University of Florida

Department of Electrical and Computer Engineering

EEL 5666

Intelligent Machines Design Laboratory

Toby the Toy-bot

Final Written Report

4/22/2003

Mike Collins

TA: Uriel Rodriguez and Jason Plew

Instructor*:* A. A Arroyo

Table of Contents

Abstract

Toby is wandering seek and acquire robot. His mission is to find toys and pick them up. Children at early ages are unable or unwilling to pick their toys. “Toby the Toy-bot” is a self-propelled mobile robot that wanders the floor in the dark and searches for toys or other small objects to pick up. While this may seem like a silly application of technology, these same behaviors do have application in seeking and manipulating objects in other contexts. I am using this as a sample of material handling, open pit mining or of debris removal.

Executive Summary

Toby the Toy-bot is mobile toy box. He is designed to wander a room at night looking for small toys and pick them up. I built his brains on the Atmel mega323 micro controller. I use IR and bump switches to sense the world around Toby. He uses three top mounted IR sensors for collision avoidance. He uses a fourth IR sensor for toy detection. In front, I added a scoop and grabbing finger that pick up the toys. The scoop is actuated by a brass, lifting arm mounted to a gear reducer. I implemented Toby's behaviors in c.

Introduction

Toby the Toy-bot is mobile toy box. He is designed to wander a room at night looking for small toys and pick them up. He uses three top mounted IR sensors for collision avoidance. He uses a fourth IR sensor for toy detection. I will explain each subsystems hardware and the code that goes along with it.

Mobile Platform

Toby's platform is a three point of contact chassis. His body and most of his frame are 1/8" 5 ply aircraft plywood. He was designed in AutoCAD and cut form sheet stock on the IMDL's T-Tech CNC router. I made additional pieces for the mounting hardware and cover by hand or from stock hardware components.

The platform houses the brains, power (13 AA batteries in banks of 8 and 5 cells) and sensors. The platform serves to both hold in place and provide any positioning required of the sensors. The platform will also provide the base link for the manipulator. The platform is able to carry a payload of 4 pounds of toys.

Lessons Learned

While building the platform, I realized that the parts don't have to be perfect, they have to work. It's better to prototype several designs badly and to find parts that work well together than to build beautiful parts that don't work. I also discovered that foamed PVC plastic is a very easily worked material. It can be cut and drilled like wood but can also be bent and twisted with the heat of a heat gun. It is also very light for its strength.

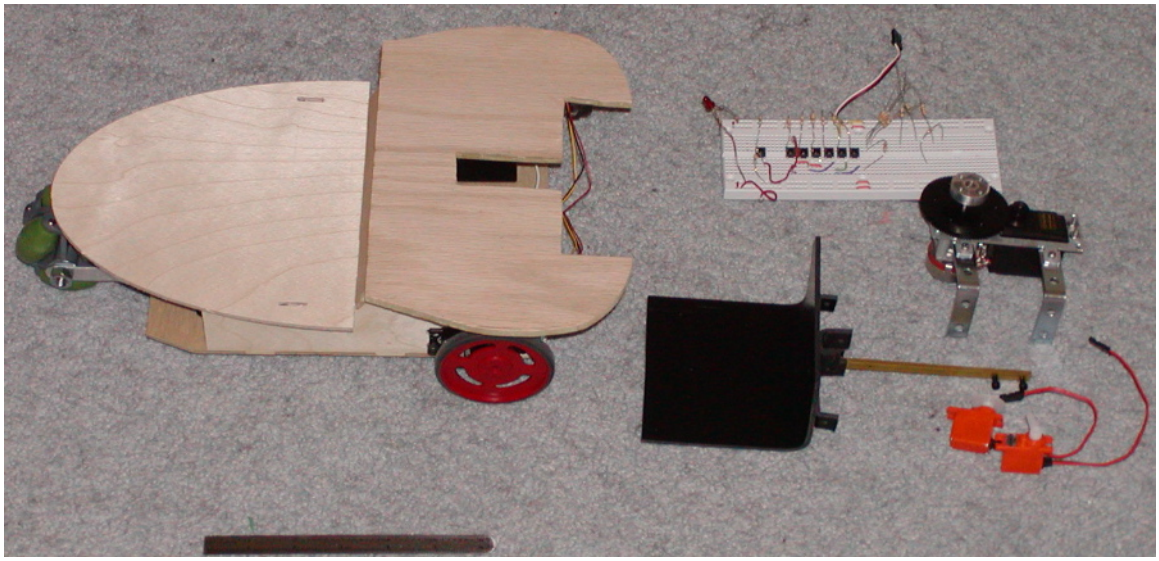

Figure 1 The platform in progress (ruler in lower left is 6” long)

