Date: August 7, 2006
Student Name: Jeremy Greene
TAs : Adam Barnet
William Otto Goethals
Instructors: A. A Arroyo
E. M. Schwartz

University of Florida
Department of Electrical and Computer Engineering
EEL 5666 Intelligent Machines Design Laboratory
Final Report

Atlas

A Ball Balancing Robot

Table of Contents

Abstract

Atlas will roam around a room, avoiding obstacles, while balancing a ball. This is done by monitoring the ball with a camera and tilting the platform when the ball is not centered. Obstacle avoidance is performed by two sonar range finders and a bump switch.

Executive Summary

When powered-up, Atlas will calmly wait for you to place a ball on the center of the platform and push the bump switch. This lets it know that the color of the ball is ready to be measured and the color tracking system should be calibrated accordingly. Once Atlas is ready to move it will let you know via the lcd display. This gives the user the opportunity to get out of the way before witting the bump switch one last time to send the robot on its way. While moving, Atlas will be looking for obstacles by using two sonar devices and a bump switch. If Atlas detects anything directly in front of it, he will move to avoid it. In order to balance the ball a vision system was used. This system tells Atlas where the ball is. Atlas can then determine the velocity of the ball from previous position values. It is these two variables (position and velocity) that determine the orientation of the platform.

Introduction

Balancing a ball on a platform is a classic example of a control dynamics problem. This problem is of particular interest because the system is unstable, meaning that once the ball is moving it will roll off the platform if no action is taken. This as opposed to a stable system, such as a ball in a bowl, where if the ball is nudged it will simply oscillate. I chose this project to show that a robot could be made to be as sensitive as a person is to this type of control problem. If you were given a ball and a plate and asked to keep the ball from falling off, you would be able to do it without even considering the complicated mathematics that you are undoubtedly performing.

Integrated System

Atlas is controlled by a Atmega128 processor that is part of the Mavric IIB. This micro-controller is interfaced with many other sensors and actuators that allow it to accomplish its task. The bump switch and sonar sensors data are read by the micro-controller which then decides the speed and direction of the motors. The data from the CMUcam is also read by the camera and the servos are then controlled by the micro-controller. A block diagram of the communication between devices can be found below.

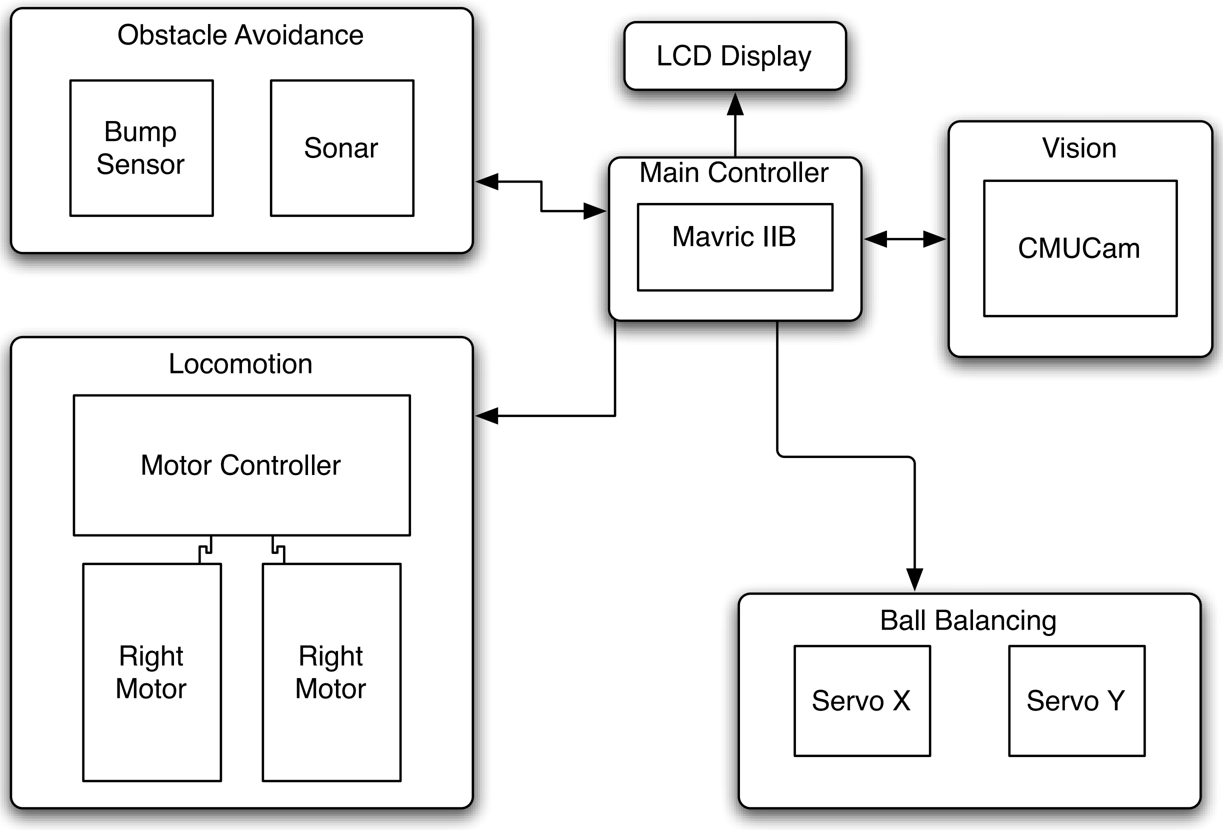
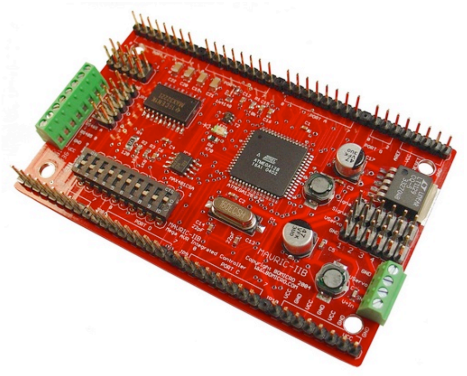


