Calibrate AR Drone’s Camera and perform online optical flow

Welcome

Lab 7

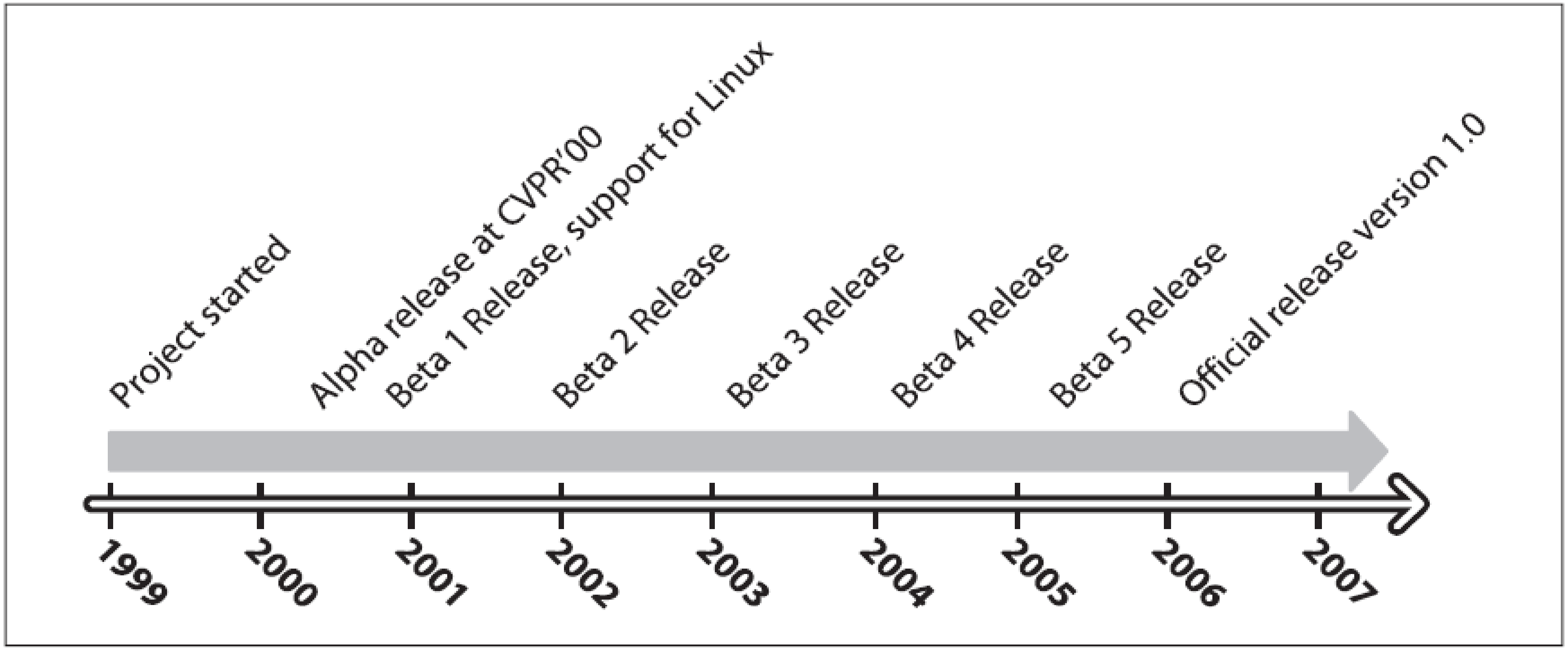
Dr. Ahmad Kamal Nasir

Today's Objectives

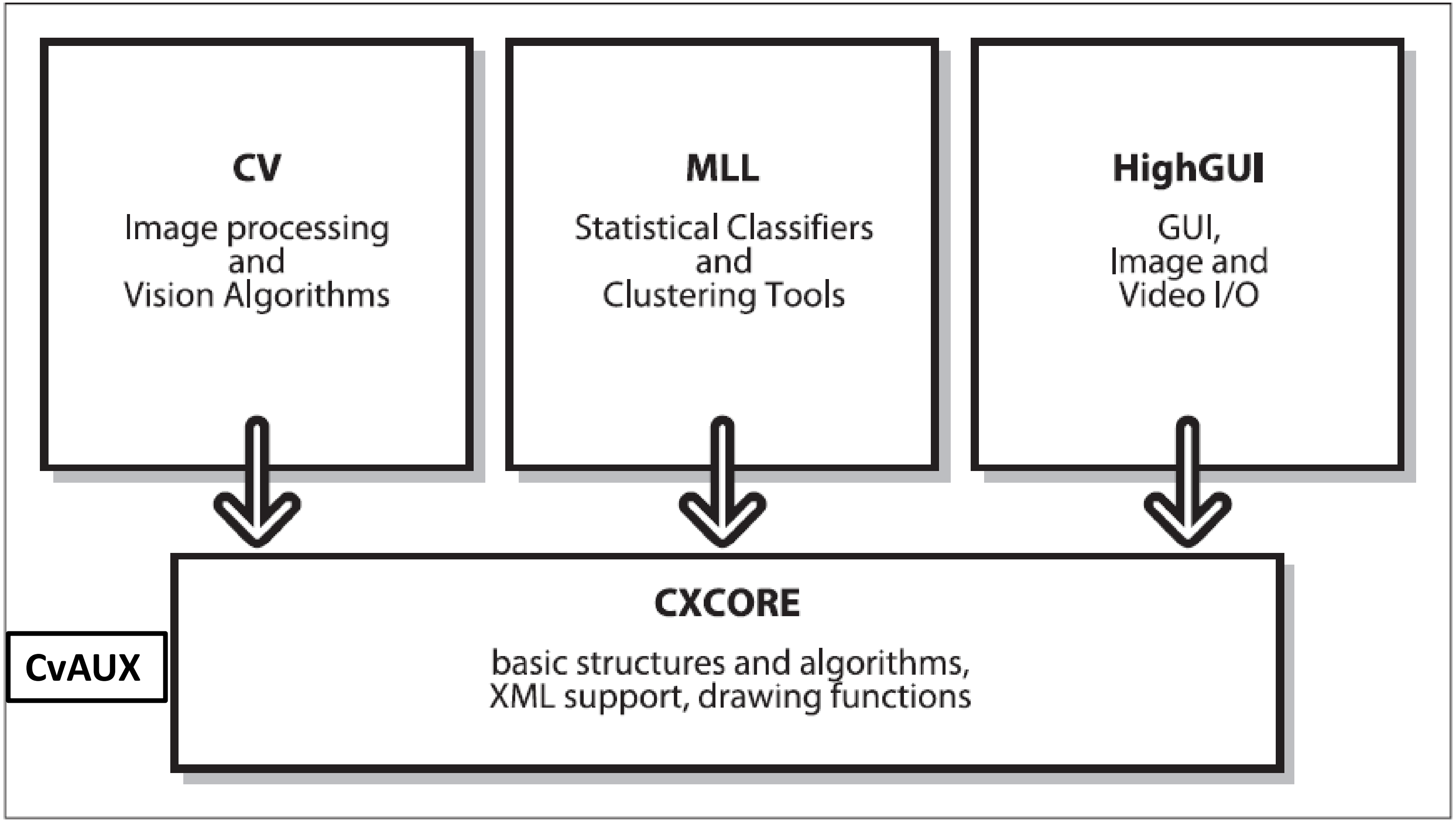
- Introduction to OpenCV
- ROS interface with OpenCV
- Camera Calibration
- Optical Flow

