

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)for more information.

Estimating camera motion trajectories in a virtual environment

Department of Automation, Tsinghua University

Name: Wang Yuqing

Date: July 26, 2021

**Catalog**

[1. Test Questions 2](#_Toc18234)

[2. Operating Environment 2](#_Toc11833)

[3. Testing algorithms 2](#_Toc22329)

[4. Experimental results 5](#_Toc12855)

[5. Problems encountered and ideas for solving them 8](#_Toc3953)

[6. Analysis and Conclusion 10](#_Toc7316)

# Test Questions

The camera trajectory is estimated from the rgb screen normal surface screen captured by the camera in the virtual environment and the 3d data from the lidar. And use the OpenCV library to draw the camera trajectory.

# Operating Environment

Windows 2010 operating system, PyCharm2021

# Testing algorithms

|  |
| --- |
| import datetime  import cv2  import numpy as np  def filter\_out(src\_frame):  if src\_frame is not None:  hsv = cv2.cvtColor(src\_frame, cv2.COLOR\_BGR2HSV)  lower = np.array([0, 0, 0])  upper = np.array([180, 255, 100])  mask = cv2.inRange(hsv, lower, upper)  return cv2.bitwise\_and(src\_frame, src\_frame, mask=mask)  def pic(image):  //Select the appropriate area for the test (including the lampholder but not the cabinet) Next  roi = image[320:512, 180:300]  \\ Acquire grayscale map  gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)  \\ Acquire image regions with high level gradient and low vertical gradient by Sobel filtering  gradX = cv2.Sobel(gray, ddepth=cv2.CV\_32F, dx=1, dy=0, ksize=-1)  gradY = cv2.Sobel(gray, ddepth=cv2.CV\_32F, dx=0, dy=1, ksize=-1)  # subtract the y-gradient from the x-gradient  gradient = cv2.subtract(gradX, gradY)  gradient = cv2.convertScaleAbs(gradient)  // Noise removal and binarization  # blur and threshold the image  blurred = cv2.blur(gradient, (9, 9))  (\_, thresh) = cv2.threshold(blurred, 70, 255, cv2.THRESH\_BINARY)  //Fill gaps to separate the object to be recognized from the background  kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (25, 25))  closed = cv2.morphologyEx(THRESH, cv2.MORPH\_CLOSE, kernel)  // Perform morphological processing and continue to remove noise  # perform a series of erosions and dilations  closed = cv2.erode(closed, None, iterations=4)  closed = cv2.dilate(closed, None, iterations=4)  (cnts, \_) = cv2.findContours(closed.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  // Sort the recognized contours from bottom to top  c = sorted(cnts, key=cv2.contourArea, reverse=True)[0]  //Select the lowermost profile  cnt = c[0]  //Drawing rectangular borders  x, y, w, h = cv2.boundingRect(c)  cv2.rectangle(roi, (x, y), (x + w, y + h), (0, 255, 0), 2)  center = (int(w/2 + x + 250), int(h/2 + y + 180)) //Get the coordinates of the center point of the rectangle border  return image, center  if \_\_name\_\_ == "\_\_main\_\_":  starttime = datetime.datetime.now()  cap = cv2.VideoCapture("savergb.avi")  if cap.isOpened(): # Whether the VideoCaputre object is successfully opened  print('The video file has been opened')  fps = cap.get(cv2.CAP\_PROP\_FPS) # Returns the fps of the video - the frame rate  width = cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH) # return the width of the video  height = cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT) # Return the height of the video  print('fps:', fps, 'width:', width, 'height:', height)  fourcc = cv2.VideoWriter\_fourcc(\*"mp4v")  video = cv2.VideoWriter('rgbB.mp4', cv2.VideoWriter\_fourcc(\*"mp4v"), 5.0, (512, 512))  i = 0  while 1:  success, frame = cap.read()  if success:  i = i + 1  frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  frame = filter\_out(frame)  frame, center = pic(frame)  data = np.load('3d.npy')  data\_i = data[i-1]  x\_i = center[0]  y\_i = center[1]  data\_i\_c = data\_i[x\_i][y\_i][2] //get the depth value of the pixel at the center of the border, i.e. the distance of the camera from the change point  num = -int(data\_i\_c\*100) // integerize the data since the depth value obtained is negative and between [-2.5916128, -0.9434452].  f = open('data.txt', 'a') //write the data to the txt file  f.write(str(num))  f.write('\n')  video.write(frame)  else:  break  else:  print('Video file failed to open')  endtime = datetime.datetime.now()  print((endtime - starttime)\*100) |
| import numpy as np  import cv2  line = cv2.VideoWriter('route.mp4', cv2.VideoWriter\_fourcc(\*"mp4v"), 5.0, (512, 512))  num\_list = []  with open('data.txt') as file\_obj:  content = file\_obj.readlines()  for j in range(0, len(content)): //extract each depth value and convert it to int form  num = content[j].split('\n')  n = int(num)  num\_list.append(n)  for i in range(0, len(num\_list)): // Draw the camera roadmap based on the depth value  img = np.zeros((512, 512, 3), np.uint8)  cv2.circle(img, (256, num\_list[i]), 2, (0, 255, 0), 4)  line.write(img)  i += 1 |

