Computer Vision, Lab 2: Homographies

In computer vision, the most important geometric transformations are 2D planar homographies, 3D Euclidean homographies, and 3D metric homographies.

Although you will be limited to a single scene plane, a 2D planar homography is the simplest way to get 3D information from a 2D image.

Today we'll boost our OpenCV programming skills and learn how to do the math we learned in class using code.

GUI programing in OpenCV

To display the results of your calculations, you'll need to be able to visualize images from a camera or video and draw on them. In OpenCV, these capabilities use the "HighGUI" and "ImgProc" libraries. For showing images, we use the following important functions:

- `imshow()` : Show an image in a window. If you use the same name as an existing window, the image will replace the old image in the same window.
- `waitKey()` : Wait for the user to press a key indefinitely or until a timer expires. Since image display runs in a separate thread that you don't (usually) control, your main thread needs to pause briefly to give time for display. `waitKey()` is a good way to do this, as it will put the calling thread to sleep for the given number of milliseconds. 1 ms is plenty of time for the display thread to do its work. If you have other actions that block the main thread such as waiting for the next image to be captured and transferred to RAM by a camera driver and you don't need user input, you don't need to use `waitKey()` .
- `destroyWindow()` : Destroy a target window.
- `destroyAllWindows()` : Destroy all windows under the program's control.

Let's start with a simple version of our solution.

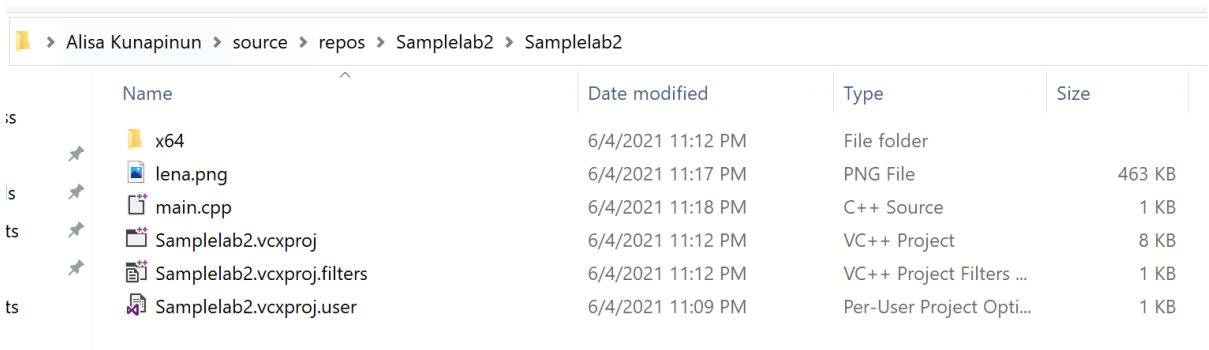
Tip on loading files before starting

Depending on how your program is started when testing it, it will always have a specific working directory.

If you want to open files by filename only without a full path, you'll need to put them in the program's working directory or change the working directory to point to where the files are.

Windows Visual Studio C++: Put resources such as images and videos in the same directory as your `.cpp` source code.

For example, suppose we create a project named `Samplelab2` under `C:\Users\alisa\source\repos` . The `.cpp` file containing the main function is in `C:\Users\alisa\source\repos\Samplelab2\Samplelab2` , so we should put the image file `lena.png` as below:



By the way, we shouldn't use the image `lena.png`, even though it is convenient and ships with the OpenCV source code. Read some of the context in Wikipedia (https://en.wikipedia.org/wiki/Lenna). The image comes from a pornographic magazine, *Playboy*, from the 1970s, and its continued use in the image processing community is given as an example of sexism in the sciences, reinforcing gender stereotypes.

So while we're being nostalgic, here's a better image from the 1970s (img/sample.jpg). Anyway, once you put it in the right place, your program your program can refer to the file without a full path:

```
Mat srcImage = imread("sample.jpg");
```

However, if you run the executable from another directory, you'll have to put the resource in the directory you're running from.

Python: Use the same idea as above.

Linux: Find out how your IDE sets the working directory when you run, or put the resource in the build directory, or use a relative path to run the executable from the directory where the resource is located.

Show an image in C++

Here's some code to show an image. Get `sample.jpg` from here (img/sample.jpg).

```
#include <iostream>
#include <opencv2/opencv.hpp> // This includes all of OpenCV. You could use just opencv2/hi
ghgui.hpp.

using namespace cv;           // Without this you would have to prefix every OpenCV call wi
th cv::
using namespace std;          // Without this you would have to prefix every C++ standard l
ibrary call with std::

int main(int argc, char* argv[])
{
    int iKey = -1;
    string sFilename = "sample.jpg";
    Mat matImage = imread(sFilename);
    if (matImage.empty())
    {
        cout << "No image to show" << endl;
        return 1;
    }
    imshow("Input image", matImage);
    // Wait up to 5s for a keypress
    iKey = waitKey(5000);
    cout << "Key output value: " << iKey << endl;
    return 0;
}
```

Show an image in Python

Things are a bit simpler in Python:

```
import cv2

if __name__ == '__main__':
    path = 'sample.jpg'
    img = cv2.imread(path)
    if img is None:
        print('No image to show')
    else
        cv2.imshow('Input image', img)
        # Wait up to 5s for a keypress
        cv2.waitKey(5000);
```

Show a video in C++

Here's how to show a video in C++:

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

// In C++, you can define constants variable using #define
#define VIDEO_FILE "robot.mp4"
#define ROTATE false

int main(int argc, char** argv)
{
    Mat matFrameCapture;
    Mat matFrameDisplay;
    int iKey = -1;

    // Open input video file
    VideoCapture videoCapture(VIDEO_FILE);
    if (!videoCapture.isOpened()) {
        cerr << "ERROR! Unable to open input video file " << VIDEO_FILE << endl;
        return -1;
    }

    // Capture loop
    while (iKey != int(' '))        // play video until user presses <space>
    {
        // Get the next frame
        videoCapture.read(matFrameCapture);
        if (matFrameCapture.empty())
        {
            // End of video file
            break;
        }

        // We can rotate the image easily if needed.
#if ROTATE
        rotate(matFrameCapture, matFrameDisplay, RotateFlags::ROTATE_180);   //rotate 180 d
egree and put the image to matFrameDisplay
#else
        matFrameDisplay = matFrameCapture;
#endif

        float ratio = 480.0 / matFrameDisplay.rows;
        resize(matFrameDisplay, matFrameDisplay, cv::Size(), ratio, ratio, INTER_LINEAR);
// resize image to 480p for showing

        // Display
        imshow(VIDEO_FILE, matFrameDisplay); // Show the image in window named "robot.mp4"
        iKey = waitKey(30); // Wait 30 ms to give a realistic playback speed
    }
    return 0;
}
```

Show a video in Python

Now let's do the same in Python:

```
import cv2
import numpy as np
import sys
```

In-Lab Exercises

1. Use `waitKey()` to wait for the user to press `<space>` to advance to the next frame or 'q' to quit. Check the documentation for waitKey() (https://docs.opencv.org/4.3.0/d7/dfc/group__highgui.html#ga5628525ad33f52eab17feebcfba38bd7), change the delay parameter to 0 for an infinite wait, and perform the necessary action on a 'spacebar' or 'q' key.
2. Display the full 1080p or 720p frame from the video without making the display window too big for your desktop. Take a look at the documentation for namedWindow() (https://docs.opencv.org/4.3.0/d7/dfc/group__highgui.html#ga5afdf8410934fd099df85c75b2e0888b) and figure out which flags you should use to set up your display window to be resizable but keep the aspect ratio and display the expanded GUI.
3. Next we probably want to give the user some useful information. Check out the documentation for displayOverlay() (https://docs.opencv.org/4.3.0/dc/d46/group__highgui__qt.html#ga704e0387318cd1e7928e6fe17e81d6aa) and add some explanatory information for the user about frame number, total frames, and user control actions.
4. Last little detail: currently, if the user closes the window, the program doesn't exit. Modify your program to exit when image display window is closed. Hint: try `cv::getWindowProperty(VIDEO_FILE, cv::WND_PROP_VISIBLE)`.

Getting four points for a homography

Now, we'd like to allow the user to select four points comprising a square in the real world then compute a rectifying homography for that square, thus rectifying the entire ground plane from the robot's point of view.

Check out the documentation for setMouseCallback() (https://docs.opencv.org/4.3.0/d7/dfc/group__highgui.html#ga89e7806b0a616f6f1d502bd8c183ad3e). Experiment with it until you can get four mouse clicks without interfering with the other GUI functions such as pan/tilt/zoom. Check that you are getting the image coordinates rather than the window coordinates of the mouse clicks.

The steps of getting 4 points for homography are:

1. Do the video captures loop
2. press any key to pause the screen and show a new pop-up for showing the pausing image
3. Use the mouse click 4 points.

Try the sample code below.

C++ variables

Create `Mat` variables to store images and a vector for storing points, then declare a `mouseHandler()` function. To keep things simple, use global variables:

```
Mat matPauseScreen, matResult, matFinal;
Point point;
vector<Point> pts;
int var = 0;
int drag = 0;

// Create mouse handler function
void mouseHandler(int, int, int, int, void*);
```

Python variables

```
matResult = None
matFinal = None
matPauseScreen = None

point = (-1, -1)
pts = []
var = 0
drag = 0
```

C++ mouse handler

```
// An OpenCV mouse handler function has 5 parameters

void mouseHandler(int event, int x, int y, int, void*)
{
    if (var >= 4) // If we already have 4 points, do nothing
        return;
    if (event == EVENT_LBUTTONDOWN) // Left button down
    {
        drag = 1; // Set it that the mouse is in pressing down mode
        matResult = matFinal.clone(); // copy final image to draw image
        point = Point(x, y); // memorize current mouse position to point var
        if (var >= 1) // if the point has been added more than 1 points, draw a line
        {
            line(matResult, pts[var - 1], point, Scalar(0, 255, 0, 255), 2); // draw a gree
n line with thickness 2
        }
        circle(matResult, point, 2, Scalar(0, 255, 0), -1, 8, 0); // draw a current green p
oint
        imshow("Source", matResult); // show the current drawing
    }
    if (event == EVENT_LBUTTONUP && drag) // When Press mouse left up
    {
        drag = 0; // no more mouse drag
```

Python mouse handler

