!=1)
{
    rightwheel(0x20);    //Hardright
    leftwheel(0x40);
}

//Found the Beacon STOP
//rightwheel            //stop before
outp(0x00,OCR1AH);
outp(0x00,OCR1AL);
//leftwheel            //you go past beacon
outp(0x00,OCR1BH);
outp(0x00,OCR1BL);

//Turn off PWM
//outp(0x00,TCCR1A);
//outp(0x00,TCCR1B);
//outp(0x00,TCNT1H);
//outp(0x00,TCNT1L);

//Let's start shooting

init_motors(numServos);

outp(0xff,DDRC);

ServoDutyCycle[DrivePort] = 0x00;
ServoDutyCycle[DriveStar] = 0x00;

x=0;
while(x<3)            //x<3 for demonstration, x<1 for testing
{
    x++;
}

//Pulling Back

    ServoDutyCycle[0x01]=0x0F;
    wait(500);

    ServoDutyCycle[0x00]=0x0F;    //Launch Arm up top
    wait(500);

    ServoDutyCycle[0x01]=0x07;    //Releasing Latched On

```

```

    outp(0xFE,PORTC);
    wait(500);

    ServoDutyCycle[0x01]=0x00;    //Clearing PWM
    wait(500);

    ServoDutyCycle[0x00]=0x02;    //Pulling back Arm
    wait(500);

//Loading Ball

    ServoDutyCycle[0x03]=0x07;    //Make sure top gate
    wait(300);                    //is closed

    ServoDutyCycle[0x03]=0x00;
    wait(500);

    ServoDutyCycle[0x02]=0x0F;    //Open bottom gate
    wait(300);
    ServoDutyCycle[0x02]=0x0F;    //Open bottom gate
    wait(300);

    ServoDutyCycle[0x02]=0x07;    //Close bottome gate
    wait(500);

    ServoDutyCycle[0x02]=0x00;
    wait(500);

    ServoDutyCycle[0x03]=0x02;    //Open top gate
    outp(0xF7,PORTC);
    wait(500);

    ServoDutyCycle[0x03]=0x07;    //Close top gate
    outp(0xEF,PORTC);
    wait(300);
    ServoDutyCycle[0x03]=0x07;    //Close top gate
    wait(300);

    ServoDutyCycle[0x03]=0x00;
    wait(500);
    outp(0xEF,PORTC);

//Firing At long Last!

    ServoDutyCycle[0x01]=0x02;    //Releasing Launch Arm

```

```

    outp(0xF7,PORTC);
    wait(500);

    ServoDutyCycle[0x01]=0x00;    //Clearing PWM
    outp(0xEF,PORTC);
    wait(500);

}

outp(0x00,PORTB);

//Done shooting time to wonder off

off_motors();

//PWM setup for wheels
outp(0xFF,DDRD);
outp(0xA1,TCCR1A);
outp(0x04,TCCR1B);
outp(0x00,TCNT1H);
outp(0x00,TCNT1L);

for(;;)
{
    if(centerIR>70)
    {
        rightwheel(0x00);    //stop before
        leftwheel(0x00);    //you hit something
    }
    else if(centerIR>55)
    {
        if(rightIR>leftIR)
        {
            rightwheel(0x02); //hardleft
            leftwheel(0x02);

            wait(20);
            rightwheel(0x02); //hardleft
            leftwheel(0x02);

            wait(20);
            rightwheel(0x02); //hardleft
            leftwheel(0x02);

            wait(20);
        }
    }
}

```

```

        rightwheel(0x02); //hardleft
        leftwheel(0x02);
    }
    else
    {
        rightwheel(0x20); //Hardright
        leftwheel(0x40);

        wait(20);
        rightwheel(0x20); //Hardright
        leftwheel(0x40);

        wait(20);
        rightwheel(0x20); //Hardright
        leftwheel(0x40);

        wait(20);
        rightwheel(0x20); //Hardright
        leftwheel(0x40);
    }
}
else if(rightIR>20 && leftIR<rightIR)
{
    // Turn Left
    rightwheel(0x02);
    leftwheel(0x00);
    outp(0xFB,PORTC);
}
else if(leftIR>20 && rightIR<leftIR)
{
    rightwheel(0x00);
    leftwheel(0x40);
    outp(0xFE,PORTC);
}
else
{
    //forward movement
    rightwheel(0x02);
    leftwheel(0x40);
    outp(0xF7,PORTC);
}
}
}

```

```

void rightwheel(u08 newSpeed)

```

```

{
    rSpeed[j]=newSpeed;
    j++;
    if(j>9)
    {
        j=1;
    }
    avgSpeed=avgArray(rSpeed);
    outp(0x00,OCR1AH);
    outp(avgSpeed,OCR1AL);
    return;
}

void leftwheel(u08 newSpeed)
{
    lSpeed[k]=newSpeed;
    k++;
    if(k>9)
    {
        k=1;
    }
    avgSpeed=avgArray(lSpeed);
    outp(0x00,OCR1BH);
    outp(avgSpeed,OCR1BL);

    return;
}

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

u08 avgArray(u08 array[9])
{

```

```
u08 value;  
value = (array[1] + array[2] + array[3] + array[4]  
+ array[5] + array[6] + array[7] + array[8] + array[9])/9;  
return value;  
}
```