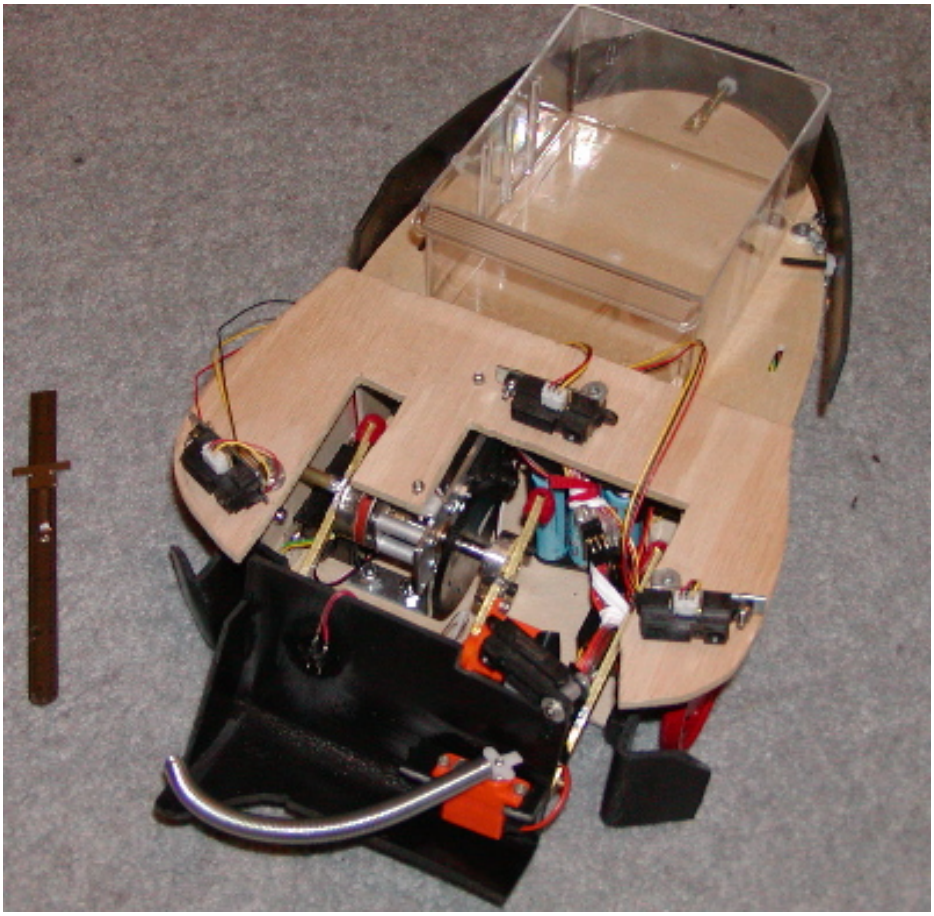

Figure 2 The final platform (ruler is 6” long)

Actuation

Toby's platform consists of two forward drive wheels and a rear omni-ball roller (Figure 3). Steering will be accomplished by driving the forward wheels at different rates. A mated pair of hacked servomotors directly drives the forward wheels.

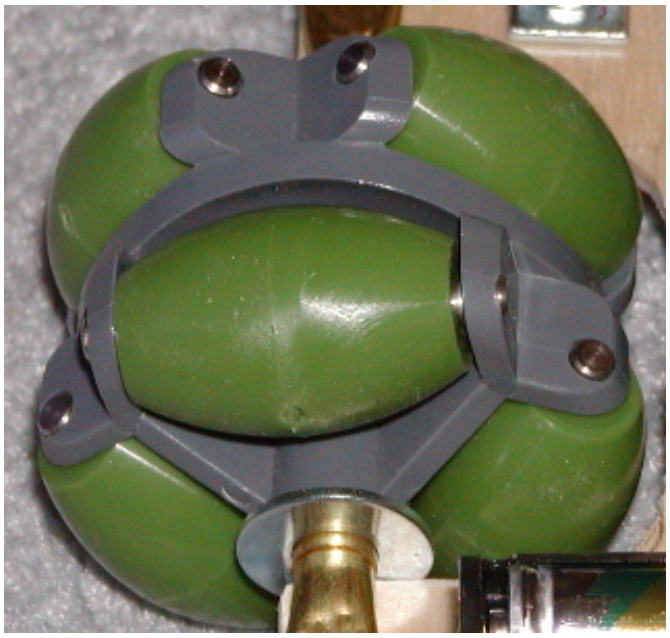

Figure 3 Rear omni-ball roller wheel

Toby moves with use of two hacked, 6-volt servomotors. The manipulator is driven with servos. The primary (shoulder) linkage uses a 6-volt, 42 oz.-in. servo. This servo has an extra 4:1 gear reducer on it (Figure 4). Attached to the final gear is large potentiometer that serves as both the shaft of the servo and position sense.

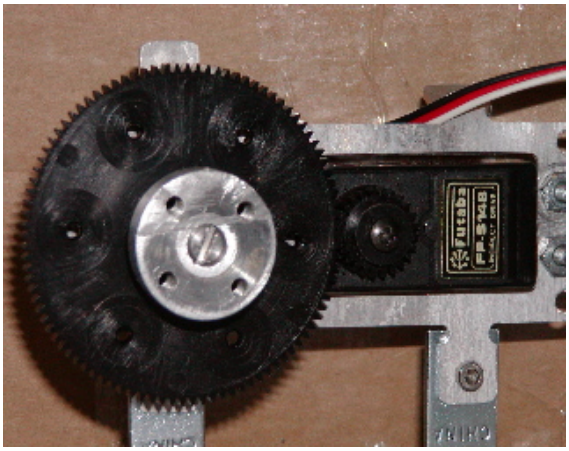

Figure 4 4:1 Gear reducer

The secondary (wrist) linkage uses a 4.8-volt, 23 oz.-in. servo. A 4.8-volt, 23 oz.-in. servo, actuates the finger.

Lessons Learned

I learned the hard way that servos function if they have less than their rated power. They function ERATICALLY however.

I also learned that the best way to calibrate a servo is to cut a small groove in to the potentiometer of the servo. This way I can insert a screwdriver through the drive gear (This can even be done with many wheels still on.) and set the potentiometer to give the desired speed at a set frequency. (see Figure 5)

I also kept learning to keep it simple . My original design used a four bar linkage to lift toys. The gear reducer did a better job. I had even intended to put counterweight on the arm but that proved to be unnecessary.