```
def mouseHandler(event, x, y, flags, param):
    global point, pts, var, drag, matFinal, matResult   # call global variable to use in th
is function

    if (var >= 4):                              # if homography points are more than 4 points,
do nothing
        return
    if (event == cv2.EVENT_LBUTTONDOWN):     # When Press mouse left down
        drag = 1                                # Set it that the mouse is in pressing down mo
de
        matResult = matFinal.copy()          # copy final image to draw image
        point = (x, y)                          # memorize current mouse position to point var
        if (var >= 1):                          # if the point has been added more than 1 poin
ts, draw a line
            cv2.line(matResult, pts[var - 1], point, (0, 255, 0, 255), 2)    # draw a green
line with thickness 2
        cv2.circle(matResult, point, 2, (0, 255, 0), -1, 8, 0)             # draw a current
green point
        cv2.imshow("Source", matResult)      # show the current drawing
    if (event == cv2.EVENT_LBUTTONUP and drag):  # When Press mouse left up
        drag = 0                                # no more mouse drag
        pts.append(point)                       # add the current point to pts
        var += 1                                # increase point number
        matFinal = matResult.copy()          # copy the current drawing image to final imag
e
        if (var >= 4):                                                  # if the homogr
apy points are done
            cv2.line(matFinal, pts[0], pts[3], (0, 255, 0, 255), 2)   # draw the last line
            cv2.fillConvexPoly(matFinal, np.array(pts, 'int32'), (0, 120, 0, 20))        #
draw polygon from points
        cv2.imshow("Source", matFinal);
    if (drag):                                      # if the mouse is dragging
        matResult = matFinal.copy()              # copy final images to draw image
        point = (x, y)                     # memorize current mouse position to point var
        if (var >= 1):                          # if the point has been added more than 1
points, draw a line
            cv2.line(matResult, pts[var - 1], point, (0, 255, 0, 255), 2)    # draw a green
line with thickness 2
        cv2.circle(matResult, point, 2, (0, 255, 0), -1, 8, 0)         # draw a current gre
en point
        cv2.imshow("Source", matResult)          # show the current drawing
```

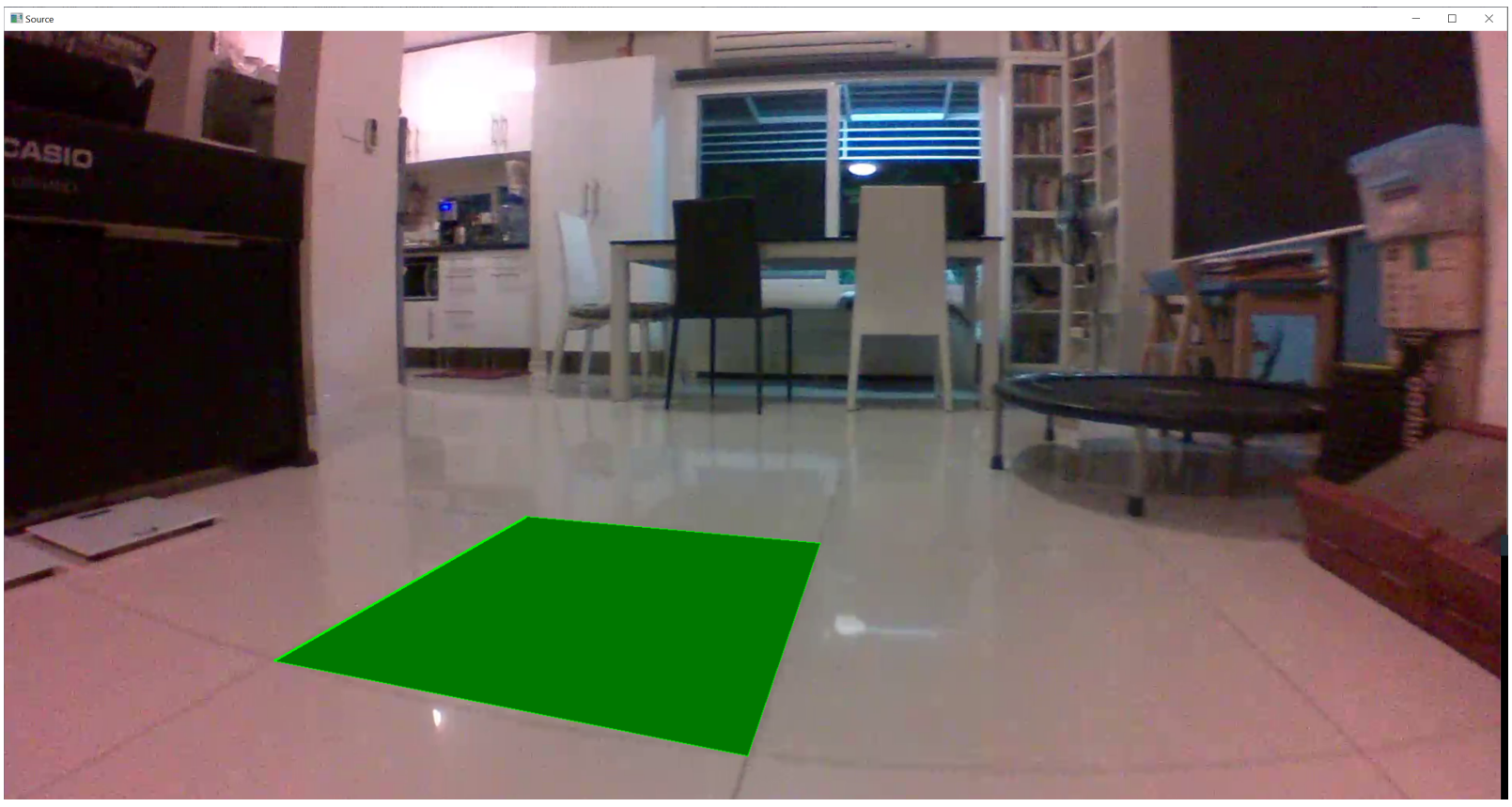
C++ main()

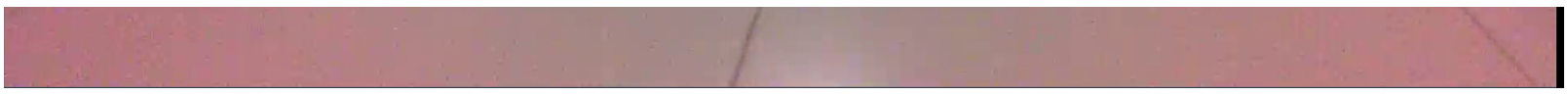
```
int main(int argc, char** argv)
{
    Mat matFrameCapture;
    Mat matFrameDisplay;
    int key = -1;

    // --------------------- [STEP 1: Make video capture from file] ---------------------
    // Open input video file
    VideoCapture videoCapture(VIDEO_FILE);
    if (!videoCapture.isOpened()) {
        cerr << "ERROR! Unable to open input video file " << VIDEO_FILE << endl;
        return -1;
    }

    // Capture loop
    while (key < 0)        // play video until press any key
    {
        // Get the next frame
        videoCapture.read(matFrameCapture);
        if (matFrameCapture.empty()) {   // no more frame capture from the video
            // End of video file
            break;
        }
        cvtColor(matFrameCapture, matFrameCapture, COLOR_BGR2BGRA);
        // Rotate if needed, some video has output like top go down, so we need to rotate i
t
#if ROTATE
        rotate(matFrameCapture, matFrameCapture, RotateFlags::ROTATE_180);   //rotate 180 d
egree and put the image to matFrameDisplay
#endif

        float ratio = 640.0 / matFrameCapture.cols;
        resize(matFrameCapture, matFrameDisplay, cv::Size(), ratio, ratio, INTER_LINEAR);

        // Display
        imshow(VIDEO_FILE, matFrameDisplay); // Show the image in window named "robot.mp4"
        key = waitKey(30);

        // --------------------- [STEP 2: pause the screen and show an image]
---------------------
        if (key >= 0)
        {
            matPauseScreen = matFrameCapture;  // transfer the current image to process
            matFinal = matPauseScreen.clone(); // clone image to final image
        }
    }

    // --------------------- [STEP 3: use mouse handler to select 4 points]
---------------------
    if (!matFrameCapture.empty())
    {
        var = 0;   // reset number of saving points
        pts.clear(); // reset all points
        namedWindow("Source", WINDOW_AUTOSIZE);  // create a windown named source
        setMouseCallback("Source", mouseHandler, NULL); // set mouse event handler "mouseHa
ndler" at Window "Source"
        imshow("Source", matPauseScreen); // Show the image
        waitKey(0); // wait until press anykey
        destroyWindow("Source"); // destroy the window
    }
    else
    {
        cout << "You did not pause the screen before the video finish, the program will sto
p" << endl;
        return 0;
```

Python `__main__`:

```
if __name__ == '__main__':
    global matFinal, matResult, matPauseScreen         # call global variable to use in thi
s function
    key = -1;

    # --------------------- [STEP 1: Make video capture from file] ---------------------
    # Open input video file
    videoCapture = cv2.VideoCapture(VIDEO_FILE);
    if not videoCapture.isOpened():
        print("ERROR! Unable to open input video file ", VIDEO_FILE)
        sys.exit('Unable to open input video file')

    width  = videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)   # float `width`
    height = videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)  # float `height`

