Fangyao Liu
No Partner
Wednesday 3:30~5:30
Date: Nov. 1st, 2016

Exercise 1

1. What is the effect of the Erode filter and what importance does this play in our object detection?

   The effect of Erode filter is to filter noise and display the shape of skeleton. If we didn't have this function, there will be too many noise on the image and skeleton will be obscure.

2. Briefly describe the functionality of the code inside the while loop that determines the object's location in the image.

   First, collect all the moments on x axis and y axis and add them up. After getting these moments, divide them by the whole area of the frame and we will get the average, which is also central point of our object.

3. What are your thoughts on the OpenCV Libraries?
   Powerful and available to process the image
   .

Exercise 2

1. Describe the code in the two functions supplied to you (initRobot and SendToCreate) and their purpose.
   InitRobot: Use "open" command to open serial port. Success, fd will be set to 1. Otherwise, fd will be 0. When Serial port has been opened successfully, the Beagle board will set baud rate, set certain serial port and receive feedback from car

   SendToCreate: Function takes fd, message and the length of the message as the inputs. Use fd to determine whether the serial port has been opened. If so, use for loop and take string length as loop times to send command to IRobot.

2. Compare this method of controlling the iRobot with the Beagleboard xM and Angstrom to using the green command module.

   For the previous lab, the method of controlling the iRobot was just setting certain pins. For this lab, the method is about using serial communication to send command to microprocessor by Beagleboard xM and then set pins of microprocessors.

Appendix

Exercise 1

```
#include <termios.h> // tcgetattr(),
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/file.h>
#include <cstdlib>
#include <iostream>
#include <cv.h>
#include <highgui.h>

#define BAUDRATE B57600
#define SERPORT "/dev/ttyUSB0"

// To compile use the following line:
// g++ `pkg-config --cflags --libs opencv` -o binaryname filename.cpp

// Prototypes
// -OpenCV Functions
IplImage* GetThresholdedImage(IplImage* img);

// -iRobot Create Functions
void SendToCreate( int fd, const char *data, int length );
int initRobot();
void FollowBall(int x, int y, double area, int imgWidth, int fd);

// Global variables
IplImage* imgYellowThresh;

int main()
{
      CvCapture* capture = 0;
     capture = cvCaptureFromCAM(0);    //start capturing frame from cameras. If failed,
return.
      if(!capture)
      {
            printf("Could not intitialize capturing...\n");
            return -1;
      }

      //Create the Windows
      cvNamedWindow("video");
```

```
cvNamedWindow("thresh");

//init iRobot
int fd = initRobot();
if(fd < 0)          //Make sure fd is valid
    return -1;      //Error will have been printed within initRobot()

while(1)
{
    IplImage* frame = 0;
    frame = cvQueryFrame(capture);

    if(!frame) break;


    IplImage* imgThresh = GetThresholdedImage(frame);

    /*
    //Calculate Ball's position
    CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
    cvMoments(imgThresh, moments, 1);

    //actual moment values
    double moment10 = cvGetSpatialMoment(moments, 1, 0);
    double moment01 = cvGetSpatialMoment(moments, 0, 1);
    double area = cvGetCentralMoment(moments, 0, 0);

    //Hold Last Position and current position of ball
    static int posX = 0;
    static int posY = 0;

    int lastX = posX;
    int lastY = posY;

    //actual new position
    posX = moment10/area;
    posY = moment01/area;

    //Follow the ball
    FollowBall(posX, posY, area, frame->width, fd);

    //draw a circle and display the images
    cvCircle(frame, cvPoint(posX, posY), 5, cvScalar(255, 0, 255), 3, 0 , 0);
    */
```

```
            cvShowImage("thresh", imgThresh);
            cvShowImage("video", frame);

            //wait for a keypress
            int c = cvWaitKey(10);
            if(c != -1)
                  break;
            /*
            cvReleaseImage(&imgThresh);
            delete moments;
            */
      }

      cvReleaseCapture(&capture); //everytime to release unused capture.
      return 0;
}

//OpenCV Functions

//Takes in an image and returns a thresholded image by a given HSV range.
IplImage* GetThresholdedImage(IplImage* img)
{
      //ToDo: Complete This Function using the following comments as guidelines.
      //Create IplImage to hold Binary Image
      IplImage* biIm=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,3);


      //Convert the image into an HSV image
      cvCvtColor(img, biIm, CV_BGR2HSV);


      //Use cvInRangeS() to create Binary Image within HSV range
      IplImage* biIm1=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
      cvInRangeS(biIm, cvScalar(30,80,150), cvScalar(110,180,220),biIm1);

      //Create IplImage and use cvErode on Thresholded Image
      IplImage* biIm2=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
      cvErode(biIm1,biIm2,NULL,1);
      //Release Unneeded Images that we created.
      cvReleaseImage(&biIm);
      cvReleaseImage(&biIm1);

      //Return Results
      return biIm2;
```

```
}

//iRobot Create Functions

//Sends an array of data to the create
//Typically 7 bytes:
//Start, Control, Packet, Data, Data, Data, Data
//E.g. 128, 131, 145, vel1, vel2, rad1, rad2
void SendToCreate( int fd,const char *data, int length )
{
	int i;
	for( i=0; i<length; i++ )
	{
		if( write(fd, &data[i], 1) == -1 )
		{
			printf( "\nUnable to write %s", SERPORT );
			printf( "\nerrno = %d", errno );
		}
		usleep( 5000 );
	}
}

//Init iRobot, return fd
//Open the serial port, set baudrate and start to transmit and receive from Robot.
int initRobot()
{
	int		fd;
	struct termios	tty;
	int		flags = fcntl(STDIN_FILENO, F_GETFL);

	fd = open(SERPORT, O_RDWR);				//open the serial port
	if ( fd < 0 )
	{
		printf( "\nUnable to open %s", SERPORT );
		printf( "\nerrno = %d", errno );
		return fd;
	}
	else
		printf( "\nSuccessfully opened the serial port" );

	tcgetattr(fd, &tty); // Read port parameters into termios structure
	// cc_t		c_cc[NCCS];	/* control chars */
	tty.c_cflag = CLOCAL | CREAD | CS8; // CLOCAL	= ignore modem control
lines
```

```
                                                    // CREAD        = enable receiver
                                                    // CS8          = 8 bit data
                                                    // ~CRTSCTS     = no hardware flow
control
        tty.c_iflag = IGNBRK | IGNPAR;              // IGNBRK       = ignore BREAK
condition
                                                    // IGNPAR       = ignore framing anf
parity error
        tty.c_oflag = 0;                            //
        tty.c_lflag = 0;                            //
        cfsetospeed(&tty, BAUDRATE );               // Set output baudrate
        cfsetispeed(&tty, BAUDRATE );               // set input baudrate
        if( tcsetattr(fd, TCSANOW, &tty) < 0 )// Set serial port - TCSANOW = Apply
changes immediately
        {
                printf( "\nUnable to tcsetattr %s", SERPORT );
                printf( "\nerrno = %d\n", errno );
        }
        fcntl(STDIN_FILENO, F_SETFL, flags | O_NONBLOCK);
        return fd;
}

void FollowBall(int x, int y, double area, int imgWidth, int fd)
{
     //ToDo: Add code to make the Robot follow the ball here
}
```