Fig 1: Block Diagram of control and processing system

Microcontroller

I chose the Mavric IIB for Atlas because of its high rate of success in this class. It features:

- Atmel ATmega128 MCU
- 128K Program FLASH, 4K Static RAM, 4K EEPROM
- dual level shifted UARTs •
- RS485 on-board
- 6 R/C Servo Headers
- I2C ready w/pull-up resistors installed
- Up to 51 digital I/O pins
- Clock frequency of 14.7456 MHz
- Voltage regulator on-board accepts 5.5-15V
- Small size at 2.2 x 3.6 inches



Mobile Platform

In order to allow Atlas to complete its objective, he would need to have a few physical requirements met. Atlas would have to hold a ball platform that would have 2 degrees of freedom, hold a camera above that platform, and have someway to attach motors. To be completely honest, I didn't have any idea how I was going to do any of those things when I had the body cut out on the Ttech. I just made a square box that was big enough to attach things to. At the time I just knew that I needed to get started building. The downside to this approach is that Atlas is probably much bigger than he needs to be. Below is a comparison of my original design to what was actually built.

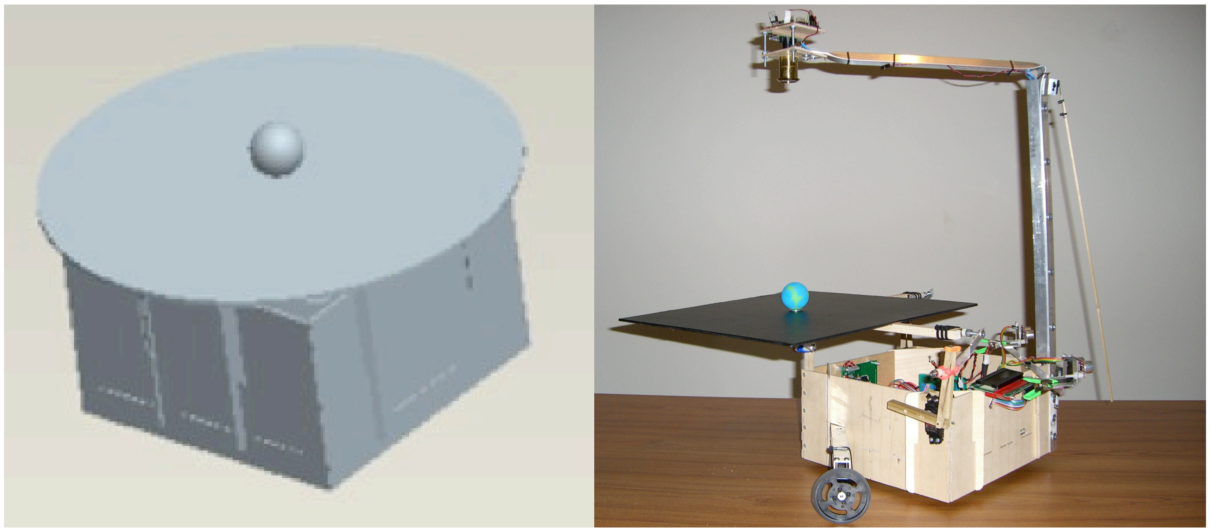


Fig 2: Comparison of Original Design VS. Final VersionActuation

Atlas has two types of actuation systems, one for ball balancing and the other for locomotion.

Ball Balancing

This was the most mechanically challenging part of building this robot. I considered many options but eventually decided that using two servos located on each axis was the best way to do it. The servos were placed at the edge of the platform and two arms were used with another mechanical piece that I made to allow each servo to move with out interference from the other servo. Below are pictures of this system

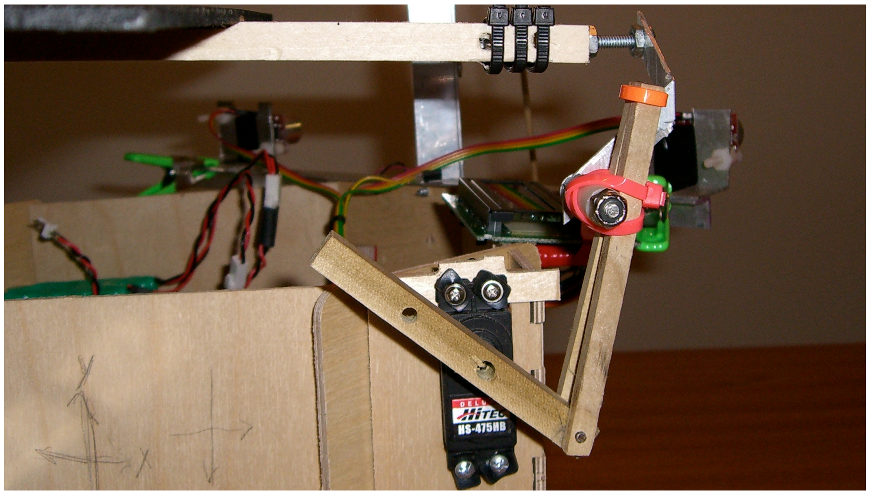


Fig 3: Fig#:Tilting System for Platform

The servos that I used are HS-475HB. Specifications for these servos are :

- Control System: +Pulse Width Control 1500usec Neutral
- Required Pulse: 3-5 Volt Peak to Peak Square Wave
- Operating Voltage: 4.8-6.0 Volts
- Operating Speed (6.0V): 0.18sec/60 degrees at no load
- Stall Torque (6.0V): 76 oz/in. (5.5kg.cm)
- Current Drain (6.0V): 8.8mA/idle and 180mA no load operating
- Gear Type: Karbonite Gears

Obstacle Avoidance

This actuation was much easier to develop. I simply added a beam through my robot to mount the motors to. I also added a caster wheel to front of my robot. Sometimes this caster wheel will get stuck and cause Atlas to stop, But this happens infrequently and a little push will get it started again.