    # Capture loop
    while (key < 0):        # play video until press any key
        # Get the next frame
        _, matFrameCapture = videoCapture.read()
        if matFrameCapture is None:   # no more frame capture from the video
            # End of video file
            break

        # Rotate if needed, some video has output like top go down, so we need to rotate it
        if ROTATE:
            _, matFrameDisplay = cv2.rotate(matFrameCapture, cv2.ROTATE_180)   #rotate 180
degree and put the image to matFrameDisplay
        else:
            matFrameDisplay = matFrameCapture;

        ratio = 640.0 / width
        dim = (int(width * ratio), int(height * ratio))
        # resize image to 480 * 640 for showing
        matFrameDisplay = cv2.resize(matFrameDisplay, dim)

        # Show the image in window named "robot.mp4"
        cv2.imshow(VIDEO_FILE, matFrameDisplay)
        key = cv2.waitKey(30)

        # --------------------- [STEP 2: pause the screen and show an image]
---------------------
        if (key >= 0):
            matPauseScreen = matFrameCapture     # transfer the current image to process
            matFinal = matPauseScreen.copy()     # copy image to final image

    # --------------------- [STEP 3: use mouse handler to select 4 points]
---------------------
    if (matFrameCapture is not None):
        var = 0                                              # reset number of saving points
        pts.clear()                                          # reset all points
        cv2.namedWindow("Source", cv2.WINDOW_AUTOSIZE)       # create a windown named source
        cv2.setMouseCallback("Source", mouseHandler)         # set mouse event handler "mous
```

Sample result





In-Lab Exercise

Global variables are not good practice in C++ (or any programming language, as far as I know).

Figure out how to allocate the state information used by your mouse handler on the heap, and request HighGUI to pass a pointer to the state information to your callback. Instead of

```
setMouseCallback("Source", mouseHandler, NULL);
```

use

```
setMouseCallback("Source", mouseHandler, pState);
```

Calculate Homography

Now, given the four points that you've collected from the user, calculate a homography to a rectified square with a desired number of pixels per meter, e.g., 1000. The tiles in the video from last week are 60cm x 60cm.

Note that OpenCV doesn't have a `null()` function like Matlab and Octave. Instead, you'll have to use the SVD operation to get the row of V associated with the smallest singular value of the design matrix. Test that you get the same result from OpenCV's SVD operation and Octave's `null()` function.

Once you've got a homography that works for the selected quadrilateral, you'll want to adjust it by incorporating a translation that maps the bounding box of the transformed image to a valid range starting at 0 for the uppermost Y coordinate and leftmost X coordinate.

Display rectified image

Once you've got that working, you'll want to display a rectified version of the original image in a second HighGUI window as we step through the video. There are two ways to do this: directly (manually) and using `cv::warpPerspective()`. For this lab's learning outcomes, it would be better for you to do it directly/manually using bilinear interpolation. This will give you a better understanding of how to render image transforms. In your own work later, go ahead and use `warpPerspective()` or whatever suits you.

Display original and rectified optical flows

Once you have the display of the original and ground-plane-rectified images working, add the optical flows from Lab 01 and render them in both images. This will be really useful.

Here are examples of using `getPerspectiveTransfrom()` to get the homograpy and `warpPerspective()` to get the warped image.

C++

Put this code in your main function.

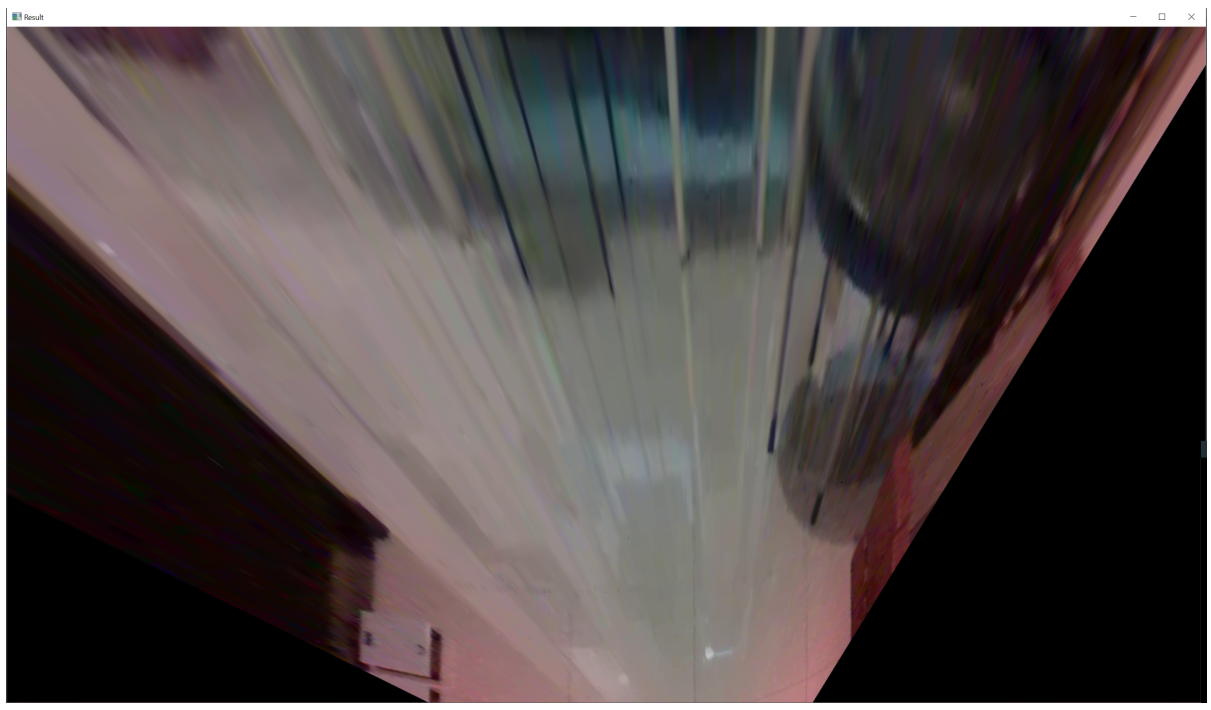
```
if (pts.size() == 4)
{
    Point2f src[4];
    for (int i = 0; i < 4; i++)
    {
        src[i].x = pts[i].x * 1.0;
        src[i].y = pts[i].y * 1.0;
    }
    Point2f reals[4];
    reals[0] = Point2f(800.0, 800.0);
    reals[1] = Point2f(1000.0, 800.0);
    reals[2] = Point2f(1000.0, 1000.0);
    reals[3] = Point2f(800.0, 1000.0);

    Mat homography_matrix = getPerspectiveTransform(src, reals);
    std::cout << "Estimated Homography Matrix is:" << std::endl;
    std::cout << homography_matrix << std::endl;

    // perspective transform operation using transform matrix
    cv::warpPerspective(matPauseScreen, matResult, homography_matrix, matPauseScreen.size
(), cv::INTER_LINEAR);
    imshow("Source", matPauseScreen);
    imshow("Result", matResult);

    waitKey(0);
}
```

Python

Here is equivalent Python code.

```
if (len(pts) == 4):
    src = np.array(pts).astype(np.float32)

    reals = np.array([(800, 800),
                      (1000, 800),
                      (1000, 1000),
                      (800, 1000)], np.float32)

    homography_matrix = cv2.getPerspectiveTransform(src, reals);
    print("Estimated Homography Matrix is:")
    print(homography_matrix)

    # perspective transform operation using transform matrix

    h, w, ch = matPauseScreen.shape
    matResult = cv2.warpPerspective(matPauseScreen, homography_matrix, (w, h), cv2.INTER_LI
NEAR)
    matPauseScreen = cv2.resize(matPauseScreen, dim)
    cv2.imshow("Source", matPauseScreen)
    matResult = cv2.resize(matResult, dim)
    cv2.imshow("Result", matResult)

    cv2.waitKey(0)
```

You should get a result similar to this:



Exercises

1. Calculate the homography manually using the SVD of the linear system design matrix similar to the null space solution from class.
2. Compute the warped image manually using the inverse of the homography and bilinear interpolation in the input image.
3. Reuse the homography from your last run: If your program finds a file `homography.yml` in the working directory, it should read the homography from that file and use it to display the transformed image. For this, you will have to learn how OpenCV stores data files in YML format using the `FileStorage` class. When the user selects four points in a frame, output the resulting homography to the data file and re-read that file when the program starts again. This way, the user only has to do the "calibration" once.

What to turn in

Write a brief report and turn in using Google Classroom before the next lab. Include your experience with both the in-class exercises and the final exercises.

Make a video showing the frames of the original video with optical flows side-by-side with the rectified image and rectified optical flows, put the video online, and point to it on the Piazza discussion board.

In Lab exercises

In [ ]:
```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <string>

using namespace cv;
using namespace std;

// In C++, you can define constants variable using #define
#define VIDEO_FILE "../robot.qt"
```

```
#define ROTATE false

int main(int argc, char** argv)
{
    Mat matFrameCapture;
    Mat matFrameDisplay;
    int iKey = -1;

    // Open input video file
    VideoCapture videoCapture(VIDEO_FILE);
    if (!videoCapture.isOpened()) {
        cerr << "ERROR! Unable to open input video file " << VIDEO_FILE << endl;
        return -1;
    }

    namedWindow(VIDEO_FILE, WINDOW_NORMAL | WINDOW_KEEPRATIO | WINDOW_GUI_EXPANDED);
    int n_frame = 0;

    // Capture loop
    while (true)        // play video until user presses <space>
    {
        // Get the next frame
        videoCapture.read(matFrameCapture);
        if (matFrameCapture.empty())
        {
            // End of video file
            break;
        }

        // We can rotate the image easily if needed.
#if ROTATE
        rotate(matFrameCapture, matFrameDisplay, RotateFlags::ROTATE_180);   //rotate 180 degree an
#else
        matFrameDisplay = matFrameCapture;
#endif

        float ratio = 480.0 / matFrameDisplay.rows;
        resize(matFrameDisplay, matFrameDisplay, cv::Size(), ratio, ratio, INTER_LINEAR); // resize

