Introduction to Robotics and Mechatronics

GROUP 2.5

Benson Chalethu

Pietro Griffa

Ingvar Groza

PostLab 06

tracking .cpp (Q1, Q2, Q4)

tracking.cpp

```cpp
/*
    Lab 06
 */

#include "tracking.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>

// include constants here:
const int edgeparam = 200; // input of function EdgeDetect (thresh = edgeparam)
int frcounter = 1; // counter for saving only every 30th image

// functions

void SvImage(IplImage* img, char* filename)
{
    int p[3] = {CV_IMWRITE_JPEG_QUALITY,95,0};
    cvSaveImage(filename, img, p);
}

IplImage* CrossTarget (IplImage* inImg, int x, int y, int size, int line_thickness)
{
    // ******* already completely implemented for you, you don't need to change anything here *******
    IplImage* outImg = cvCloneImage(inImg);
    /* We use opencv function called "cvLine" to create a cross on the given coordinate (x,y)
     * This function is used in color tracking image. You will integrate this function into the color tracking function during the lab
     * You can find the defintion of cvLine from the website.    *
     */

    // horizontal line
    CvPoint pt1 = cvPoint(x-size/2,y);
    CvPoint pt2 = cvPoint(x+size/2,y);
    cvLine(outImg, pt1, pt2, cvScalar(0, 200,0), line_thickness, 8, 0);
    // verical line
    pt1.x = x; pt1.y = y-size/2;
    pt2.x = x; pt2.y = y+size/2;
    cvLine(outImg, pt1, pt2, cvScalar(0,200,0), line_thickness, 8, 0);


    return outImg;
}
```

```cpp
int ColorTrackingSetColors(IplImage* img, int* hmax, int* hmin, int* smax, int* smin, int* vmax, int* vmin)
{
    // ******** PostLab Q1 *********
    /* This function will allow us to manually set the thresholds, while observing the result.
       The example code on color tracking with OpenCV in the lab manual can be used as a reference.*/\

    IplImage *imgHSV, *imgThresh, *imgShow;

    // Create a new image with the same size as IplImage* img, a depth of 8 and appropriate number of channels (use cvCreateImage)
    // Call that new image "imgHSV"
    imgHSV = cvCreateImage(cvGetSize(img), 8, 3);

    // Convert Source Image to HSV, use cvCvtColor()
    // make sure to store the converted image in "imgHSV"
    cvCvtColor(img,imgHSV,cv::COLOR_RGB2HSV);

    // Create Threshold Trackbars using cvCreateTrackbar()
    // the window_name can be any name (e.g. "Set"), just make sure that a window with the same name is created in main()
    cvCreateTrackbar("Hmin", "Track", hmin, 255, 0);
    cvCreateTrackbar("Hmax", "Track", hmax, 255, 0);
    cvCreateTrackbar("Smin", "Track", smin, 255, 0);
    cvCreateTrackbar("Smax", "Track", smax, 255, 0);
    cvCreateTrackbar("Vmin", "Track", vmin, 255, 0);
    cvCreateTrackbar("Vmax", "Track", vmax, 255, 0);

    // Create a new image to apply thresholds and one to save the mask
    // call the mask "imgThresh" and the masked image "imgShow", think about how many channels should be used for each image
    imgThresh = cvCreateImage(cvGetSize(img), 8, 1);
    imgShow = cvCreateImage(cvGetSize(img), 8, 3);

    // Threshold the image using the function cvInRangeS() and save the mask in imgThresh (already done)
    cvInRangeS(imgHSV, cvScalar(*hmin,*smin,*vmin), cvScalar(*hmax,*smax,*vmax), imgThresh);

    // filter the original image using our mask, save the filtered image in imgShow
    // use the function cvCopy()
    cvCopy(img, imgShow, imgThresh);

    // Display the filtered image imgShow using the function cvShowImage()
    cvShowImage("Track",imgShow);

    // Release created Images (HSV image, filtered image and threshed/mask image)
    // use the function vcReleaseImage()
    cvReleaseImage(&imgHSV);
    cvReleaseImage(&imgThresh);
    cvReleaseImage(&imgShow);

    return 0;
}


int ColorTracking (IplImage* img, int* positionX , int* positionY, CvScalar min, CvScalar max)
{
    // ******** PostLab Q2 ************
    /* In this function we will implement our Color Tracking algorithm
       The thresholds min and max are passed to the function and are determined beforehad using ColorTrackingSetColors*/

    int size = 10, linet = 2;
    IplImage *imgHSV, *imgThresh, *imgShow;

    // Create new HSV image
    imgHSV = cvCreateImage(cvGetSize(img), 8, 3);

    // Convert Source Image to HSV (with parameters cv::COLOR_BGR2HSV), use cvCvtVolor()
    cvCvtColor(img,imgHSV,cv::COLOR_BGR2HSV);

    // Create new image to apply thresholds, think about the number of channels needed
    imgThresh = cvCreateImage(cvGetSize(img), 8, 1);

    // Threshold the image with CvScalar min and CvScalar max, use cvInRangeS()
    cvInRangeS(imgHSV, min, max, imgThresh);

    // Create memory space for moments (already done)
    CvMoments *moments_y = (CvMoments*)malloc(sizeof(CvMoments));

    // Calculate moments (already done)
    cvMoments(imgThresh,moments_y,1);

    // Extract spatial moments and area (already done)
    double moment10_y = moments_y->m10;
    double moment01_y = moments_y->m01;
    double area_y = moments_y->m00;

    // Determine Center (see: https://docs.opencv.org/3.1.0/d8/d23/classcv_1_1Moments.html)
    (*positionX) = (int)(moment10_y/area_y);
    (*positionY) = (int)(moment01_y/area_y);

    // Add a cross at center using the function (CrossTarget())
    // you will need to use cvCloneImage to duplicate the original image first
    imgShow = cvCloneImage(img);
    imgShow = CrossTarget(imgShow, *positionX, *positionY, size, linet);

    // display the image (the one with the cross), use cvShowImage()
    cvShowImage("Target",imgShow);