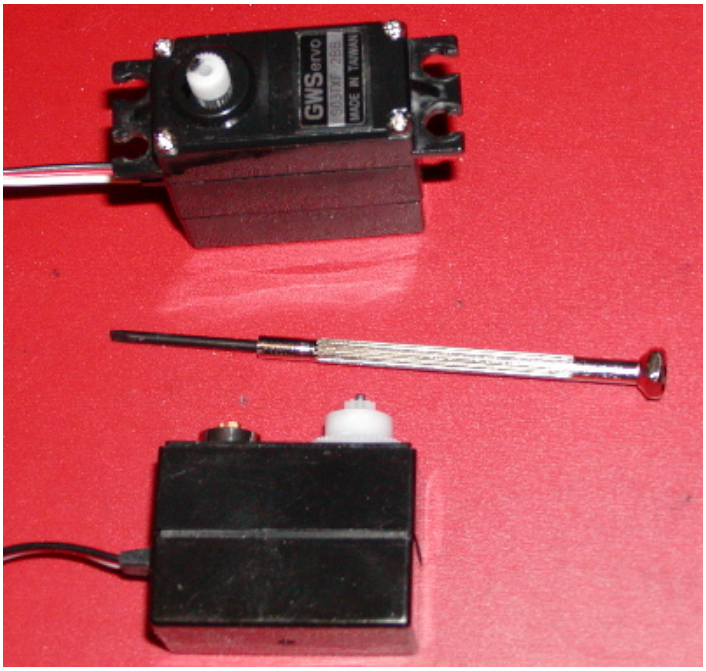

Figure 5 How to calibrate a servo without opening

Power

Toby will be battery powered. He currently carries 13 AA cells in two banks of five and eight cells. This allows for a group for motor power and a group of cells for brainpower. It also allows for isolation of the servo motors from the sensors and microcontroller. Time permitting a recharging system would have been added.

Manipulator

The manipulator is able to pick up half pound, non-rolling objects and lift them into the hopper. These objects must have a center of mass less than two inches from the object's edge. The manipulator is based on a high torque servo and a reducing gear train. I attached a set screw collar to the external potentiometer of the shoulder. Moving down the manipulator arm, I then added a brass frame that holds the scoop and the wrist servo.

This small servo allows the scoop to have an additional rotational degree of freedom. With this wrist servo, I am able to tilt the scoop and dump the contents of the scoop into the hopper and to change the angle of attack when it is picking objects up. Additionally I can tilt the scoop for angling the IR sensor mounted on the top of the scoop for sensing toys. A second small servo actuates the finger attached to the scoop. The finger keeps lighter objects from being pushed away from Toby when he moves in to pick them up. The finger itself is made from an extension spring and piece of formed PVC piece that acts as a return spring. (see Figure 16.)

Sensors

The sensors will allow Toby to find out what is in the world around him. Toby's world consists of toys and obstacles. A toy is any half-pound, non-rolling object. An obstacle is every other thing. Sensor use:

IR proximity detectors: (4 Sharp GP2D120) to find obstacles and toys. Used also to tell the difference based on height and width. I mounted three sensors on top of Toby for collision avoidance. These sensors are mounted at 0°, 25°, and 335° off of the forward line of travel. I mounted a fourth sensor on the scoop to detect toys within the grasping range of the finger. This sensor is mounted looking 10° down and to 15° to the right. It also helps me to detect toys that are further away if I tilt the scoop up.

I used a multiplexed analog to digital converter to read the data from these sensors. I read once and discard the data (to let the ADC settle down), read again and store the data in an array and switch to the next sensor on the ADC. If the values fall out side the range of the IR sensor, I ignore the new value. I repeat this cycle 16 times for each of the IR sensors.

To find the distance from this sensor, I take the average of these values at the time I need it. (see appendix for more information on these sensors)

Bump switches: (3+1) to find obstacles and toys. I use three of the bump switches to detect obstacles. I mounted these on single voltage divider. Each of the bump switches has skirt or swivel that is attached to the platform. I use the other contact switches to find out if there is anything in the scoop. (these are mounted on a separate voltage divider because I had an error in my code and this solved the problem)