Introduction to OpenCV

- Open Source Computer Vision Library
  - Created by Intel and Maintained by Willow Garage
  - Written in C/C++ for Linux, Windows
  - 500+ functions



Basic Structure of OpenCV



CV

Image processing
and
Vision Algorithms

MLL

Statistical Classifiers
and
Clustering Tools

HighGUI

GUI,
Image and
Video I/O

CXCORE

basic structures and algorithms,
XML support, drawing functions

CvAUX

A basic example

```
#include <opencv2\opencv.hpp>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

using namespace cv;

int main( int argc, char** argv )
{
        VideoCapture cap(0);  // Open the default camera
        if(!cap.isOpened())   // Exit, if camera is not successfully acquired
                return -1;
        Mat frame;            // Create image storage
        cap >> frame;         // Acquire a new frame from camera
        imshow("Captured Frame", frame); // Display image
        waitKey(0);           // Wait for a keystroke in the window
        if (cap.isOpened())   // Close camera device
                cap.release();
        return 0;
}
```

OpenCV Basics: Mat

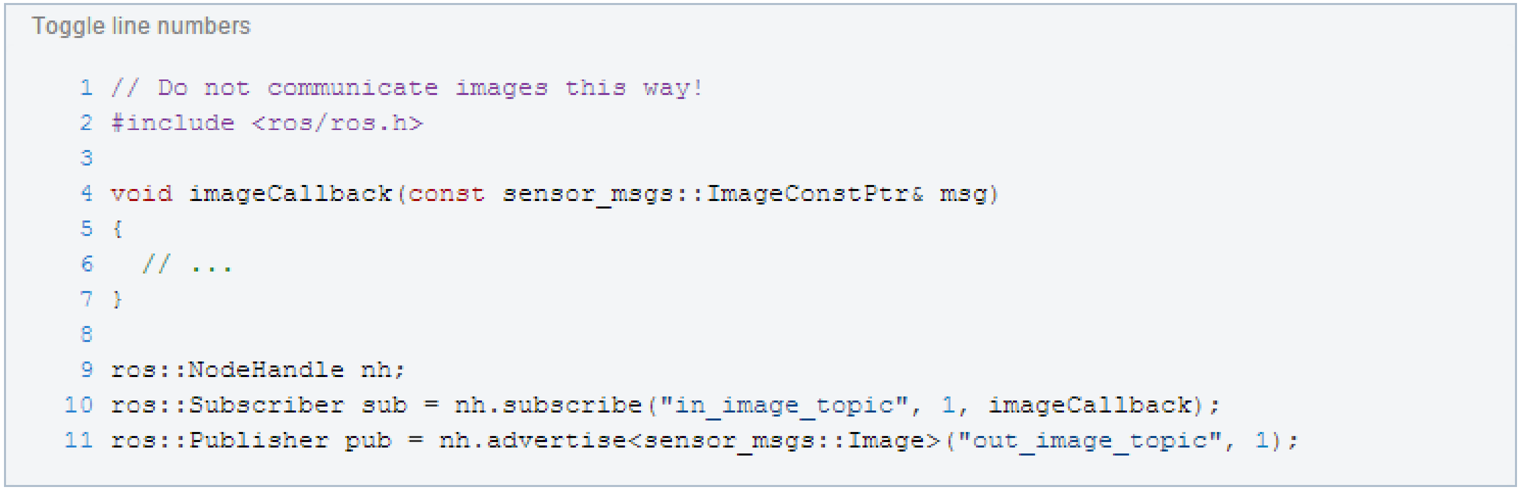
```
#include <opencv2\opencv.hpp>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
Mat C = (Mat_<double>(3,3) << 0, -1.2, 0, -1, 5, -1, 0, -1, 0);
Mat a(4,4,CV_32S);
randu(a,Scalar::all(1), Scalar::all(10));
cout << a << endl << a(Rect(1,1,2,2));

Mat image = imread(argv[1], IMREAD_COLOR);
image.at<Vec3b>(row,col)[0] = 0;
image.at<Vec3b>(row,col)[1] = 0;
image.at<Vec3b>(row,col)[2] = 0;

vector<Vec3d> vv;
vv.push_back(Vec3d(1,2,3));
vv.push_back(Vec3d(4,5,6));
vv.push_back(Vec3d(7,8,9));
Mat vvv(vv);

return 0;
}
```

```
C:
+000.00 -001.20 +000.00
-001.00 +005.00 -001.00
+000.00 -001.00 +000.00
A:
 [8, 6, 5, 3;
  5, 8, 6, 3;
  8, 9, 6, 3;
  1, 6, 1, 7]
A(rect(1,1,2,2)):
 [8, 6;
  9, 6]
vvv:
+001.00 +002.00 +003.00
+004.00 +005.00 +006.00
+007.00 +008.00 +009.00
```

