Team Battle Bots: The Shoe Monster Edition

Rachel Boy, Victoria Coleman, Brooks Willis

11/10/2014

The Goal

Our goal was to turn the innocent looking derpy neato vacuum cleaners into incognito RUTHLESS KILLER ROBOTS. The robot uses image recognition to locate, track, and attack shoes. We wanted to compare a couple of strategies of image processing for identification of shoes. We wanted to be able to track a manually identified shoe with the stretch goal being to identify a generic shoe.

SHOE MONSTERS
SHOE MONSTERS ATTACK IN PAIRS AND DEVOUR THEIR VICTIMS FROM THE FEET UP.
Chris Madden

Figure 1: Cartoon warning the public about the dangers of shoe monsters. They can come in many forms, so always be on the lookout.

Videos of our neato chasing a sock and a shoe can be found here:
https://drive.google.com/file/d/0B2V5eGVz0HJGVmpCaUwxRE1ONDg/view?usp=sharing

Our Solution

We adapted keypoint and histogram-based tracking methods to track shoes, then used visual servoing techniques to drive towards, and eventually eat, the shoe. In our final implementation, we initially identify the shoe to attack for the neato, then use both keypoint and histogram based tracking to identify the probable location of the shoe on the image. The neato then navigates towards this location using a proportional controller, with angle based on the image and speed based on lidar data.

When building our code, we drew heavily on examples provided in class and on the internet. (Though mostly the examples in class were what ended up in our final code - the internet helped us with things like specific color tracking, which we used only as an intermediate, proof-of-concept step). Since we didn't have a firm grasp of the libraries involved at the beginning of the project, this gave us good working code to start from so we could work on understanding, modifying, and refactoring it to fit our actual needs.

In implementing our solution, we decided to abstract the object tracking from the navigational code, so that we could develop separately and more easily swap out implementations. We therefore have one node which handles navigation based on the location provided to it by another node, which handles object tracking.

For the object tracking node, we decided to structure it as a single mediator which could have any number of individual tracking objects, using different algorithms or implementations. All these tracking objects supplied the same interface for tracking - a function which takes an image and returns a center and a confidence - which allowed us to easily switch out and combine basically any tracking algorithms in a consistent manner.

To successfully use keypoint tracking in real time, we had to choose which keypoint extractors and descriptors to use. In the end, we actually decided to use different algorithms for pulling out keypoints and getting their descriptors, because SURF, which appeared to give us the best keypoints, was far too slow for real-time tracking. FAST, however, which found features much more quickly, was not actually available as an algorithm for descriptor extraction in the library we were using. So we combined the two, using FAST for keypoint detection and SURF for descriptor extraction, resulting in a system that may have been slightly less accurate, but still worked well, and ran approximately ten times faster.

The Code

Our code was designed to be very modular because we used multiple types of object tracking and initially considered adding more. We split the code into objects with discrete responsibilities such as navigating (servoing.py), and implementing a specific tracking technique (ex, HistTracker in track_shoe.py). A high level diagram of our code can be found in Figure 2. Each tracking method is isolated from the rest of the code so that they can be added or removed easily. To add a new method we simply add a new object to “track_shoe.py” and call it in “testing_track_shoe.py” along with the others. “testing_track_shoe.py” is the data middleman of our code. It gives data to the tracking objects, combines their results, and spits out a target to the navigation node. The navigation node houses the visual servoing portion of the code, and has the structure for object avoidance or other behaviours to be added on once they have been created. This script also determines the distance to the object we are tracking. In general, this script deals with all of the navigation and behaviour of the neato.

