Assignment 2

Collision Avoidance with Optical-Flow using OpenCV

Submitted by
Taylor Whitaker
Ying Fan
Md Jubaer Hossain Pantho
Pankaj Bhowmik

Objective

The main object of this project is to:

- Familiarization with openCV.
- Implementation of Optical Flow in openCV
- Tracking an object
- Design and implementation of a collision avoidance system

Optical Flow:

The optical flow methods try to calculate the motion between two image frames which are taken at times *t* and *t+dt* at every pixel position. The objective of the Optical Flow method is to reconstruct the displacement vector field of objects captured with a sequence of images. In fact, based on a set of images capturing the motion of one or multiple objects, we want to be able to reconstruct the displacement field associated to each pixel during the time difference from one frame to another frame. The common application of optical flow in robotics, motion detection system, surveillance system, and autonomous cars.

Implementation:

We used Farneback optical flow algorithm in this project. This algorithm is basically a

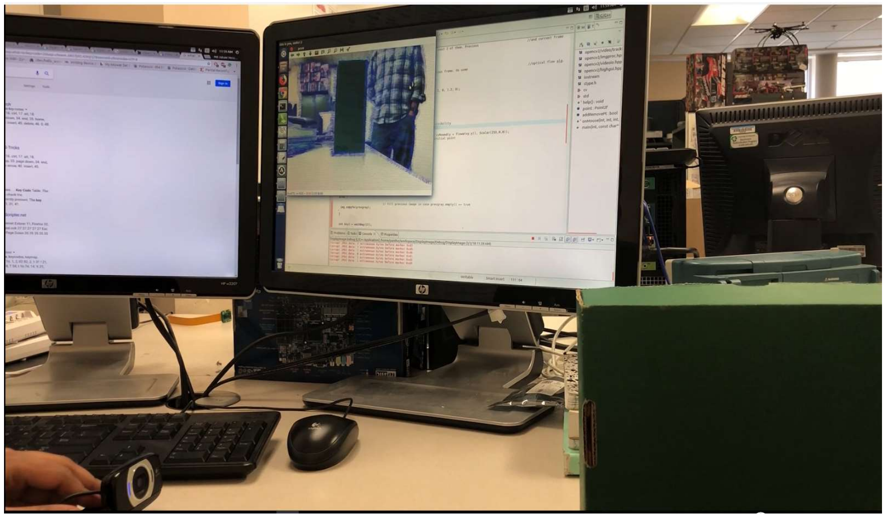


Figure 1: Experimental Setup

two-frame motion estimation based on polynomial expansion. The idea of polynomial expansion is to approximate some neighborhood of each pixel with a polynomial expression.

To implement this algorithm we used a web-camera which captures the video and the Farneback implemented using openCV library detects the real-time optical flow in the video file. We demonstrated the motion vector as optical flow by moving the camera towards an object. The experimental setup is shown in the Figure 1.

In figure 2, we represent the initial condition where there is no relative motion; object and camera are stable. But when we start moving the camera, we observed the motion vector as blue around objects. The vector represents the optical flow. We illustrated it in figure 3. The figure shows that, some blue lines are visible surrounding an object when we move the camera in real-time. If there is flow, the console generates an output "Object detected", which gives the authenticity of successful implementation of optical flow algorithm. The attached video file (optical_flow.mov) gives the proof of the implementation.