# Experimental results



Figure 1 Foreground subject outline drawing rectangular border 1



Figure 2 Foreground subject outline drawing rectangular border 2



Figure 3 Foreground subject outline drawing rectangular border 3



Figure 4 Foreground subject outline drawing rectangular border 4



Figure 5 Foreground subject outline drawing held border 5



Figure 6 Foreground subject outline drawing rectangular border 6

Attachment: depth value of pixels at the center of the rectangular border



Example of a pixel depth value at the center of a rectangular border

-2.76443

-2.640259

-2.6488626

-2.7945151

-2.7633045

-2.3250287

-2.276642

-2.2261915

-2.179585

-2.1289585

-2.0784903

-2.024272

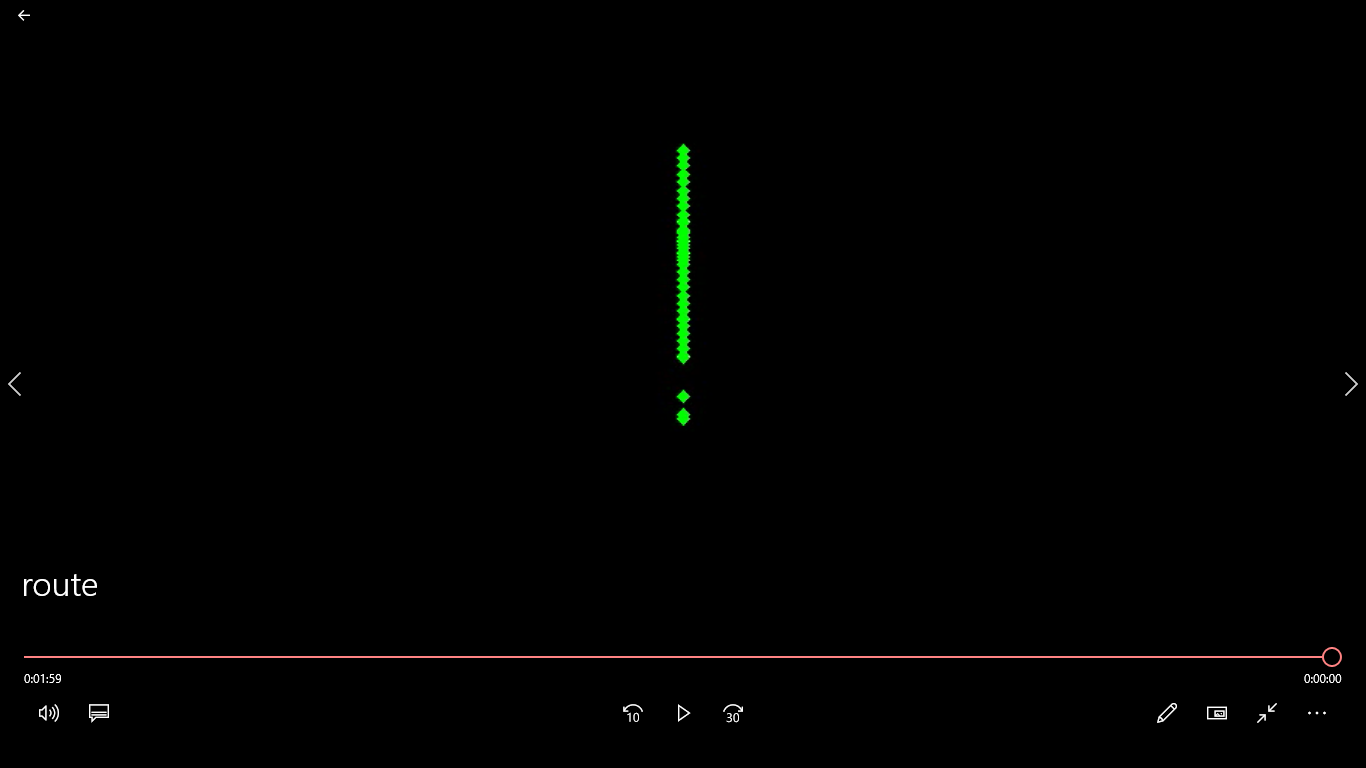


Figure 7 Drawing a path based on the depth value of the pixels at the center of the rectangular border

# Problems encountered and ideas for solving them

Problem: When fitting image contours, all foreground objects except walls and floors are included.

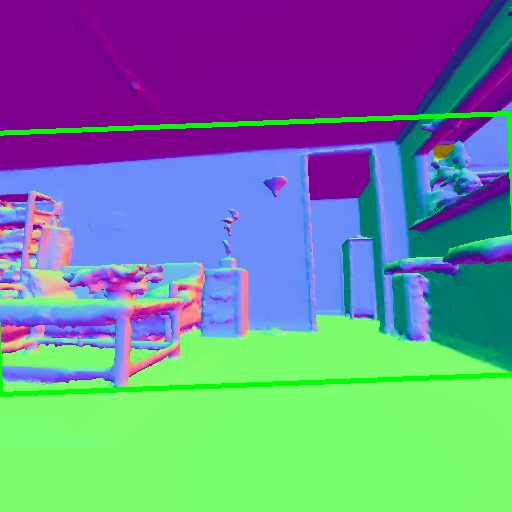


Figure 8 Surface normal directly recognizes the rectangular border of the foreground outline

Solution idea: Segment the image in HSV color space so that the rectangular boundary can contain a range more precisely.

Problem: When recognizing the outline of the foreground subject, the cabinet is usually recognized because of its large size, but it cannot be used as a control object all the time because the cabinet will gradually go outside the picture.



Figure 9 The cabinet and the lamp base with the foreground subject

Solution idea: The lamp as well as the base of the lamp will always be in the picture. Since the color difference between the lamp base and the color difference around the lamp base is more obvious than the color difference between the lamp and the lamp surroundings, you can let the contour extraction focus on the lamp base by ROI segmentation of the image. Then the query fitting of the contour and the drawing of the rectangular boundary are carried out.