        // Display
        imshow(VIDEO_FILE, matFrameDisplay); // Show the image in window named "robot.mp4"
        n_frame++;
        displayOverlay(VIDEO_FILE, "Frame #"+to_string(n_frame));

        iKey = -1;
        while (iKey!=int(' ') && iKey!=int('q') ) {
            iKey = waitKey(30); // Wait 30 ms to give a realistic playback speed
            if (getWindowProperty(VIDEO_FILE, WND_PROP_VISIBLE)==0) {
                    iKey = int('q');
            }
        }
        if (iKey==int('q')) {
            break;
        }

    }
    return 0;
}
```

In [ ]:

```
import cv2
import numpy as np
import sys

VIDEO_FILE = 'robot.qt'
ROTATE = False

if __name__ == '__main__':

    key = -1;

    # Open input video file
    videoCapture = cv2.VideoCapture(VIDEO_FILE);
    if not videoCapture.isOpened():
        print('Error: Unable to open input video file', VIDEO_FILE)
        sys.exit('Unable to open input video file')

    width  = videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)   # float `width`
    height = videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)  # float `height`
    cv2.namedWindow(VIDEO_FILE, cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO | cv2.WINDOW_GUI_EXPANDED)
    n_frame = 0

    # Capture loop
    while (1):        # play video until user presses <space>
        # Get the next frame
        _, matFrameCapture = videoCapture.read()
        if matFrameCapture is None:
            # End of video
            break
```

```
# Rotate if needed
if ROTATE:
    _, matFrameDisplay = cv2.rotate(matFrameCapture, cv2.ROTATE_180)
else:
    matFrameDisplay = matFrameCapture;

ratio = 480.0 / height
dim = (int(width * ratio), int(height * ratio))
# resize image to 480p for display
# matFrameDisplay = cv2.resize(matFrameDisplay, dim)

# Show the image in window named "robot.mp4"
cv2.imshow(VIDEO_FILE, matFrameDisplay)
n_frame+=1
cv2.displayOverlay(VIDEO_FILE, "Frame: "+str(n_frame))

key = -1
while ( key != ord(' ') and key != ord('q') ):
    key = cv2.waitKey(30)
    if cv2.getWindowProperty(VIDEO_FILE, cv2.WND_PROP_VISIBLE)==0:
          key = ord('q')
if key == ord('q'):
    break
```

Calculating and saving Homography to file

In [ ]:

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <cmath>

using namespace cv;
using namespace std;

//#define VIDEO_FILE "../myvid1.mp4"

struct sState {
    Mat matPauseScreen, matResult, matFinal;
    Point point;
    vector<Point> pts;
    int var;
    int drag;
};

// Create mouse handler function

void mouseHandler(int, int, int, int, void*);

Mat calcHomography(struct sState *pState, cv::Mat *homography_matrix);

int main(int argc, char* argv[])
{
    Mat matFrameCapture;
    Mat matFrameDisplay;
    int key = -1;
    struct sState state;
    state.var = 0;
    state.drag = 0;

    string VIDEO_FILE = argv[1];

    // ---------------------- [STEP 1: Make video capture from file] ----------------------
    // Open input video file
    VideoCapture videoCapture(VIDEO_FILE);
    if (!videoCapture.isOpened()) {
        cerr << "ERROR! Unable to open input video file " << VIDEO_FILE << endl;
        return -1;
    }

    // string filename = "../homography.xml";
    // string filename = argv[2];
    // FileStorage fs(filename, FileStorage::WRITE);

    // fs.open(filename, FileStorage::WRITE);

    // Capture loop
    while (key < 0)        // play video until press any key
    {
        // Get the next frame
        videoCapture.read(matFrameCapture);
        if (matFrameCapture.empty()) {   // no more frame capture from the video
            // End of video file
            break;
        }
        cvtColor(matFrameCapture, matFrameCapture, COLOR_BGR2BGRA);

        // Rotate if needed, some video has output like top go down, so we need to rotate it
```

```
#if ROTATE
        rotate(matFrameCapture, matFrameCapture, RotateFlags::ROTATE_180);   //rotate 180 degree an
#endif

        float ratio = 640.0 / matFrameCapture.cols;
        resize(matFrameCapture, matFrameDisplay, cv::Size(), ratio, ratio, INTER_LINEAR);

        // Display
        imshow(VIDEO_FILE, matFrameDisplay); // Show the image in window named "robot.mp4"
        key = waitKey(30);

        // ---------------------- [STEP 2: pause the screen and show an image] ---------------------
        if (key >= 0)
        {
            state.matPauseScreen = matFrameCapture;  // transfer the current image to process
            state.matFinal = state.matPauseScreen.clone(); // clone image to final image
            imwrite("image.jpg", matFrameCapture);
        }
    }

    // ---------------------- [STEP 3: use mouse handler to select 4 points] ---------------------
    if (!matFrameCapture.empty())
    {
        state.var = 0;   // reset number of saving points
        state.pts.clear(); // reset all points
        namedWindow("Source", WINDOW_AUTOSIZE);  // create a windown named source
        setMouseCallback("Source", mouseHandler, &state); // set mouse event handler "mouseHandler"
        imshow("Source", state.matPauseScreen); // Show the image
        waitKey(0); // wait until press anykey

        if (state.pts.size() == 4)
        {
            Mat homography_matrix = Mat::zeros(1, 9, CV_32F);
            homography_matrix = calcHomography(&state, &homography_matrix);
            std::cout << "Estimated Homography Matrix is:" << std::endl;
            std::cout << homography_matrix << std::endl;
            // fs << "homography_matrix" << homography_matrix; // save Homography to file

            imshow("Source", state.matPauseScreen);
            imshow("Result", state.matResult);

            waitKey(0);
        }
    }
    else
    {
        cout << "You did not pause the screen before the video finish, the program will stop" << en
        return 0;
    }

    return 0;
}

// An OpenCV mouse handler function has 5 parameters: the event that occurred,
// the position of the mouse, and a user-specific pointer to an arbitrary data
// structure.

void mouseHandler(int event, int x, int y, int, void *pVoid)
{
    struct sState *pState = (struct sState *)pVoid;

    if (pState->var >= 4) // If we already have 4 points, do nothing
        return;
    if (event == EVENT_LBUTTONDOWN) // Left button down
    {
        pState->drag = 1; // Set it that the mouse is in pressing down mode
        pState->matResult = pState->matFinal.clone(); // copy final image to draw image
        pState->point = Point(x, y); // memorize current mouse position to point var
        if (pState->var >= 1) // if the point has been added more than 1 points, draw a line
        {
            line(pState->matResult, pState->pts[pState->var - 1], pState->point, Scalar(0, 255, 0,
        }
        circle(pState->matResult, pState->point, 2, Scalar(0, 255, 0), -1, 8, 0); // draw a current
        imshow("Source", pState->matResult); // show the current drawing
    }
    if (event == EVENT_LBUTTONUP && pState->drag) // When Press mouse left up
    {
        pState->drag = 0; // no more mouse drag
        pState->pts.push_back(pState->point);  // add the current point to pts
        pState->var++; // increase point number
        pState->matFinal = pState->matResult.clone(); // copy the current drawing image to final im
        if (pState->var >= 4) // if the homograpy points are done
        {
            line(pState->matFinal, pState->pts[0], pState->pts[3], Scalar(0, 255, 0, 255), 2); // d
            fillPoly(pState->matFinal, pState->pts, Scalar(0, 120, 0, 20), 8, 0); // draw polygon f

            setMouseCallback("Source", NULL, NULL); // remove mouse event handler
        }
        imshow("Source", pState->matFinal);
    }
```

```
    if (pState->drag) // if the mouse is dragging
    {
        pState->matResult = pState->matFinal.clone(); // copy final images to draw image
        pState->point = Point(x, y); // memorize current mouse position to point var
        if (pState->var >= 1) // if the point has been added more than 1 points, draw a line
        {
            line(pState->matResult, pState->pts[pState->var - 1], pState->point, Scalar(0, 255, 0,
        }
        circle(pState->matResult, pState->point, 2, Scalar(0, 255, 0), -1, 8, 0); // draw a current
        imshow("Source", pState->matResult); // show the current drawing
    }
}

cv::Mat calcHomography(struct sState *pState, cv::Mat *homography_matrix)
{
    cout << "Calculating homography..." << endl;
    for (int i = 0; i < pState->pts.size(); i++)
    {
        cout << "Point " << i << ": " << pState->pts[i] << endl;
    }
    if (pState->pts.size() != 4) {
        cout << "Four points are needed for a homography..." << endl;
        Mat error_mat;
        return error_mat;
    }
    Mat matA = Mat::zeros(8, 9, CV_32F);
    int xprimes[] = { 300, 400, 400, 300 };
    int yprimes[] = { 300, 300, 400, 400 };
    for (int i = 0; i < pState->pts.size(); i++) {
        float x = pState->pts[i].x;
        float y = pState->pts[i].y;
        float xprime = xprimes[i];
        float yprime = yprimes[i];
        matA.at<float>(i*2, 0) = 0;
        matA.at<float>(i*2, 1) = 0;
        matA.at<float>(i*2, 2) = 0;
        matA.at<float>(i*2, 3) = -x;
        matA.at<float>(i*2, 4) = -y;
        matA.at<float>(i*2, 5) = -1;
        matA.at<float>(i*2, 6) = yprime * x;
        matA.at<float>(i*2, 7) = yprime * y;
        matA.at<float>(i*2, 8) = yprime;
        matA.at<float>(i*2+1, 0) = x;
        matA.at<float>(i*2+1, 1) = y;
        matA.at<float>(i*2+1, 2) = 1;
        matA.at<float>(i*2+1, 3) = 0;
        matA.at<float>(i*2+1, 4) = 0;
        matA.at<float>(i*2+1, 5) = 0;
        matA.at<float>(i*2+1, 6) = -xprime * x;
        matA.at<float>(i*2+1, 7) = -xprime * y;
        matA.at<float>(i*2+1, 8) = -xprime;
    }
    cout << "Matrix A:" << endl;
    cout << matA << endl;
    SVD svdA( matA, SVD::FULL_UV );
    cout << "U:" << endl << svdA.u << endl;
    cout << "W:" << endl << svdA.w << endl;
    cout << "Vt:" << endl << svdA.vt << endl;

    Mat matH = Mat::zeros(1, 9, CV_32F);
    for (int i=0; i<9; i++) {
        matH.at<float>(0, i) = svdA.vt.at<float>(8, i);
    }
    matH = matH / matH.at<float>(0,8);
    // cout << "H:" << endl << matH << endl;
    matH = matH.reshape(3,3);
    string filename = "homography.xml";
    FileStorage fs(filename, FileStorage::WRITE);
    fs.open(filename, FileStorage::WRITE);
    fs << "homography_matrix" << matH; // save Homography to file
    fs.release()
    return matH;
}
```

```
...Passport/MS DSAI/InterSem  ...terSem/CVLabs/Lab02/build  dell@dell: ~/Documents
QSettings::value: Empty key passed
Calculating homography...
Point 0: [203, 291]
Point 1: [380, 289]
Point 2: [344, 407]
Point 3: [125, 410]
Matrix A:
[0, 0, 0, -203, -291, -1, 60900, 87300, 300;
 203, 291, 1, 0, 0, 0, -60900, -87300, -300;
 0, 0, 0, -380, -289, -1, 114000, 86700, 300;
 380, 289, 1, 0, 0, 0, -152000, -115600, -400;
 0, 0, 0, -344, -407, -1, 137600, 162800, 400;
 344, 407, 1, 0, 0, 0, -137600, -162800, -400;
 0, 0, 0, -125, -410, -1, 50000, 164000, 400;
 125, 410, 1, 0, 0, 0, -37500, -123000, -300]
U:
[0.23375797, 0.063044012, -0.28080451, -0.097716555, 0.50584477, 0.19023266, -0.72838002, -0.17429353;
 -0.233758, -0.063039683, -0.28801304, 0.21114901, -0.44896159, 0.10934015, -0.43451095, 0.64150316;
 0.30505961, -0.36163202, -0.40866163, -0.50391084, 0.10130378, -0.46470669, 0.16430618, 0.31940427;
 -0.40674579, 0.48218194, -0.32128847, -0.35658291, -0.30884916, -0.34442252, -0.13192664, -0.37451482;
 0.46865681, -0.072934829, -0.43865389, 0.019339334, -0.47941908, 0.4581632, 0.1770916, -0.33336571;
 -0.46865684, 0.072941318, -0.41620293, -0.12937775, 0.40348804, 0.48423821, 0.4012242, 0.16372833;
 0.35139227, 0.62897384, -0.26471925, 0.46414351, 0.17937435, -0.25851652, 0.20913924, 0.2294873;
 -0.26354456, -0.47172672, -0.36365959, 0.57640976, 0.10992776, -0.32564721, 0.013425621, -0.35385185]
W:
[454573.97;
 99051.469;
```