Image_transport

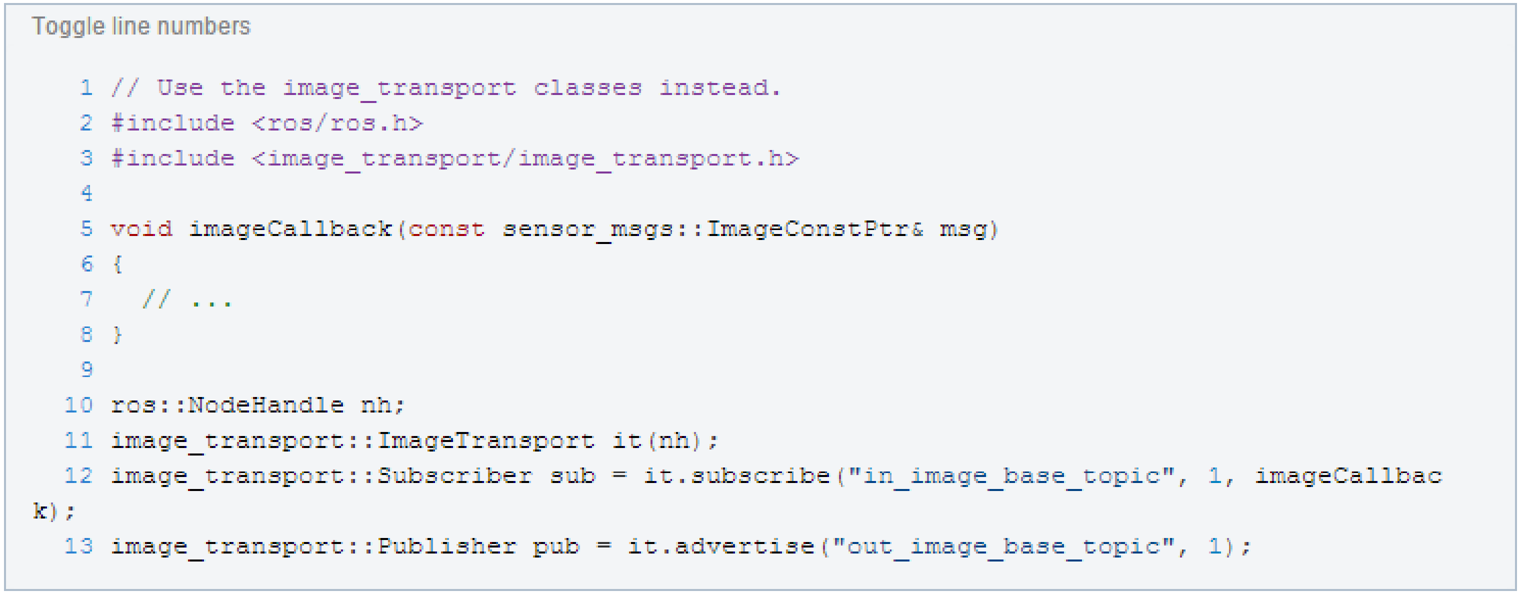
- Image Transport is a package in ROS that makes image usage streamlined by specialized transport strategies of image compression or video codecs.
- `image_transport` should always be used to publish and subscribe to image
- You will use this package to subscribe to your camera feed (from AR Drone) and then run optical flow on it.
- Also, once you have optical flow results, you’d publish them as rostopic using this package again
- Find publisher and subscriber tutorials on wiki.ros.org for image_transport

Instead of:



