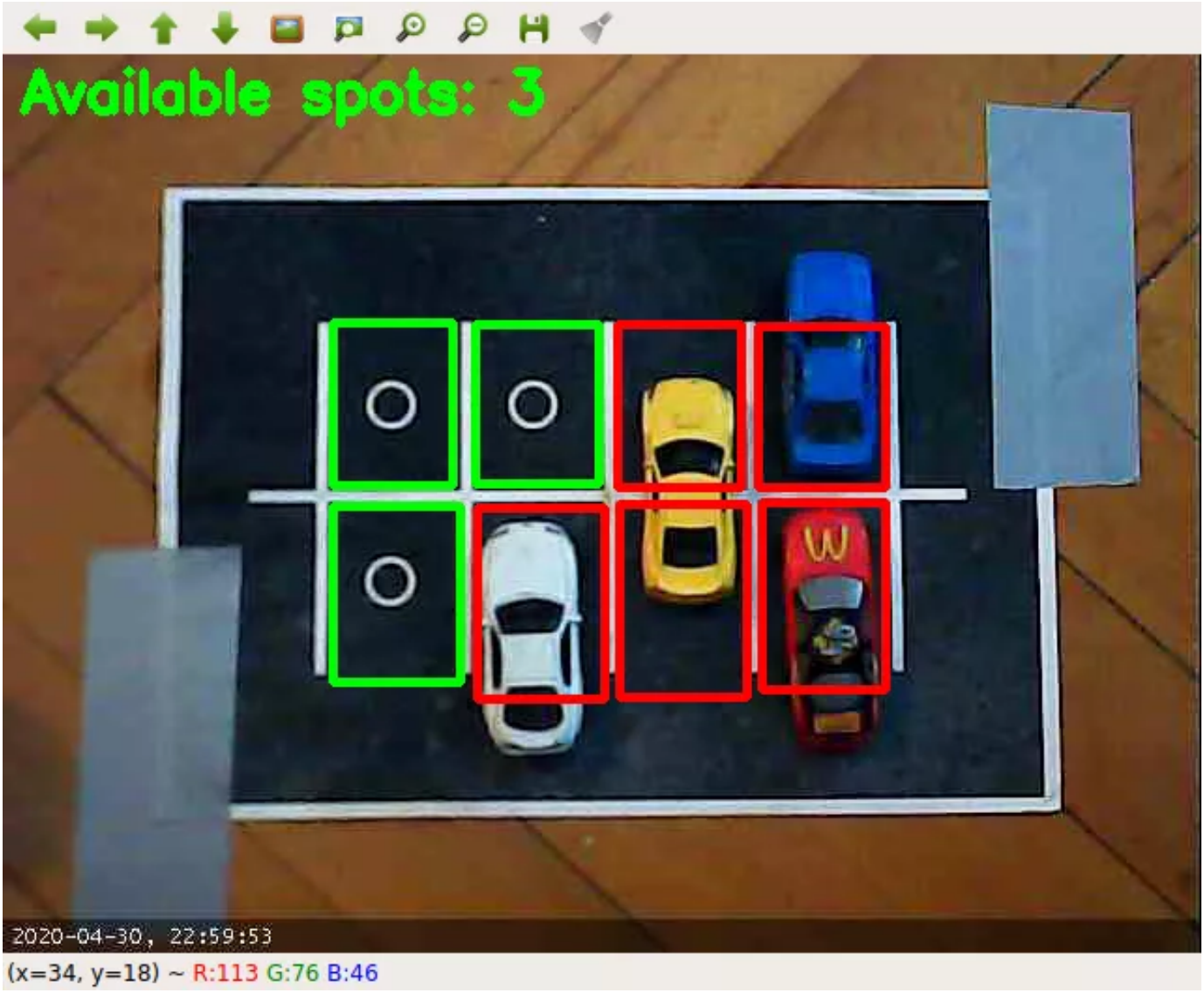
基於python和OpenCV構建智能停車系統

原創 努比 小白學視覺 2022-01-17 10:05

收錄於話題
#OpenCV視覺實戰項目

58個



當今時代最令人頭疼的事情就是找不到停車位，尤其是找20分鐘還沒有找到停車位。

根據複雜性和效率的不同，任何問題都具有一個或多個解決方案。目前智能停車系統的解決方案，主要包括基於深度學習實現，以及基於重量傳感器、光傳感器實現等。

本期我們將一起通過使用攝像頭和少量代碼來實現最簡單的智能停車系統。該解決方案所使用的概念非常簡單。它由具有以下兩個腳本組成：

1.選擇停車位的坐標並將其保存到文件中。

2.從文件中獲取坐標，並確定該點是否可用。

將該解決方案分成兩個腳本的原因是，避免在每次確定是否有可用停車位的時候，就進行停車位的選擇。

为了使这一过程尽可能简单，从现在开始，我们将这两个脚本称为**selector**和**detector**。

相关依赖

在本文中，我们使用python 3.7.6，但其他版本（例如3.6或3.8）当然也可以使用。首先我们要检查python的版本，我们通过在控制台中编写`python -version`，即可返回已安装的python版本。

```
C:\Users\Razvan>python --version
Python 3.7.6
```