```

```
    // save the the image
    // uncomment the following code and use the correct image
    if (frcounter%30 == 0)
    {
        char filename[50];
        sprintf(filename,"Crossed_frame%d.jpg",frcounter);
        SvImage(imgShow,filename); // you will need to change "image" to the correct variable name
    }
    frcounter++;

    // Release created images and free memory (used by moments_y), use cvReleaseImage() and free()
    cvReleaseImage(&imgHSV);
    cvReleaseImage(&imgThresh);
    cvReleaseImage(&imgShow);
    free(moments_y);

    return 0;
}


int EdgeDetect (IplImage* img, int thresh)
{   // ********** Prelab Q2 ***********
    // note: you can find the function definitions online under https://docs.opencv.org/2.4/index.html

    // Create a new image for converting the  original image to gray image. Use cvCreateImage() and assign
    // it to variable called "gray_img" of type IplImage*. Use cvGetSize(img) within cvCreateImage() to get the right size
    // If you are not sure, you can refer to the example in the lab manual.
    IplImage *gray_img, *smooth_gray_img, *edge_detect, *edge_img;
    gray_img = cvCreateImage(cvGetSize(img), 8, 1);

    // Convert your image "img" to gray (store it in "gray_img") with the function cvCvtColor(), using the parameters called cv::COLOR_BGR2GRAY
    cvCvtColor(img,gray_img,CV_BGR2GRAY);

    // Smooth the gray image by using "cvSmooth" function and using gaussian method (CV_GAUSSIAN).
    // Make sure to create a new image called "smooth_gray_img" first (cvCreateImage) where you can store the smoothed image
    smooth_gray_img = cvCreateImage(cvGetSize(img), 8, 1);
    cvSmooth(gray_img,smooth_gray_img,CV_GAUSSIAN);

    // Create another new image called "edge_detect" which we will use for edge detection from the converted gray image
    edge_detect = cvCreateImage(cvGetSize(img), 8, 1);

    // Detect edges using canny edge detection by using "cvCanny" function with the parameter of thresh to define the range. Use "thresh" for the
    // first threshold for the hysteresis procedure and "thresh*2" for the second treshhold for the hysteris procedure and an aperture size of 3.
    // refer to https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny#cv.Canny
    cvCanny(smooth_gray_img,edge_detect,thresh,thresh*2,3);

    // Create variables to store contours (already done)
    CvMemStorage *mem;
    mem = cvCreateMemStorage(0);
    CvSeq *contours = 0;

    // Find contours in the canny output using the cvFindContours() function
    cvFindContours(edge_detect, mem, &contours);

    // Create a new image called "edge_img" for storing edge tracking image from gray image. Use 3 channels.
    // you can use cvSet to set the whole image to a specific color (background)
    edge_img = cvCreateImage(cvGetSize(img), 8, 3);
    cvSet(edge_img, cvScalar(0, 0, 0));

    // define the color of the contour using cvScalar (make sure it's consistent with the number of channels)
    //CvScalar cont_color(255, 255, 255);
    //cont_color = cvScalar(255, 255, 255);

    //define the color of contours using cvScalar()

    while (contours != 0)
    {
        // draw  contours by using the cvDrawContours() function
        // set maxLevel = 0
        cvDrawContours(edge_img, contours, cvScalar(0, 255, 0), cvScalar(255, 0, 0), 0);

        // pointer to the next sequence
        contours = contours->h_next;
    }

    // Display images by using cvShowImage() function

    //cvNamedWindow("image", CV_WINDOW_AUTOSIZE);
    //cvShowImage("image",img);
    //cvNamedWindow("gray", CV_WINDOW_AUTOSIZE);
    //cvShowImage("gray",gray_img);
    //cvNamedWindow("edge", CV_WINDOW_AUTOSIZE);
    //cvShowImage("smooth-gray",smooth_gray_img);
    //cvShowImage("edge_detect",edge_detect);
    cvShowImage("Edge",edge_img);


    // Save Images (already done, just uncomment)

    if (frcounter%30 == 0)
    {
        char filename[50];
        sprintf(filename,"Contour_frame%d.jpg",frcounter);
        SvImage(edge_img,filename);
    }
    frcounter++;

    //release the used images by using cvReleaseImage() function. Pass the address of your image pointers to cvReleaseImage
    // cvReleaseImage(&img);
    cvReleaseImage(&gray_img);
    cvReleaseImage(&smooth_gray_img);
    cvReleaseImage(&edge_detect);
    cvReleaseImage(&edge_img);

    return 0;
}
```

```
245
246
247 int main ()
248 {
249     // Variable initialization
250     IplImage* frame = 0;
251     int key;
252     int hmax=255, hmin=0, smax=255, smin=0, vmax=255, vmin=0;
253     CvCapture* capture;
254     int xc,yc, tresh;
255
256     FILE *coordinate;
257     FILE *imageOut;
258
259     // ************ PostLab Q1 ************ //
260
261     // open the .avi file using cvCaptureFromFile()
262     // if you get an error after reading the first framem try to define multiple channels (e.g. call cvCaptureFromFile() multiple times)
263     capture = cvCaptureFromFile("capture.avi");
264     //capture = cvCaptureFromFile("capture.avi");
265     //capture = cvCaptureFromFile("capture.avi");
266
267     if (!capture)
268     {
269         printf("Could not initialize capturing...\n");
270         return -1;
271     }
272
273     /******* set color tracking parameters  *******/
274     // in this part of our program we want to use the function ColorTrackingSetColors()
275     // to specify the color tracking parameters for the color tracking algorithm
276
277
278     // create a window using cvNameWindow() for setting the color tracking parameters (you can use the option CV_WINDOW_AUTOSIZE)
279     // make sure to use the same window name as in ColorTrackingSetColors()
280     // the window can be moved with cvModeWindow()
281
282     cvNamedWindow("Track", CV_WINDOW_AUTOSIZE);
283     // cvMoveWindow("Track", int x_pos, int y_pos);
284
285     // write a while loop that loops over all the frames in the .avi file and calls the function ColorTrackingSetColors() on each of them
286     // once at the end, start again until the user presses any key
287     while (1)
288     {
289         // use cvQueryFrame() to grab a single video frame
290         // use cvSetCaptureProperty() to start again from the beginning if necessary (cvQueryFrame() returns NULL if at the end of movie)
291         // cvWaitKey(10) can be used to receive user inputs
292         frame = cvQueryFrame(capture);
293         if(!frame)
294         {
295             cvSetCaptureProperty(capture,CV_CAP_PROP_POS_AVI_RATIO, 0);
296             frame = cvQueryFrame(capture);
297         }
298
299         ColorTrackingSetColors(frame, &hmax, &hmin, &smax, &smin, &vmax, &vmin);
300
301         key = cvWaitKey(10);
302         if (key != -1)
303         {
304             break;
305         }
306
307     }
308
309     // close the window with cvDestroyWindow()
310     cvDestroyWindow("Track");
311     //cvDestroyAllWindows();
312
313     cvWaitKey(100);
314
315     // ************ PostLab Q2 ************ //
316
317     /******* color tracking algorithm ******/
318     // similar to before, call the ColorTracking() function on each frame of the .avi file
319     // to track the red dot, also use your color tracking parameters you specified before
320     // make sure to display (show images) how the dot is being tracked
321
322     // write the coordinates of the center of the red dot in a text file
323
324     cvNamedWindow("Target", CV_WINDOW_AUTOSIZE);
325
326     cvSetCaptureProperty(capture,CV_CAP_PROP_POS_FRAMES, 1);
327     coordinate = fopen("Coordinates.txt", "w+");
328
329     while (1)
330     {
331         frame = cvQueryFrame(capture);
332         if(!frame)
333         {
334             cvSetCaptureProperty(capture,CV_CAP_PROP_POS_AVI_RATIO,0);
335             frame = cvQueryFrame(capture);
336             break;  // if movie has ended
337         }
338
339         ColorTracking (frame, &xc , &yc, cvScalar(hmin, smin, vmin), cvScalar(hmax, smax, vmax));
340         fprintf(coordinate, "\n%d\t%d", xc, yc);
341
342         cvWaitKey(10);
343
344     }
345
346     fclose(coordinate);
347
348     cvDestroyWindow("Target");
349     //cvDestroyAllWindows();
350
```

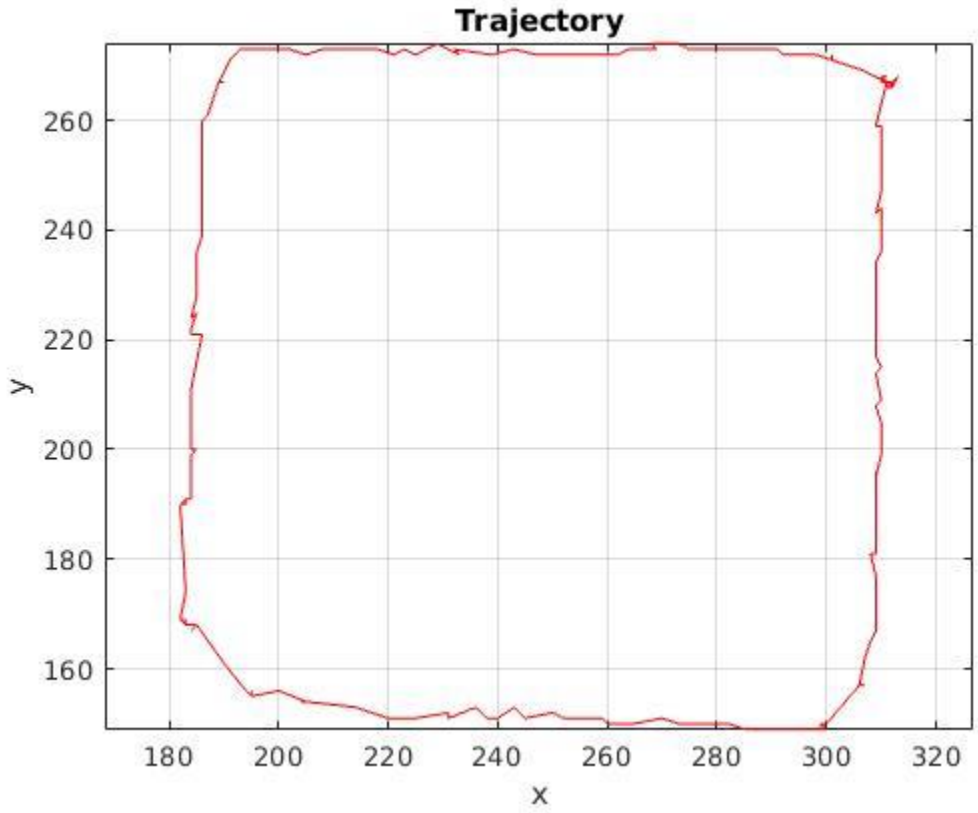
```
    // ************ PostLab Q3 ************ //

    /******* edge tracking algorithm ******/
    // now use your edge tracking algorithm to track the dot
    // make sure to display (show images) how the dot is being tracked

    frcounter = 0;

    cvSetCaptureProperty(capture,CV_CAP_PROP_POS_FRAMES, 1);
    thresh = 2*edgeparam;

    cvNamedWindow("Edge", CV_WINDOW_AUTOSIZE);

    while (1)
    {
        frame = cvQueryFrame(capture);
        if(!frame)
        {
            cvSetCaptureProperty(capture,CV_CAP_PROP_POS_AVI_RATIO,0);
            break;  // if movie has ended
        }
        EdgeDetect (frame, thresh);

        cvWaitKey(10);

    }



    //release the used images and capture (use cvDestroyAllWindows(), cvReleaseCapture() and cvReleaseImage())
    // make sure to close all windows
    cvDestroyAllWindows();
    cvReleaseCapture(&capture);
    cvReleaseImage(&frame);


    return 0;
}
```

tracking .h

tracking.h

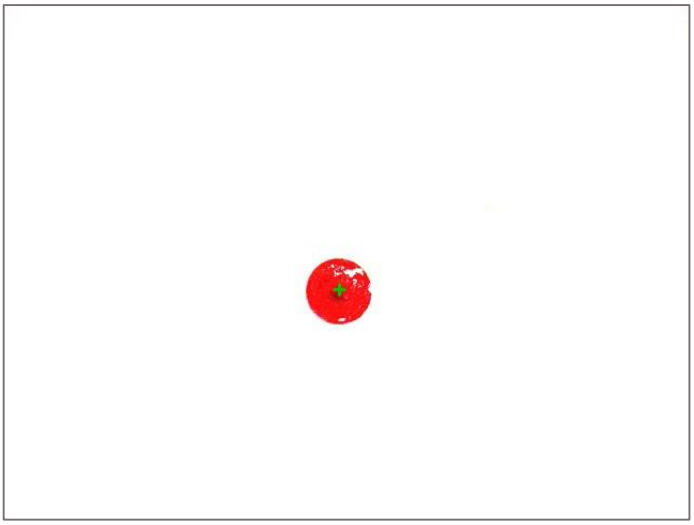
```
#ifndef tracking_h
#define tracking_h

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <iostream>
#include <sys/time.h>
#include <opencv2/opencv.hpp>
#include <opencv2/legacy/legacy.hpp>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"


// Draw a cross on image
IplImage* CrossTarget (IplImage* inImg, int x, int y, int size, int line_thickness);
 /*
  * inImg: input image
  * x,y: position of center of cross
  * size: size of cross
  * line_thickness: thickness of cross lines
 */


// Set parameters for Color Tracking
int ColorTrackingSetColors (IplImage* img,
int* hmax, int* hmin, int* smax, int* smin, int* vmax, int* vmin);
 /*
  * img: input image
  * hmax,..., vmin: pointers to integers to store values of minimal\
  *  and maximal parameters for thresholding
 */

// Color Tracker
int ColorTracking (IplImage* img, int* positionX , int* positionY,
 CvScalar min, CvScalar max);
 /*
  * img: input image
  * positionX,positionY pointers to integer to store position
  * min: minimal (hue,saturation,value,0)
  * min: maximal (hue,saturation,value,0)
 */


// Edge Detection
int EdgeDetect (IplImage* img, int thresh);
 /*
  * img: input image
  * thresh: threshold for canny edge
 */

// Save Image
void SvImage(IplImage* img, char* filename);
 /*
  * img: input image
  * filename: name of image file where image will be stored
  */

#endif
```