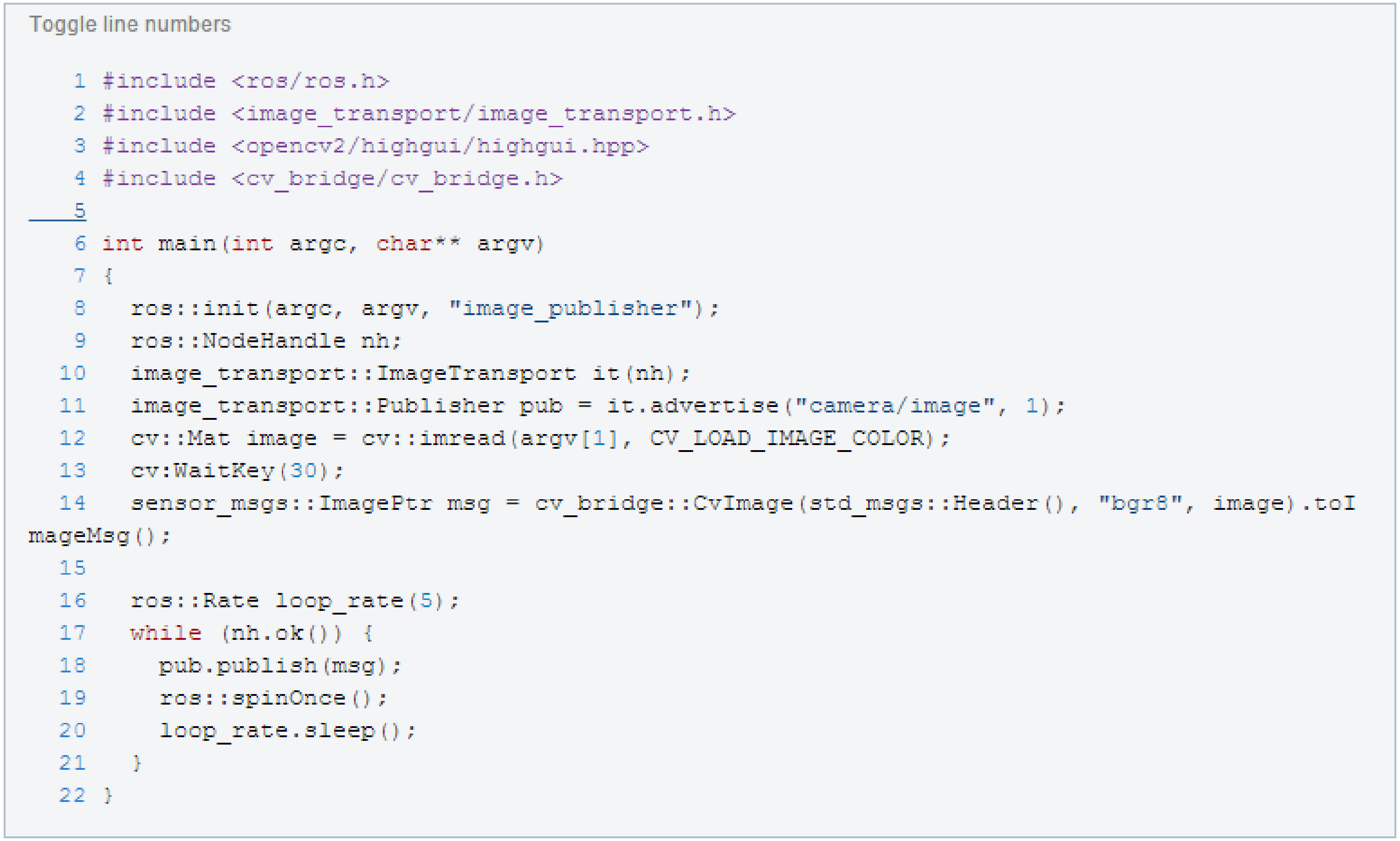
```
// Do not communicate images this way!
#include <ros/ros.h>

void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
  // ...
}

ros::NodeHandle nh;
ros::Subscriber sub = nh.subscribe("in_image_topic", 1, imageCallback);
ros::Publisher pub = nh.advertise<sensor_msgs::Image>("out_image_topic", 1);
```

Do:



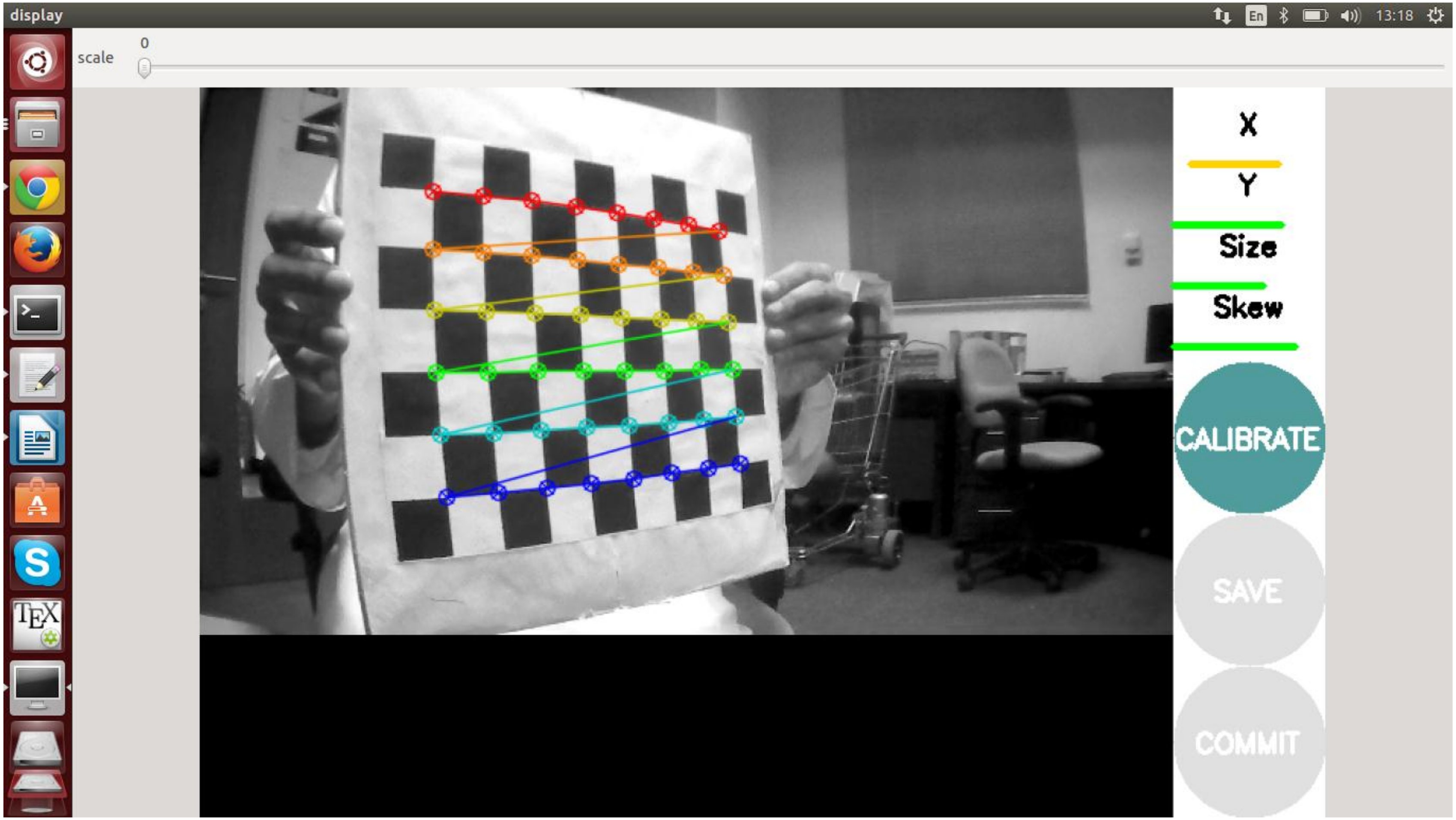
```
// Use the image_transport classes instead.
#include <ros/ros.h>
#include <image_transport/image_transport.h>

void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
  // ...
}

ros::NodeHandle nh;
image_transport::ImageTransport it(nh);
image_transport::Subscriber sub = it.subscribe("in_image_base_topic", 1, imageCallback);
image_transport::Publisher pub = it.advertise("out_image_base_topic", 1);
```