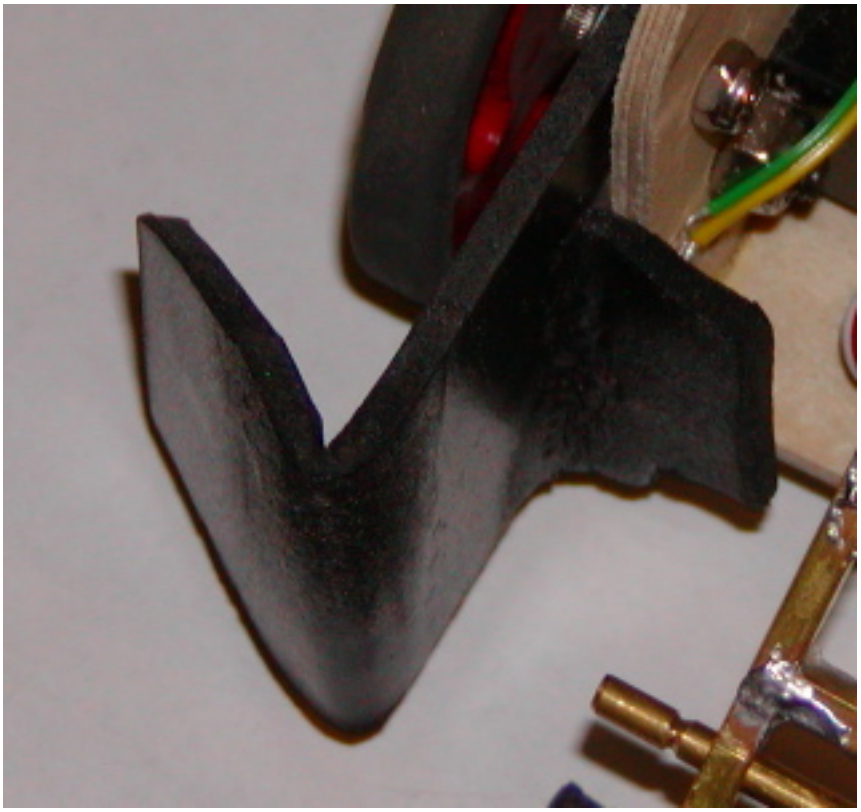

Figure 6 Bumper

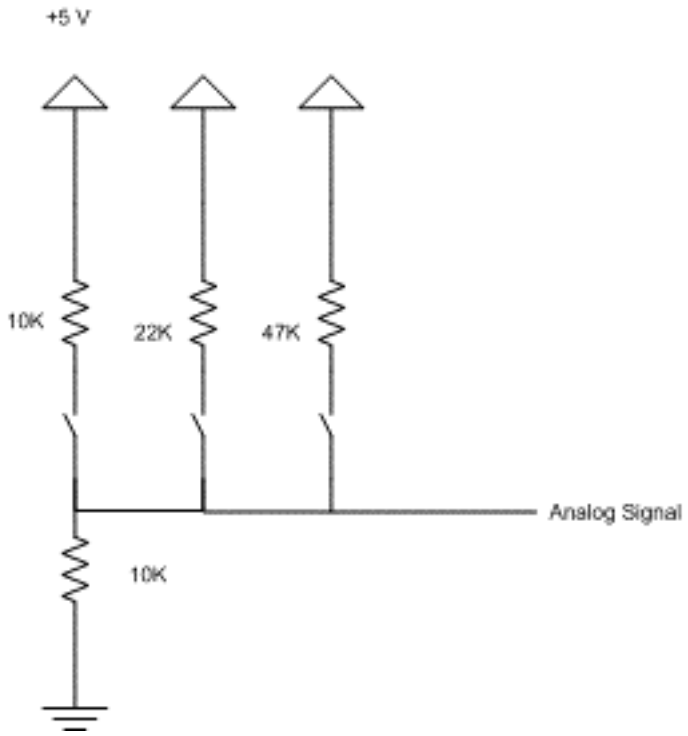

Figure 7 Voltage divider interface

Light detector: (1 CdS) used to sense if it is "day." I used this light sensor to detect light, if it is light enough, Toby stops working.

Figure 8 CdS Light detector

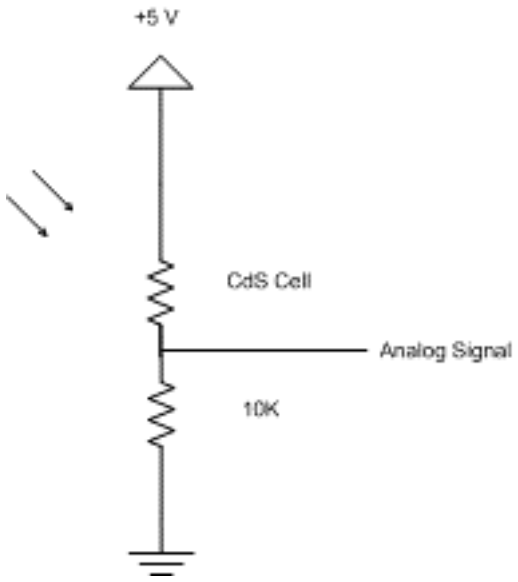

Figure 9 Light detector interface

Behaviors

Toby's brains are instantiated in a Progressive Resources, LLC, MegaAVR developers board with an on board ATmega323 microcontroller. His mind is written in c and allows him to have useful behaviors that will include:

Pick up /store toys (but not anything so heavy it can't lift it) If a toy is detected, pick it up, put it in the hopper.

Wander: If there is nothing to do wander around.

Sleep during the day. If bright light is detected it must be "day," stop and wait for night.

Seek toys: use sensor to look for toys.

Avoid walls/obstacles (anything taller than a toy or too heavy to lift) maneuver around anything that can't be picked up.

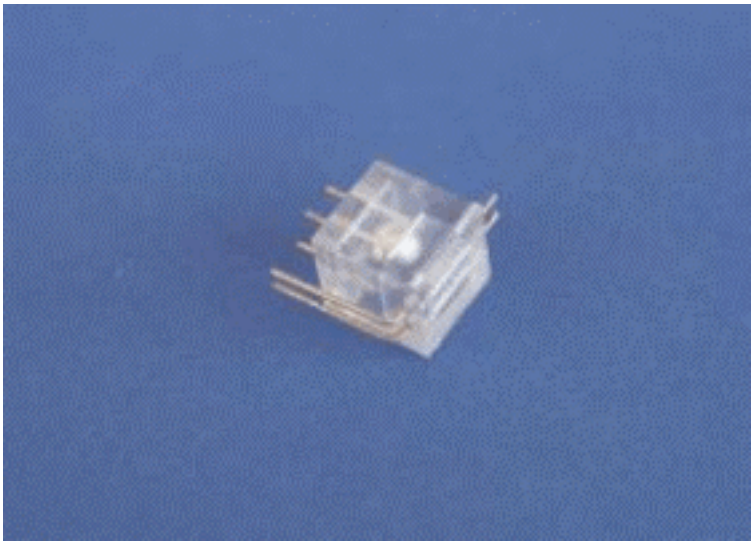

Figure 10 Tilt sensor for further experimentation

Conclusion

As a test bed for sensors and control behaviors, I find that Toby serves well. I think that he can be use for expansion of his current capabilities too. I shortchanged Toby's higher level behaviors by taking so long to build his platform and implement his collision avoidance. Toy seeking leaves much to be desired.

Future work

If I were to continue working on Toby, I'd like to add more bump switches and a tilt switch to the scoop. (see Figure 10) I'd also like to build a homing routine with the light sensors and a light source on a charging station. If I were just doing this for fun I'd be adding many more behaviors. I am very interested in trying a more modular arbitrator. If I were starting over, I would definitely get more ideas (i.e. code and circuits) from my

classmates. It seems to be much easier to buy, beg, borrow or steal (code anyway) things than to “roll your own.”