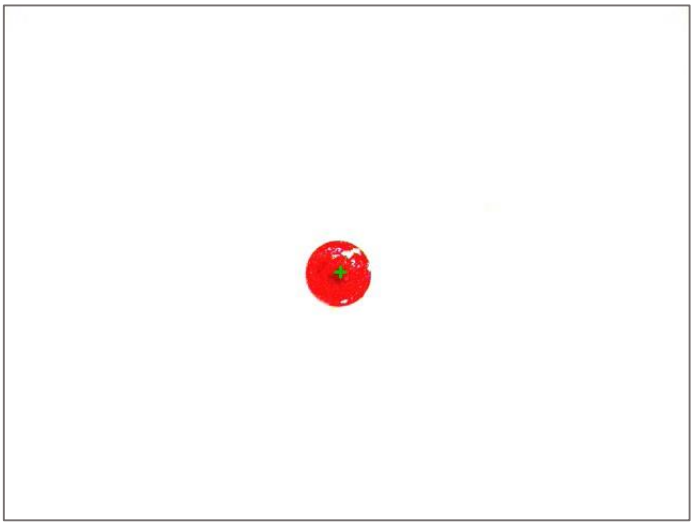
Q3

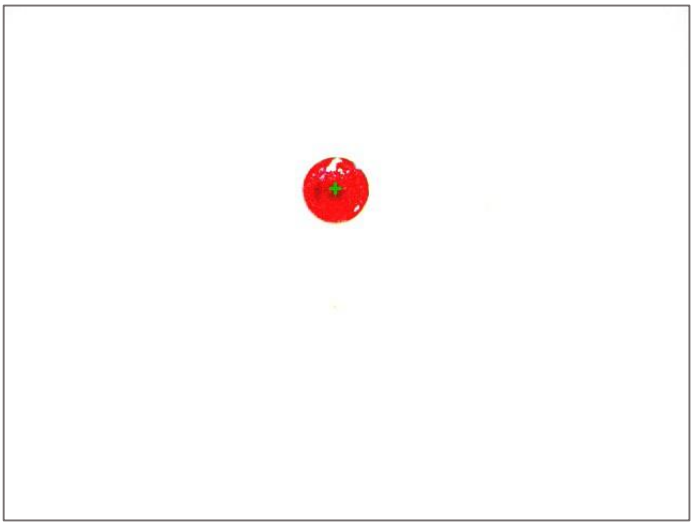
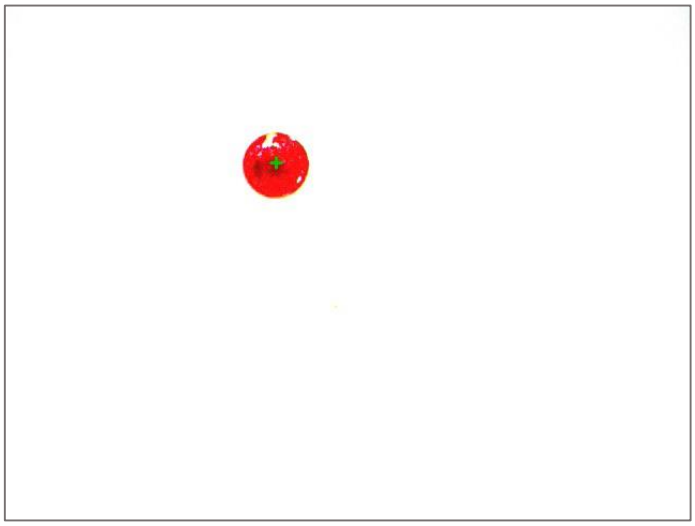
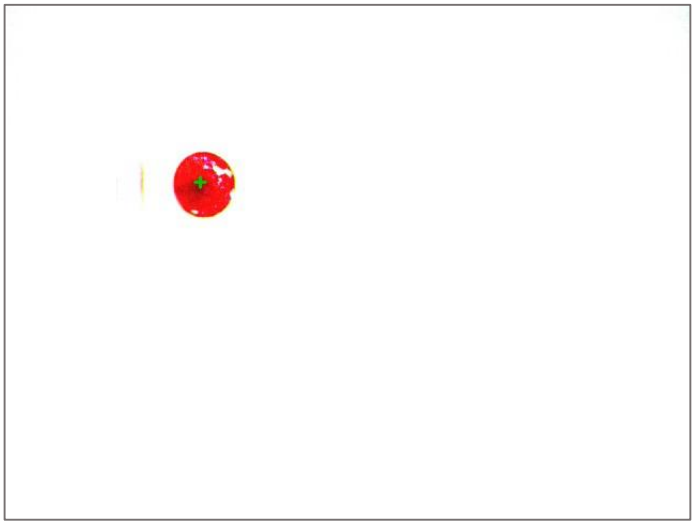
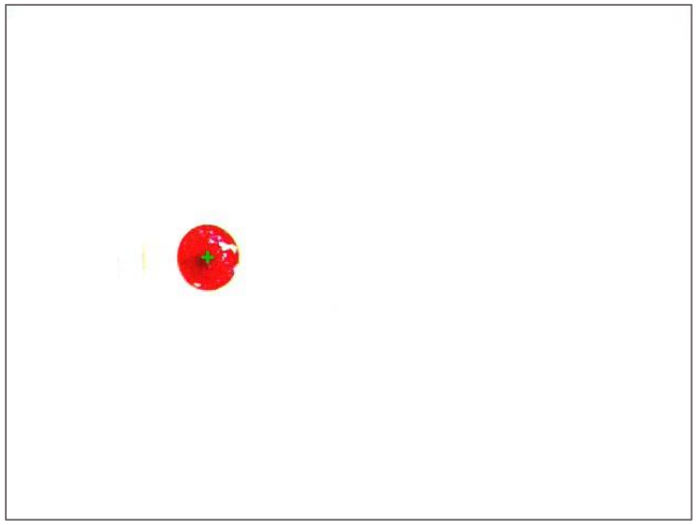
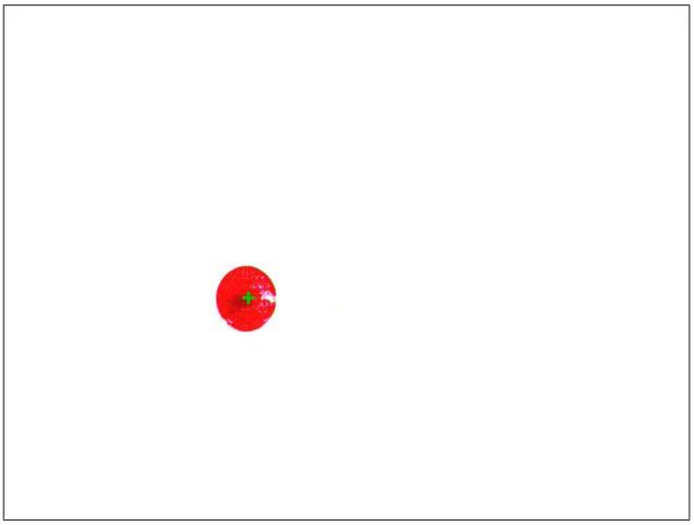
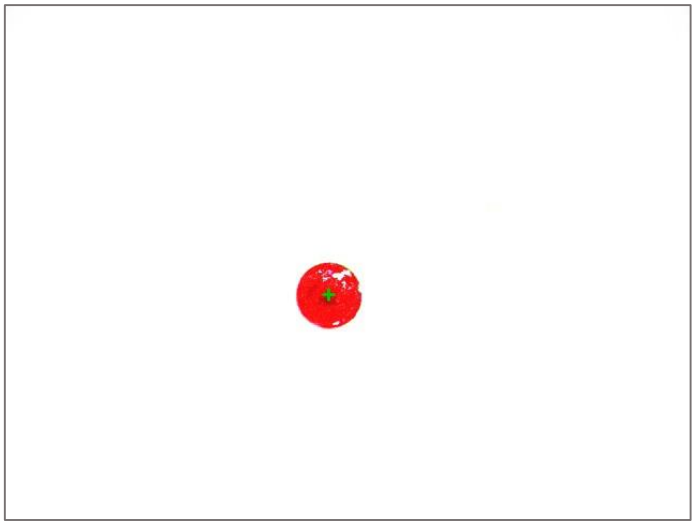