Image Publisher

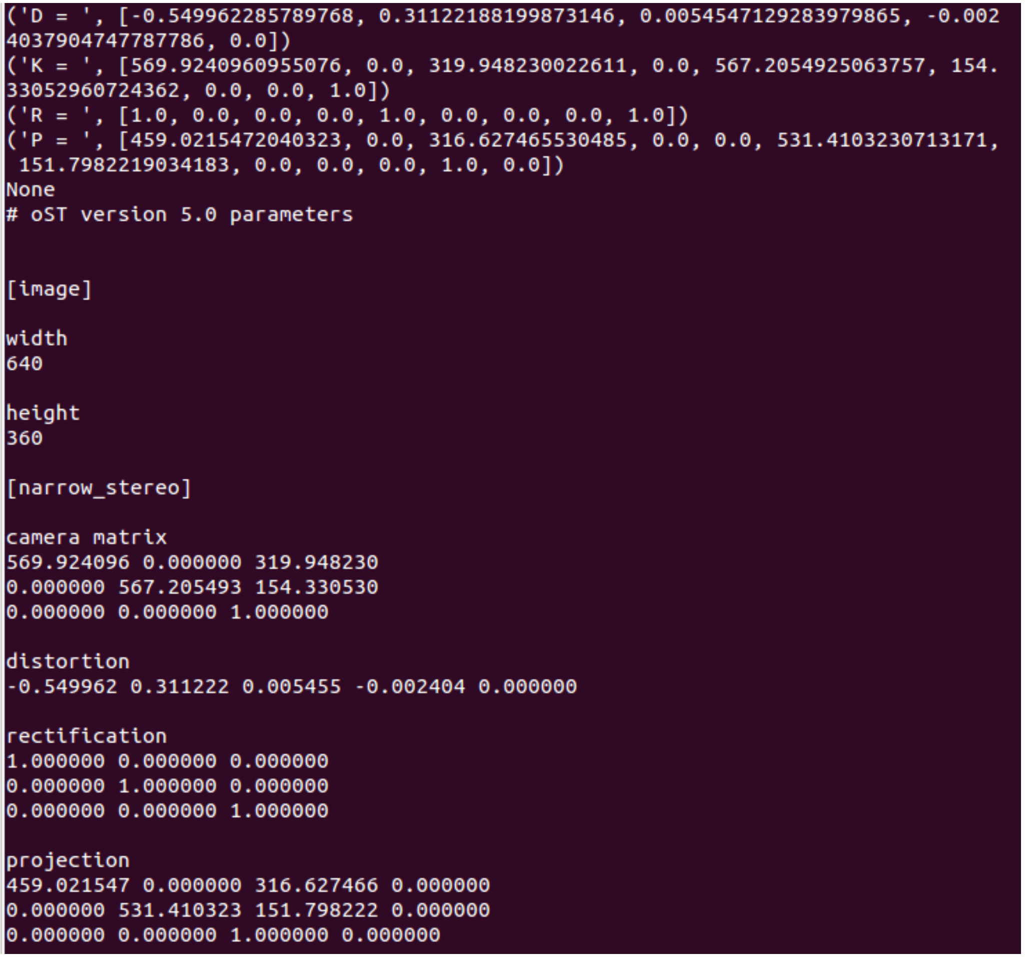


```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <opencv2/highgui/highgui.hpp>
#include <cv_bridge/cv_bridge.h>

int main(int argc, char** argv)
{
  ros::init(argc, argv, "image_publisher");
  ros::NodeHandle nh;
  image_transport::ImageTransport it(nh);
  image_transport::Publisher pub = it.advertise("camera/image", 1);
  cv::Mat image = cv::imread(argv[1], CV_LOAD_IMAGE_COLOR);
  cv:WaitKey(30);
  sensor_msgs::ImagePtr msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", image).toImageMsg();

  ros::Rate loop_rate(5);
  while (nh.ok()) {
    pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
  }
}
```

Camera Calibration



Camera Calibration (Cont.)

