Computer Vision Project

Team Member:
Yiqun LIN, 1552705
Yukai JIANG, 1552667
Hejie CUI, 1552746

Detection and Distance Measurement of Speed Bumps

26th June 2018

Background Introduction

When developing no-man vehicles, detection and distance measurement of speed bumps is a specific and important task.

The basic functions and conditions are as follows:

1. The only sensor provided is a monocular camera. The camera (and your notebook) is mounted on a bracket with wheels, which acts as our “car”.
2. When the bracket is moving, the system can detect the speed bumps in the video and also can output the distance between the detected bump and the bracket.
3. We use our own server with a GPU to complete the training process.

* We can also detect the passengers on the road and measure their distances from the “car”.

System Structure Design

Processing flow:

1. Video => Frame ➔ cv2.VideoCapture

   Split the image frame from the video.

2. Frame => Distort Frame ➔ cv2.undistort

   Use the distort function in OpenCV to correct the distortion induced by fish-eye camera.

3. Distort Frame => Object Detection & Bounding Box ➔ darknet.detect_np

Use Yolo to detect the speed bumps and obtain the position and size of bounding box.

4. Bounding Box => Distance ➔ D=d^′+d × u

Firstly, transform the distortion corrected image into aerial view. According to the bounding box and the corresponding real distance of each pixel, calculate the distance from the “car”.

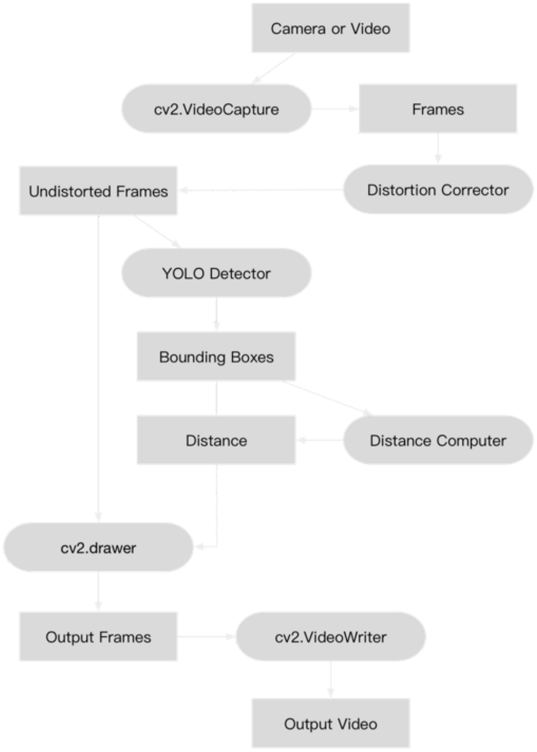
5. Attach the text ➔ cv2.rectangle/putText

Attach the distance text on the image frame.

6. Final Frame => Video ➔ cv2.VideoWriter

Write the image frame into a video.

Flow Chart



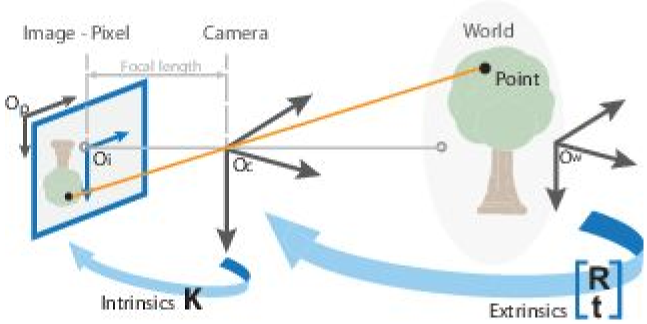
Key Algorithms Used

Camera Calibration and Distortion Correcting

Use Matlab App to get the **intrinsic parameters** and the **distortion coefficients**.

The model of image-forming principle are shown as below:

The world points are transformed to camera coordinates using the extrinsics parameters. The camera coordinates are mapped into the image plane using the intrinsics parameters.