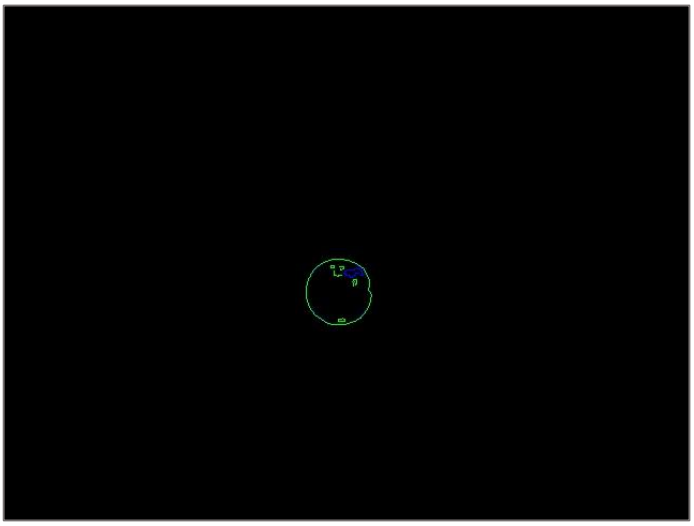
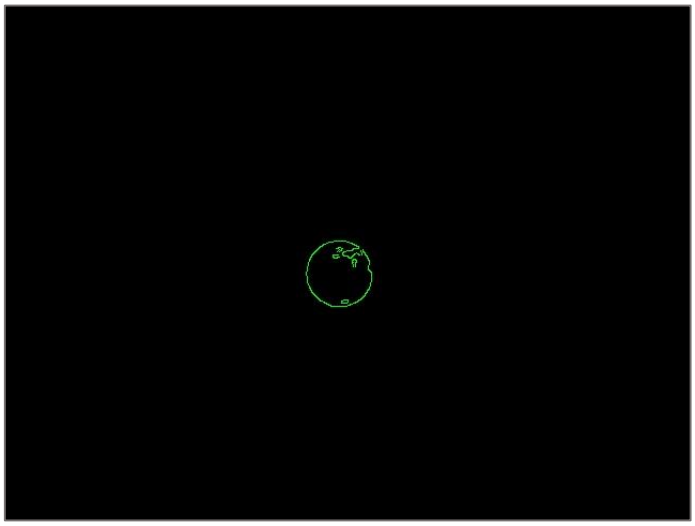
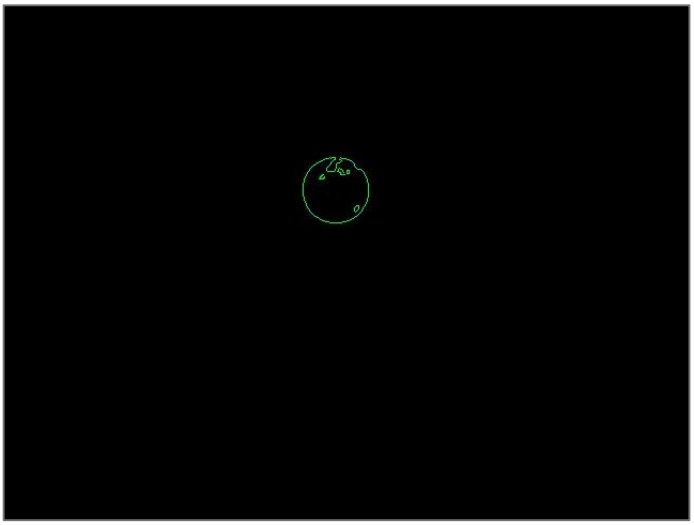
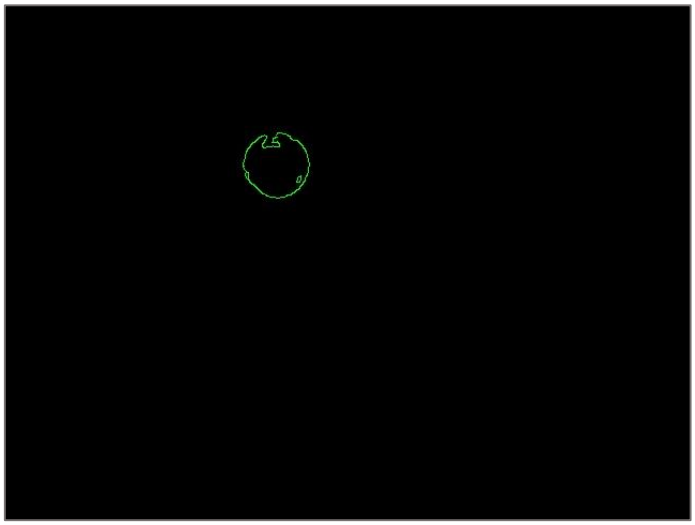
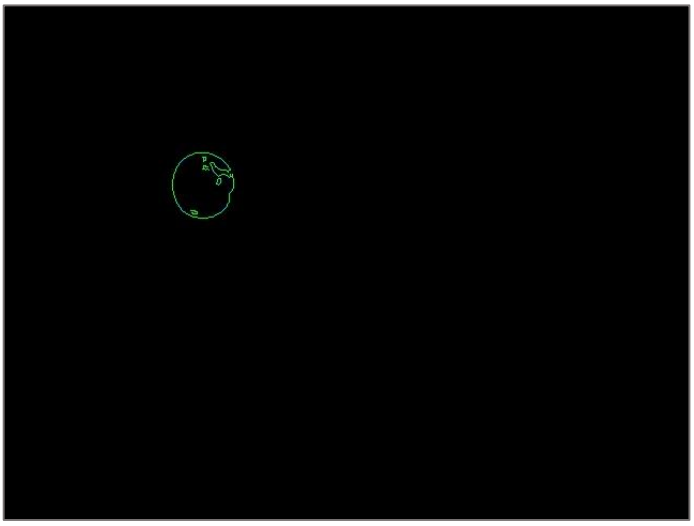
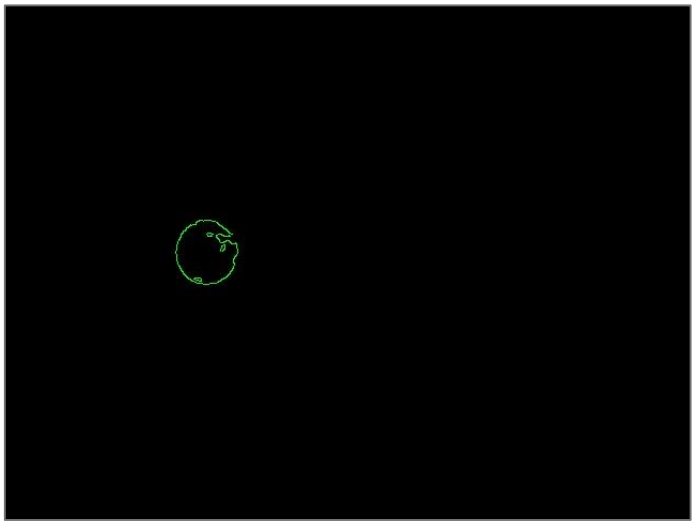
Q5

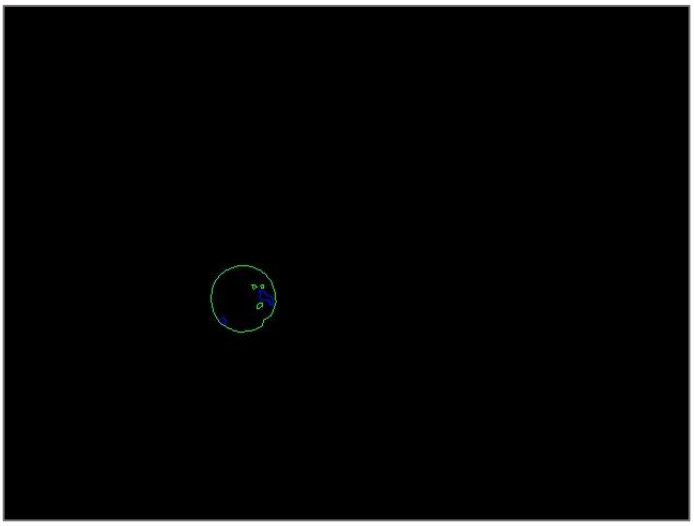
thresh = edgeparam :