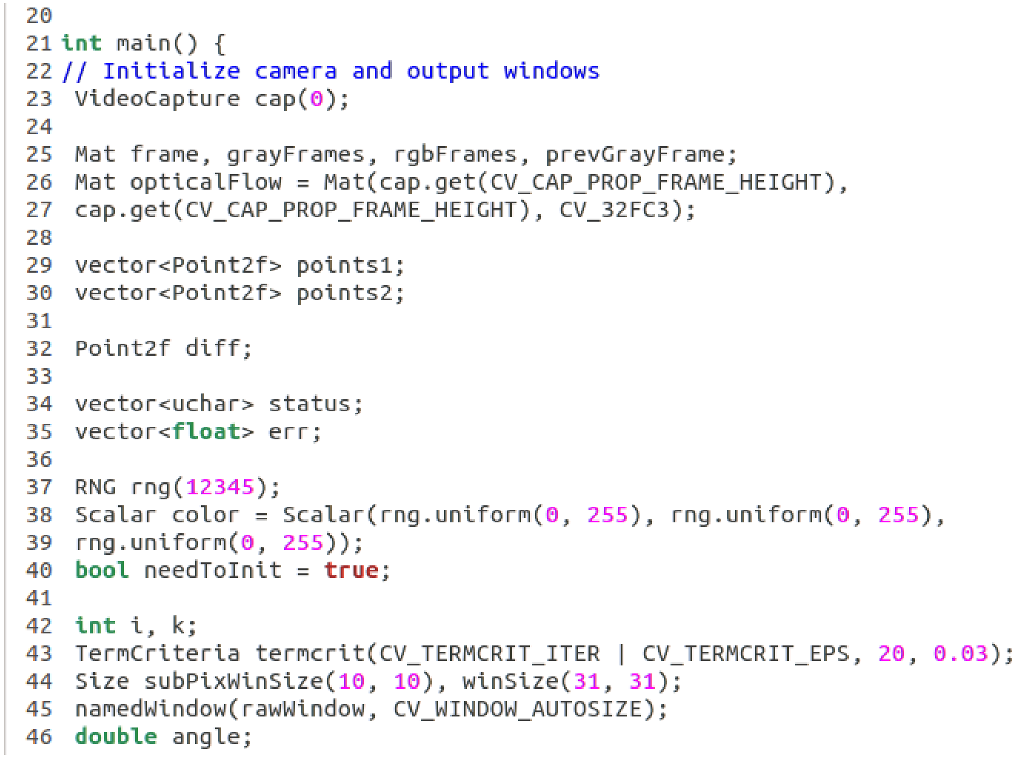


```
('D = ', [-0.549962285789768, 0.31122188199873146, 0.0054547129283979865, -0.002
4037904747787786, 0.0])
('K = ', [569.9240960955076, 0.0, 319.948230022611, 0.0, 567.2054925063757, 154.
33052960724362, 0.0, 0.0, 1.0])
('R = ', [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0])
('P = ', [459.0215472040323, 0.0, 316.627465530485, 0.0, 0.0, 531.4103230713171,
 151.7982219034183, 0.0, 0.0, 0.0, 1.0, 0.0])
None
# oST version 5.0 parameters


[image]

width
640

height
360

[narrow_stereo]

camera matrix
569.924096 0.000000 319.948230
0.000000 567.205493 154.330530
0.000000 0.000000 1.000000

distortion
-0.549962 0.311222 0.005455 -0.002404 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
459.021547 0.000000 316.627466 0.000000
0.000000 531.410323 151.798222 0.000000
0.000000 0.000000 1.000000 0.000000
```

Task 1: Camera Calibration

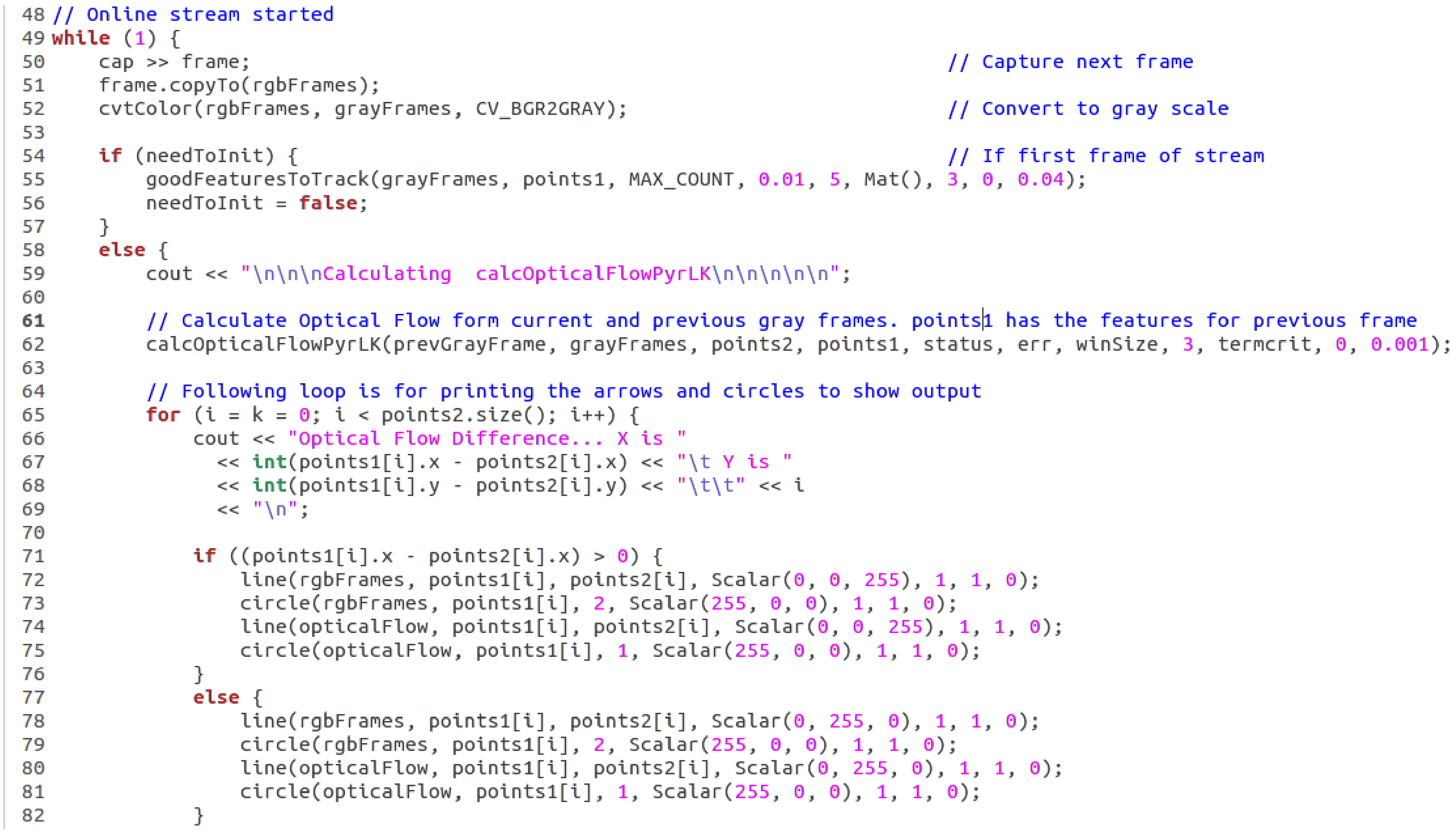
- Camera calibration is the process of inferring camera parameters (intrinsic and extrinsic) with the help of dataset of images of a known object e.g. chessboard, circles, etc.
- You will use chessboard of 9x6 size. With known square size, and it's black and white nature, a calibration utility can easily detect the corners in an image of chessboard.
- Upon each consecutive frame the utility itself can find correspondences and based on