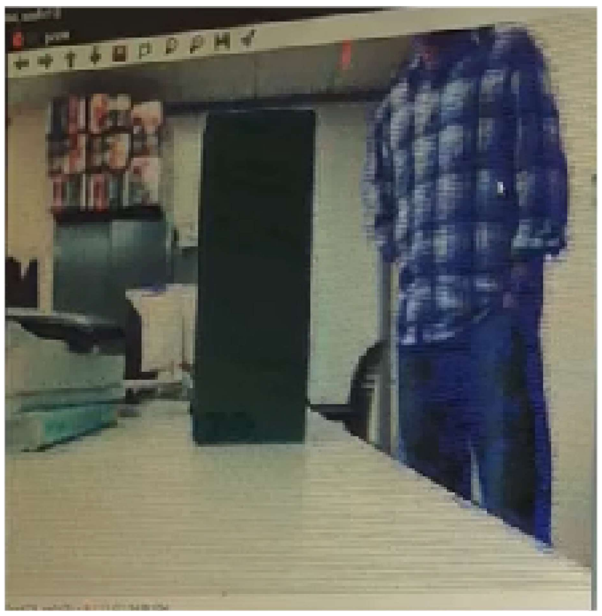


Figure 2: Stable Objects and Camera

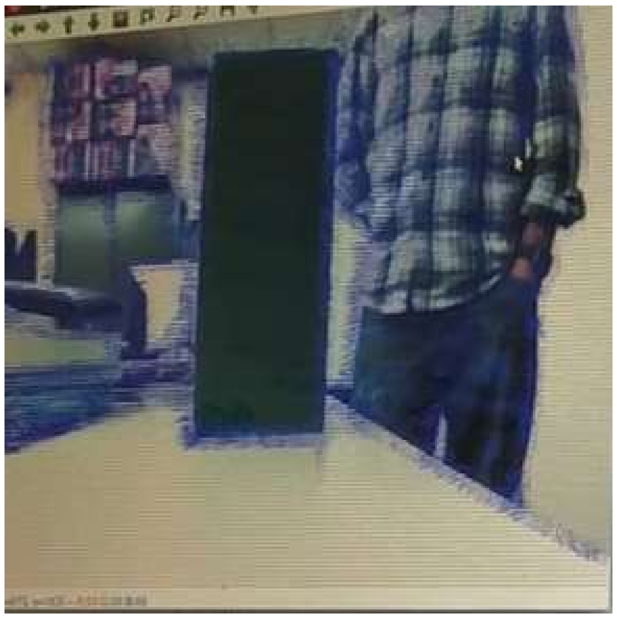
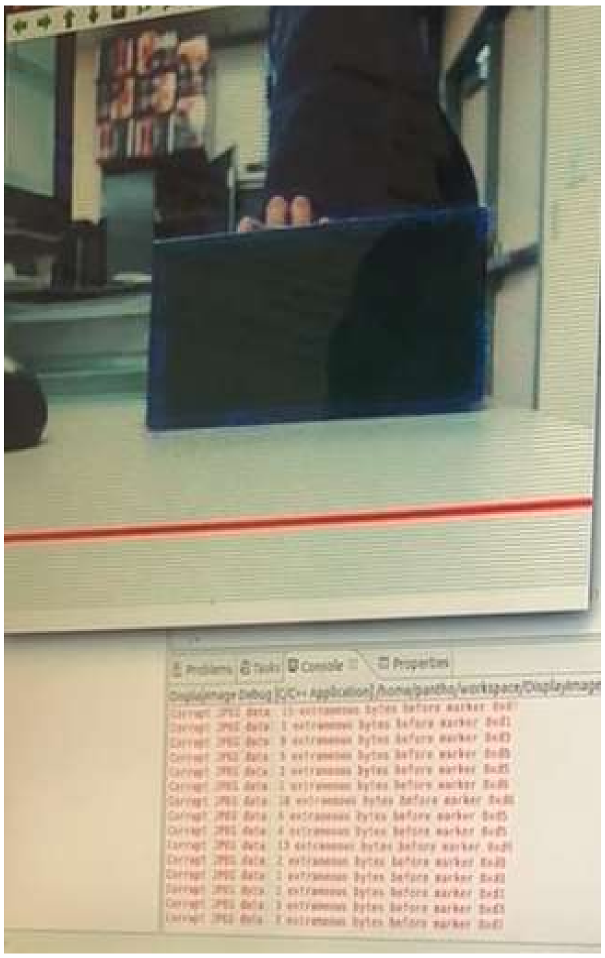