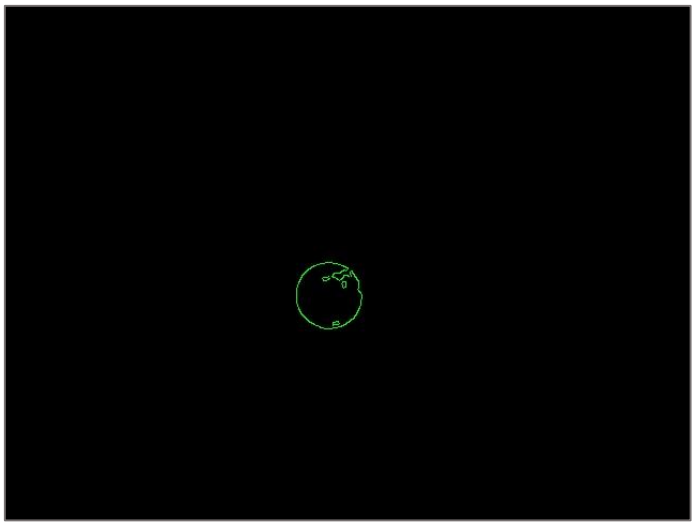




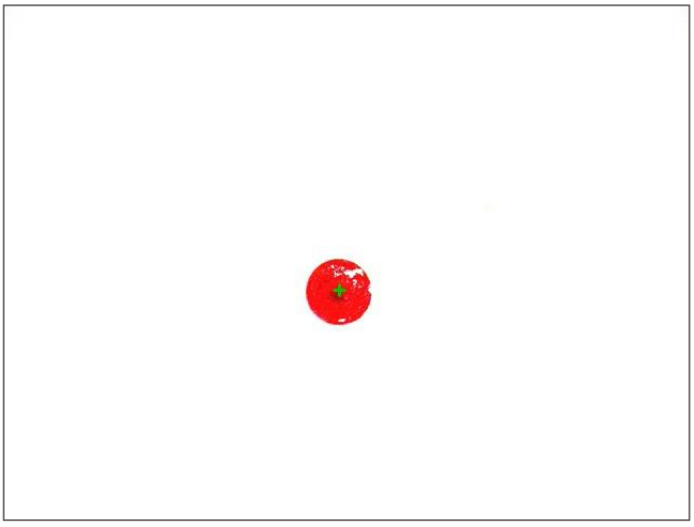







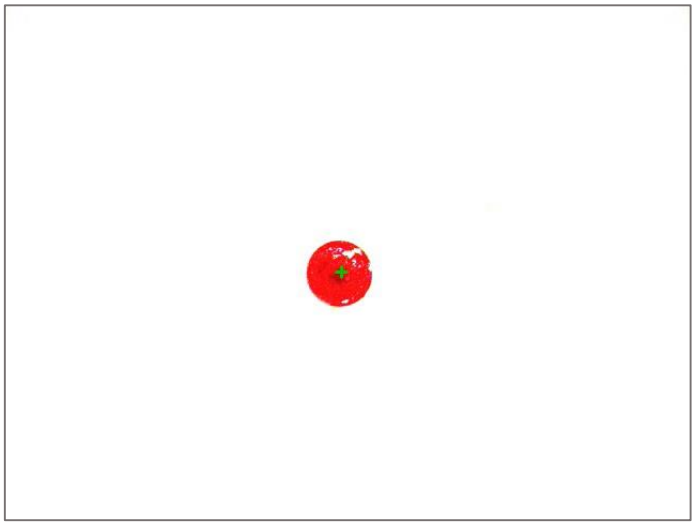







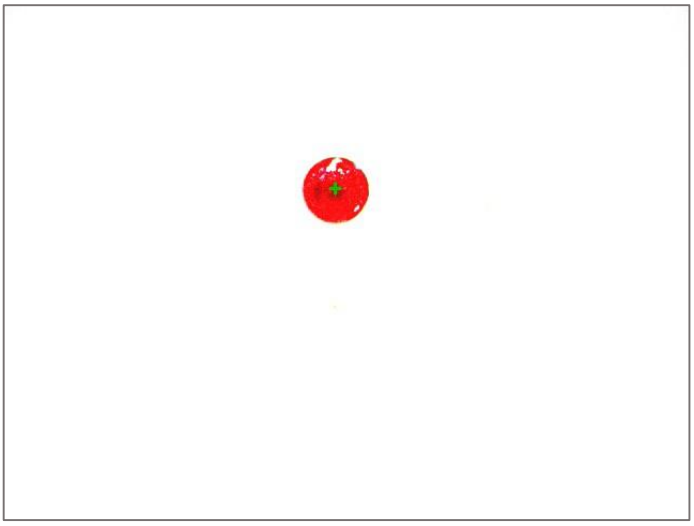


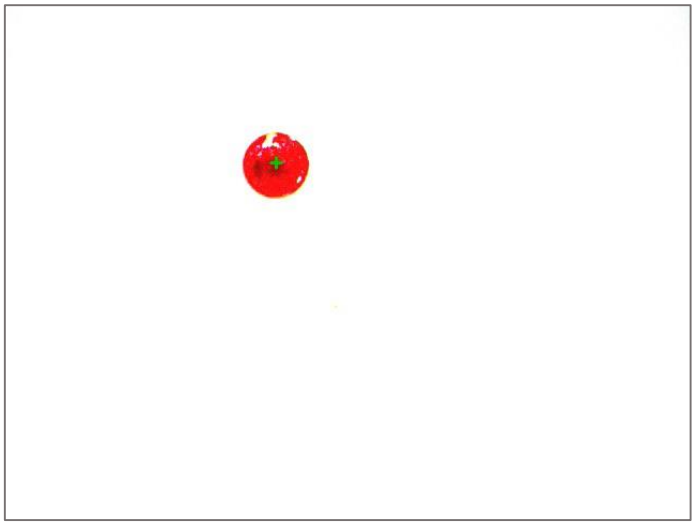


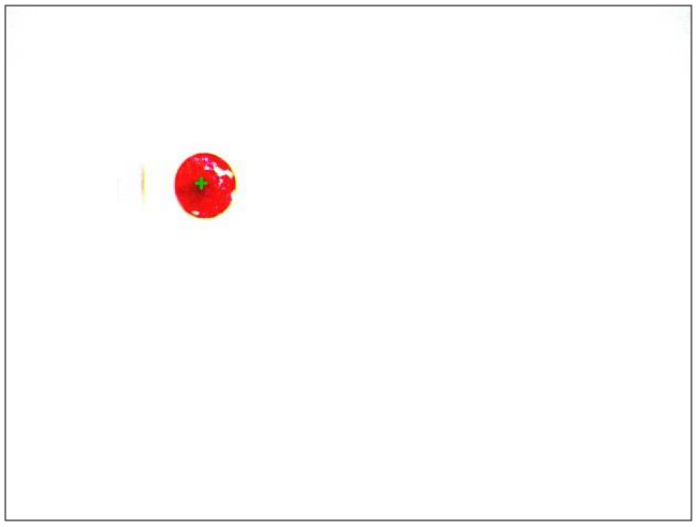
thresh = edgeparam/2 :