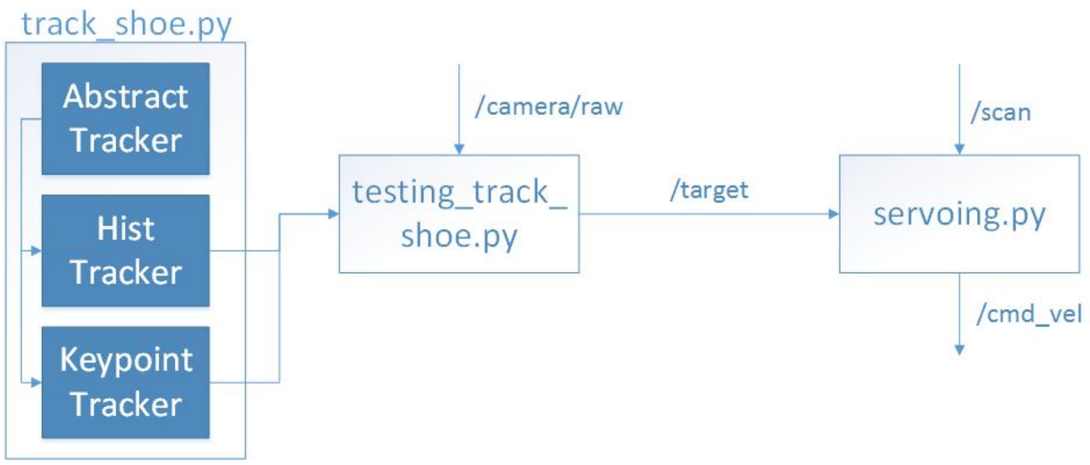

Figure 2: This diagram shows the high level overview of our code. White boxes represent scripts, and blue boxes represent objects (only ones which are called by other objects are shown).

Challenges

One of the most difficult challenges with this goal was the “moving shoe” problem. The most promising tracking algorithms were color histogram, keypoint matching, and template reference, all of which would require some sort of template to match. Unfortunately, the shoe has a lot of variation as a person walks around, both rotationally and in size. As it rotates, the view of the shoe becomes very narrow and can look dramatically different when viewed from the front and back as opposed the sides. Also, because the shoe not only looks different but rapidly changes in size, template matching is virtually impossible since it is quite sensitive to changes in size and orientation. With a relatively large time delta because template matching is a computationally heavy operation, there is too much variation from image to image for it to be effective. Because there is lots of variation in the shoe and image, keypoint matching and color histogram also have issues with consistency. Frequent changes in exposure could throw off histogram tracking, while changes in size and orientation confused keypoint matching. Tuning those two methods proved difficult, but we did have some success.

Our robot unfortunately had an overly refined taste for shoes. It only liked distinctive shoes with lots of character, none of these boring run-of-the mill shoes most people have. As a result, it only noms stylish people with vibrant or excitingly patterned shoes.

But in the end, we were right about what the biggest challenge of this project would be… We had a team of three seniors with two major higher priorities than CompRobo. This was unfortunate because we think comprobo is awesome and wish we could spend more time on it. But between SCOPE and job interviews, we had trouble being in the same room as each other for a little over a week, which is less than ideal for group projects. We tried to put in as much time as we could anyway, though!

Lessons Learned and Next Steps

Next Steps

We have a decently long list of things we would like to add to the project if we had the time. The biggest ones are:

- Add probability to keypoints! Keypoint tracking and color histogram tracking work very well in totally different situations (one where there's a lot of color variation, one where the object has a lot of interesting points). We can imagine how we would get the probability of a key point match, by looking at either the number of matches found inside our bounding box, or how consistently spaced the matches are (are they spaced in the image to track in approximately the same way they are in the template image?). With a reasonable guess for the accuracy of keypoint tracking and color histogram tracking, we would be better at choosing the more accurate one.
- Change how objects are identified in histogram tracking! Right now meanshift is used, which works decently well, but never updates the assumed size of the object, which can lead to inaccuracies as it gets closer or farther away. In order to improve this approach, we would identify the shoe using connected components instead. By finding a connected component of a similar size to the last detection in color heatmap of the new image, and updating the proposed size based on that finding, we could improve our accuracy when detecting, for example, white shoes, which are frequently confused with the wall in our current setup.
- Try having more voting algorithms! There are a number of parameters to tweak and tune in both the histogram and keypoint algorithms (and there are other algorithms we could try). As long as we think that our algorithms return roughly accurate confidence scores with their results, then we could try running more algorithms, or more instances of our current algorithms with different parameters. With histogram tracking, for instance, we could vary the number of histogram bins, or the number of cycles to convergence, or the use of mean shift versus some other method for finding the center. This would slow our process down (so we would probably only want to do this with relatively fast algorithms to maintain real-time operability), but could increase our accuracy (as we would theoretically disregard inaccurate, low-confidence results, and follow whichever setup returned good

results!). We could also only run other methods when the main algorithm(s) fall below a certain confidence threshold to help mitigate the computation time issue.

Lessons Learned

The modularity of our code was hugely beneficial to the project as it allowed us to quickly swap in vision processing methods to try, and easily remove ones that didn't work well (such as template matching. Poor template matching). It also led us to a code base which we could very easily expand upon in the future by simply plugging in new vision modules. For the larger final project this will be very useful since it makes intermediate steps require less work on the integration side.

The camera itself has a huge effect on the effectiveness of the tracker, and subtle changes in the image to track can drastically change performance. For instance, we originally tried having the neato find a shoe on the basis of an image we had taken with the webcam. But the differences in the exposure, coloring, and resolution of the images meant that this was just totally ineffective, and we went back to specifying the example image from the neato's camera. Improved white balancing or a better camera could increase the accuracy of the algorithms.