Exercise 2

```
//Compile with
//g++ main.cpp -lpthread

#include <termios.h> // tcgetattr(),
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/file.h>
#include <cstdlib>
#include <iostream>
#include <pthread.h>

#define BAUDRATE B57600
#define SERPORT "/dev/ttyUSB0"

//Commands to Drive the iRobot
```

```
//use 131 instead of 132 to prevent driving off cliff automatically
//it will allow you to back up if cliff sensor is active but that is all
const char cmdStop[7]          = {128, 131, 145, 0, 0, 0, 0};
const char cmdForward[7]       = {128, 131, 137, 0x00, 0x64, 0x80, 0x00};
const char cmdRight[7]         = {128, 131, 137, 0x00, 0x32, 0xFF, 0xFF};
const char cmdLeft[7]          = {128, 131, 137, 0x00, 0x32, 0x00, 0x01};
const char cmdBackward[7]      = {128, 131, 137, 0xFF, 0x9C, 0x80, 0x00};


// Prototypes
// -iRobot Create Functions
void SendToCreate( int fd, const char *data, int length );
void RecieveFromCreate( int fd, char *data, int length );
int initRobot();



int main()
{
    //init iRobot
    int fd = initRobot();
    if(fd < 0)          //Make sure fd is valid
        return -1;      //Error will have been printed within initRobot()
    SendToCreate(fd, cmdStop, 7);


    //Create pthreads

    while(true)
    {
        SendToCreate(fd, cmdForward, 7);
        usleep(5000);


    }

}
//iRobot Create Functions

//Sends an array of data to the iRobot
void SendToCreate( int fd,const char *data, int length )
{
     int i;
     for( i=0; i<length; i++ )
```

```
	{
		if( write(fd, &data[i], 1) == -1 )
		{
			printf( "\nUnable to write %s", SERPORT );
			printf( "\nerrno = %d", errno );
		}
		usleep(5000);
	}
}

//Recieves an array of data with "length" from the iRobot
void RecieveFromCreate( int fd, char *data, int length )
{
	int i;
	for( i=0; i<length; i++ )
	{
		if( read(fd, &data[i], 1) == -1 )
		{
			printf( "\nUnable to write %s", SERPORT );
			printf( "\nerrno = %d", errno );
		}
		usleep( 5000 );
	}
}

//Init iRobot, return fd
int initRobot()
{
	int		fd;
	struct termios	tty;
	int		flags = fcntl(STDIN_FILENO, F_GETFL);

	fd = open(SERPORT, O_RDWR);				//open the serial port
	if ( fd < 0 )
	{
		printf( "\nUnable to open %s", SERPORT );
		printf( "\nerrno = %d", errno );
		return fd;
	}
	else
		printf( "\nSuccessfully opened the serial port\n" );

	tcgetattr(fd, &tty); // Read port parameters into termios structure
	// cc_t		c_cc[NCCS];	/* control chars */
```

```
    tty.c_cflag = CLOCAL | CREAD | CS8; // CLOCAL      = ignore modem control
lines
                                        // CREAD       = enable receiver
                                        // CS8         = 8 bit data
                                        // ~CRTSCTS    = no hardware flow
control
    tty.c_iflag = IGNBRK | IGNPAR;      // IGNBRK      = ignore BREAK
condition
                                        // IGNPAR      = ignore framing anf
parity error
    tty.c_oflag = 0;                    //
    tty.c_lflag = 0;                    //
    cfsetospeed(&tty, BAUDRATE );       // Set output baudrate
    cfsetispeed(&tty, BAUDRATE );       // set input baudrate
    if( tcsetattr(fd, TCSANOW, &tty) < 0 )// Set serial port - TCSANOW = Apply
changes immediately
    {
        printf( "\nUnable to tcsetattr %s", SERPORT );
        printf( "\nerrno = %d\n", errno );
    }
    fcntl(STDIN_FILENO, F_SETFL, flags | O_NONBLOCK);
    return fd;
}
```

