To add a new method for scouting around in the data to situate, we need to include basically 4 new functions.

- The first looks at training data and does any machine-learningy stuff to build models that will be used later.
- The second takes an image and an interest type and generates an initial distribution structure that will have all the needed information for sampling a box when asked to do so.
- The third updates the distribution structure when information is added to the workspace.
- The fourth is the actual scouting code. Here, that means generating a candidate bounding box based on the information in the distribution structure. More detail below.

1

```
learned_stuff_struct = train_all_of_my_models( fn_lb_train, fn_im_train );
```

We need to build the 'learned_stuff' structure, which will consist of the trained models that will be selected when initializing and updating distributions later on. This is essentially the one look at training data, so build any models you'll want, store any information from the training data that you want, and name them whatever you want. Then store all of it as fields in a single struct that will be passed around during the run.

have access to:

fnames_lb_train a cell array of training label file names.

fnames_im_train has the image file names. note: these two will match (i.e., images and labels in the same order, every label has its image, every image has its label, and all of them exist)

parameters_struct contains parameters used for the run. mostly just stuff about what method to use, how many iterations, etc, but also includes the target resize for input images in pixels (if that helps with scaling anything), and a list of the types of objects that we'll build distribution structures for (which is just 'dog','person','leash' for all of our experiments. not really anything to worry about).

This function will be used in `situate_experiment.m`

2

```
[box_generating_data_struct, location_map_if_possible] =
    my_distribution_struct_initializer( image, learned_stuff, interest_string );
```

We need to build and initialize distribution structures. This will be done for each image, and for all possible objects of interest. Basically it defines the brain of an agent looking for a specific object in a specific image. You need to produce a struct that contains information for generating bounding boxes for the current image and interest. That could consist of a bunch of independent parts, like a box shape distribution, a box size distribution, and a location distribution; or a single object that has information for all of that. The struct you make will be used when sampling boxes. Since that too will be a function you define, you can store whatever you want in this data. Just combine it all into one struct.

If you want to have an image that's visible in the GUI version of situate, that should also be generated here. If so, it should be a single layer matrix that has the same dimensions as the input image.

This function will have access to:

interest string containing the object of interest this map will be used to locate

parameters_struct situate runtime parameters (probably not needed)

image the image itself, which can also be used to get the desired output size

learned_stuff the structure of learned models that was built in the last step. the whole thing will be passed in, so you'll have to do things like pick the right part to use for dogs vs the right part to use for people.

This function will be used in `situate_distribution_struct_initialize.m`

3

```
[box_generating_data_struct, location_map_if_possible] =
  my_distribution_struct_updater( image, learned_stuff, 'dog' );
```

Here we update the distribution struct (initialized previously) whenever the workspace changes. Essentially, we need the same stuff and have access to the same stuff, but we now have a workspace, so if you condition on anything, this is where conditioning will happen.

This function will have access to:

distribution_struct: what was built in the last step. contains (at least) the object of interest description, the image dimensions, a record of the boxes that have been sampled, and a copy of the whole 'learned_stuff_structure'

parameters_struct: This contains the check-in criteria for objects in the workspace. If you want to treat provisionally checked-in objects different from fully committed objects, you have to compare their IOR in the workspace against the thresholds here. Talk to me if you want to do any of this. I've just been treating everything that's made it into the workspace the same.

workspace: the workspace contains bounding boxes for objects that have been checked-in. workspace.boxes is in $[r_{initial}, r_{final}, c_{initial}, c_{final}]$ format, where r and c are *row* and *column*, respectively.

An example workspace with 2 objects, both checked in with scores over the provisional check-in threshold (`parameters_struct.total_support_threshold_1`), but under the full threshold (`p.total_support_threshold_2`).

```
workspace =
               boxes:
                    [ 86   417   294   394
                     222   292   329   354 ]
             labels: {'person'  'leash'}
         labels_raw: {'person'  'leash'}
    internal_support: [0.4600 0.3400]
       total_support: [0.4600 0.3400]
```

This function will be used in `situate_distribution_struct_update.m`

4

`box_r0rfc0cf = sample_a_box_using_my_data( distribution_struct );`

This is where we actually generate candidate bounding boxes. Situate will select an agent and that agent will respond with a bounding box.

This function will have access to:

distribution_struct whatever model you've built for this object of interest type will be stored in here as a struct that you named, so you should have whatever you need.

This function will be used in `situate_sample_box.m`

notes for max

situate_gui.m need to make sure that 'train_all_of_my_models' get's placed here too.

situate_visualize.m if there is visualization data, make sure it's looked for here. otherwise, it'll default to the 'not-implemented' drawing