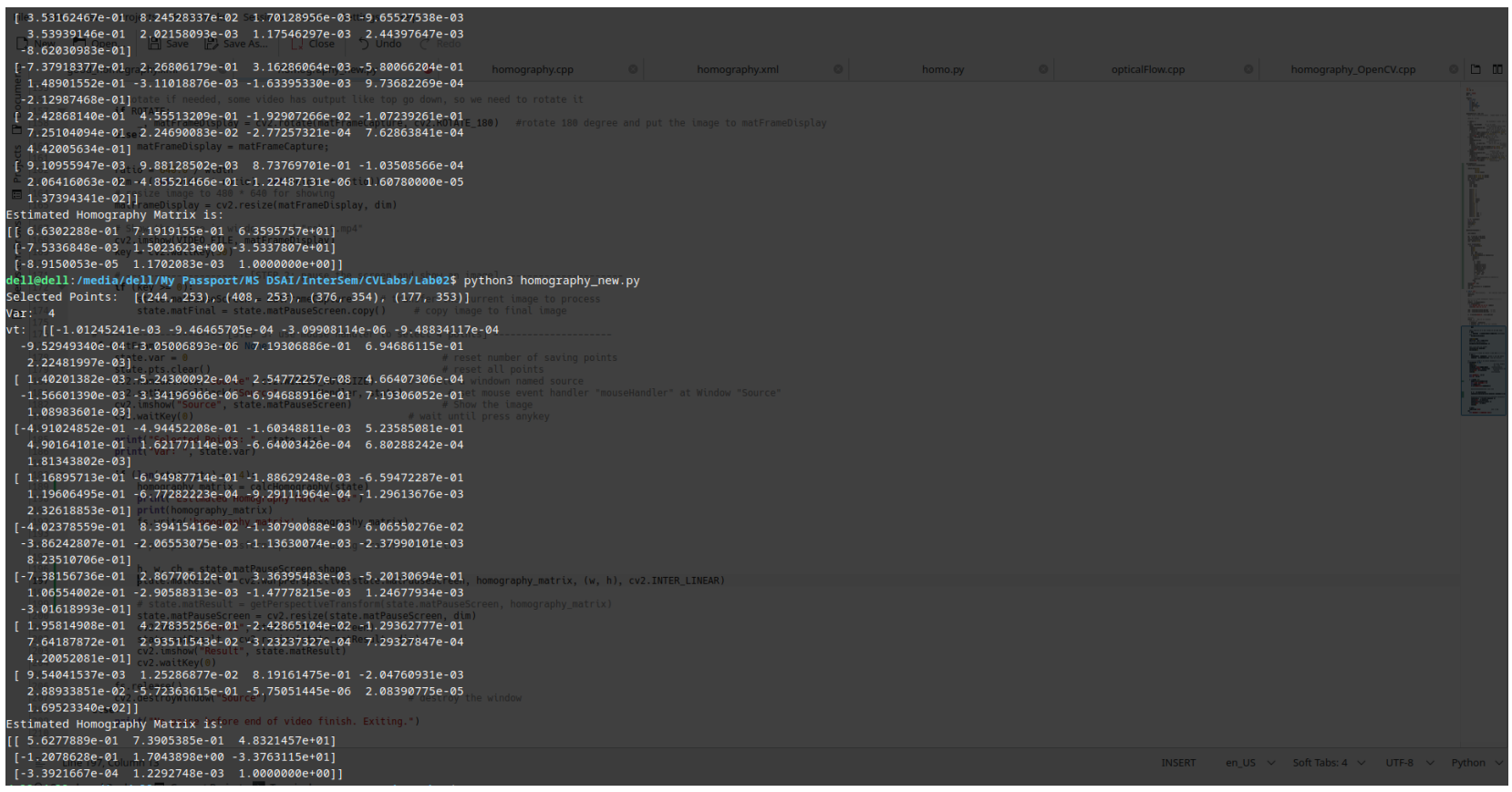
```
 880.67438;
 217.59308;
 148.76633;
 22.754387;
 9.042696;
 0.087251902]
Vt:
[-0.00087153527, -0.0010655464, -3.0197614e-06, -0.0008106876, -0.0010801322, -2.9893188e-06, 0.61926287, 0.78517836, 0.0021757004;
 0.0013786571, -0.00043123856, 2.0551768e-07, 0.00071770849, -0.0014338905, -2.5991642e-06, -0.78518075, 0.61926246, 0.00071901869;
 -0.41921002, -0.56225002, -0.0015773865, 0.44997525, 0.55285412, 0.00158156, -0.00063589384, 0.00049105461, 0.0026221042;
 -0.29914993, 0.65288442, 0.0013860654, 0.67397332, -0.11077646, 0.00054296054, 0.00059736695, 0.00099534087, -0.1331773;
 -0.37616751, -0.071354978, -0.0016428106, 0.0088489093, -0.36901888, -0.0020643375, -0.0011179739, -0.0024777218, 0.84684056;
 0.75534976, -0.18240702, -0.0033616116, 0.55715585, -0.067595579, 0.003288486, 0.0015683096, -0.00095315126, 0.28488097;
 0.15055749, 0.46814051, -0.016785696, -0.18104668, 0.73547655, 0.01966702, -0.00047001097, 0.00080997631, 0.42872339;
 0.0062560309, 0.0085913222, 0.88095164, -0.0015293041, 0.018533146, -0.4725652, -4.8235452e-06, 1.4239205e-05, 0.012152119;
 0.0027690276, 0.0047212495, -0.4728891, 6.0770664e-05, 0.0080131823, -0.88106608, 1.0299166e-08, 9.6942877e-06, 0.0020538927]
Estimated Homography Matrix is:
[1.3481851, 2.2986836, -230.2404;
 0.029588042, 3.9014609, -428.97375;
 5.0144613e-06, 0.0047199582, 1]
dell@dell:/media/dell/My Passport/MS DSAI/InterSem/CVLabs/Lab02/build$
```