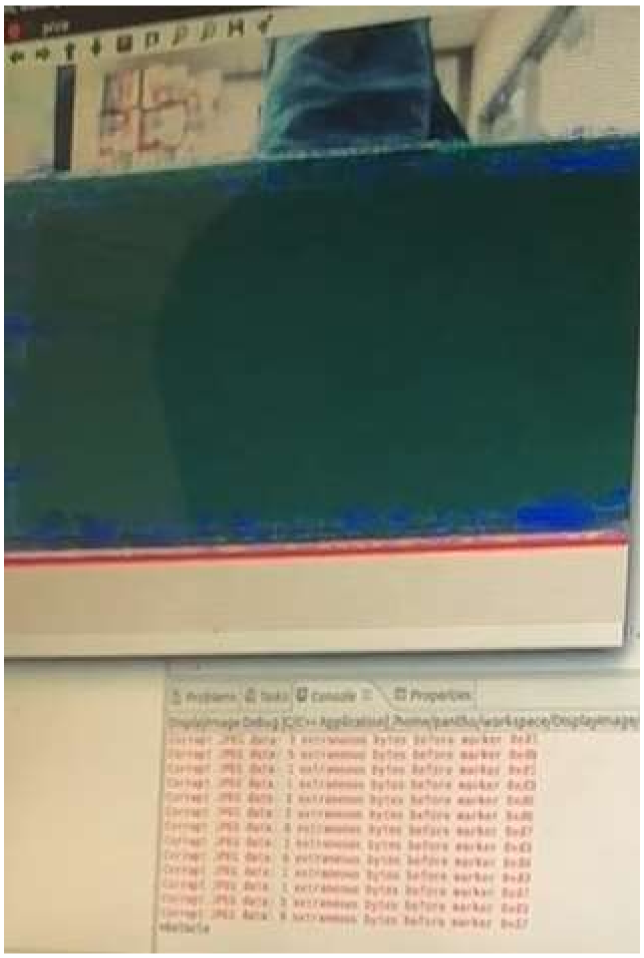
Figure 3: Optical flow generated by moving camera

In the second phase of the project, we implemented a pragmatic application of optical flow which is collision avoidance. In the system, we have considered the camera is stable and an object is heading towards it. When the object will come closer to it by a sudden amount with a sudden amount of speed so that the consecutive frames are distinguishable in terms of optical flow, then our system detects that optical flow and generates a signal

in the console as “Obstacle Detected”. We represent the maximum allowed distance by the red line as shown in figure 4. In this figure, the object is moving but this is far enough from the camera. As the object comes to the boundary line, we observed the blue vector lines surrounding the object as depicted in figure 5. When the motion is detected on the boundary lines we generated the interrupt and which is also shown in the programming console. The signal is helpful for the robots to avoid a collision.
In addition the attached video file (Object Detection.MOV) gives the proof of the implementation.



```
DisplayImage Debug [C/C++ Application] /home/pantho/workspace/DisplayImage
Corrupt JPEG data: 13 extraneous bytes before marker 0xd7
Corrupt JPEG data: 1 extraneous bytes before marker 0xd1
Corrupt JPEG data: 9 extraneous bytes before marker 0xd3
Corrupt JPEG data: 5 extraneous bytes before marker 0xdb
Corrupt JPEG data: 3 extraneous bytes before marker 0xd5
Corrupt JPEG data: 1 extraneous bytes before marker 0xd5
Corrupt JPEG data: 18 extraneous bytes before marker 0xd6
Corrupt JPEG data: 4 extraneous bytes before marker 0xd5
Corrupt JPEG data: 4 extraneous bytes before marker 0xd5
Corrupt JPEG data: 13 extraneous bytes before marker 0xd6
Corrupt JPEG data: 2 extraneous bytes before marker 0xd0
Corrupt JPEG data: 1 extraneous bytes before marker 0xd2
Corrupt JPEG data: 2 extraneous bytes before marker 0xd1
Corrupt JPEG data: 3 extraneous bytes before marker 0xd3
Corrupt JPEG data: 7 extraneous bytes before marker 0xd2
```



```
DisplayImage Debug [C/C++ Application] /home/pantho/workspace/DisplayImage
Corrupt JPEG data: 3 extraneous bytes before marker 0xd1
Corrupt JPEG data: 5 extraneous bytes before marker 0xdb
Corrupt JPEG data: 2 extraneous bytes before marker 0xd1
Corrupt JPEG data: 1 extraneous bytes before marker 0xd3
Corrupt JPEG data: 8 extraneous bytes before marker 0xd0
Corrupt JPEG data: 3 extraneous bytes before marker 0xd0
Corrupt JPEG data: 8 extraneous bytes before marker 0xd7
Corrupt JPEG data: 2 extraneous bytes before marker 0xd3
Corrupt JPEG data: 6 extraneous bytes before marker 0xd6
Corrupt JPEG data: 2 extraneous bytes before marker 0xd3
Corrupt JPEG data: 1 extraneous bytes before marker 0xd1
Corrupt JPEG data: 3 extraneous bytes before marker 0xd3
Corrupt JPEG data: 8 extraneous bytes before marker 0xd7
[illegible]
```

