Date: *12/6/10*
Student Name: Anup Parikh
TAs : Mike Pridgen
Tim Martin

Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

University of Florida
Department of Electrical and Computer Engineering
EEL 4665/5666
Intelligent Machines Design Laboratory

Final Formal Written Report

Robot: Minesweeper

Table of Contents

Abstract

This report details the design and testing of an autonomous robot that seeks and disables mines in the form of LED beacons. The robot uses the Boe-Bot CMUcam with built in image processing to determine the relative location of the mine with respect to the robot by tracking a blue colored blob. The movement of the robot is controlled with visual servoing (proportional control). Once the robot is within range of the beacon, a Hall Effect sensor on the beacon is triggered by the magnetic field from a magnet mounted on the robotic platform, disabling the mine. After deactivation, the robot backs away and continues searching for other active beacons.

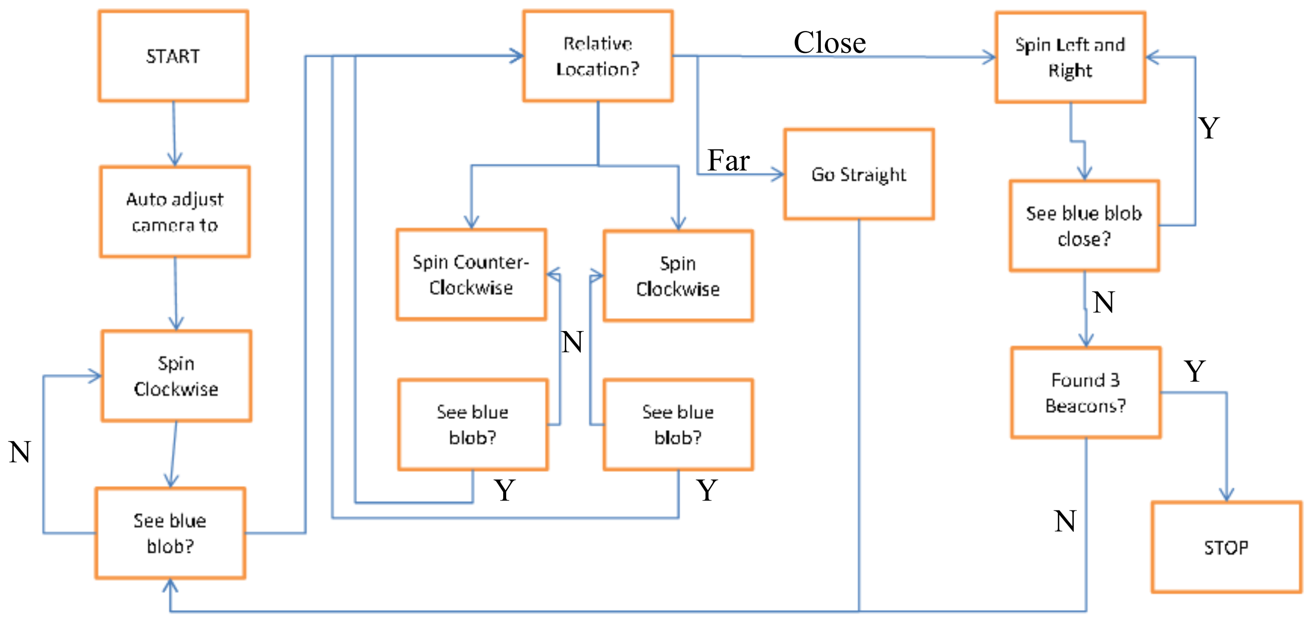
The robot was successful in locating and deactivating the mines. The main issues with the robot's performance were mutual interference between the sonar modules and false positive detections by the camera. Methods for eliminating these problems and future improvements are also outlined.

Introduction

The purpose of the project is to create a robot that is able to find and disable mines. The mines are simple light beacons that turn off when a magnetic field is applied to them. The robot uses a camera to determine the direction to move in order to reach the mines, and then waves a small magnet over them in order to deactivate them. Sonar rangefinders and bump sensors are used to avoid obstacles and walls in the field.

Integrated Systems

Process Flow



Mobile Platform and Drive System

The platform consists of a stack two circular plates made of wood, separated by 2 inch spacers. The servos are mounted onto the sides of the bottom plate, where flats have been cut. The microcontroller board is mounted on the top plate for easy access. The sonar, camera, and magnet arm are also mounted onto the top plate. By mounting the sonar high, it should not sense the small beacons. A schematic of the robot platform is shown in Appendix A.