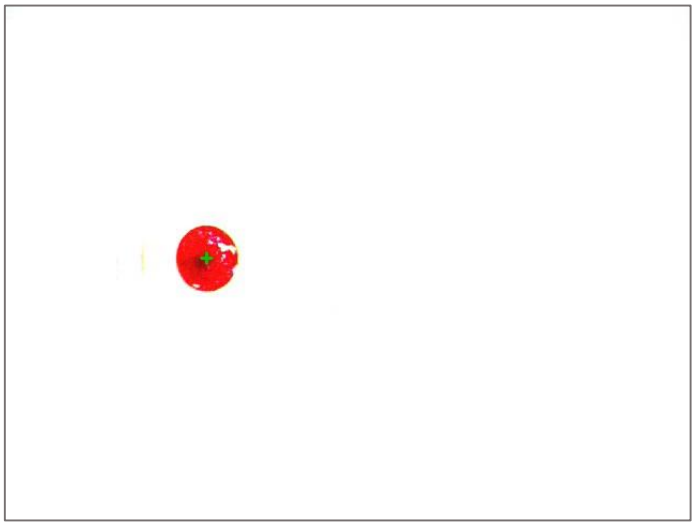


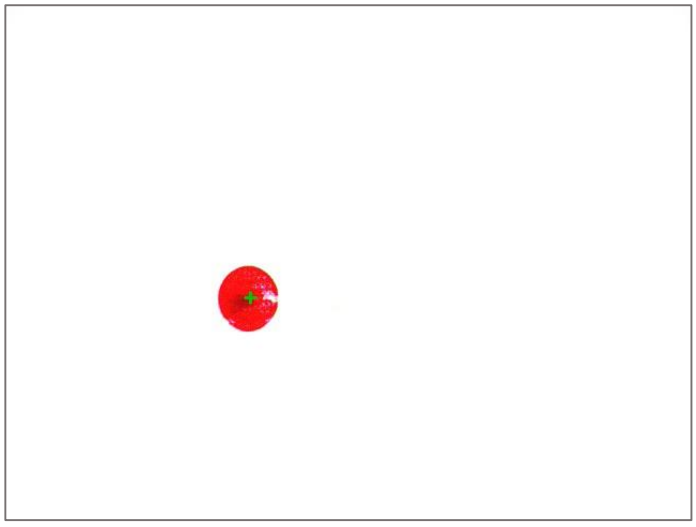


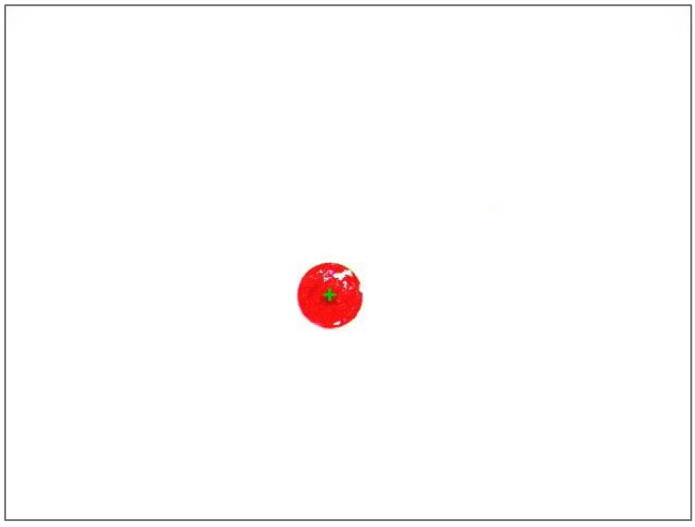


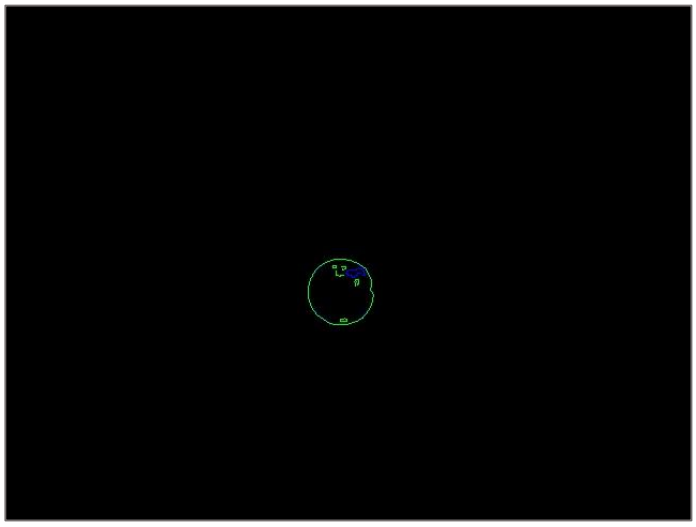


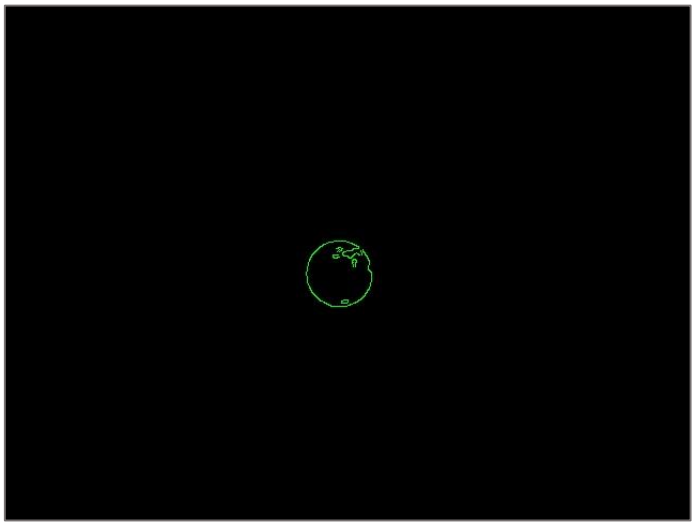


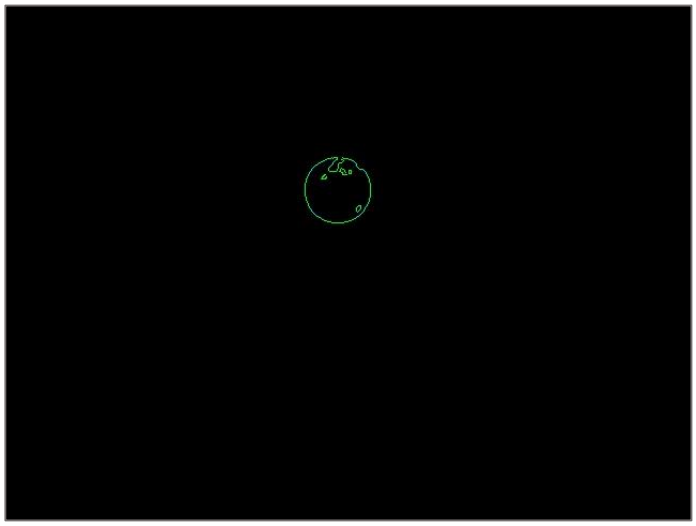
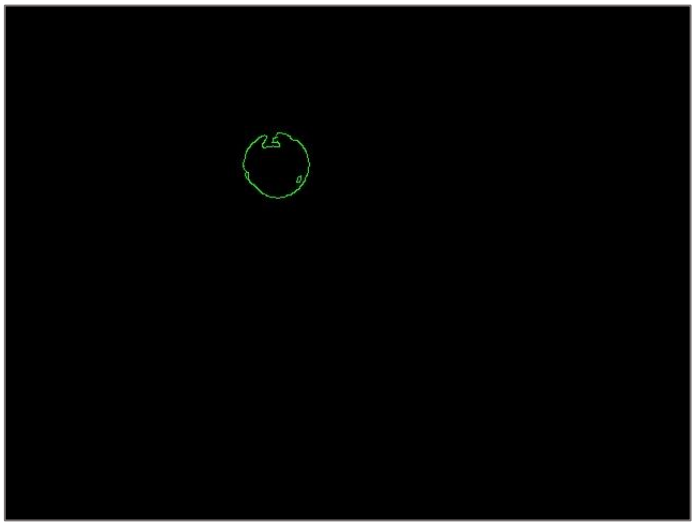
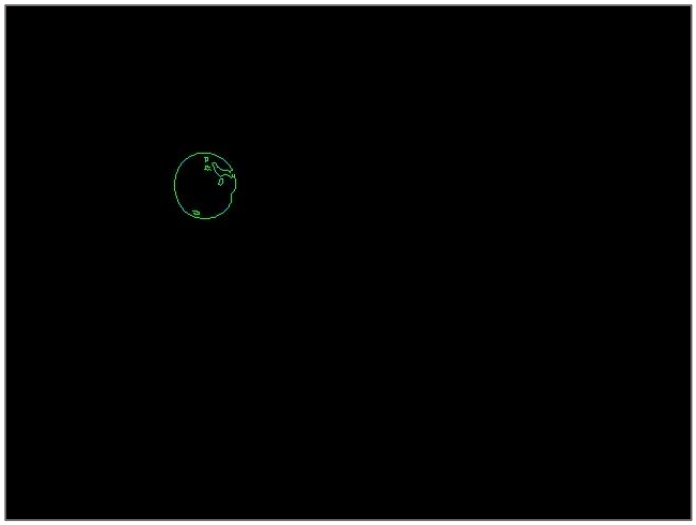
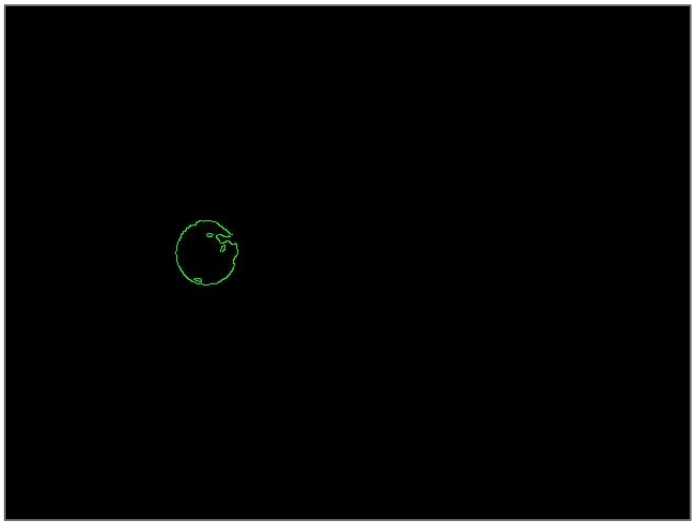
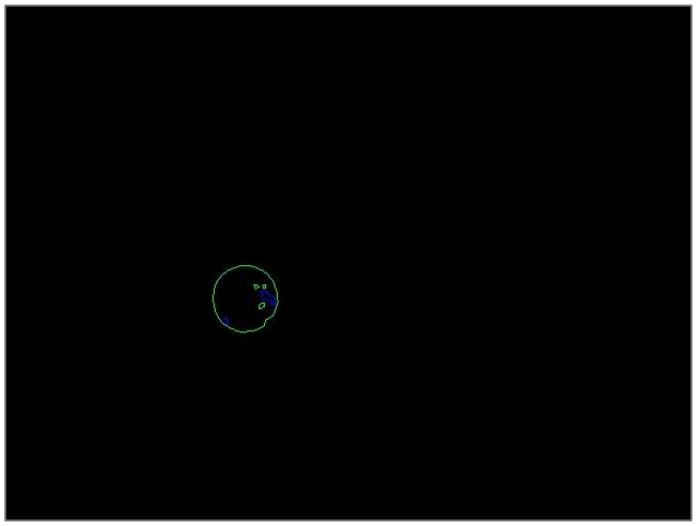
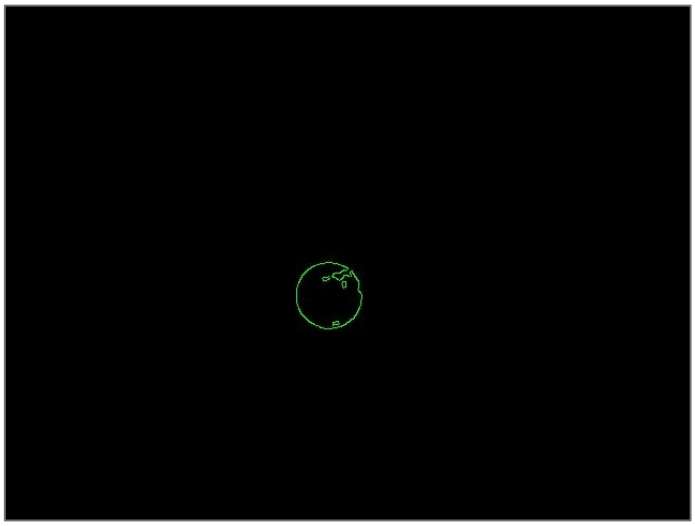