That said, I must now thank those whose help I should have asked for sooner:

Professors Arroyo and Schwartz. Teaching Assistants Uriel Rodriguez and Jason Plew

The former students of IMDL for their understanding of the Atmel microcontroller and the code they shared with us. Specifically, C. Andrew Davis and Amit Jayakaran. Ashish Jain whose support and interest kept me going when all seemed lost.

My classmates: Steve Vanderploeg, Jordan Wood, Kyle Tripician, Brian Ruck, Danny Kent and Roberto Montane.

Documentation

Progressive Resources, LLC	http://www.prllc.com
Acroname	http://www.acroname.com
Mark III Robot Store	http://www.junun.org/MarkIII/Store.jsp
Servo City	http://www.servocity.com
Zell’s Ace Hardware	3727 W. University Ave. (phone 352-378-4650)

Much more useful for small parts than the home improvement stores

Budget Robotics	http://www.budgetrobotics.com/
Robot Builder's Sourcebook	http://www.robotoid.com/

by Gordon McComb

Appendices

Code

```
// Toby the Toy Bot
// Coll avoid and arm action on ir4
#include <io.h>
#include <interrupt.h>
#include <math.h>
#include <SIG-AVR.h>

#define irLeft sample[0x00]
#define irRight sample[0x01]
#define irMid sample[0x02]

#define close 0x22  //  10
#define very_close 0x38  //  40

#define irScoop sample[0x03]

#define Toy_GripClose 0x3F
// This is only  useful if scoop is tilted aft
#define Toy_Ho 0x30



#define phaseLength 0x74  //116*1024/6MHz = 20ms
#define Servo_Min 0x03
#define Servo_Max 0x0d
#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
#define DriveStar 0x01
#define LiftServo 0x02
#define TiltServo 0x03
#define FingerServo 0x04

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

#define speedMax        0x06
#define speedMid        0x07



typedef signed char  s08;
typedef unsigned char  u08;
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;


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;
      }
}

SIGNAL(SIG_OUTPUT_COMPARE0){ // This timmer 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;
}

void SetPortMotor(u08 speed) {
// Port:forward high --- backward low
      if (speed == 0) {
            ServoDutyCycle[DrivePort] = 0;
      } else {
            if (speed < Servo_Min) {
                  ServoDutyCycle[DrivePort] = Servo_Min;
```

```
            } else if (speed > Servo_Max) {
                  ServoDutyCycle[DrivePort] = Servo_Max;
            } else ServoDutyCycle[DrivePort] = speed;
      }
      return;
}

void SetStarboardMotor(u08 speed) {
// Starboard Motor has LOW values for forward
      if (speed == 0) {
            ServoDutyCycle[DriveStar] = 0;
      } else {
            if (Servo_Max-speed < Servo_Min) { // OK since "0" case
already done
                  speed = Servo_Max - Servo_Min;
            }
            ServoDutyCycle[DriveStar] = Servo_Max - speed + Servo_Min;
      }
      return;
}



void setDrives(u08 leftSpeed, u08 rightSpeed){
      SetPortMotor(leftSpeed);
      SetStarboardMotor(rightSpeed);
      return;
}



// Definitions for ADC long sample IR
#define sampleMax 0x0F
#define sampleBeforeSwitch 4
#define maxAnalogChannels 0x06 //  shall be 0 based -i.e. 0 means one
ana chana
#define baseChannelCnt 0x20
#define IRMax 0xA0
#define IRMin 0x02
#define leftIRChannel 0x00
#define rightIRChannel 0x01
#define midIRChannel 0x02
#define scoopIRChannel 0x03
#define lightSense 0x04
#define bumpChannel1 0x05
#define bumpChannel2 0x06
#define ImABump(inp) ((inp==bumpChannel1)||(inp==bumpChannel2))

// Globals for ADC
volatile u08 sample[maxAnalogChannels];
volatile u08 holding,toss;
volatile u08 channel = baseChannelCnt;
volatile u08 offsetChan = baseChannelCnt+1;
//                channel             sampleN
volatile u08 wideSample[maxAnalogChannels][sampleMax];
volatile u08 switchIndex = 0;// 0-4 sampleBeforeSwitch
volatile u08 sampleIndex = 0;// 0-0x0F sampleMax
```

```
volatile u08 channelIndex = 0;// 0-7 maxAnalogChannels

// Global values for sensors
// for bumpChannel1
#define bumpFinger 0x05
#define bumpScoop 0x06
#define bumpBottom 0x07
// for bumpChannel2
#define bumpLeft 0x08
#define bumpRight 0x09
#define bumpBack 0x0A

volatile u08 sensorState[11];// 4 IR;3+3 bumps; CdS


SIGNAL(SIG_ADC) {
// when the ADC is finished reading value this interupt is called
// read each channel "sampleBeforeSwitch" times then add it to the
sample array and average it with the total array
//                      far   close
// legitimate voltage range is .35 to 2.75V so legit values for the ADC
are 0x02||xx to 0xA0||xx

      u08 holdADC;
      u08 incList;


      if (ImABump(channelIndex)) {// this is for bumps changing
            toss = inp(ADCL); // get low 8 bits
            holdADC = inp(ADCH);// get high bits
            // sample in legit range increment and average else inc
            if ( switchIndex != 0) {// discard first sample of ADC
                  wideSample[channelIndex][0] = holdADC;
            }
            switchIndex++;
            if (switchIndex > sampleBeforeSwitch ) {
                  switchIndex = 0;
                  channelIndex++;
                  if (channelIndex> (maxAnalogChannels) ) channelIndex
= 0;
            }
      } else {// this is for IR's

            toss = inp(ADCL); // get low 8 bits
            holdADC = inp(ADCH);// get high bits
            // sample in legit range increment and average else inc
            if ( switchIndex != 0) {// discard first sample of ADC
                  if((holdADC < IRMax) && (holdADC > IRMin)) {
                        wideSample[channelIndex][sampleIndex] =
holdADC;
                        sampleIndex++;
                        if (sampleIndex > sampleMax ) sampleIndex = 0;
                  }
            }
            switchIndex++;
            if (switchIndex > sampleBeforeSwitch ) {
                  switchIndex = 0;
```

```
                  channelIndex++;
                  if (channelIndex> (maxAnalogChannels) ) channelIndex
= 0;
            }
      }

      // select channel via ADMUX
      outp((baseChannelCnt+channelIndex), ADMUX);// baseChannelCnt sets
the apropriate control bit
      return;
}
void init_ADC(void)
{
      outp(0x00,DDRA);        //set port a as input
      channel=baseChannelCnt;
      outp(baseChannelCnt,ADMUX);
      outp((1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADFR)|(1<<ADPS2)|(1<<ADPS1
)|(0<<ADPS0), ADCSR);
      switchIndex = 0;
      sampleIndex = 0;
      channelIndex = 0;
      return;
}

u08 get_ADC(u08 channel)
{
      u08 getADCCt;
      u16 runningTotal = 0;

      if (ImABump(channel)){
            runningTotal = wideSample[channel][0];
      } else {// I'm an IR
            for (getADCCt = 0;getADCCt<=sampleMax;getADCCt++) {
                  if ((wideSample[channel][getADCCt]>IRMin) &&
(wideSample[channel][getADCCt]<IRMax)) { // valid value