The mobility of the robot is achieved through the use of 2 continuous servos attached to 3 inch wheels. The servos were originally restricted to only 90° rotation, but the potentiometer and other limiting components were removed to allow for continuous rotation. A metal ball caster used on the back of the platform to allow for easy rotation and linear motion. When the robot is first activated, the servos run in opposite directions to allow the robot to spin until the camera "sees" a beacon. Proportional control (or visual servoing) was implemented in moving the robot to the beacon, where the input error was the offset of the beacon from camera’s center of field of view.

Sensors

Camera

The camera used on the robot is the Boe-Bot CMUcam. This camera can capture up to 17 frames per second at a resolution of 80x143. More importantly, this camera has image processing circuitry built-in, therefore the location of a color blob can be easily determined with only a few commands. The camera communicates with the controller via RS232.

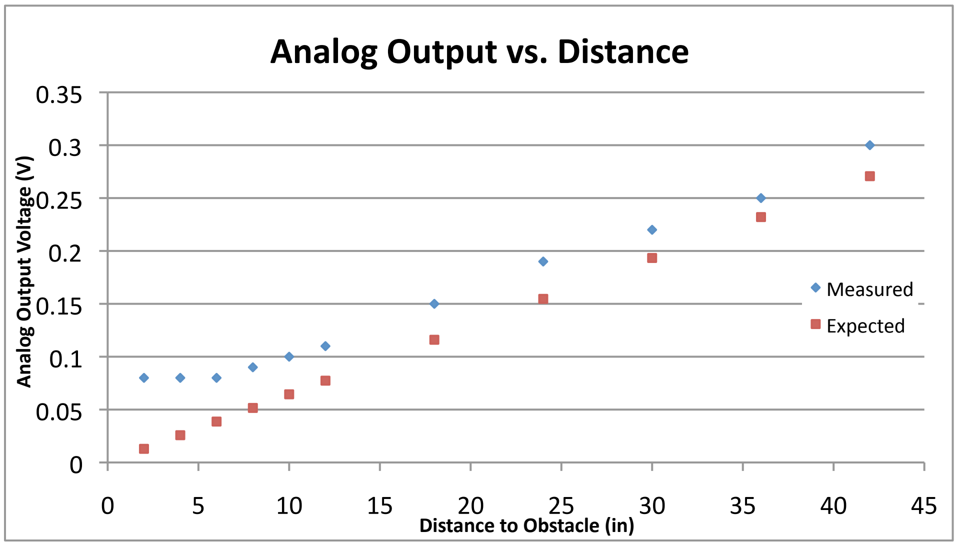
After communication was established, 2 calibration steps were taken. The lens focus was adjusted in order to get the sharpest images and therefore the most accurate beacon locations. Also, the white balance and auto-gain are adjusted for the lighting conditions. The robot automatically performs this calibration upon activation. This causes a 5 second delay before the robot begins searching for a beacon.

The camera used in this robot was a modified version of the original CMUcam. As a result, circuitry had to be added to bring the native TTL level transmission lines to RS232 levels. Communication with the camera with UART was also quite difficult to establish, however once established, the camera performed quite well and was able to track objects at a considerable distance at a fast rate. The limiting factor in tracking the correct object was determining the narrowest RGB color tracking bandwidths the object would be in. Too wide a range results in many false positives, while too narrow a range results in the camera not detecting the object at all.

Sonar

Sonar rangefinders continuously monitor the distance to the nearest object in the field of view of the sensors. When they sense an object about 6 inches from the robot, the robot turns away from the object and then continues finding beacons. A 4 step running average of the range data is used to reduce noise. Bump sensors are also used in case the sonar does not detect an obstacle.

Using an oscilloscope, the output voltage versus range relationship was determined. The following data was obtained by measuring the output voltage for various distances between the sonar and a wall. The expected values were determined from the sonar datasheet.



One of the main issues with using multiple sonar modules was mutual interference. Since the cameras were pointed sufficiently away from each other such that no part of their field of views overlapped, this interference was mostly mitigated. However, when the robot was near an obstacle, sonar pings from one sonar would reflect off the walls and be intercepted by the other. One way of eliminating this was to only power one sonar module at a time. This method was not used however because the sonar modules have a 50 ms delay after power on and before reliable sonar data is available, and this delay would significantly slow the main loop frequency of the robot.