Exercise 3

```
#include <termios.h> // tcgetattr(),
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/file.h>
#include <cstdlib>
#include <iostream>
#include <cv.h>
#include <highgui.h>
#include <pthread.h>

#define BAUDRATE B57600
#define SERPORT "/dev/ttyUSB0"

//Commands to Drive the iRobot
//use 131 instead of 132 to prevent driving off cliff automatically
//it will allow you to back up if cliff sensor is active but that is all
const char cmdStop[7]       = {128, 131, 145, 0, 0, 0, 0};
const char cmdForward[7]    = {128, 131, 137, 0x00, 0x64, 0x80, 0x00};
```

```
const char cmdRight[7]      = {128, 131, 137, 0x00, 0x32, 0xFF, 0xFF};
const char cmdLeft[7]       = {128, 131, 137, 0x00, 0x32, 0x00, 0x01};
const char cmdBackward[7]   = {128, 131, 137, 0xFF, 0x9C, 0x80, 0x00};

// To compile use the following line:
// g++ `pkg-config --cflags --libs opencv` -o binaryname filename.cpp

// Prototypes
// -OpenCV Functions
IplImage* GetThresholdedImage(IplImage* img);

// -iRobot Create Functions
void SendToCreate( int fd, const char *data, int length );
int initRobot();
void FollowBall(int x, int y, double area, int imgWidth, int fd);
void delay(int a);

// Global variables
IplImage* imgYellowThresh;

int main()
{
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(0);
    if(!capture)
    {
        printf("Could not intitialize capturing...\n");
        return -1;
    }

    //Create the Windows
    cvNamedWindow("video");
    cvNamedWindow("thresh");

    //init iRobot
    int fd = initRobot();
    if(fd < 0)          //Make sure fd is valid
        return -1;      //Error will have been printed within initRobot()

    while(1)
    {
        IplImage* frame = 0;
        frame = cvQueryFrame(capture);
```

```
        if(!frame) break;


        IplImage* imgThresh = GetThresholdedImage(frame);


        //Calculate Ball's position
        CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(imgThresh, moments, 1);

        //actual moment values
        double moment10 = cvGetSpatialMoment(moments, 1, 0);
        double moment01 = cvGetSpatialMoment(moments, 0, 1);
        double area = cvGetCentralMoment(moments, 0, 0);

        //Hold Last Position and current position of ball
        static int posX = 0;
        static int posY = 0;

        int lastX = posX;
        int lastY = posY;

        //actual new position
        posX = moment10/area;
        posY = moment01/area;

        //Follow the ball
        FollowBall(posX, posY, area, frame->width, fd);

        //draw a circle and display the images
        cvCircle(frame, cvPoint(posX, posY), 5, cvScalar(255, 0, 255), 3, 0 , 0);

        cvShowImage("thresh", imgThresh);
        cvShowImage("video", frame);

        //wait for a keypress
        int c = cvWaitKey(10);
        if(c != -1)
              break;
        /*
        cvReleaseImage(&imgThresh);
        delete moments;
        */
}
```

```
    cvReleaseCapture(&capture);
    return 0;
}

//OpenCV Functions

//Takes in an image and returns a thresholded image by a given HSV range.
IplImage* GetThresholdedImage(IplImage* img)
{
    //ToDo: Complete This Function using the following comments as guidelines.
    //Create IplImage to hold Binary Image
    IplImage* biIm=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,3);


    //Convert the image into an HSV image
    cvCvtColor(img, biIm, CV_BGR2HSV);


    //Use cvInRangeS() to create Binary Image within HSV range
    IplImage* biIm1=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
    cvInRangeS(biIm, cvScalar(30,80,150), cvScalar(110,180,220),biIm1);

    //Create IplImage and use cvErode on Thresholded Image