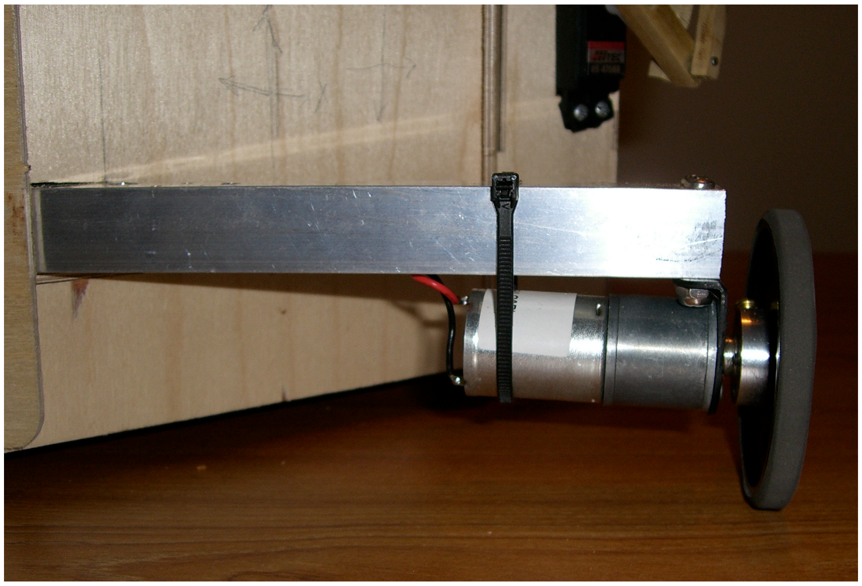


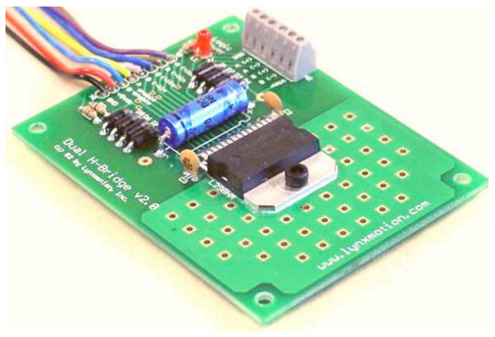
Fig 4: Motors Mounting Configuration

The motors that I used are MT-200 from BatterySpace.com. The specifications for these motors are:

- Horse Power Cont: 0.8 W (60 mA no load)
- Toque: 3.3 Lb-Inch
- Gear Ratio: 1:20
- Length of Motor (excluding spindle):54 mm
- Diameter of Motor: 25 mm
- Voltage & Current: 12 V DC
- Length of Spindle: 8 mm
- RPM: 200

These motors were controlled by the micro-controller through a motor driver from Lynxmotion. The specifications for this motor controller are:

- 4.8v - 12vdc
- 4 amp Peak
- 2 amp Continuous
- Dual Channel, TTL Input
- Chip rated current = 2 amps max
- I/O required = Four TTL level lines (outputs)
- Logic voltage = 5vdc regulated
- Onboard regulator = None
- Current requirements (5v) = 36mA
- PC board size = 2.3" x 3.0"



One lesson that I leaned is to buy motors with metal gears. I stripped both of my plastic gear motors within an hour of getting them.

Sensors

Atlas was built using three different types sensors:

Vision

For the image processing system I chose the CMUCam. More Information on this device can be found in my Special Sensor Report in the appendix.

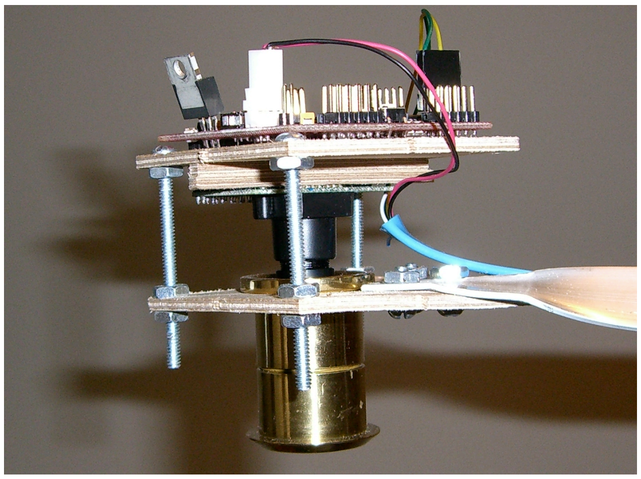


Fig 5: Close-up of Camera Mount.

Sonar

The sonars that I used are two SRF08. I chose these because of there I2C interface. Having never used I2C, I was curious as to how it worked. I must say from my experience with the sonar, that they are a little hard to get set-up correctly. I had to add a capacitor to the power input in order to satisfy the sonar's instantaneous need for current when it would ping. Otherwise I would get errors.

After I got them working correctly, they produced very good data. It's rare that Atlas doesn't detect an obstacle before hitting it.

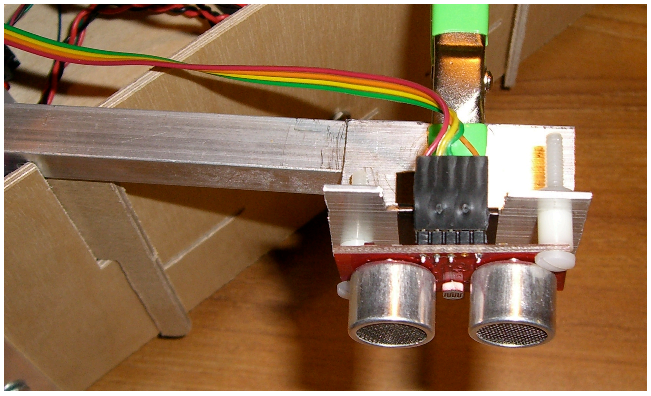


Fig 6: Close-up of Sonar

Sonar rangers work by emitting a high frequency sound wave and measuring the time delay for its echo to return. This time delay is directly proportional to the distance of the object. This particular sonar takes care of all of these actions via its own on board processor. All that is required for it to work is to send an command on the I2C bus, wait ~65ms and read the distance value off the I2C bus. This integer value can be represented in inches, cm, or ms.