Beacon

The beacons use a simple latching Hall Effect sensor in order to store an ON and OFF state. When the beacon is activated with a mechanical switch, all the LEDs on the beacon light continuously. When the Hall Effect sensor senses a magnetic field, power transmission to the LEDs is terminated. The system is reset by turning the switch OFF and ON. The threshold range for state switching is approximately 4 inches using the same neodymium magnets that are mounted onto the robot. The LEDs are embedded in a small Styrofoam ball that diffuses the light and increases the size of the color blob the camera detects. One interesting aspect of the Hall Effect sensors used on the beacon was that they were much more sensitive to a positive magnetic field (north) than a negative magnetic field (south).

Behaviors

Once the robot is activated, it calibrates the camera by sending an auto-adjust command and waiting 5 seconds. The robot then spins in place until the camera “sees” a blue blob of light. After aligning the robot so that the beacon is directly in front of it, the robot moves toward the beacon, using visual servoing to maintain path toward the beacon. Once the robot comes within range of the beacon, the Hall Effect sensor is triggered by the magnets mounted on the robot, causing the LEDs to deactivate (indicating the mine has been found and disabled). Once the blue blob of light disappears from the camera’s view, the robot will back away, turn and look for a new beacon. Throughout the entire run time, the sonar detects any obstacles, such as walls, and navigates away from them.

Conclusion

The robot was successful in detecting and maneuvering toward the beacon, deactivating the LEDs and continuing its search for other mines, all while avoiding obstacles. One of the main issues I had was the camera tracking objects other than the beacons. Although the LEDs were blue, the camera saw them as having large green and blue intensities. As a result, the camera would detect and track bright lights and windows in addition to the beacon. Using red LEDs was suggested, however the camera detected a variety of common objects (tables, floors, etc) with large red intensities and the number of false positives increased.

Future work would include adding a home base to move the mines after deactivating and capturing them. One of the main changes I would need to make to the platform in order to accomplish this task would be moving the camera to a higher vantage point. Currently, the camera is mounted at the same height as the LEDs on the mines. Therefore, the limiting factor in how far the camera can detect the mine is the light intensity of the beacon and not the vertical field of view of the camera. One downside to this placement is that the distance between the robot and the beacon cannot be determined from the y coordinate of the center of mass of the tracked color blob. I also plan on implementing a new method of chaining the sonar to eliminate mutual interference without severely decreasing the run frequency of the robot.