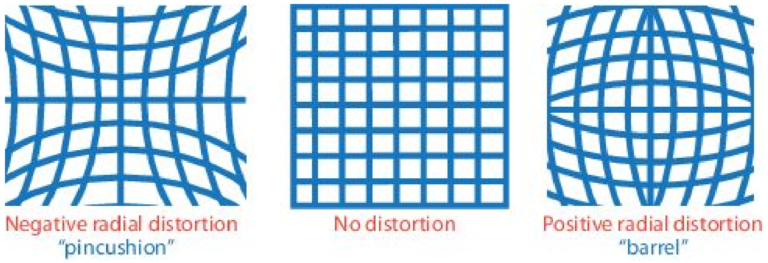
Intrinsic Parameters

The intrinsic parameters include the focal length, the optical center, also known as the principal point, and the skew coefficient. The camera intrinsic matrix, K, is defined as:

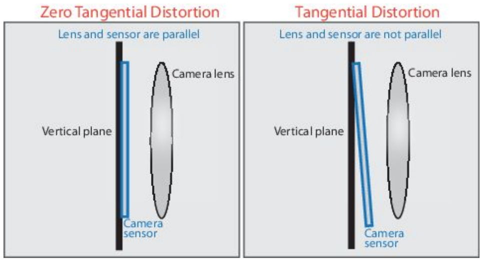
$$\begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}$$

Distortion in Camera Calibration

To accurately represent a real camera, the camera model includes the radial and tangential lens distortion.



Radial Distortion



Tangential Distortion

Typically, these two types of distortion can be represented in the following models:

Radial Distortion

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Tangential Distortion

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

So, typically, these 5 parameters are enough for use:

$$distortion\_coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Experimental Results

In our case, the distortion only involve the Radial Distortion, so the p1 and p2 are both zero.

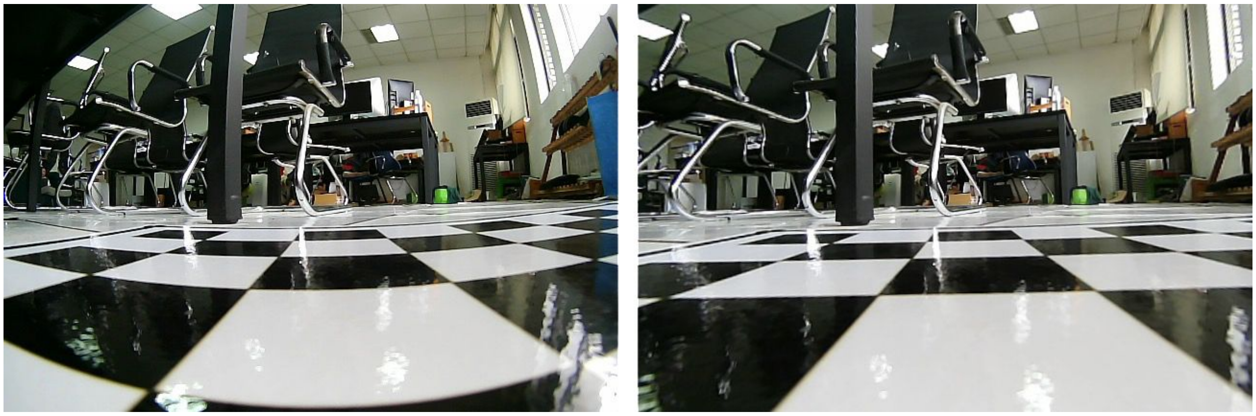
Our intrinsic parameters are:

```
np.array([[366.8414,          0, 337.4132],
          [       0,   366.1242, 182.8138],
          [       0,          0,   1.0000  ],])
```

Our distortion parameters are:

```
np.array([-0.347049837762,  0.165398310868,   0, 0, -0.0444392622226])
```

And the results are as follow:



Aerial View Transformation

Use the matlab to set the calibrating points and set the corresponding coordinates in reality.