Figure: 4 Object is in motion but not closer to camera   Fig 5: Object is heading to camera and crosses max distance

Conclusion:

Optical flow is being used in robotics applications, primarily where there is a need to measure visual motion or relative motion between the robot and other objects in the vicinity of the robot. Using this algorithm the robots can avoid collision among them and with different objects. Our experimental results show that the system we have developed, successfully tracks objects and generates signal for possible collision avoidance.

In addition, we have attached the video links of our implementation in Appendix A and provided the CPP code in Appendix B

Appendix A:

1. Optical_Flow.MOV
2. Object_detection.MOV

Appendix B:

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"

#include <iostream>
#include <ctype.h>

using namespace cv;
using namespace std;

static void help()
{
   // print a welcome message, and the OpenCV version
   cout << "\nThis is a demo of Lukas-Kanade optical flow lkdemo(),\n"
         "Using OpenCV version " << CV_VERSION << endl;
   cout << "\nIt uses camera by default, but you can provide a path to video as an argument.\n";
   cout << "\nHot keys: \n"
         "\tESC - quit the program\n"
         "\tr - auto-initialize tracking\n"
         "\tc - delete all the points\n"
         "\tn - switch the \"night\" mode on/off\n"
         "To add/remove a feature point click it\n" << endl;
}

Point2f point;
```

```
bool addRemovePt = false;

static void onMouse( int event, int x, int y, int /*flags*/, void* /*param*/ )
{
   if( event == EVENT_LBUTTONDOWN )
   {
      point = Point2f((float)x, (float)y);
      addRemovePt = true;
   }
}

int main(int argc, const char** argv)
{

 // add your file name
 //VideoCapture cap("yourFile.mp4");

   VideoCapture cap;
   // open the default camera, use something different from 0 otherwise;
   // Check VideoCapture documentation.
   if(!cap.open(0))
      return 0;
   for(;;)
   {
        Mat frame;
        cap >> frame;
        if( frame.empty() ) break; // end of video stream
        imshow("this is you, smile! :)", frame);
        if( waitKey(10) == 27 ) break; // stop capturing by pressing ESC
   }


 Mat flow, frame;
        // some faster than mat image container
 UMat  flowUmat, prevgray;

 for (;;)
 {
```

```
 bool Is = cap.grab();
 if (Is == false) {
                    // if video capture failed
  cout << "Video Capture Fail" << endl;
  break;
 }
 else {



  Mat img;
  Mat original;

  // capture frame from video file
  cap.retrieve(img, CV_CAP_OPENNI_BGR_IMAGE);
  resize(img, img, Size(800, 600));