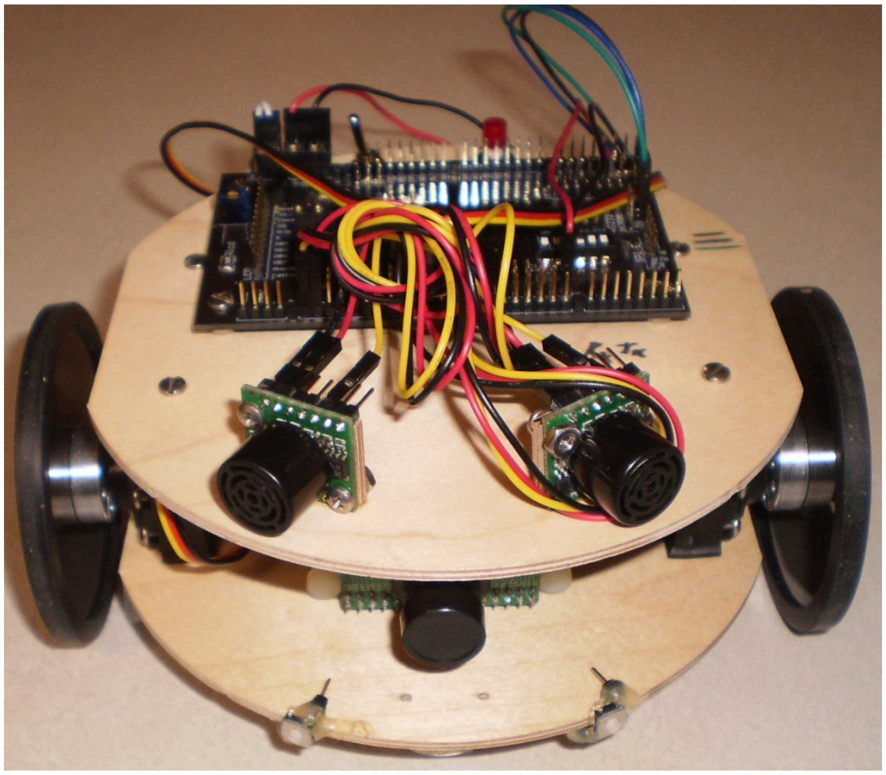
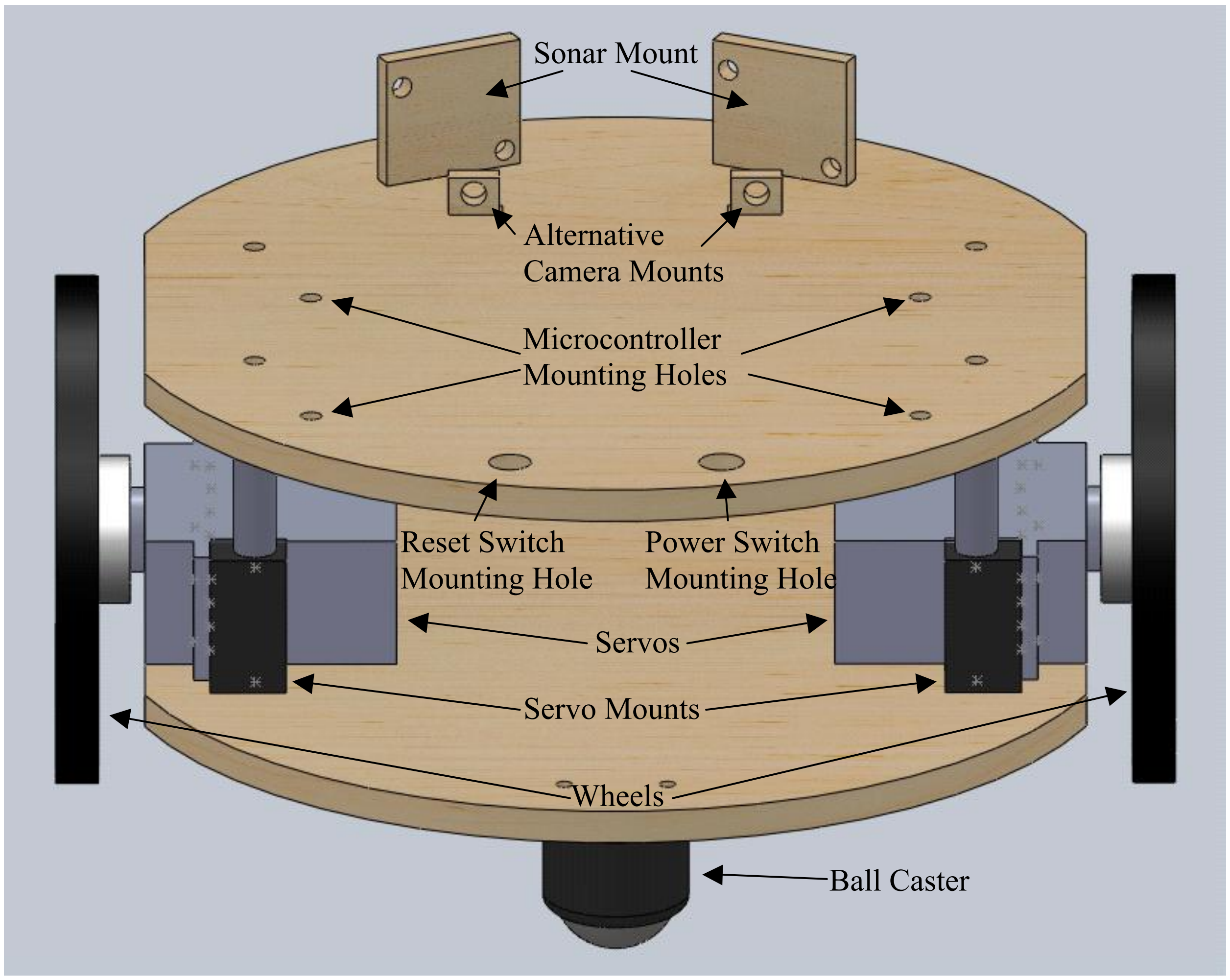