Then we can get the points pair in two coordinate system, apply the least square method to get the homography matrix. Use this matrix to transform the image into bird view.

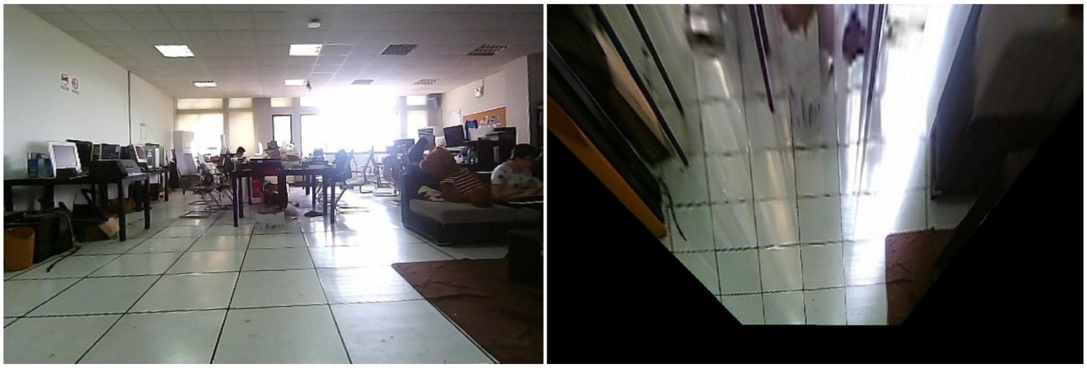
For convenience, the ratio of distance in reality and pixel numbers are set approximately as 1.

Experimental Results

The homography matrix:

```
np.array([[-0.5115,  -1.4774,  428.8531],
          [-0.0880,  -3.6296,  847.1490],
          [-0.0001,  -0.0054,    1.0000]])
```

The bird view transform results:



The pixel distance in the bird view:

- 5*60cm per 298pix
- 4*60cm per 238pix
- 3*60cm per 182pix

Object Detection

Use the Yolo v3 net to detect the bumps and pedestrians. The basic procedure can refer to the homework 3 *(instant noodles detection in supermarket video).*

Here we will give an overview of the dataset we use:

- Bump Dataset:
    a. Photos captured by the smartphone
    b. Photos captured by the camera provided by this course
    c. Images from Internet (Baidu & Google)

  After the data augmentation we have total 279 images.

- Pedestrian Dataset:
    a. VOC 2007
    b. VOC 2012

Training data set:

- Bump Dataset * 3 (repeat 3 times)
- VOC 2007-test * 0.5 (use a half of the data)

Testing data set:

- Bump Dataset * 7 (repeat 7 times)
- VOC 2012-train+val * 0.5 (use a half of the data)

Training steps: 30K

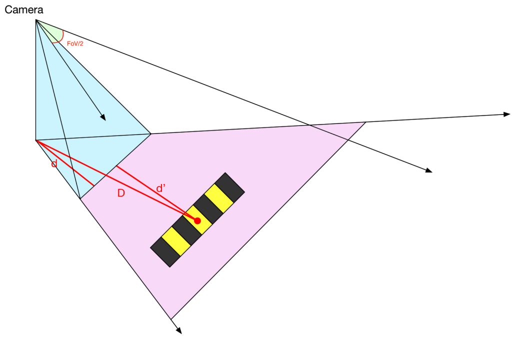
Distance Calculation

D = d + d’

D → the distance result.

d’ ➔ the calculated distance from the bottom of the bump box to the bottom of the camera view. It is calculated by the pixel number and the pixel scaling factor.

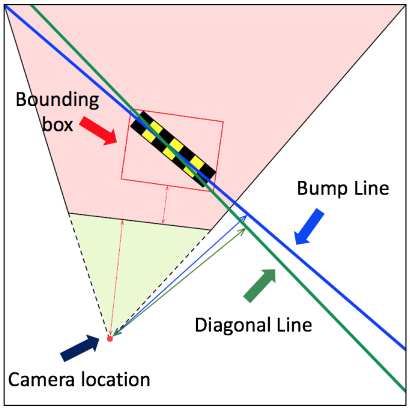
d ➔ the distance from the bottom of the camera view to the camera location. It is calculated during the calibration process.