Problem: Longer time consumption when all steps are run in the same python file.

Solution idea: Put the drawing route in another python file to do it.

Problem: After extracting the depth data of the center point of the rectangular border of the lampholder and importing it into a txt file, and then importing that txt file in the program that draws the path, the data is a list in the form of a string and cannot be processed numerically.

Solution: After importing the data, each data in the list is split into numeric part and carriage return part, and then the numeric part is converted to floating point data and can be processed as well as converted to integer data and subsequent processing.

# Analysis and Conclusion

Analysis: Identification and tracking of a fixed object during camera movement and calculation of the camera's trajectory by obtaining the distance between the camera and the object from 3d data. In order to reduce the error, it is necessary to select an object whose color is as prominent as possible and always within the frame for tracking. The object that meets these two conditions is the lampholder. After color segmentation of the video frame, the cabinet next to the lampholder still has a significant impact on the recognition of the lampholder, so a more accurate region processing is needed. After ROI processing of the frames, the rectangular borders can contain the lampholders as much as reasonably possible. Next, the center distance of the pixel corresponding to the center of the rectangular border of each frame is obtained, and the path is depicted with this data. After obtaining the depth of each frame, it is possible to depict what kind of motion the camera makes in a straight line based on this data.

Conclusion: The motion trajectory of the camera relative to the starting point is obtained, and the motion and rest of the camera, as well as the relative velocity of the motion, can be seen through the video of the depicted motion trajectory.