    IplImage* biIm2=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
    cvErode(biIm1,biIm2,NULL,1);
    //Release Unneeded Images that we created.
    cvReleaseImage(&biIm);
    cvReleaseImage(&biIm1);

    //Return Results
    return biIm2;
}

//iRobot Create Functions

//Sends an array of data to the create
//Typically 7 bytes:
//Start, Control, Packet, Data, Data, Data, Data
//E.g. 128, 131, 145, vel1, vel2, rad1, rad2
void SendToCreate( int fd,const char *data, int length )
{
     int i;
     for( i=0; i<length; i++ )
```

```
        {
                if( write(fd, &data[i], 1) == -1 )
                {
                        printf( "\nUnable to write %s", SERPORT );
                        printf( "\nerrno = %d", errno );
                }
                usleep( 5000 );
        }
}

//Init iRobot, return fd
int initRobot()
{
        int             fd;
        struct termios  tty;
        int             flags = fcntl(STDIN_FILENO, F_GETFL);

        fd = open(SERPORT, O_RDWR);                             //open the serial port
        if ( fd < 0 )
        {
                printf( "\nUnable to open %s", SERPORT );
                printf( "\nerrno = %d", errno );
                return fd;
        }
        else
                printf( "\nSuccessfully opened the serial port" );

        tcgetattr(fd, &tty); // Read port parameters into termios structure
        // cc_t         c_cc[NCCS];      /* control chars */
        tty.c_cflag = CLOCAL | CREAD | CS8; // CLOCAL       = ignore modem control
lines
                                                // CREAD        = enable receiver
                                                // CS8          = 8 bit data
                                                // ~CRTSCTS     = no hardware flow
control
        tty.c_iflag = IGNBRK | IGNPAR;          // IGNBRK       = ignore BREAK
condition
                                                // IGNPAR       = ignore framing anf
parity error
        tty.c_oflag = 0;                        //
        tty.c_lflag = 0;                        //
        cfsetospeed(&tty, BAUDRATE );           // Set output baudrate
        cfsetispeed(&tty, BAUDRATE );           // set input baudrate
        if( tcsetattr(fd, TCSANOW, &tty) < 0 )// Set serial port - TCSANOW = Apply
```

```
changes immediately
        {
                printf( "\nUnable to tcsetattr %s", SERPORT );
                printf( "\nerrno = %d\n", errno );
        }
        fcntl(STDIN_FILENO, F_SETFL, flags | O_NONBLOCK);
        return fd;
}

void delay(int a)
{
      int i,j,k;
      for(i=0;i<a;i++)
            for(j=0;j<1000;j++)
                  for(k=0;k<1000;k++)
                        ;
}

void FollowBall(int x, int y, double area, int imgWidth, int fd)
{
      //ToDo: Add code to make the Robot follow the ball here
      //if skeleton is too close to the robot, stop
      if(area>50000)
      {
            SendToCreate(fd,cmdStop,7);
            delay(30);
      }
      //if skeleton is on the left side of the robot, turn left
      else if(x<200)
      {
            printf("%d, %f\n",y,area);
            SendToCreate(fd,cmdLeft,7 );
            delay(30);
            SendToCreate(fd,cmdStop,7);
      }
      //if skeleton is on the right side of the robot, turn right
      else if(x>450)
      {
            printf("%d, %f\n",y, area);
            SendToCreate(fd,cmdRight,7);
            delay(30);
            SendToCreate(fd,cmdStop,7);
      }
      else
```

```
    {
        printf("%d, %f\n",y, area);
        SendToCreate(fd,cmdForward,7);
        delay(50);
        SendToCreate(fd, cmdStop, 7);
    }
}
```