Model Optimization

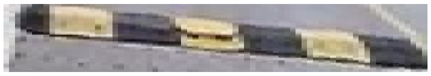
The former method to calculate the distance that we use seems not that reliable. Since the actual distance from the camera to the bump should not be calculated from the bottom of the bounding box when the bump is not horizontal.

So, we optimize the results by redefine the distance calculation.

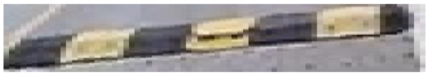


- The actual distance should be calculated from the centered line of the bump (*blue line in the figure*). So we use the diagonal line to approximate it.
- Then we can also calculate the camera location using the two boundary line of the camera view (*the pink area in the figure*).
- Use the camera location point and the diagonal line, we can calculate the distance directly.

But there is still a problem: how can we identify the type of the diagonal line, since there are two types of it.

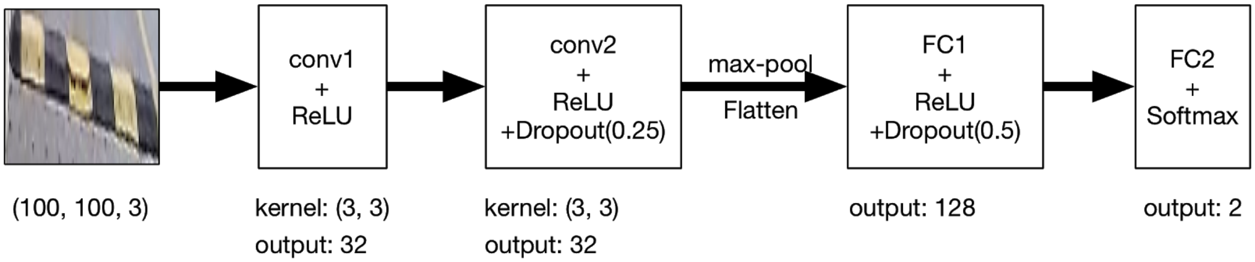


Type 0



Type 1

So, our solution is to write a rather simple network to do the classification. Here is our network structure:



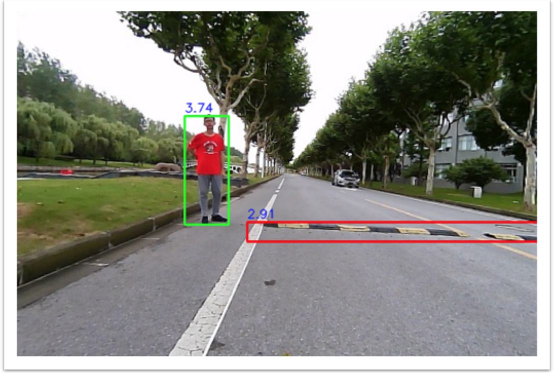
The optimization results are shown in the next document section.

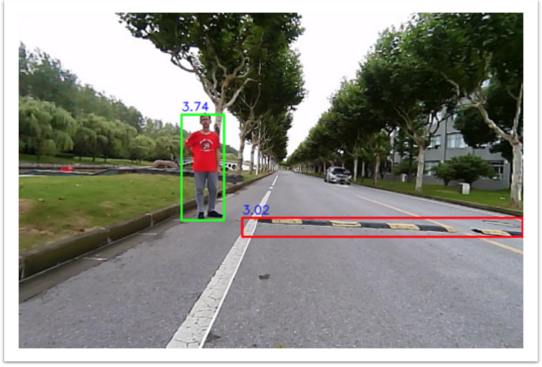
Experimental Results

Detection & measurement

Here, we will represent the results both before the optimization and after the optimization.