Bump Switch

For those few and far between times when the sonar doesn't see something, Atlas will need to know when it's touching something. For this reason I attached a long stick to a limit switch and mounted it to Atlas's camera arm.

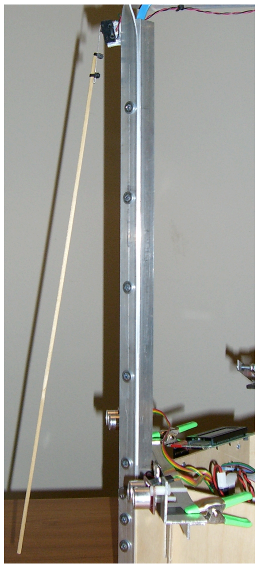


Fig 7: Close-up of Bump Switch

Behaviors

Atlas performs two behaviors, ball balancing and obstacle avoidance. Both of which are completely independent of each other.

Ball Balancing

This, by far, was the most complicated and hardest part of this robot. Balancing a ball on a platform is really child's play, for us. For a robot though, everything has to be broken down into numbers that it can perform math on. What Atlas needs to needs to know in order to control the ball is at most, three things; position, velocity, and acceleration. Since velocity and acceleration can be derived from position, that is the only real measurement that Atlas needs to make. So the ball balancing behavior goes as follows:

1. Take position measurement from camera.
2. Derive velocity from previous position measurements.
3. Multiply them both by constants and add them together.
4. Move the servos to the new position.

This algorithm works so well that I didn't even need to add acceleration to the equation. The ball usually settles within a few seconds.

Obstacle Avoidance

obstacle avoidance was a mush easier task to complete. Motor speed and direction were controlled by sonar and bump switch values. The behavior for obsticle avoidance goes as follows:

1. Ping sonar.
2. Read sonar and bump switch values.
3. If sonar reading is below a given threshold or if bump switch is pressed, stop.
4. Back up.
5. Turn in the appropriate direction.
6. Move forward.

When the above actions require a change in motor speed, a smoothing function is applied to make sure Atlas doesn't change direction too quickly. This provides a much nicer environment for the ball balancing behavior.

Experimental Layout and Results

Before fully assembling the robot, each sensor and behavior was tested individually.

Vision

The CMUCam2 vision system was by far the most throughly tested component of this robot. I spent a great amount of time testing this camera with my computer trying to determine what kind of background and ball color combination would work the best. I tried two different approaches; white background with a dark colored ball and black background with a light colored ball. Both methods seemed to produce similar results when tested in optimal lighting conditions. But when forced to deal with lower lighting and shadows, the black background turned out to be the winner. If the lighter colored ball is darkened by a shadow, the worst case problem would be that the camera would not see the ball. Although this is very bad, it is not as detrimental as what happens with a white background. If a shadow is cast on a white background the worst thing that could happen is that the camera thinks the shadow is the ball. I'd much rather the ball be lost momentary then for the camera think that the ball has jumped from one side of the platform to the other. In the first case the ball may roll off, in the second case it is thrown off.

Sonar

After the sonar ranger was working properly, there was no real need to test it that throughly. Because the SRF08's on bard processor gives the results in the units of cm, inches, or uS, all I needed to do was confirm that is was correct; It was.

Bump Switch

I wrote a simple test that would display the results of pin 3 on port D. After I connected the bump switch, I confirmed that it was giving me the correct results.

Conclusions

I am very happy to say that Atlas does almost everything that I wanted it to do. I can place a ball on top of the platform with little fear of it falling off. If I were able to redo this project, I would do somethings differently. First, and most importantly, I would not use the CMUcam. This, by far, is the weakest link in Atlas. Whenever Atlas does drop the ball, most of the time it is because the camera was unable to find it correctly. I've tried illuminating the platform with a few different methods but was unable to find a method that would create even and consistent lighting conditions across the entire platform.

Another thing that I would do is spend more time on the control algorithm. Most of the time the ball will not return to the exact center. There is usually some error that I blame on a variety of things; such as the ping pong ball having a large seam and the friction between platform and the ball. Dynamically changing the coordinates of the center position would allow me to program Atlas to "draw" shapes with the ball. Unfortunately I was unable to control the ball precisely enough to do that.

Documentation

(1) AVR Freaks

www.avrfreaks.net

(2) Sparkfun Electronics

www.sparkfun.com

(3) BDMicro Mavric128 Development Board

www.bdmicro.com

(4) CMUCam

http://www.seattlerobotics.com/

(5) CodeVision

http://www.hpinfotech.ro/

Appendices

Appendix A:

Source Code

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.6 Professional
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project :
Version :
Date    : 7/29/2006
Author  : Jeremy Greene
Company :
Comments:


Chip type           : ATmega128
Program type        : Application
Clock frequency     : 14.745600 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 1024
*****************************************************/

/////////////////////////////////

#define FrontBumpSwitch  PIND.4
#define LeftIR  PINA.0
#define RightIR PINA.1
#define BothSonar 0x00
#define LeftSonar 0xE0
#define RightSonar 0xFe

#define TurnSpeed 60
#define FullSpeed 65
#define BackSpeed -65
#define LeftWheelTweak  0
#define RightWheelTweak 0
#define BackUpTime 8
#define TurnTime   15


#include <mega128.h>
#include <math.h>
#include <string.h>

// I2C Bus functions
#asm
   .equ __i2c_port=0x12 ;PORTD
   .equ __sda_bit=1
   .equ __scl_bit=0
#endasm
#include <i2c.h>
// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
```