In [ ]:

```python
import cv2
import numpy as np

VIDEO_FILE = "robot.mp4"
ROTATE = False

class sState:
    def __init__(self):
        self.pts = []
        self.matResult = None
        self.matFinal = None
        self.matPauseScreen = None
        self.point = (-1, -1)
        self.var = 0
        self.drag = 0

def mouseHandler(event, x, y, flags, state):
    # global point, pts, var, drag, matFinal, matResult   # call global variable to use in this fun

    # print("Var: ", state.var)

    if (state.var >= 4):                            # if homography points are more than 4 points, d
        return
    if (event == cv2.EVENT_LBUTTONDOWN):     # When Press mouse left down
        state.drag = 1                              # Set it that the mouse is in pressing down mode
        state.matResult = state.matFinal.copy()          # copy final image to draw image
        state.point = (x, y)                        # memorize current mouse position to point var
        if (state.var >= 1):                        # if the point has been added more than 1 points
            cv2.line(state.matResult, state.pts[state.var - 1], state.point, (0, 255, 0, 255), 2)
        cv2.circle(state.matResult, state.point, 2, (0, 255, 0), -1, 8, 0)             # draw a cur
        cv2.imshow("Source", state.matResult)      # show the current drawing
    if (event == cv2.EVENT_LBUTTONUP and state.drag):  # When Press mouse left up
        state.drag = 0                              # no more mouse drag
        state.pts.append(state.point)                    # add the current point to pts
        state.var += 1                              # increase point number
        state.matFinal = state.matResult.copy()          # copy the current drawing image to final
        if (state.var >= 4):                                                  # if the homograp
            cv2.line(state.matFinal, state.pts[0], state.pts[3], (0, 255, 0, 255), 2)   # draw the
            cv2.fillConvexPoly(state.matFinal, np.array(state.pts, 'int32'), (0, 120, 0, 20))
        cv2.imshow("Source", state.matFinal);
    if (state.drag):                                    # if the mouse is dragging
        state.matResult = state.matFinal.copy()              # copy final images to draw image
        state.point = (x, y)                    # memorize current mouse position to point var
        if (state.var >= 1):                            # if the point has been added more than 1 p
            cv2.line(state.matResult, state.pts[state.var - 1], state.point, (0, 255, 0, 255), 2)
        cv2.circle(state.matResult, state.point, 2, (0, 255, 0), -1, 8, 0)         # draw a current
        cv2.imshow("Source", state.matResult)           # show the current drawing

def calcHomography(state):
    src = np.array(state.pts).astype(np.float32)

    reals = np.array([(300, 300),
                      (400, 300),
                      (400, 400),
                      (300, 400)], np.float32)

    xprimes = np.array([300, 400, 400, 300], np.float32);
    yprimes = np.array([300, 300, 400, 400], np.float32);
    matA = np.zeros((8,9), dtype=np.float32)
    for i in range(src.shape[0]):
        x = src[i][0];
        y = src[i][1];
        xprime = xprimes[i];
        yprime = yprimes[i];

        matA[i*2, 0] = 0;
        matA[i*2, 1] = 0;
        matA[i*2, 2] = 0;
        matA[i*2, 3] = -x;
        matA[i*2, 4] = -y;
        matA[i*2, 5] = -1;
        matA[i*2, 6] = yprime * x;
        matA[i*2, 7] = yprime * y;
        matA[i*2, 8] = yprime;
        matA[i*2+1, 0] = x;
        matA[i*2+1, 1] = y;
        matA[i*2+1, 2] = 1;
```

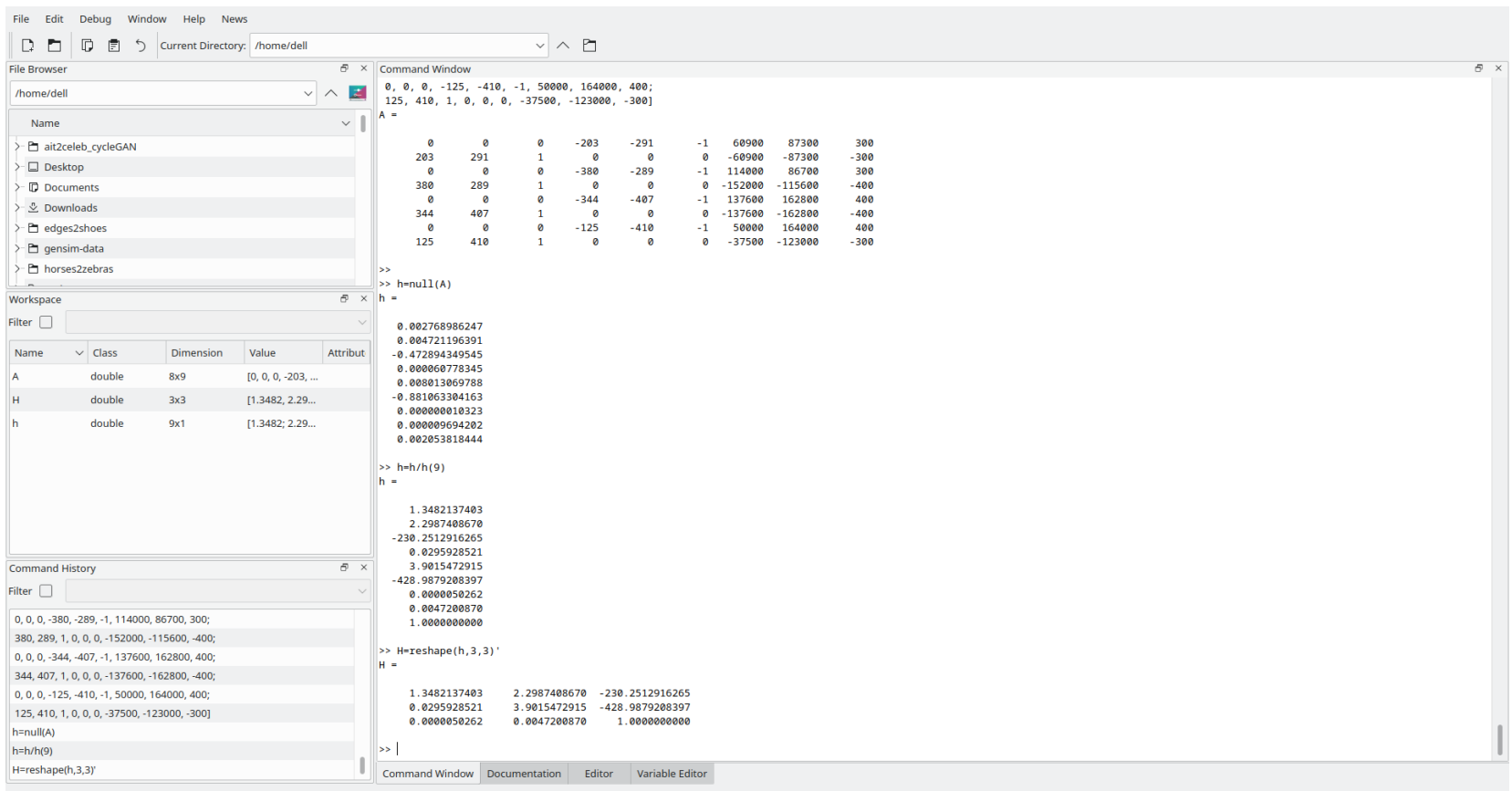
```
        matA[i*2+1, 3] = 0;
        matA[i*2+1, 4] = 0;
        matA[i*2+1, 5] = 0;
        matA[i*2+1, 6] = -xprime * x;
        matA[i*2+1, 7] = -xprime * y;
        matA[i*2+1, 8] = -xprime;

    w, u, vt = cv2.SVDecomp(matA)
    print("vt: ", vt)
    h = vt[7,:]/vt[7,8]
    h = h.reshape(3,3)
    #print("H:", h)
    return h

if __name__ == '__main__':
    # global matFinal, matResult, matPauseScreen         # call global variable to use in this func
    state = sState()
    key = -1;

    # -------------------- [STEP 1: Make video capture from file] --------------------
    # Open input video file
    videoCapture = cv2.VideoCapture(VIDEO_FILE);
    if not videoCapture.isOpened():
        print("ERROR! Unable to open input video file ", VIDEO_FILE)
        sys.exit('Unable to open input video file')

    width  = videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)   # float `width`
    height = videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)  # float `height`

    fs = cv2.FileStorage('homography.yml', cv2.FILE_STORAGE_WRITE)

    # Capture loop
    while (key < 0):        # play video until press any key
        # Get the next frame
        _, matFrameCapture = videoCapture.read()
        if matFrameCapture is None:   # no more frame capture from the video
            # End of video file
            break

        # Rotate if needed, some video has output like top go down, so we need to rotate it
        if ROTATE:
            _, matFrameDisplay = cv2.rotate(matFrameCapture, cv2.ROTATE_180)   #rotate 180 degree a
        else:
            matFrameDisplay = matFrameCapture;

        ratio = 640.0 / width
        dim = (int(width * ratio), int(height * ratio))
        # resize image to 480 * 640 for showing
        matFrameDisplay = cv2.resize(matFrameDisplay, dim)

        # Show the image in window named "robot.mp4"
        cv2.imshow(VIDEO_FILE, matFrameDisplay)
        key = cv2.waitKey(30)

        # -------------------- [STEP 2: pause the screen and show an image] --------------------
        if (key >= 0):
            state.matPauseScreen = matFrameCapture     # transfer the current image to process
            state.matFinal = state.matPauseScreen.copy()     # copy image to final image

    # -------------------- [STEP 3: use mouse handler to select 4 points] --------------------
    if (matFrameCapture is not None):
        state.var = 0                                       # reset number of saving points
        state.pts.clear()                                   # reset all points
        cv2.namedWindow("Source", cv2.WINDOW_AUTOSIZE)      # create a windown named source
        cv2.setMouseCallback("Source", mouseHandler, state)         # set mouse event handler "mouse
        cv2.imshow("Source", state.matPauseScreen)          # Show the image
        cv2.waitKey(0)                                      # wait until press anykey

        print("Selected Points: ", state.pts)
        print("Var: ", state.var)

        if (len(state.pts) == 4):
            homography_matrix = calcHomography(state)
            print("Estimated Homography Matrix is:")
            print(homography_matrix)
            fs.write('homography_matrix', homography_matrix)

            # perspective transform operation using transform matrix

            #h, w, ch = state.matPauseScreen.shape
            # state.matResult = cv2.warpPerspective(state.matPauseScreen, homography_matrix, (w, h)

            state.matResult = getPerspectiveTransform(state.matPauseScreen, homography_matrix)
            state.matPauseScreen = cv2.resize(state.matPauseScreen, dim)
            cv2.imshow("Source", state.matPauseScreen)
            state.matResult = cv2.resize(state.matResult, dim)
            cv2.imshow("Result", state.matResult)
            cv2.waitKey(0)

        fs.release()
```

```
        cv2.destroyWindow("Source")                         # destroy the window
    else:
        print("No pause before end of video finish. Exiting.")
```