                        runningTotal += wideSample[channel][getADCCt];
                  } else {
                        runningTotal += sample[channel];// add prevous
value to the running average
                  }
            }
            runningTotal = runningTotal/sampleMax;
      }
      getADCCt = runningTotal;
      sample[channel] = runningTotal;
      return getADCCt;//      wideSample[channel][0x01]
}

void updateADCChannels(void)
{
      u08 chanInc;
      u08 chanValue;
      u08 toss=0x0;

      for (chanInc = 0;chanInc<=maxAnalogChannels;chanInc++) {
            chanValue = get_ADC(chanInc);
            switch (chanInc) {
```

```
                  case leftIRChannel:
                  case rightIRChannel:
                  case midIRChannel:
                  case scoopIRChannel:
                  case lightSense:
                        sensorState[chanInc] = chanValue;
                        break;

                  case bumpChannel1:
                        if(chanValue>0xA9) {
                              sensorState[bumpFinger]=0x01;
                              chanValue -=0xA9;
                        }
                        if((chanValue>0x7F)&&(chanValue<0x88)) {
                              sensorState[bumpScoop]=0x01;
                              chanValue -=0x7f;
                        }
                        if((chanValue>0x4E)&&(chanValue<0x58)) {
                              sensorState[bumpBottom]=0x01;
                              chanValue -=0x4e;
                        }
                        break;
                  case bumpChannel2:
                        //sensorState[bumpChannel2] = chanValue;
                        if(chanValue>0xA9) {
                              sensorState[bumpLeft]=0x01;
                              chanValue =chanValue - 0xA9;
                              toss += 0x01;
                        }
                        if((chanValue>0x7F)&&(chanValue<0x88)) {
                              sensorState[bumpRight]=0x02;
                              chanValue -=0x7f;
                              toss += 0x02;
                        }
                        if((chanValue>0x4E)&&(chanValue<0x58)) {
                              sensorState[bumpBack]=0x04;
                              chanValue -=0x4e;
                              toss += 0x04;
                        }


                  //    sensorState[bumpLeft]= toss;
                        sensorState[bumpRight]= toss;
                        sensorState[bumpBack]= toss;


                        break;


            }
      }
}

void Wait_opt(int time) {
      volatile int a, b, c, d;

      for (a = 0; a < time; ++a) {
```

```
            for (b = 0; b < 10; ++b) {
                  for (c = 0; c < 66; ++c) {
                        d = a + 1;
                  }
            }
      }
      return;
}

// lifting
#define Servo_Nuetral 0x00
// lift
#define Lift_Down 0x09
// nomial with no others running 0x07
#define Lift_Half 0x0a
#define Lift_Up 0x0d
// nomial with no others running 0x0d
#define Lift_Nuetral 0x00
// Tilt
#define Tilt_For 0x0a
#define Tilt_Aft 0x08
#define Tilt_Nuetral 0x00
// finger
#define Finger_Open 0x03
#define Finger_Closed 0x07
#define Finger_Nuetral 0x00
      u08 led = 0x00;

void liftUp( void )
{
      ServoDutyCycle[LiftServo] = Lift_Up;
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      ServoDutyCycle[LiftServo] = Servo_Nuetral;
}
void liftDown( void )
{
// Shoulder_Down
      ServoDutyCycle[LiftServo] = 0x07;
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      ServoDutyCycle[LiftServo] = Servo_Nuetral;
}

void tiltFor( void )
{
// Tilt_For
      ServoDutyCycle[TiltServo] = Tilt_For;
      Wait_opt(0xff);
      ServoDutyCycle[TiltServo] = Servo_Nuetral;
}

void tiltAft( void )
{
```

```
// Tilt_Aft
      ServoDutyCycle[TiltServo] = Tilt_Aft;
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      ServoDutyCycle[TiltServo] = Servo_Nuetral;
}

void grab( void )
{
// Finger_Closed
      ServoDutyCycle[FingerServo] = Finger_Closed;
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      ServoDutyCycle[FingerServo] = Servo_Nuetral;
}

void release( void )
{
// Finger_Open
      ServoDutyCycle[FingerServo] = Finger_Open;
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      Wait_opt(0xff);
      ServoDutyCycle[FingerServo] = Servo_Nuetral;
}


void resetArm(void)
{
      release();
      tiltFor();
      liftDown();
}

void grabToy(void)
{
      grab();
      tiltAft();
      liftUp();
      resetArm();
}


int main( void )
{
      u08 led;
      u08 test = 0;
      u08 Servtes = 0;
      u08 leftIRTest = 0x00;
      u08 rightIRTest = 0x00;
      u08 midIRTest = 0x00;

      init_motors(numServos);
```

```
outp(0xFE,ADCSR);
outp(0x20,ADMUX);

outp(0xFF,DDRC);        //set portc as output
outp(0xFF,PORTC);       //set all leds at portc on
sei();                  //set global interrupt enable

sbi(ADCSR, ADSC);

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

resetArm();

test = 0xff;
outp(~test,PORTC);
while(1) {
      test = 0;
      Wait_opt(100);

      updateADCChannels();

      if (irLeft>0x1f){
            leftIRTest = 0x01;
      } else {
            leftIRTest = 0x00;
      }

      if (irRight>0x1f){
            rightIRTest = 0x01;
      } else {
            rightIRTest = 0x00;
      }

      if (irMid>0x16){
            midIRTest = 0x01;
      } else {
            midIRTest = 0x00;
      }

      test = (leftIRTest<<2)+(midIRTest<<1)+(rightIRTest);


      switch (test) {
      case 0x00:// str fast
            setDrives(0x0a,0x0a);
            break;

      case 0x01:// left
            setDrives(0x00,0x0A);
            outp(0x00,PORTC);
            break;

      case 0x02:// ???? slow rt  *** not for final
            setDrives(0x09,0x00);
            break;
```

```
            case 0x03:// hard left
                  setDrives(0x06,0x09);
                  break;

            case 0x04:// right
                  setDrives(0x09,0x00);
                  break;

            case 0x05:// str slow
                  setDrives(0x08,0x08);
                  break;

            case 0x06:// hard right
                  setDrives(0x09,0x06);
                  break;

            case 0x07:// stop
                  setDrives(0x06,0x06);
                  Wait_opt(0xff);
                  Wait_opt(0xff);
                  setDrives(0x00,0x00);
                  Wait_opt(0xff);
                  setDrives(0x06,0x09);
                  Wait_opt(0xA0);
                  break;
            }// end switch
            if(irScoop>Toy_GripClose) {
                  setDrives(0x00,0x00);
                  grabToy();
            }
            outp(~test,PORTC);
      }

}
```