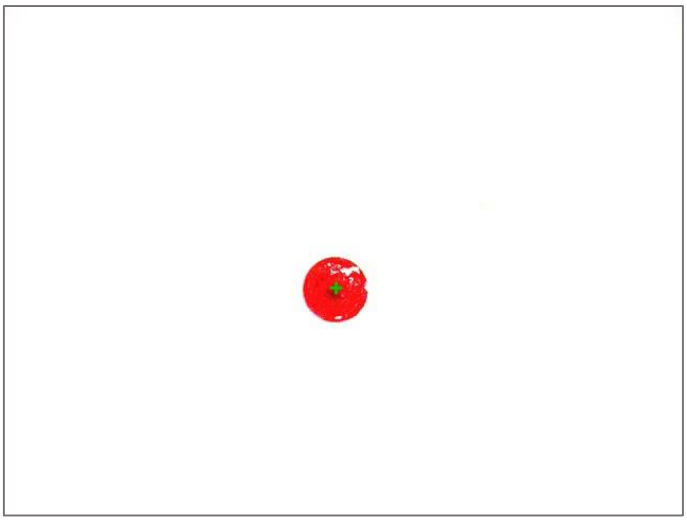


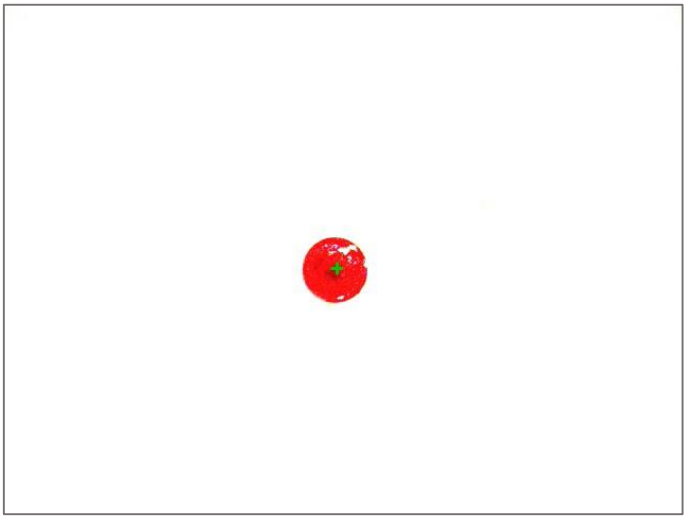


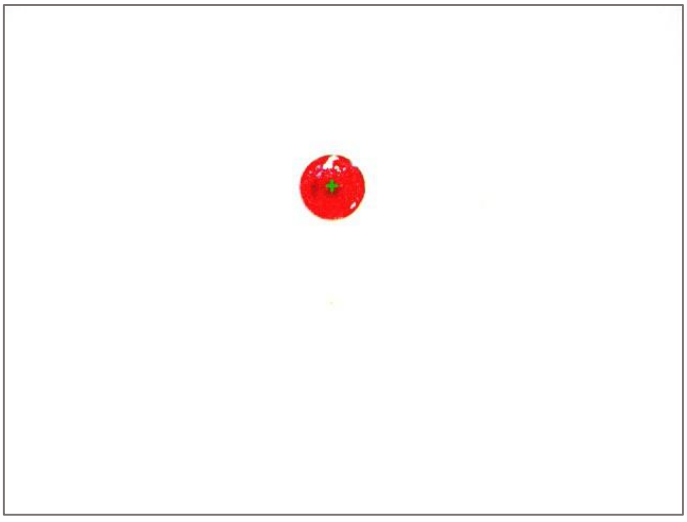


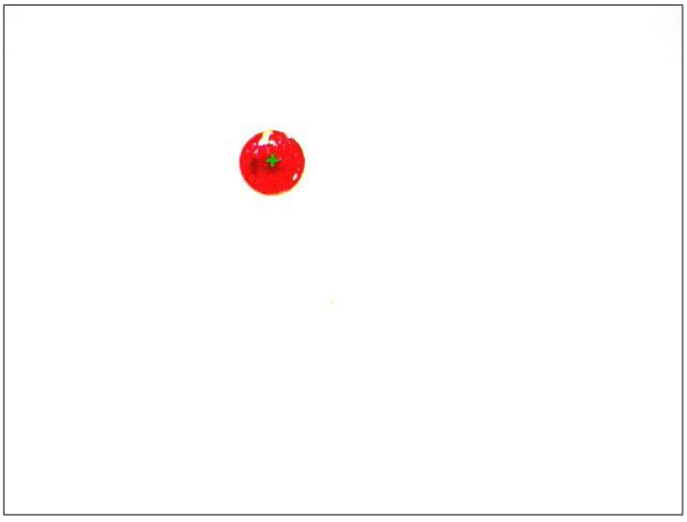







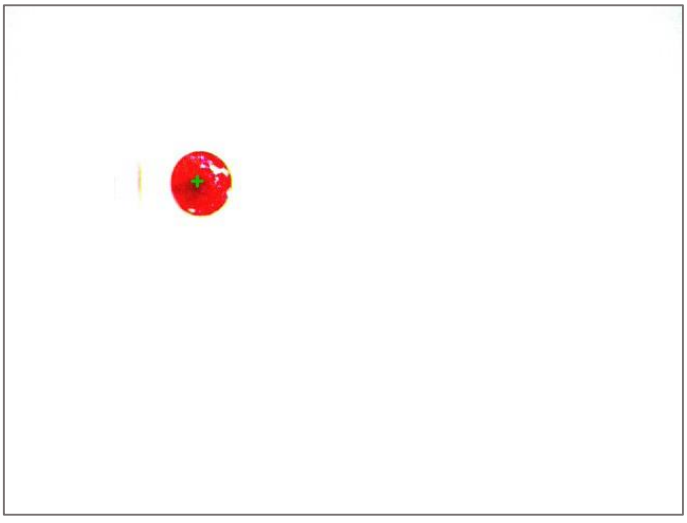
thresh = 2*edgeparam :