在开始构建该系统依赖项之前，我们可以设置一个**虚拟环境**。通过以下链接我们可以了解更多有关虚拟环境的信息https://docs.python.org/3.7/tutorial/venv.html。

也可以使用**conda**创建和管理环境。有关更多信息见https://docs.anaconda.com/anaconda/。

在python中设置完所有内容后， 最重要的依赖关系将是**OpenCV**库。通过pip将其添加到虚拟环境中，可以运行`pip install opencv-python`。

要检查所有设置是否正确，我们可以使用以下`cv2.__version__`命令打印环境中可用的当前OpenCV版本。

```
(OpenCV) C:\Users\Razvan>python
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] ::
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print(cv2.__version__)
4.2.0
```

```
>>>
```

在第一行中，我们可以看到在该项目中使用了名为**OpenCV**的虚拟环境。

步骤

首先，我们需要安装一个停车场摄像头。由于我们并没有一个窗户可以看到的任何停车场，因此我们选择使用旧汽车玩具和印刷纸。另外，我在停车场上方设置了一个网络摄像头，以获取良好的图像，因此我们正在处理的图像如下所示：

selector选择器

接下来，我们来介绍编码部分。首先，我们需要构建选择器。我们从导入所需模块开始

```
import cv2
import csv
```

之后，我们开始获取图像，在该图像上选择停车位。为此，我们可以选择摄网络摄像头提供的第一帧，保存并使用该图像选择停车位。下面的代码是这样的：

1. 打开image变量中的视频流；suc确定流是否成功打开。

2. 将第一帧写入**frameo.jpg**。

3. 流被释放，所有窗口都关闭。

4. 新保存的图片将以img变量形式读取。

```
VIDEO_SOURCE = 1

cap = cv2.VideoCapture(VIDEO_SOURCE)
suc, image = cap.read()
cv2.imwrite("frame0.jpg", image)
cap.release()
cv2.destroyAllWindows()
img = cv2.imread("frame0.jpg")
```

现在，我们已经保存了第一帧并在img变量中将其打开，可以使用**selectROIs**函数标记停车位。ROI被定义为感兴趣的**区域**，代表图像的一部分，我们将在其上应用不同的函数以及滤波器来获取结果。

返回到**selectROIs**函数，这将返回一个列表（**类型：numpy.ndarray**），其中包含我们组装图像所需的数字及其边界ROI。

```
r = cv2.selectROIs('Selector', img, showCrosshair = False, fromCenter = False)
```

我们的列表将保存在变量r中。赋予cv2.selectROIs函数的参数如下：

1. **“选择器”**是允许我们选择投资回报率的窗口的名称。

2. **img**是包含我们要选择图像的变量。

3. **showCrosshair = Flase**删除选区内部的中心线。可以将其设置为**True**，因为对结果没有影响。

4. **fromCenter = False**是一个非常重要的参数，因为如果将其设置为**True**，则正确的选择会困难得多。

选择所有停车位之后，是时候将它们写入**.csv**文件了。为此，我们需要将r变量转换为python列表，可以使用`rlist = r.tolist()`命令实现。

拥有适当的数据后，我们将其保存到**.csv**文件中，以备将来使用。

```
with open('data/rois.csv', 'w', newline='') as outf:
  csvw = csv.writer(outf)
  csvw.writerows(rlist)
```

detector探测器

现在我们已经选择了停车位，是时候进行一些图像处理了。解决这个问题的方法如下：

1. 从.csv文件获取坐标。

2. 从中构建新图像。

3. 应用OpenCV中可用的**Canny**函数。

4. 计算新图像内的白色像素。

5. 建立一个点内的像素范围将被占用。

6. 在实时供稿上绘制一个红色或绿色矩形。

对于所有这些操作，我们需要定义一个要应用于每个位置的函数。该函数如下所示：

```
def drawRectangle(img, a, b, c, d):
    sub_img = img[b:b + d, a:a + c]
    edges = cv2.Canny(sub_img, lowThreshold, highThreshold)
    pix = cv2.countNonZero(edges)

    if pix in range(min, max):
        cv2.rectangle(img, (a, b), (a + c, b + d), (0, 255, 0), 3)
        spots.loc += 1
    else:
        cv2.rectangle(img, (a, b), (a + c, b + d), (0, 0, 255), 3)
```