And sensor stuff

IR Object size detection voltage as target area and range change

Voltages as Measured by ADC

	Target Area [sq. in.]		
Range [in.]	**9**	**4**	**1**
3	94	111	88
6	48	53	30
9	23	13	17

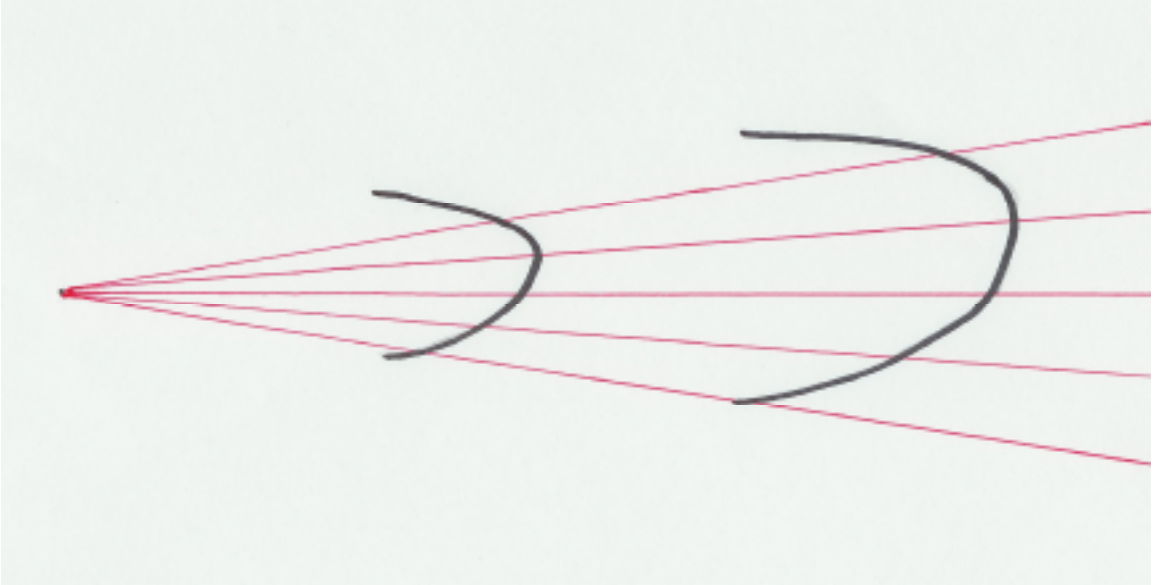

Figure 11 Top View of IR isometric voltages

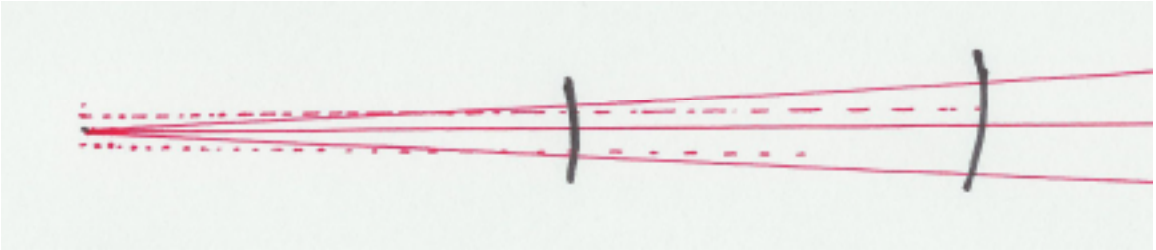

Figure 12 Side View of IR isometric voltages

Black lines are Isometric voltages as measured by ADC

Red lines are angle references

Dashed red lines are parallel to top and bottom of sensor casing

These IR sensors seem to be most sensitive a few degrees off of their center, side to side but are consistent top to bottom. (see Figure 11 & Figure 12)

Lessons Learned

Here are some images taken of an IR sensor working. I wish I'd thought of using this camera before I fiddled with everything else.