  // save original for later
  img.copyTo(original);

  // just make current frame gray
  cvtColor(img, img, COLOR_BGR2GRAY);


     // For all optical flow you need a sequence of images.. Or at least 2 of them. Previous
//and current frame
  //if there is no current frame
  // go to this part and fill previous frame
  //else {
  // img.copyTo(prevgray);
  //   }
      // if previous frame is not empty.. There is a picture of previous frame. Do some
//optical flow alg.

  if (prevgray.empty() == false ) {

   // calculate optical flow
   calcOpticalFlowFarneback(prevgray, img, flowUmat, 0.4, 1, 12, 2, 8, 1.2, 0);
   // copy Umat container to standard Mat
```

```
    flowUmat.copyTo(flow);


            // By y += 5, x += 5 you can specify the grid
    for (int y = 0; y < original.rows; y += 5) {
     for (int x = 0; x < original.cols; x += 5)
     {
               // get the flow from y, x position * 10 for better visibility
               const Point2f flowatxy = flow.at<Point2f>(y, x)*1;
                     // draw line at flow direction
               line(original,  Point(x,  y),  Point(cvRound(x  +  flowatxy.x),  cvRound(y  +  flowatxy.y)),
Scalar(255,0,0));
                                                     // draw initial point
       circle(original, Point(x, y), 1, Scalar(0, 0, 0), -1);

       if(y>=520 && y<600 && x>0 && x<700)
       {
              if((x-((int) (x + flowatxy.x)))>5 && (y-((int) (y + flowatxy.y)))>5)
              {
                                  cout<< "obstacle\n";
                                  getchar();
              }

       }

      }

     }


            // Boundary line for object being too close
    line(original, Point(0, 520), Point(800, 520), Scalar(0,0,255), 5);


    // draw the results
    namedWindow("prew", WINDOW_AUTOSIZE);
    imshow("prew", original);

                              // fill previous image again
    img.copyTo(prevgray);

   }
   else {

                              // fill previous image in case prevgray.empty() == true
    img.copyTo(prevgray);

   }


   int key1 = waitKey(27);

  }
 }
}
```

****

```
/*#include <opencv2/opencv.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "iostream"
#include <time.h>
using namespace std;

using namespace cv;

void help()
{
 cout << "\nThis program demonstrates line finding with the Hough transform.\n"
         "Usage:\n"
         "./houghlines <image_name>, Default is pic1.jpg\n" << endl;
}

int main( int argc, char** argv )
{
  clock_t tStart = clock();
  Mat src;
 src = imread("Lane.jpg");

 Mat dst, cdst;
 Canny(src, dst, 50, 180, 3);
 cvtColor(dst, cdst, COLOR_GRAY2BGR);

 #if 0
  vector<Vec2f> lines;
  HoughLines(dst, lines, 1, CV_PI/180, 100, 0, 0 );

  for( size_t i = 0; i < lines.size(); i++ )
  {
     float rho = lines[i][0], theta = lines[i][1];
     Point pt1, pt2;
     double a = cos(theta), b = sin(theta);
     double x0 = a*rho, y0 = b*rho;
     pt1.x = cvRound(x0 + 1000*(-b));
     pt1.y = cvRound(y0 + 1000*(a));
     pt2.x = cvRound(x0 - 1000*(-b));
     pt2.y = cvRound(y0 - 1000*(a));
     line( cdst, pt1, pt2, Scalar(0,0,255), 3, LINE_AA);
  }
 #else
  vector<Vec4i> lines;
  HoughLinesP(dst, lines, 1, CV_PI/180, 50, 50, 15 );
  for( size_t i = 0; i < lines.size(); i++ )
  {
    Vec4i l = lines[i];
    line( cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, CV_AA);
  }
 #endif
 imshow("source", dst);
 imshow("detected lines", cdst);
 waitKey(0);

 return 0;
}
```