```
#endasm
#include <stdlib.h>
#include <lcd.h>
#include <delay.h>

// Standard Input/Output functions
#include <stdio.h>

// Declare your global variables here
int ps = 25; //PacketSize
int ColorTolerance = 50;
unsigned char CamPacket[25];
char Tx[1];
int i,j,cr,x,y,z,r,g,b,SignOfx,SignOfy,Px,Py,BufferSize,Vx,Vy;
int PrevX,PrevY,ServoX,ServoY,DirectionX,DirectionY,MovingToEdgeOrOrigin;
int PrevDelay;
int TrackingColor[5]; //Rmin,Rmax,Gmin,Gmax,Bmin,Bmax
int WhiteSpace[4];
int Avg_MovingToEdgeOrOrigin_sum,Avg_MovingToEdgeOrOrigin,Avg_Counter,AvgMax;
int YOffset,XOffset,ServoXWithOffset,ServoYWithOffset,ServoPositionX,ServoPositionY,TweakX,TweakY;
unsigned char ColorToTrack[35];
int PrevXPoints[11],PrevYPoints[11];
int MSR,NewMSR,KR;
int MSL,NewMSL,KL;
int NoObsticalCounter,TurnDirection;
//int TurnSpeed,FullSpeed,BackSpeed,LeftWheelTweak,RightWheelTweak;
//int BackUpTime,TurnTime;
int LeftSonarDistance,RightSonarDistance;
int TestCount,ObstacleAvoidCount;


void lcd_PrintInt(int number)
{
char character[1];
      itoa(number, character);
      lcd_puts(character);
 }

 void WriteMyName(void)
{
        lcd_clear();
        lcd_putsf("Jeremy Greene");
        lcd_gotoxy(0,1);
        lcd_putsf("Is My Name ");
        delay_ms(500);
}

void GotHere(void)
{
       // lcd_clear();
        lcd_putsf("got this far");
        //delay_ms(50);
       // lcd_clear();
}

 void GetSonarData(void) {
unsigned char data;


//Read Value Left
i2c_start();
i2c_write(LeftSonar);  //Sonar address
i2c_write(0x03);
i2c_start();
i2c_write(LeftSonar | 1);
data=i2c_read(0); //Distance value
i2c_stop();

if(data != 128){LeftSonarDistance = data - 129;}



//Read Value Right
i2c_start();
i2c_write(RightSonar);  //Sonar address
i2c_write(0x03);
i2c_start();
i2c_write(RightSonar | 1);
data=i2c_read(0); //Distance value
i2c_stop();

if(data != 128){RightSonarDistance = data - 129;}



i2c_start();
i2c_write(BothSonar); //Sonar address
i2c_write(0x00); //Sonar Command Reg Address
i2c_write(0x51); //Value in (0x50 for inches)(0x51 for cm)(0x52 for uS)
i2c_stop();

}

 void ChangeSonarAddress(void) {

i2c_start();
i2c_write(0xE0); //Sonar address
i2c_write(0x00); //Sonar Command Reg Address
i2c_write(0xA0);
i2c_stop();

delay_ms(100);

i2c_start();
i2c_write(0xE0); //Sonar address
i2c_write(0x00); //Sonar Command Reg Address
i2c_write(0xAA);
i2c_stop();

delay_ms(100);

i2c_start();
i2c_write(0xE0); //Sonar address
i2c_write(0x00); //Sonar Command Reg Address
i2c_write(0xA5);
i2c_stop();

delay_ms(100);

i2c_start();
i2c_write(0xE0); //Sonar address
i2c_write(0x00); //Sonar Command Reg Address
i2c_write(0xE4);
i2c_stop();

}

////////////////////////////////
```

```
void TxTrackColorToCamera()
{
unsigned char character[6];

/*lcd_putsf("tc");  //old way
printf("tc");
for(i=0; i< 6; i++)
    {
    printf(" ");
      itoa(TrackingColor[i], character);
      puts(character);
    lcd_putsf(" ");
    lcd_puts(character);
      }
 printf('\r');
 lcd_putsf("\r");

 delay_ms(10000);*/

 //new way
 lcd_clear();
ColorToTrack[0]='t';
ColorToTrack[1]='c';

for(i=0; i< 6; i++)
    {
      itoa(TrackingColor[i], character);
      //strcat(ColorToTrack,32);
      ColorToTrack[strlen(ColorToTrack)]=' ';

      strcat(ColorToTrack,character);
      }

  ColorToTrack[strlen(ColorToTrack)]='\r';

      //lcd_putsf("Length=");
      //lcd_PrintInt(strlen(ColorToTrack));

       for(i=0; i<strlen(ColorToTrack); i++)
      {
      putchar(ColorToTrack[i]);
      }
      //printf(ColorToTrack[]);
 delay_ms(1000);


 }

void GetCamPacketAndFormat()
{

// Declare your local variables here

int d[4];
int num[4];
       for(i=0; i< ps; i++)//Get camera packet
      {
      CamPacket[i]=getchar();
      }

      for(i=0; i< ps; i++) //Find carriage return (cr)
      {
```

```
      if(CamPacket[i]==13) {   //13 is ascii for cr
       cr=i;
       break;
       }
      }

      j=0;
      for(i=cr; i<ps; i++)  //Find first 3 white spaces after cr
      {
      if(CamPacket[i]==' ') {
        WhiteSpace[j]=i;
        if(j==3) {
         break;
        }
        else
         {
         j++;
         }
        }
      }


      d[0]= ((WhiteSpace[1] - WhiteSpace[0]) - 1);
      d[1]= ((WhiteSpace[2] - WhiteSpace[1]) - 1);
      d[2]= ((WhiteSpace[3] - WhiteSpace[2]) - 1);

      if (d[0]==3) { //num0 calculation
          num[0] = 100*(CamPacket[WhiteSpace[0]+1] - 48) + 10*(CamPacket[WhiteSpace[0]+2] - 48) + (CamPack-
et[WhiteSpace[0]+3] - 48);
        }
      else if (d[0]==2) {
          num[0] = 10*(CamPacket[WhiteSpace[0]+1] - 48) + (CamPacket[WhiteSpace[0]+2] - 48);
        }

      if (d[1]==3) { //num1 calculation
          num[1] = 100*(CamPacket[WhiteSpace[1]+1] - 48) + 10*(CamPacket[WhiteSpace[1]+2] - 48) + (CamPack-
et[WhiteSpace[1]+3] - 48);
        }
      else if (d[1]==2) {
          num[1] = 10*(CamPacket[WhiteSpace[1]+1] - 48) + (CamPacket[WhiteSpace[1]+2] - 48);
        }