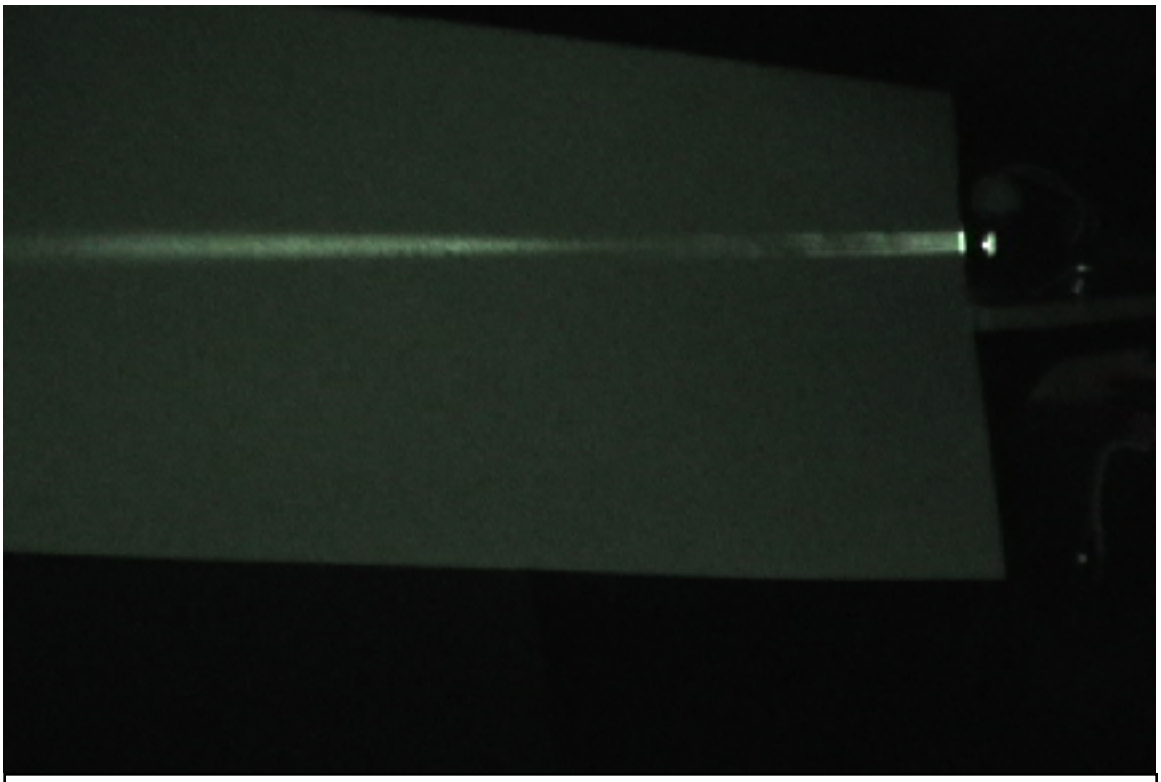

Figure 13 IR Beam on a white sheet of paper (side)

Figure 14 IR Beam on a white sheet of paper (front)

Figure 15 Toby as seen in the Infrared

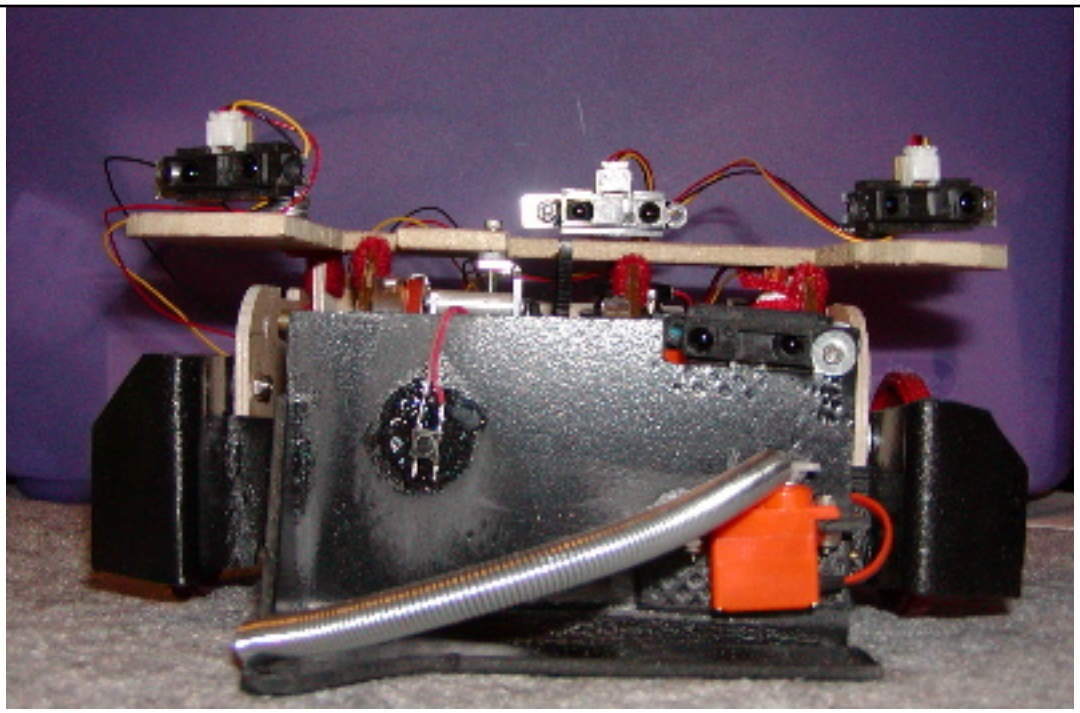

Figure 16 Toby as seen in visible light