现在我们已经实现了所需的功能，如果我们直接将其应用于.csv文件中的每组坐标效果可能并不好。因此我们做如下处理

首先，我们的有一些参数如果可以在脚本运行时（也可以在通过GUI）实时调整它们，那就更好了。为此，我们需要构建一些轨迹栏。OpenCV为我们提供这项功能。

我们需要一个**回调**函数，该函数不执行任何操作，但作为使用OpenCV创建轨迹栏的参数是必需的。实际上，回调参数具有明确定义的用途，但我们在此不使用它。要了解有关此内容的更多信息，查阅OpenCV文档。

```
def callback(foo):
    pass
```

现在我们需要创建轨迹栏。我们将使用`cv2.namedWindow`和`cv2.createTrackbar`功能。

```
cv2.namedWindow('parameters')
cv2.createTrackbar('Threshold1', 'parameters', 186, 700, callback)
cv2.createTrackbar('Threshold2', 'parameters', 122, 700, callback)
cv2.createTrackbar('Min pixels', 'parameters', 100, 1500, callback)
cv2.createTrackbar('Max pixels', 'parameters', 323, 1500, callback)
```

现在，我们已经创建了用于操作参数的GUI，只剩下一件事了。这就是图像中可用斑点的数量。在**drawRectangle**中定义为**spot.loc**。这是一个静态变量，必须在程序开始时进行定义。该变量为静态变量的原因是，我们希望调用的每个**drawRectangle**函数都将其写入相同的全局变量，而不是每个函数都使用一个单独的变量。这样可以防止返回的可用空间数量大于实际的可用空间数量。

为了实现这一点，我们只需要使用它的**loc**静态变量创建**spots**类。

```
class spots:
    loc = 0
```

现在我们已经准备就绪，只需要从.csv文件中获取数据，将其所有数据转换为整数，然后在无限循环中应用构建的函数即可。

```
with open('data/rois.csv', 'r', newline='') as inf:
    csvr = csv.reader(inf)
    rois = list(csvr)

rois = [[int(float(j)) for j in i] for i in rois]
VIDEO_SOURCE = 1
cap = cv2.VideoCapture(VIDEO_SOURCE)

while True:
    spots.loc = 0

    ret, frame = cap.read()
    ret2, frame2 = cap.read()
    min = cv2.getTrackbarPos('Min pixels', 'parameters')
    max = cv2.getTrackbarPos('Max pixels', 'parameters')
    lowThreshold = cv2.getTrackbarPos('Threshold1', 'parameters')
    highThreshold = cv2.getTrackbarPos('Threshold2', 'parameters')

    for i in range(len(rois)):
```

```
            drawRectangle(frame, rois[i][0], rois[i][1], rois[i][2], rois[i][3])

        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(frame, 'Available spots: ' + str(spots.loc), (10, 30), font, 1
        cv2.imshow('Detector', frame)

        canny = cv2.Canny(frame2, lowThreshold, highThreshold)
        cv2.imshow('canny', canny)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

拓展

在我们的循环中实际上只是调用的构建函数要复杂一点。

首先，我们将空间的数量初始化为0，以防止每帧添加数字。

其次，我们进入两个处理流以显示真实图像和已处理的图像。这有助于更好地了解此脚本的工作方式以及图像的处理方式。

然后，我们需要在每次迭代中获取我们创建的**参数** GUI中的**参数**值。这是通过`cv2.getTrackbarPos`功能完成的。

接下来最重要的部分，将**drawRectangle**函数应用到**Selector**脚本获取的所有坐标上。

最后，在结果图像上写下可用斑点的数量，显示**Canny**函数的结果，显然，这是一种众所周知的停止循环的方法。

我们现在便完成了一个智能停车项目！

总结

如今，智能停车已成为热门话题之一，并且有许多实现方式可以导致良好的功能系统。我们这处理方法并不是完美的，有许多方法可以更好地优化结果，并且可以在更多情况下使用。但是，即使这不能解决停车场危机，也可能是导致危机 的主要原因。

收录于话题 #OpenCV视觉实战项目 58

下一篇 · 基于OpenCV修复表格缺失的轮廓--如何识别和修复表格识别中的虚线

喜欢此内容的人还喜欢

基於Python利用OpenCV實現Hough變換的形狀檢測

小白學視覺