      if (d[2]==3) { //num2 calculation
          num[2] = 100*(CamPacket[WhiteSpace[2]+1] - 48) + 10*(CamPacket[WhiteSpace[2]+2] - 48) + (CamPack-
et[WhiteSpace[2]+3] - 48);
        }
      else if (d[2]==2) {
          num[2] = 10*(CamPacket[WhiteSpace[2]+1] - 48) + (CamPacket[WhiteSpace[2]+2] - 48);
        }

      if(CamPacket[cr+1]=='S') {
      r=num[0];
      g=num[1];
      b=num[2];
      }

      if(CamPacket[cr+1]=='T') {

      x=num[0]-98;
      y=num[1]-170;
      SignOfx=sign(x);
      SignOfy=sign(y);
      x=abs(x);
      y=abs(y);
      x=x*x/10;
      y=y*y/50;
      x=x*SignOfx;
      y=y*SignOfy;
      num[2]=x;//used num[2] so to save a variable
      x=y;
      y=num[2];
      }
 }

void SetUpCamera(void)
{
   printf("hr 1\r");
   delay_ms(250);
  // printf("vw 100 170 105 184\r"); //Round
  printf("vw 99 159 104 173\r"); //Square

 delay_ms(500);

   printf("gm\r");
   delay_ms(250);

/////////////////////////
 while(FrontBumpSwitch == 0)  {
   GetCamPacketAndFormat();
      lcd_clear();
      lcd_gotoxy(0,0);lcd_putsf("r=");lcd_PrintInt(r);
      lcd_gotoxy(5,0);lcd_putsf(" g=");lcd_PrintInt(g);
      lcd_gotoxy(11,0);lcd_putsf(" b=");lcd_PrintInt(b);
      delay_ms(400);
  }
   TrackingColor[0] = r - ColorTolerance;
   TrackingColor[1] = r + ColorTolerance;
   TrackingColor[2] = g - ColorTolerance;
   TrackingColor[3] = g + ColorTolerance;
   TrackingColor[4] = b - ColorTolerance;
   TrackingColor[5] = b + ColorTolerance;

    for(i=0; i< 6; i++)  //test for min and max
    {
      if(TrackingColor[i]>255){
      TrackingColor[i] = 255;
      }

      if(TrackingColor[i]<0){
      TrackingColor[i] = 0;
      }
     }
   lcd_gotoxy(0,0);lcd_putsf("  ");lcd_PrintInt(TrackingColor[0]);
   lcd_gotoxy(5,0);lcd_putsf("   ");lcd_PrintInt(TrackingColor[2]);
   lcd_gotoxy(11,0);lcd_putsf("   ");lcd_PrintInt(TrackingColor[4]);

   lcd_gotoxy(0,1);lcd_putsf("  ");lcd_PrintInt(TrackingColor[1]);
   lcd_gotoxy(5,1);lcd_putsf("   ");lcd_PrintInt(TrackingColor[3]);
   lcd_gotoxy(11,1);lcd_putsf("   ");lcd_PrintInt(TrackingColor[5]);

 delay_ms(1000);

   printf("rs\r");
   delay_ms(200);
   printf("hr 1\r");
   delay_ms(200);
   //printf("vw 73 120 125 234\r"); // tracking window round
   printf("vw 77 115 125 219\r"); // tracking window square
   delay_ms(50);

TxTrackColorToCamera();

  // printf("tc 105 165 210 254 190 250\r");
   delay_ms(50);
}

void TestServos(void)
{

while(1){
 OCR0= 43;
 OCR2= 53;
 }
 OCR0= 33;
 OCR2= 43;
  delay_ms(100);
 OCR0= 43;
 OCR2= 33;
  delay_ms(100);
 OCR0= 53;
 OCR2= 43;
 delay_ms(100);
 OCR0= 43;
 OCR2= 53;
 delay_ms(100);

}

void FindVelocity(void)
{

 //GetVelocity
BufferSize=3;
PrevXPoints[BufferSize-1]=PrevX;
PrevYPoints[BufferSize-1]=PrevY;

for(i=0; i<BufferSize-1; i++)
      {
      PrevXPoints[i]=PrevXPoints[i+1];
      PrevYPoints[i]=PrevYPoints[i+1];
      }

      Vx=0;
      Vy=0;
for(i=0; i<BufferSize-1; i++)
      {
      Vx=Vx+(PrevXPoints[i+1]-PrevXPoints[i]);
      Vy=Vy+(PrevYPoints[i+1]-PrevYPoints[i]);
      }
}

void MoveServos(void)
{

Px=(43+(-x/4));
Py=43+y/3;

ServoPositionX = Px+3-(1.5*Vx);
ServoPositionY = (Py+2)+(1.5*Vy);

 if(x>0){ServoPositionX=(ServoPositionX-43)/1.3+43;};
 if(y>0){ServoPositionY=(ServoPositionY-43)/1.3+43;};
 TweakX=-4;
 TweakY=-2;

OCR2= ServoPositionX+TweakX;
OCR0= ServoPositionY+TweakY;

 PrevX=x;
 PrevY=y;
}

void BalanceBall(void)
{
GetCamPacketAndFormat();
FindVelocity();
MoveServos();
}

void ObstacleAvoid(void)
{
 KL = 6;
 KR = 6;

  GetSonarData();

   //Obstical Advoidance Code

  //Bump Switches
      if (FrontBumpSwitch==1 && NoObsticalCounter==(BackUpTime+TurnTime)) {   //stop, go back
        MSL=BackSpeed+LeftWheelTweak;
        MSR=BackSpeed+RightWheelTweak;
        NoObsticalCounter=0;
        TurnDirection=1;  // 0=left 1=right
        KL=10;
        KR=10;
         };

      if (LeftSonarDistance<20 && NoObsticalCounter==(BackUpTime+TurnTime)) {   //stop, go back
        MSL=BackSpeed+LeftWheelTweak;
        MSR=BackSpeed+RightWheelTweak;
        NoObsticalCounter=0;
        TurnDirection=1;  // 0=left 1=right
        KL=10;
        KR=10;
         };


      if (RightSonarDistance<20 && NoObsticalCounter==(BackUpTime+TurnTime)) {   //stop, go back
        MSL=BackSpeed+LeftWheelTweak;
        MSR=BackSpeed+RightWheelTweak;
        NoObsticalCounter=0;
        TurnDirection=0;  // 0=left 1=right
        KL=10;
        KR=10;
         };

     if ((NoObsticalCounter > BackUpTime) && (TurnDirection=0)) { //turn
        MSR=TurnSpeed+RightWheelTweak;
        MSL=-1*TurnSpeed+LeftWheelTweak;}
     else if((NoObsticalCounter > BackUpTime) && (TurnDirection=1)){
        MSR=-1*TurnSpeed+RightWheelTweak;
        MSL=TurnSpeed+LeftWheelTweak;};
```