Octave Code

```
In [ ]: A=[0, 0, 0, -203, -291, -1, 60900, 87300, 300;
         203, 291, 1, 0, 0, 0, -60900, -87300, -300;
         0, 0, 0, -380, -289, -1, 114000, 86700, 300;
         380, 289, 1, 0, 0, 0, -152000, -115600, -400;
         0, 0, 0, -344, -407, -1, 137600, 162800, 400;
         344, 407, 1, 0, 0, 0, -137600, -162800, -400;
         0, 0, 0, -125, -410, -1, 50000, 164000, 400;
         125, 410, 1, 0, 0, 0, -37500, -123000, -300]

        h=null(A)
        h=h/h(9)
        H=reshape(h,3,3)'
```



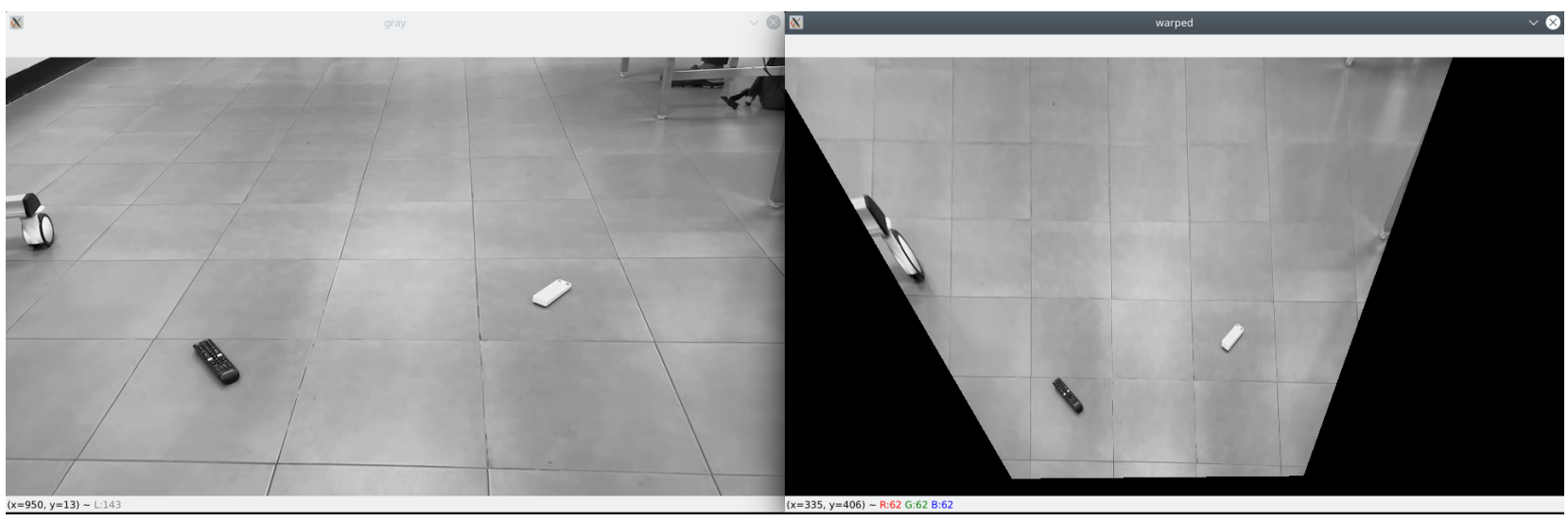
I get the same homography matrix from Octave using null space and from OpenCV using SVD and last row of Vt.

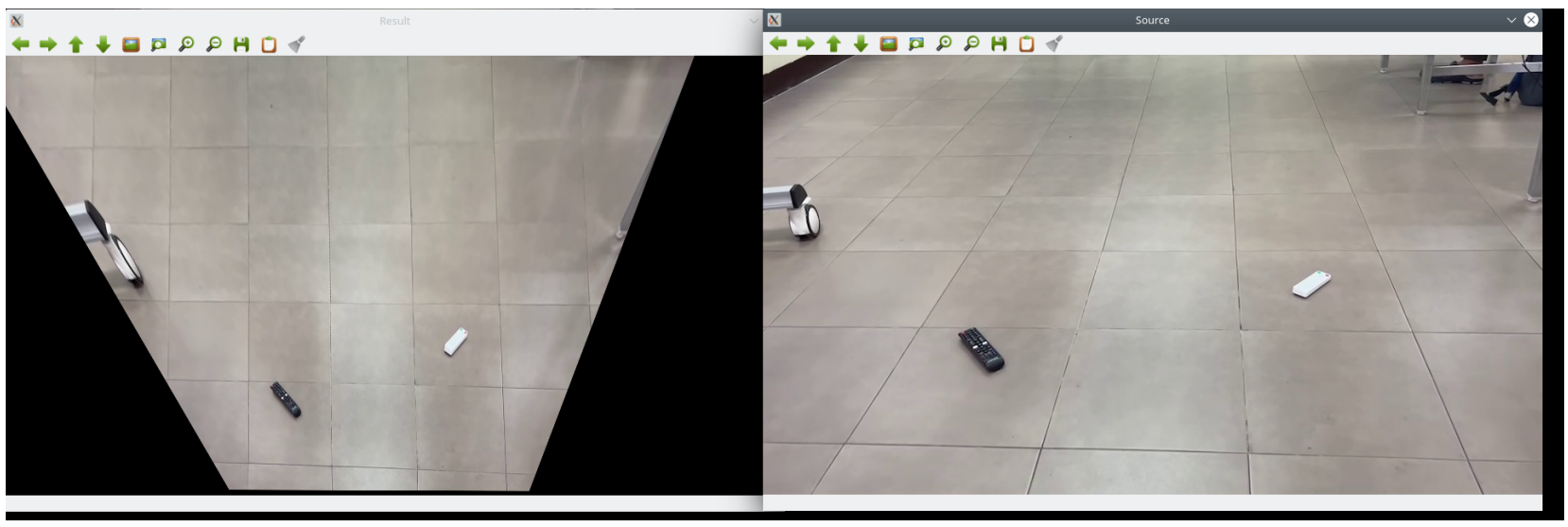
Computing rectified image using inverse image warping and previously calculated homograpgy

Implemented separately because it takes a long time to run.

```
In [ ]: # CODE FOR PERSPECTIVE TRANSFORM
        import cv2
        import numpy as np
        from math import *

        if __name__=="__main__":
```

```
img = cv2.imread("image.jpg")

homography_file = cv2.FileStorage("good_homography.xml", cv2.FILE_STORAGE_READ)
h = homography_file.getNode("homography_matrix").mat()
homography_file.release()

print(h)
h = np.linalg.inv(h)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
result = np.zeros(img.shape, dtype=np.uint8)
point = np.array([[0],[0],[0]])
tpoint = np.array([[0],[0],[0]])

for i in range(result.shape[0]):
    for j in range(result.shape[1]):
        point[0,0] = j
        point[1,0] = i
        point[2,0] = 1
        tpoint = h@point
        tpoint = tpoint / tpoint[2,0]
        if tpoint[0,0]>=0 and tpoint[1,0]>=0 and tpoint[2,0]>=0:
            x = tpoint[1,0]
            y = tpoint[0,0]
            try:
                dx = x-floor(x)
                dy = y-floor(y)
                foo = gray[floor(x),floor(y)]
                flo = gray[ceil(x), floor(y)]
                fol = gray[floor(x), ceil(y)]
                fll = gray[ceil(x), ceil(y)]
                fy = (1-dx)*foo + dx*flo
                fy1 = (1-dx)*fol + dx*fll
                fi = (1-dy)*fy + dy*fy1
                #print(fi)
                # result[i,j] = gray[round(tpoint[1,0]), round(tpoint[0,0])]
                result[i,j] = fi
            except:
                pass


cv2.imshow("gray", gray)
cv2.imshow("warped", result)

cv2.waitKey(0)
```