Optical Flow



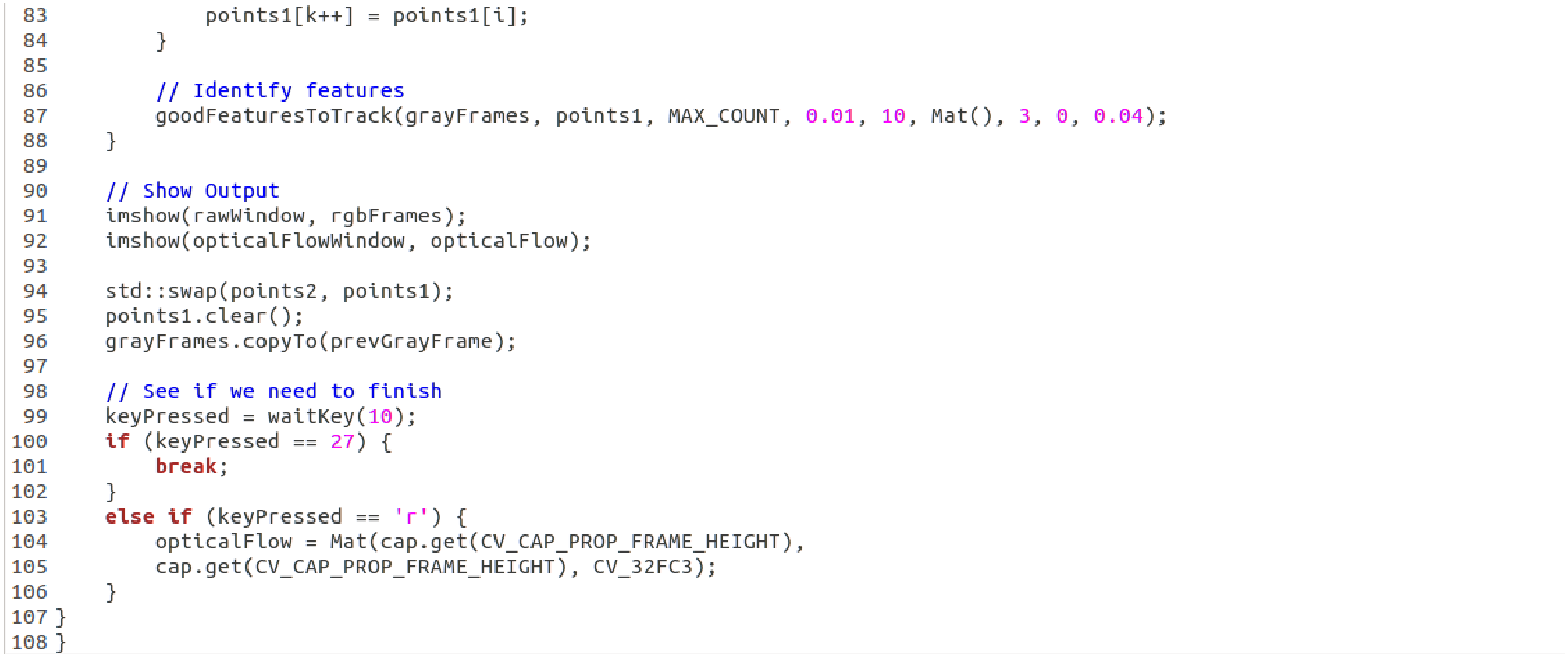
```

int main() {
// Initialize camera and output windows
 VideoCapture cap(0);

 Mat frame, grayFrames, rgbFrames, prevGrayFrame;
 Mat opticalFlow = Mat(cap.get(CV_CAP_PROP_FRAME_HEIGHT),
 cap.get(CV_CAP_PROP_FRAME_HEIGHT), CV_32FC3);

 vector<Point2f> points1;
 vector<Point2f> points2;

 Point2f diff;

 vector<uchar> status;
 vector<float> err;

 RNG rng(12345);
 Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255),
 rng.uniform(0, 255));
 bool needToInit = true;

 int i, k;
 TermCriteria termcrit(CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, 0.03);
 Size subPixWinSize(10, 10), winSize(31, 31);
 namedWindow(rawWindow, CV_WINDOW_AUTOSIZE);
 double angle;
```

Optical Flow (Cont.)