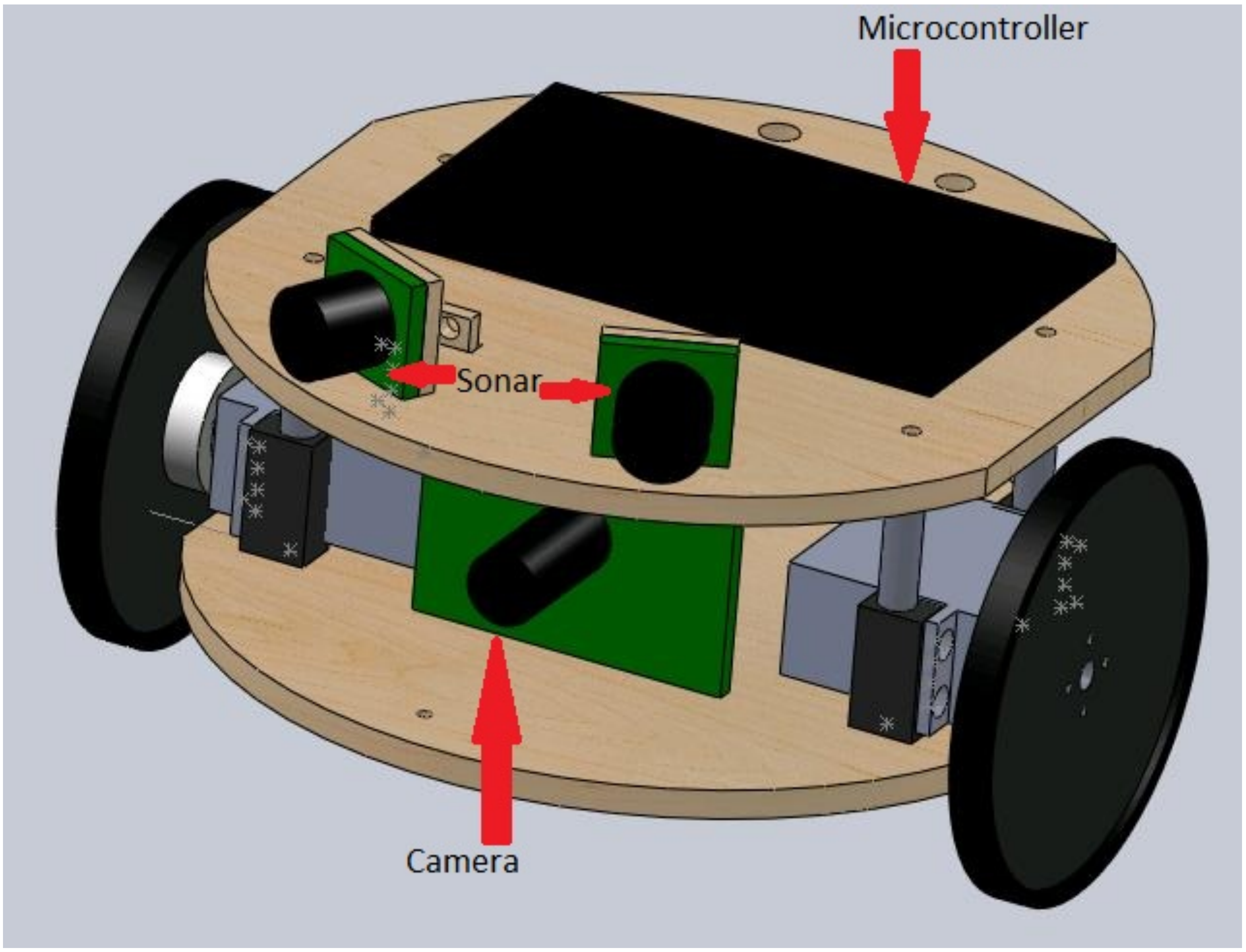