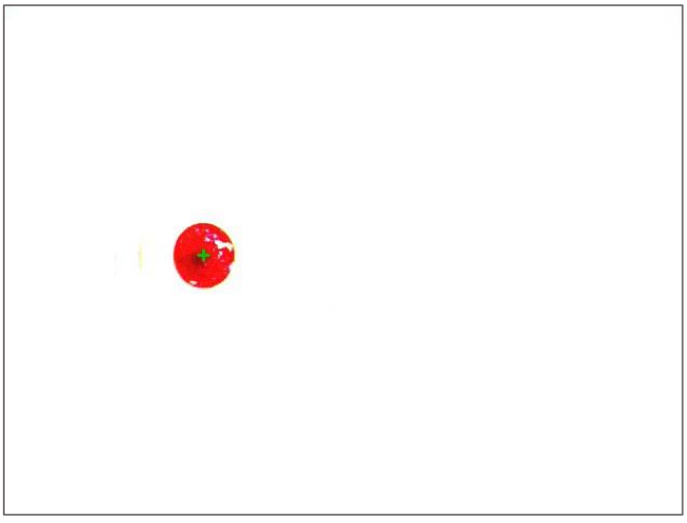


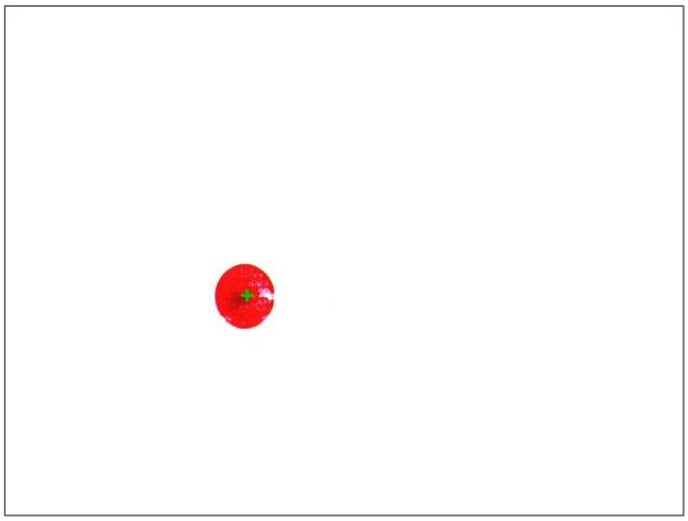
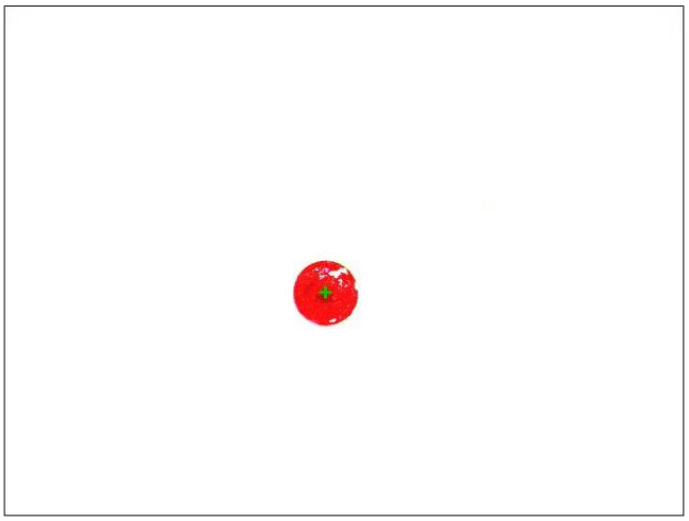
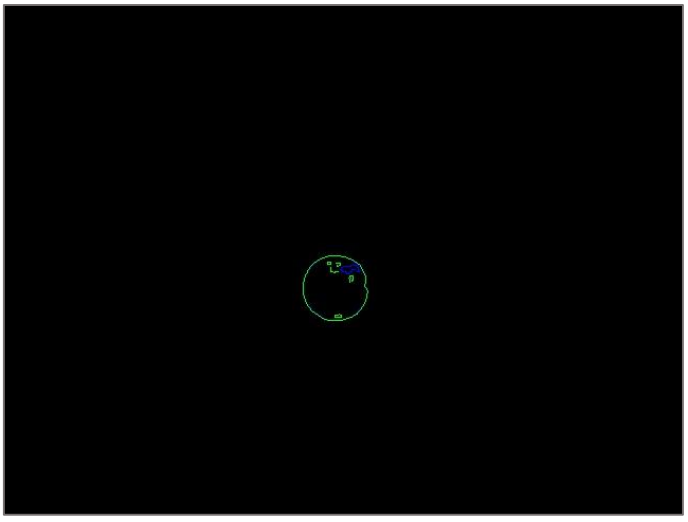
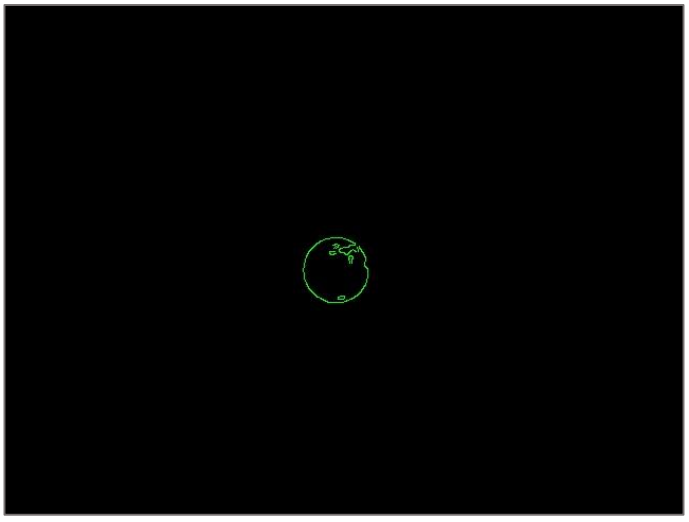
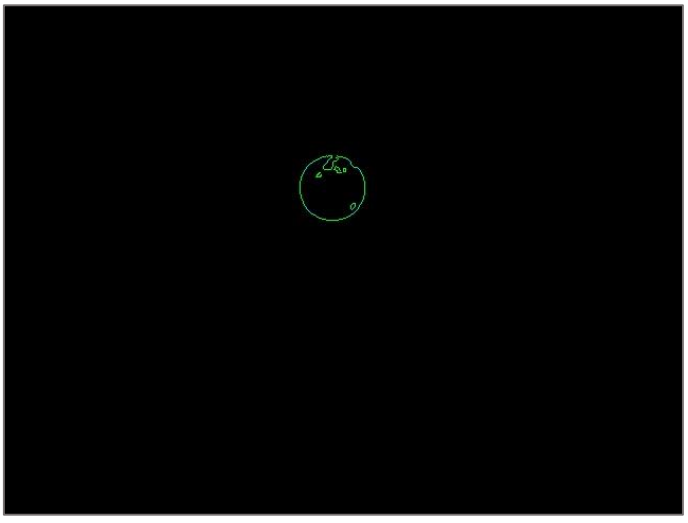
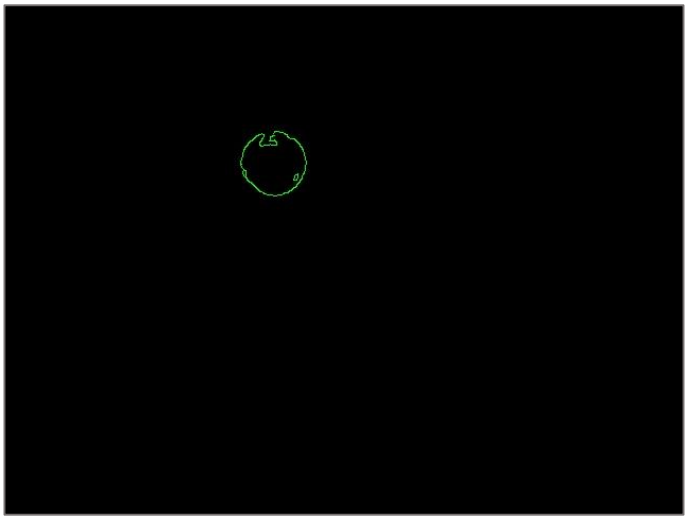


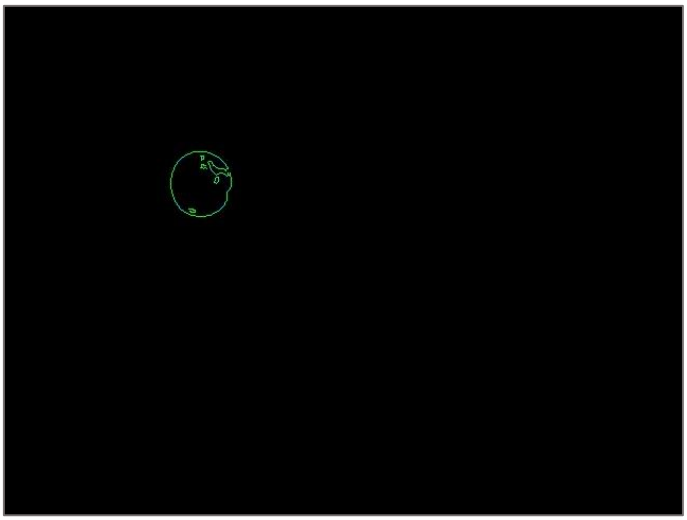
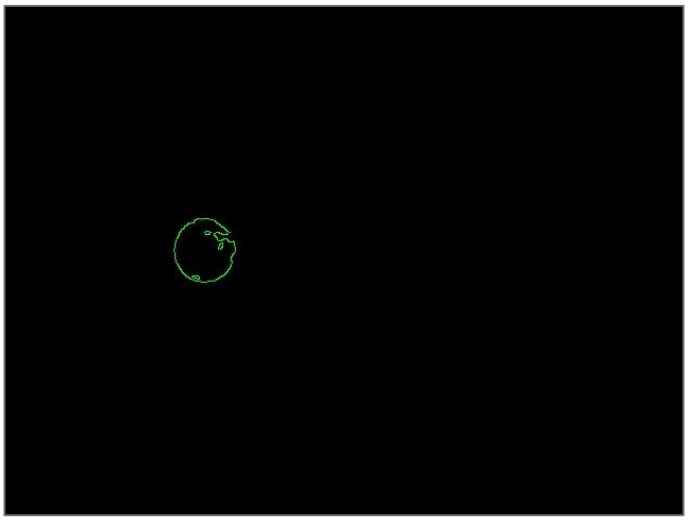
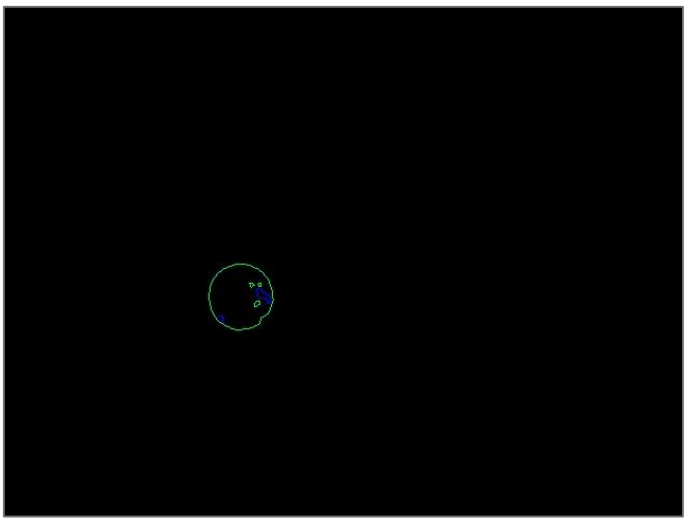
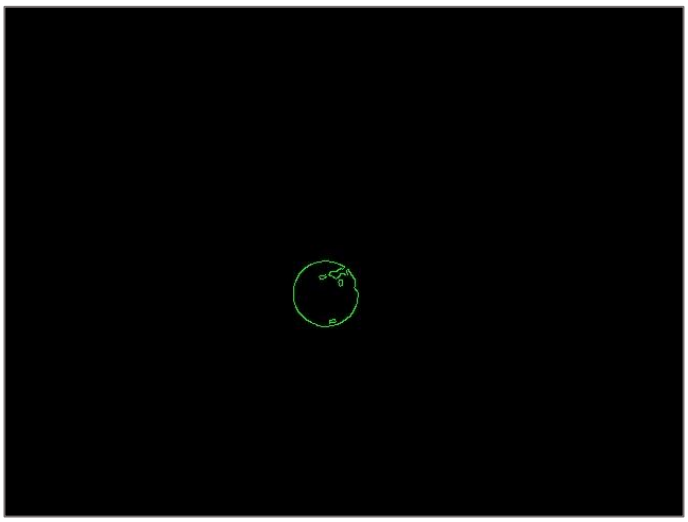






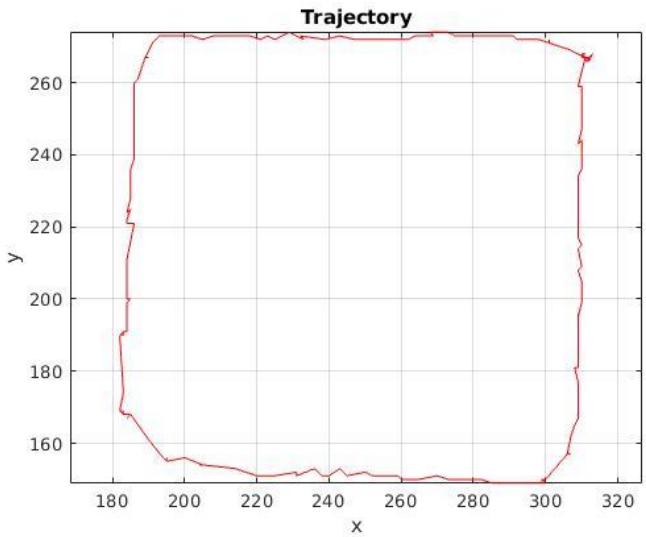


comparison tracked trajectories:

thresh = edgeparam :



thresh = 2*edgeparam :