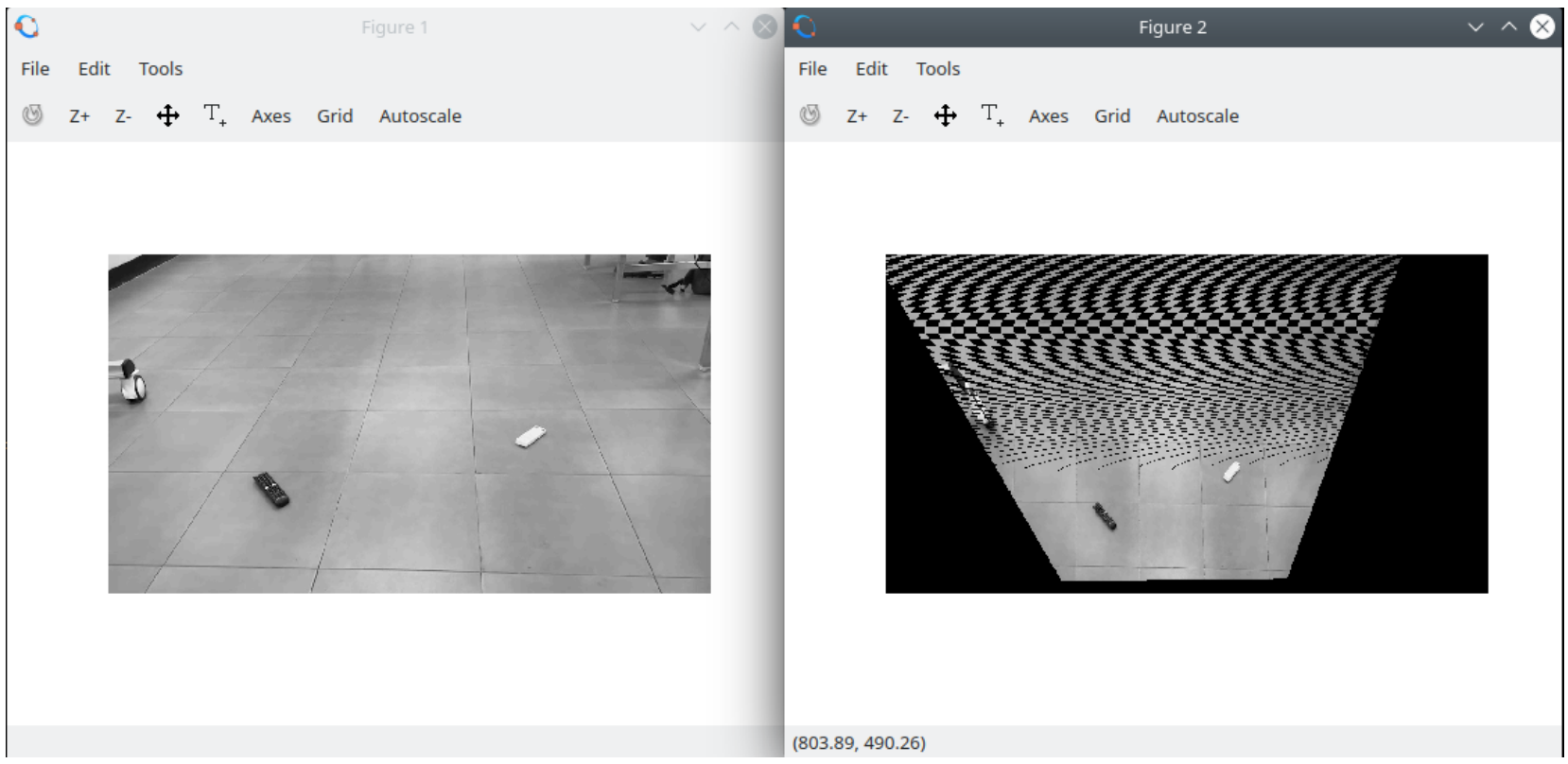




OCTAVE Code - without bilinear interpolation

In [ ]:
```
H=[1.34262430, 2.09180830, -220.45955000;
0.08859860, 3.75966310, -341.46072000;
0.00019525, 0.00417576, 1.00000000];
```

```
img = imread('/home/dell/Documents/image.jpg');
img = rgb2gray(img);
transformed_img = uint8(zeros(size(img)));

for x=1:size(img,1)
  for y=1:size(img,2)
    point = H*[y,x,1]';
    point = point/point(3,1);
    yp = point(1,1);
    xp = point(2,1);
    zp = point(3,1);
    if xp>0 && yp>0 && zp >0
      transformed_img(round(1+xp), round(1+yp)) = img(x, y);
    endif
  endfor
endfor

figure(1);
imshow(img);
figure(2);
imshow(transformed_img);
```



We see a lot of aliasing artifacts in the image when we use nearest neighbour and forward image warping instead of bilinear interpolation and inverse image warping.

Optical Flow (Original + Rectified)

Reusing previously saved homography from an xml file using the FileStorage class.

Accompanying Videos at https://youtu.be/f7eR8FBYqVU (https://youtu.be/f7eR8FBYqVU) and https://youtu.be/ZS9EJTYVJo8 (https://youtu.be/ZS9EJTYVJo8)

Using optical flows from the warped image gives us better feature points to track and results in a better optical flow more accurately reflective of the actual motion of camera.

In [ ]:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/video.hpp>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
       string filename = argv[1];
       VideoCapture cap(filename);     // declaring a capture object to grab video frames
       if (!cap.isOpened()) {          // check if capture object can access the file
            cerr << "Unable to open file!" << endl;
            return 0;
       }

       string homography_file = "../homography.xml";
       FileStorage fs(homography_file, FileStorage::READ);
       fs.open(homography_file, FileStorage::READ);

       Mat homography_matrix;
       fs["homography_matrix"] >> homography_matrix;
       cout << "homography_matrix:" << endl << homography_matrix << endl;

       // Create some random colors
```

```
vector<Scalar> colors;
RNG rng;
for (int i=0; i<100; i++) {
     int r = rng.uniform(0,256);
     int g = rng.uniform(0,256);
     int b = rng.uniform(0,256);
     colors.push_back(Scalar(r,g,b));
}

// GET IMAGE FROM VIDEO AND PROCESS TO GET FEATURES TO TRACK
Mat old_frame, old_gray; // Declare matrices to hold previous frames (color and grayscale)
vector<Point2f> p0, p1;  // Declare two point vectors (float dtype)
cap >> old_frame; // Copy from video stream to old frame
cvtColor(old_frame, old_gray, COLOR_BGR2GRAY);  // convert to grayscale
goodFeaturesToTrack(old_gray, p0, 100, 0.3, 7, Mat(), 7, false, 0.04);  // get corner featur
Mat mask = Mat::zeros(old_frame.size(), old_frame.type());  // declare a zero matrix of same

// WARP THE IMAGE AND PROCESS IT TO GET FEATURES TO TRACK
Mat old_warped, old_warped_gray;
vector<Point2f> rp0, rp1;
cv::warpPerspective(old_frame, old_warped, homography_matrix, old_frame.size(), cv::INTER_LI
cvtColor(old_warped, old_warped_gray, COLOR_BGR2GRAY);
goodFeaturesToTrack(old_warped_gray, rp0, 100, 0.3, 7, Mat(), 7, false, 0.04);
Mat mask_warped = Mat::zeros(old_warped.size(), old_warped.type());

// VIDEO WRITER TO SAVE THE OPTICAL FLOW VIDEO
VideoWriter writer;
int codec = VideoWriter::fourcc('M', 'P', '4', 'V');
double fps = 30.0;
string saveFilename = argv[2];
Size sizeFrame(int(old_frame.rows/2), int(old_frame.cols/2));
writer.open(saveFilename, codec, fps, sizeFrame, true);

while(true) {
     Mat frame, frame_gray;  // Declare current frame for every loop
     Mat warped, warped_gray;

     cap >> frame;           // Get current frame from the video stream
     if (frame.empty()) {   // Check if frame is empty
         break;
     }

     cvtColor(frame, frame_gray, COLOR_BGR2GRAY);  // convert to grayscale
     cv::warpPerspective(frame, warped, homography_matrix, frame.size(), cv::INTER_LINEAR);
     cvtColor(warped, warped_gray, COLOR_BGR2GRAY);


     vector<uchar> status, status_warped;
     vector<float> err, err_warped;     // Error vector
     TermCriteria criteria = TermCriteria((TermCriteria::COUNT) + (TermCriteria::EPS), 10, 0

     // Calculate optical flow between successive frames
     try {
         calcOpticalFlowPyrLK(old_gray, frame_gray, p0, p1, status, err, Size(15,15), 2, cri
     }
     catch (...) {
         old_gray = frame_gray.clone();
         old_warped_gray = warped_gray.clone();
         goodFeaturesToTrack(old_gray, p0, 100, 0.3, 7, Mat(), 7, false, 0.04);
         goodFeaturesToTrack(old_warped_gray, rp0, 100, 0.3, 7, Mat(), 7, false, 0.04);
         continue;
     }
     // calcOpticalFlowPyrLK(old_gray, frame_gray, p0, p1, status, err, Size(15,15), 2, crit
     /*
     try {
         calcOpticalFlowPyrLK(old_warped_gray, warped_gray, rp0, rp1, status_warped, err_war
     }
     catch (...) {
         old_gray = frame_gray.clone();
         old_warped_gray = warped_gray.clone();
         goodFeaturesToTrack(old_gray, p0, 100, 0.3, 7, Mat(), 7, false, 0.04);
         goodFeaturesToTrack(old_warped_gray, rp0, 100, 0.3, 7, Mat(), 7, false, 0.04);
         continue;
     }*/
     calcOpticalFlowPyrLK(old_warped_gray, warped_gray, rp0, rp1, status_warped, err_warped,

     vector<Point2f> good_new;
     for (uint i=0; i < p0.size(); i++) {
         if (status[i]==1) {
             good_new.push_back(p1[i]);
             line(mask, p1[i], p0[i], colors[i], 2);
             circle(frame, p1[i], 5, colors[i], -1);
         }
     }

     vector<Point2f> good_new_warped;
     for (uint i=0; i < rp0.size(); i++) {
         if (status_warped[i]==1) {
             good_new_warped.push_back(rp1[i]);
```

```
            line(mask_warped, rp1[i], rp0[i], colors[i], 2);
            circle(warped, rp1[i], 5, colors[i], -1);
        }
    }

    Mat img, warped_img;
    add(frame, mask, img);
    add(warped, mask_warped, warped_img);

    int rows = max(img.rows, warped_img.rows);
    int cols = img.cols + warped_img.cols;

    Mat combined_Matrix(rows, cols, img.type());
    img.copyTo(combined_Matrix(Rect(0, 0, img.cols, img.rows)));
    warped_img.copyTo(combined_Matrix(Rect(img.cols, 0, warped_img.cols, warped_img.rows)))

    Mat resized_result;
    resize(combined_Matrix, resized_result, sizeFrame);
    writer.write(resized_result);

    imshow("Video", resized_result);

    int keyboard = waitKey(30);
    if (keyboard == 'q' || keyboard == 27) {
        break;
    }

    old_gray = frame_gray.clone();
    old_warped_gray = warped_gray.clone();
            p0 = good_new;
            rp0 = good_new_warped;
        }

        writer.release();
        cap.release();
}
```