Figure 1: Final Robot.

Appendix A – Robot Platform Schematic


Sonar Mount
Alternative Camera Mounts
Microcontroller Mounting Holes
Reset Switch Mounting Hole
Power Switch Mounting Hole
Servos
Servo Mounts
Wheels
Ball Caster

Appendix B – Final Robot Schematic



Appendix C – Code

C.1 – Main Code

```
#include <avr/io.h>
#include "PVR.h"
#include "CMUcam.h"


//sonar variable initialization
int threshold = 280;                   //sonar threshold, 0 - 4096
#define avgLen 4                         //running average length
int Lsonar[avgLen];                     //Left sonar array
int Rsonar[avgLen];                     //Right sonar array
int sensCount = 0;                      //sonar counter
int Lavg;                               //Sonar average value, used for threshold comparison
int Ravg;

void ObstAvoid(void)
{
       Lsonar[sensCount] = ADCA1();         //get sonar values
       Rsonar[sensCount] = ADCA0();
    if (sensCount == 3)                 //reset sensCount
    {
        sensCount = 0;
    }
    else
    {
        sensCount++;
    }
    Lavg = 0;
    Ravg = 0;
    for(int i = 0; i < avgLen;i++){     //calculate average sonar values
        Lavg += Lsonar[i];
        Ravg += Rsonar[i];
    }
    Lavg = Lavg/avgLen;
    Ravg = Ravg/avgLen;


    lcdGoto(0,0);                       //Display sonar values
    lcdString("                ");
    lcdGoto(0,0);
    lcdString("Lsonar ");
    lcdInt(Lavg);
    lcdGoto(1,0);                       //Display sonar values
    lcdString("                ");
    lcdGoto(1,0);
    lcdString("Rsonar ");
    lcdInt(Ravg);


       if (Lavg < threshold)                //detect obstacle
       {
               ServoC0(-100);
               ServoC1(-100);
               lcdGoto(0,0);
               lcdString("Obstacle on left");
               for(int j = 0; j < 4; j++)
               {
                delay_ms(100);
                Lsonar[j] = ADCA1();
                Rsonar[j] = ADCA0();
           }
```

```
            sensCount = 0;
                lcdGoto(0,0);
                lcdString("                ");

        }
        else if (Ravg < threshold)          //detect obstacle
        {
                ServoC0(100);
                ServoC1(100);
                lcdGoto(0,0);
                lcdString("Obstacle on right");
                for(int j = 0; j < 4; j++)
                {
                 delay_ms(100);
                 Lsonar[j] = ADCA1();
                 Rsonar[j] = ADCA0();
            }
            sensCount = 0;
                lcdGoto(1,0);
                lcdString("                ");
        }


}

void main(void)
{
    //initialization
        xmegaInit();                                                //setup XMega
        delayInit();                                                //setup delay functions
        ServoCInit();                                               //setup PORTC Servos
        ADCAInit();                                                 //setup PORTA analog readings
        lcdInit();                                                  //setup LCD on PORTK
    USARTinit();
    camInit();
    PORTQ_DIR |= 0x01;                                   //set Q0 (LED) as output

    //variable initialization
        int maxMotor = 50;
    int spinTime = 0;
    int maxSpin = 100;
    //int targetLocked = 0;
    //int centerThreshold = 10;
        float propConst = 2;                    //Proportional Control constant kp
        int ctrl;
        int MX;
        //char clear[] = "                ";
    int i = 0;                                  //LED toggle
    float j = 1;                             //Homing toggle and scale

        for(int s = 0; s < avgLen;s++)          //Set all initial sonar values to twice the
threshold
        {
         Lsonar[s] = 2*threshold;
         Rsonar[s] = 2*threshold;
        }

    while(1)
        {
                //Code to get coordinates of beacon. All are zeros if beacon is not in sight
                MX = BeaconX();

         ObstAvoid();

         while(MX == 0)
         {
             spinTime++;
             if (spinTime < maxSpin)
             {
                 ServoC0((int) (j*10));
                 ServoC1((int) (j*10));
             }
             else
             {
                 ServoC0(maxMotor);
                 ServoC1(-1*maxMotor);
                 delay_ms(1500);
                 spinTime = 0;
             }

             MX = BeaconX();

             i++;
             if(i%2 == 1)
             {
                 PORTQ_OUT = 1;                                    //  turn on LED
             }
             else
             {
                 PORTQ_OUT = 0;                                    //  turn on LED
             }

              ObstAvoid();

         }

                i = 0;


         while(MX != 0)
         {
             if(MX < 40)
                       {
                              ctrl = (int) (MX*-1*propConst + 80);
                 /*
                              lcdGoto(1,5);
                              lcdString(clear);
                              lcdGoto(1,5);
                              lcdInt(ctrl);
                 */
                              ServoC0((int) (maxMotor - ctrl));
                              ServoC1(-1*maxMotor);

                       }
                       else
                       {
                              ctrl = (int) (MX*propConst - 80);
                 /*
                              lcdGoto(1,5);
                              lcdString(clear);
                              lcdGoto(1,5);
                              lcdInt(ctrl);
                 */
                              ServoC0(maxMotor);
                              ServoC1((int) (-1*maxMotor + ctrl));

                       }

             MX = BeaconX();
             /*
                lcdGoto(1,0);
                lcdString(clear);
                lcdGoto(1,0);
                lcdInt(MX);
             */

             i++;

             if(i%2 == 1)
             {
                 PORTQ_OUT = 1;                                    //  turn on LED
             }
```

```
            else
            {
                PORTQ_OUT = 0;                                      //  turn on LED
            }

               ObstAvoid();

        }

        if(i > 10)
        {
            j = 2*(i%2) - 1;
            ServoC0(-1*maxMotor);
            ServoC1(maxMotor);
            delay_ms(500);
        }
        else
        {
            j = -0.95*j;
        }
        i = 0;


        }
}
```