```
       if (NoObsticalCounter > BackUpTime+TurnTime-1) {
        MSR=FullSpeed+RightWheelTweak;
        MSL=FullSpeed+LeftWheelTweak;
        NoObsticalCounter=BackUpTime+TurnTime-1;};

NoObsticalCounter++;

//Motor Controlling Code (for smooth motors)

//Left Motor
if (NewMSL > MSL){NewMSL= NewMSL + (MSL-NewMSL)/KL;}
if (NewMSL < MSL){NewMSL= NewMSL - (NewMSL-MSL)/KL;}
if (NewMSL > MSL){ NewMSL--;}
if (NewMSL < MSL){ NewMSL++;}

//Right Motor
if (NewMSR > MSR){NewMSR= NewMSR + (MSR-NewMSR)/KR;}
if (NewMSR < MSR){NewMSR= NewMSR - (NewMSR-MSR)/KR;}
if (NewMSR > MSR){ NewMSR--;}
if (NewMSR < MSR){ NewMSR++;}

      if (sign(NewMSL)==1){
      PORTB.0=1;
      PORTB.1=0;      }
else if (sign(NewMSL)==255) {
      PORTB.0=0;
      PORTB.1=1;
      }
else if (sign(NewMSL)==0){
      PORTB.0=0;
      PORTB.1=0;
      } ;

     if (sign(NewMSR)==1){
      PORTB.2=1;
      PORTB.3=0;;
      }
else if (sign(NewMSR)==255) {
      PORTB.2=0;
      PORTB.3=1;
      }
else if (sign(NewMSR)==0){
      PORTB.2=0;
      PORTB.3=0;
      } ;

OCR1AL=abs(NewMSL);
OCR1BL=abs(NewMSR);

}
// Timer 3 output compare A interrupt service routine
interrupt [TIM3_COMPA] void timer3_compa_isr(void)
{

//Numbers are out of 255
 KL = 4;
 KR = 4;

  GetSonarData();

   //Obstical Advoidance Code

  //Bump Switches
     if (FrontBumpSwitch==1 && NoObsticalCounter==(BackUpTime+TurnTime)) {   //stop, go back
```

```
        MSL=BackSpeed+LeftWheelTweak;
        MSR=BackSpeed+RightWheelTweak;
        NoObsticalCounter=0;
        TurnDirection=1;  // 0=left 1=right
        KL=10;
        KR=10;
         };

      if (LeftSonarDistance<20 && NoObsticalCounter==(BackUpTime+TurnTime)) {   //stop, go back
        MSL=BackSpeed+LeftWheelTweak;
        MSR=BackSpeed+RightWheelTweak;
        NoObsticalCounter=0;
        TurnDirection=1;  // 0=left 1=right
        KL=10;
        KR=10;
         };


      if (RightSonarDistance<20 && NoObsticalCounter==(BackUpTime+TurnTime)) {   //stop, go back
        MSL=BackSpeed+LeftWheelTweak;
        MSR=BackSpeed+RightWheelTweak;
        NoObsticalCounter=0;
        TurnDirection=0;  // 0=left 1=right
        KL=10;
        KR=10;
         };

     if ((NoObsticalCounter > BackUpTime) && (TurnDirection=0)) { //turn
        MSR=TurnSpeed+RightWheelTweak;
        MSL=-1*TurnSpeed+LeftWheelTweak;}
     else if((NoObsticalCounter > BackUpTime) && (TurnDirection=1)){
        MSR=-1*TurnSpeed+RightWheelTweak;
        MSL=TurnSpeed+LeftWheelTweak;};


       if (NoObsticalCounter > BackUpTime+TurnTime-1) {
        MSR=FullSpeed+RightWheelTweak;
        MSL=FullSpeed+LeftWheelTweak;
        NoObsticalCounter=BackUpTime+TurnTime-1;};

NoObsticalCounter++;




//Motor Controlling Code (for smooth motors)

//Left Motor
if (NewMSL > MSL){NewMSL= NewMSL + (MSL-NewMSL)/KL;}
if (NewMSL < MSL){NewMSL= NewMSL - (NewMSL-MSL)/KL;}
if (NewMSL > MSL){ NewMSL--;}
if (NewMSL < MSL){ NewMSL++;}

//Right Motor
if (NewMSR > MSR){NewMSR= NewMSR + (MSR-NewMSR)/KR;}
if (NewMSR < MSR){NewMSR= NewMSR - (NewMSR-MSR)/KR;}
if (NewMSR > MSR){ NewMSR--;}
if (NewMSR < MSR){ NewMSR++;}


     if (sign(NewMSL)==1){
      PORTB.0=1;
      PORTB.1=0;      }
else if (sign(NewMSL)==255) {
      PORTB.0=0;
      PORTB.1=1;
      }
else if (sign(NewMSL)==0){
      PORTB.0=0;
      PORTB.1=0;
      } ;

     if (sign(NewMSR)==1){
      PORTB.2=1;
      PORTB.3=0;;
      }
else if (sign(NewMSR)==255) {
      PORTB.2=0;
      PORTB.3=1;
      }
else if (sign(NewMSR)==0){
      PORTB.2=0;
      PORTB.3=0;
      } ;




OCR1AL=abs(NewMSL);
OCR1BL=abs(NewMSR);




TCNT3H=0x00;
TCNT3L=0x01;
}

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTB=0x00;
DDRB=0xFF;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;
```

```
// Port E initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x00;

// Port F initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTF=0x00;
DDRF=0x00;