Case 1



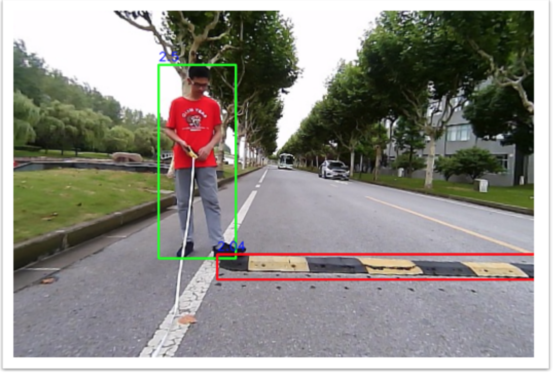


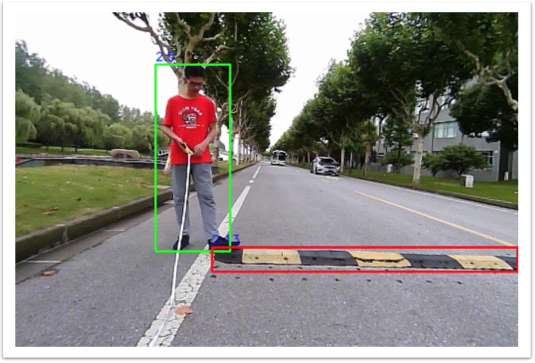
Actual Distance = 3.40 meters

Results (Before) = 2.91 meters

Results (After) = 3.02 meters

Case 2



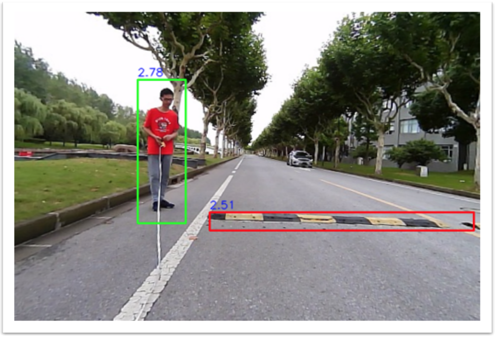


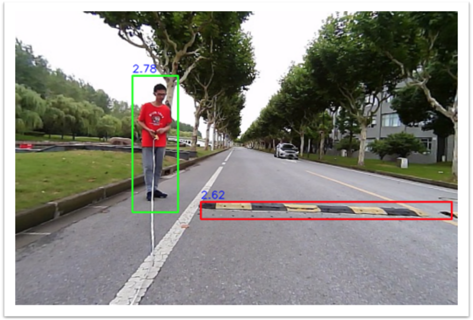
Actual Distance = 2.20 meters

Results (Before) = 2.04 meters

Results (After) = 2.13 meters

Case 3





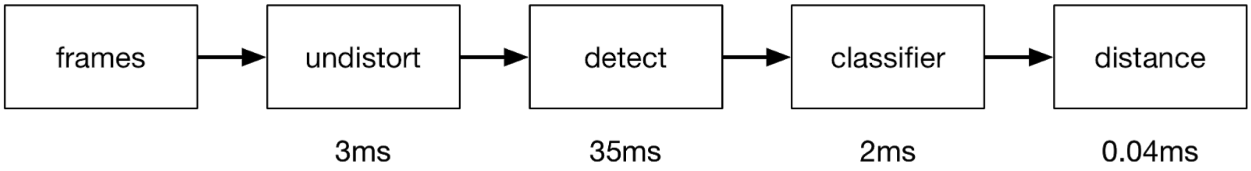
Actual Distance = 2.80 meters

Results (Before) = 2.51 meters

Results (After) = 2.62 meters

Performance Assessment

GPU: Nvidia GeForce GTX 1080
Total: 41ms



Reference

1. Camera Calibration in Matlab
2. Remove Distortion in Matlab
3. YOLOv3: An Incremental Improvement
4. Measurement in Single Camera in DIP Course Slides