Full-stack project

January 2020

This folder contains a list of series of images of a warehouse captured by a drone in a single mission, named as `imgNNNN.jpg`, where `NNNN` corresponds to a number. For each of the images there is an accompanying JSON file, with the same name, save for the file extension. For example:

Image file	JSON file
img0000.jpg	img0000.json
img0001.jpg	img0001.json
img0002.jpg	img0002.json

...

These JSON files contain metadata describing the types of objects in the image, which are of two types: boxes (as in packages, or cases), and barcodes. The structure of the JSON file is a list of dictionaries, such as the following:


```
{
   "className": "barcode",
   "score": 0.48609521985054016,
   "imgSize": [
     1000,
     750
   ],
   "rect": [
     [
       0.17028214037418365,
       0.6822006702423096
     ],
     [
       0.017573440447449684,
       0.027124278247356415
     ]
   ],
   "code": "12345"
 }
```

The meaning of each dictionary is as follows:

- `className`: indicates the type of object. It can be “box” or “barcode”.
- `score`: is a confidence score, i.e. how confident we are in that this is a box or label.
- `imgSize`: this is the width and height (in pixels) of the corresponding image.
- `rect`: this is a list with the following structure: `[[origin_x, origin_y], [width, height]]`. This encodes a bounding box rectangle showing the location of the barcode or box. There are a few things to consider when interpreting this rectangle:
  - The coordinates are normalized to the (0, 1) range. Therefore, to get the pixel coordinates, `origin_x` and `width` have to be multiplied by `imgSize.width`; `origin_y` and `height` have to be multiplied by `imgSize.height`.
  - The coordinates are using the convention of having the origin in the bottom left, common for iOS but not that common elsewhere (the other common convention having the origin in the top left).
  - Therefore, `origin_x` and `origin_y` correspond to the bottom left corner of each bounding box.
- `code`: For the case of entries where className is “barcode”, this field will have the code that was read from the barcode. In several cases, the drone was unable to read the barcode; in this case, the field will be “NA”.
- The bounding boxes are not perfect; sometimes there are erroneous detections, or the bounding boxes are not very tight. This is OK. Figure 1 shows a real example, where the yellow rectangles are barcodes, the magenta rectangles are boxes, and the black on green text show the read barcode information.

Task

The task, given the raw images and JSON files, is to create a simple visualization and summarization of the inventory scanned in this mission.

The visualization is fairly open-ended - but here are some things would be cool to see:

- A “gallery thumbnail” view of the images, where selection of a thumbnail opens a bigger version of the image.
- In the larger version of the image, the JSON metadata could be visualized in an overlay similar to this example for each image. Ideally, the overlaid rectangles should be able to be toggled on and off.
- Having a per-image “item summary” view, structured as a table, with one column for “product code” (the “code” field in the JSON file) and another column for “count”, with a total count of that product code over the currently selected image. Something like:

Figure 1: Captured image with overlay

Code	Count
20046045021109	22
00049202002726	13
0794504291629	9
...	

- Having a warehouse-level “item summary” view, structured as in the above table, but with total counts of that product code for the whole dataset.
- Feel free to implement other features you may think of!

Notes on the solution

- The solution should have a frontend / backend separation; in other words, a backend server process should serve the data (the images and JSON files that we provide), and a browser frontend should retrieve data from this backend as needed.
- We’ll leave the specifics of the implementation stack up to you, up to and including the language (JS, Python, .Net, etc), libraries and framework (React, Angular, Bootstrap, etc.), for both frontend and backend.
- For this task, we prefer functionality over style; in reality, we have designer-created wireframes to guide the appearance of the solution, so the solution doesn’t need to be super polished in terms of UI.