C.2 – “CMUcam.c”

```
#include <avr/io.h>
#include "PVR.h"
#include "CMUcam.h"

void USARTinit(void)
{
    //Initializaiton
    PORTE_DIR    = PIN3_bm;                  //Pin 3 of port E is output
    PORTE_OUT    = PIN3_bm;                  //Pin 3 of port E is TXO
    PORTE.DIRCLR    = PIN2_bm;               //Pin 2 of port E is RXO
    USARTE0_CTRLC = 0x03;                    // USART Control Register C: ASYNCHRONOUS, no parity
1 stop bit 8 bit word
    USARTE0_BAUDCTRLA = 0x06;                // Page 238 of Atmel manual, fbaud=115200 (my
case)=32MHz/(16*(((2^BSCALE) * BSEL)+1))
    USARTE0_BAUDCTRLB = 0xC1;                // BSEL= 262 and BSCALE= -4 in 2s comp .
    USARTE0_CTRLB |= 0x08;                   // TX0 is on
    USARTE0_CTRLB |= 0x10;                   // RX0 is on
}

//CMUCAM functions
void CMUsend(char *command)
{
  int i = 0;
  while (command[i] != '\0')   //While command does not end do...
  {
    // Data Register Empty Flag: check if data register is empty
    while (!(USARTE0_STATUS & (1<<USART_DREIF_bp)));

    //USARTE0_DATA is shared by the transmit and receive
    USARTE0_DATA = command[i];

    //if data is sent TXCIF is set
    while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp)));
    i++;
  }
}

char CMUreceive(void)
{
  int i=0;
  static char *data;
  do
    {   // wait for receive is complete
        while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));

        //Put data into string
        data[i] = USARTE0_DATA;
    } while (data[i++] != '\r');  //Since all trasnfers are ended by \r wait for it to happen
  return data;
}

void camInit(void)
{

    CMUsend("RS\r");                        //Reset the camera
    lcdString("Calibrating...");
    int j;
    for(j = 0; j < 5; j++)              //Long enough to wait for ACK and to get the camera ready
    {
        lcdGoto(1,0);
        lcdInt((5-j));
        delay_ms(1000);
    }
    CMUsend("PM 1\r");                      //Activate polling mode
    //lcdGoto(0,0);
    //lcdString("                ");
    //lcdString("ready");
    /*
    delay_ms(2000);
    CMUsend("TW\r");
    temp = CMUreceive();
    temp = CMUreceive();
    */
    lcdGoto(0,0);
    lcdString("                ");
    lcdGoto(1,0);
    lcdString("                ");

}

int BeaconX(void)
{
       CMUsend("TC 0 50 150 250 200 250\r");        //Track beacon
       char *temp;
       temp=CMUreceive();                    //Receive ACK
    temp=CMUreceive();
    /*
       int i = 0;
       while(temp[i] != 'M')
       {
              i++;
       }
       int j = i + 2;
       while(temp[j] != ' ')
       {
              j++;
       }
       char xstr[5];
       strncpy(xstr,(temp + i + 1),(j - (i + 2)));
       */
    int x;
    temp[0] = ' ';
       x = atoi(temp);
    delay_ms(5);
       return x;

}
```