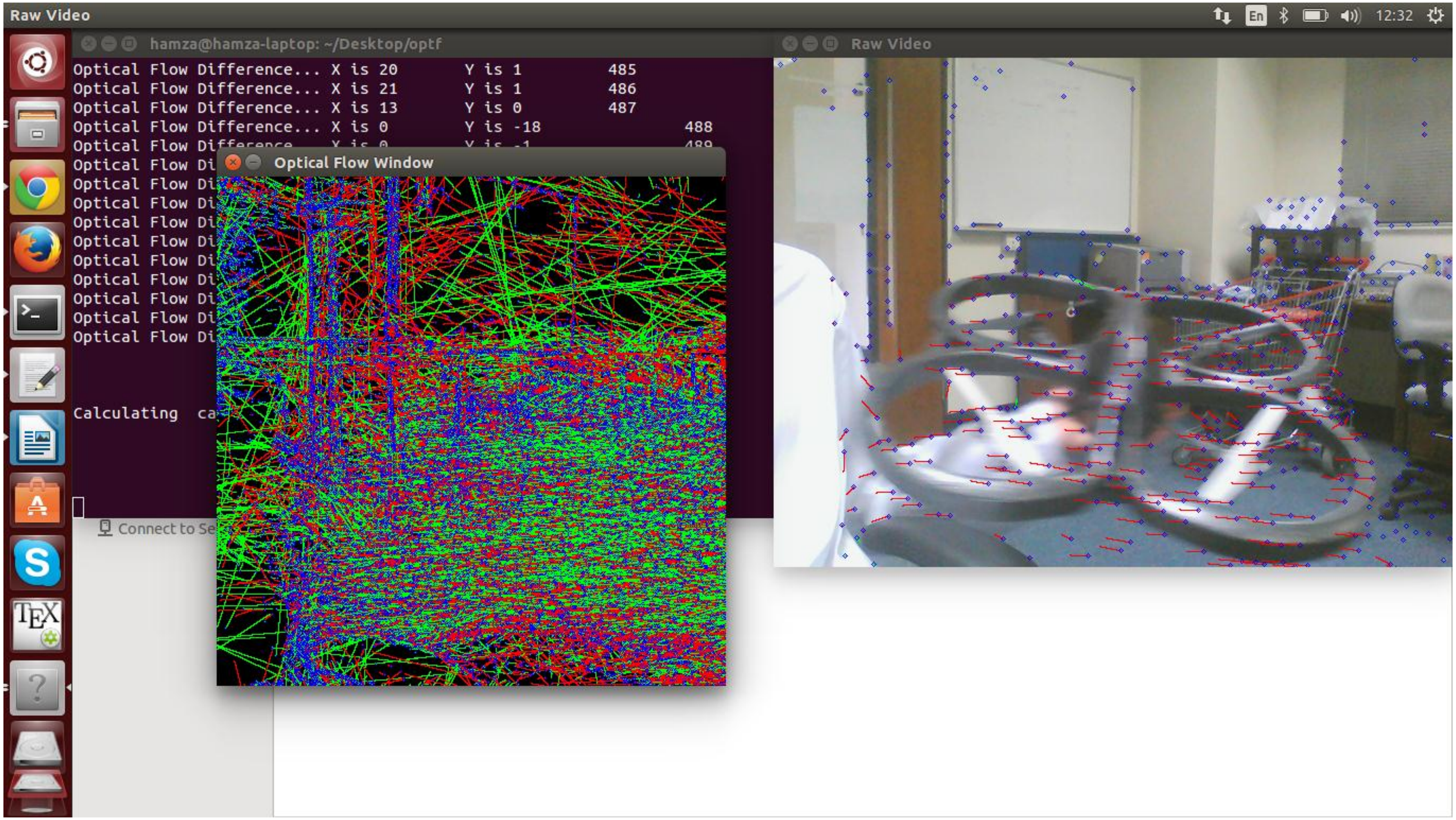


```
// Online stream started
while (1) {
    cap >> frame;                                                           // Capture next frame
    frame.copyTo(rgbFrames);
    cvtColor(rgbFrames, grayFrames, CV_BGR2GRAY);                           // Convert to gray scale

    if (needToInit) {                                                       // If first frame of stream
        goodFeaturesToTrack(grayFrames, points1, MAX_COUNT, 0.01, 5, Mat(), 3, 0, 0.04);
        needToInit = false;
    }
    else {
        cout << "\n\n\nCalculating  calcOpticalFlowPyrLK\n\n\n\n\n";

        // Calculate Optical Flow form current and previous gray frames. points1 has the features for previous frame
        calcOpticalFlowPyrLK(prevGrayFrame, grayFrames, points2, points1, status, err, winSize, 3, termcrit, 0, 0.001);

        // Following loop is for printing the arrows and circles to show output
        for (i = k = 0; i < points2.size(); i++) {
            cout << "Optical Flow Difference... X is "
              << int(points1[i].x - points2[i].x) << "\t Y is "
              << int(points1[i].y - points2[i].y) << "\t\t" << i
              << "\n";

            if ((points1[i].x - points2[i].x) > 0) {
                line(rgbFrames, points1[i], points2[i], Scalar(0, 0, 255), 1, 1, 0);
                circle(rgbFrames, points1[i], 2, Scalar(255, 0, 0), 1, 1, 0);
                line(opticalFlow, points1[i], points2[i], Scalar(0, 0, 255), 1, 1, 0);
                circle(opticalFlow, points1[i], 1, Scalar(255, 0, 0), 1, 1, 0);
            }
            else {
                line(rgbFrames, points1[i], points2[i], Scalar(0, 255, 0), 1, 1, 0);
                circle(rgbFrames, points1[i], 2, Scalar(255, 0, 0), 1, 1, 0);
                line(opticalFlow, points1[i], points2[i], Scalar(0, 255, 0), 1, 1, 0);
                circle(opticalFlow, points1[i], 1, Scalar(255, 0, 0), 1, 1, 0);
            }
```

Optical Flow (Cont.)



```
            points1[k++] = points1[i];
        }

        // Identify features
        goodFeaturesToTrack(grayFrames, points1, MAX_COUNT, 0.01, 10, Mat(), 3, 0, 0.04);
    }

    // Show Output
    imshow(rawWindow, rgbFrames);
    imshow(opticalFlowWindow, opticalFlow);

    std::swap(points2, points1);
    points1.clear();
    grayFrames.copyTo(prevGrayFrame);

    // See if we need to finish
    keyPressed = waitKey(10);
    if (keyPressed == 27) {
        break;
    }
    else if (keyPressed == 'r') {
        opticalFlow = Mat(cap.get(CV_CAP_PROP_FRAME_HEIGHT),
        cap.get(CV_CAP_PROP_FRAME_HEIGHT), CV_32FC3);
    }
}
}
```



```
Optical Flow Difference... X is 20       Y is 1          485
Optical Flow Difference... X is 21       Y is 1          486
Optical Flow Difference... X is 13       Y is 0          487
Optical Flow Difference... X is 0        Y is -18                488
Optical Flow Difference... X is 0        Y is -1                 489
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di
Optical Flow Di



Calculating  ca
```

Task 2: Optical Flow

Questions