// Port G initialization
// Func4=In Func3=In Func2=In Func1=In Func0=In
// State4=T State3=T State2=T State1=T State0=T
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 57.600 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
ASSR=0x00;
TCCR0=0x66;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 57.600 kHz
// Mode: Ph. correct PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0xA1;
TCCR1B=0x04;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: 57.600 kHz
// Mode: Phase correct PWM top=FFh
// OC2 output: Non-Inverted PWM
TCCR2=0x64;
TCNT2=0x00;
OCR2=0x00;
```

```
// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: 14.400 kHz
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x05;
TCNT3H=0x00;///////
TCNT3L=0x00;///////
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x07;///3
OCR3AL=0xF0;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x10;

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: On
// USART0 Mode: Asynchronous
// USART0 Baud rate: 115200
UCSR0A=0x00;
UCSR0B=0x18;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x07;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// I2C Bus initialization
```

```
i2c_init();

// LCD module initialization
lcd_init(20);


// My code starts here!!!!!!!!!!!

 //TestServos();

 OCR0= 43;;// y
 OCR2= 43;;// x
//delay_ms(5000);


WriteMyName();

SetUpCamera();

GotHere();
while(FrontBumpSwitch == 0)  {
lcd_clear();
lcd_putsf("Ready To Go!");
delay_ms(100);
}

// Global enable interrupts
//#asm("sei")

while (1)
      {
      BalanceBall();
      lcd_clear();

      lcd_putsf("L");
      lcd_PrintInt(LeftSonarDistance);
      lcd_gotoxy(0,1);
      lcd_putsf("R");
      lcd_PrintInt(RightSonarDistance);

      lcd_gotoxy(3,0);
      lcd_putsf("l");
      lcd_gotoxy(3,1);
      lcd_putsf("l");


      lcd_gotoxy(4,0);
      lcd_putsf("x");
      lcd_PrintInt(x);
      lcd_gotoxy(4,1);
      lcd_putsf("y");
      lcd_PrintInt(y);

      lcd_gotoxy(8,0);
      lcd_putsf("l");
      lcd_gotoxy(8,1);
      lcd_putsf("l");

      lcd_gotoxy(9,0);
      lcd_putsf("X");
      lcd_PrintInt(OCR2);
      lcd_gotoxy(9,1);
      lcd_putsf("Y");
      lcd_PrintInt(OCR0);
```

```
    lcd_gotoxy(13,0);
    lcd_putsf("l");
    lcd_gotoxy(13,1);
    lcd_putsf("l");

  /*   lcd_gotoxy(14,0);
    lcd_putsf("Vx");
    lcd_PrintInt(Vx);
    lcd_gotoxy(14,1);
    lcd_putsf("Vy");
    lcd_PrintInt(Vy);
    */

     lcd_gotoxy(14,0);
     lcd_PrintInt(TestCount);


     if (ObstacleAvoidCount == 3){
     ObstacleAvoid();
     ObstacleAvoidCount=0;

     TestCount++ ;
     }
     ObstacleAvoidCount++;
    };
}
```

Date: August 7, 2006
Student Name: Jeremy Greene
TAs : Adam Barnet
William Otto Goethals
Instructors: A. A Arroyo
E. M. Schwartz

University of Florida

Department of Electrical and Computer Engineering

EEL 5666 Intelligent Machines Design Laboratory

Formal Proposal

Special Sensor Report: CMUcam2 Vision System

Table of Contents

Introduction

Delivery Boy will need to be able to have knowledge of where the ball is located on the platform. The best way that this 2-dimensional tracking can be done is with the use of a camera. I chose the CMUcam2 to meet this requirement.

The CMUcam2 is a low resolution camera that is interfaced with a microcontroller. This allows high level data to be accessed from the camera. Although the cameras quality leaves much to be desired, I do believe that it will be able to provide Delivery Boy with the necessary ability to track a ball.

Purpose

Delivery Boy will need to be able to keep a ball on it's platform while moving around a room. A key part of accomplishing this tack will be knowing the location of the ball. The purpose of this camera will be to provide Delivery Boy with this information. The CMUcam2 will be placed so that is can view the entire platform. One advantage that I will have while using this camera is that I will have control over what is in view of the camera. That is, the camera will only be looking at the ball and the platform.

Benefits

The CMUcam2 does provide many benefits that makes it an attractive candidate for image processing. The first reason that I chose this camera is its size. Although the camera did not provide a good mounting option, It is small and light enough to place on a stand over the platform. Another benefit this camera provides is the on-board image processing. This allows me to simply send commands to the camera such as TC (track color) and the camera will stream back the position of the color that is being tracked. Cost was another factor in choosing a camera. At $160, the camera does provide an affordable vision solution.

Problems

The main problem that I have found while using this camera is that it requires a great amount of light in order to properly differentiate colors. In normal lighting conditions, everything the camera sees has a dark red tone. This must be addressed before any reliable data can be obtained. My only other complaint is that the camera does not seem to be designed so that it can be mounted easily. There are only two holes available and they are too small to fit normal board-mounting screws into them. Also, the actual camera is located on a separate chip and the only thing holding the two together are the header pins. So even if you are able to securely mount the CMUCam with the two holes provided, the actual camera is still able to rock back and fourth with no way of securing it.

Interface

The CMUCam's communication interface turned out to be very easy to use. It is controlled and provides data by the use of serial data transmission. It is configurable to use seven different baud rates and can accept or send data at either TTL levels (0V to 5V) or RS-232 (-12V to12V) levels. For example, If tracking a color was the objective of the camera, the command could be sent as:

```
TC 200 255 0 65 20 72/r
```

This would tell the camera to Track the Color which red value is in between 200 and 255 and green color is in between 0 and 65 and blue color is in between 20 and 72. The camera would then return a packet that would look like this:

```
T 23 45/r
```

This tells the user that the object that was to be tracked is located on in the cameras view on (23,45). That is, 23 pixels to the right and 45 pixels down.

Conclusion

Although the CMUcam may not be the best choice for complicated image processing, It should provide Delivery Boy with enough information to be able to complete its task, balancing a ball on top of its platform.