Text Localization for Rescue Robots

Computational Robotics

Diana Vermilya and Nathan Lintz

December 18, 2014

1 Goal

In search and rescue scenarios, robots must explore unfamiliar places and identify signs of human presence. Robots can be helped immensely if they can read written information in real time such as the exit label on an emergency exit. By developing a suite of tools which convert images into a digital format, we will improve the Neato's search and rescue capabilities. We implemented the algorithms described in Detecting Text in Natural Scenes with Stroke Width Transform by Microsoft[1]. These algorithms take in a base image, calculate a Stroke Width Transform for the image, discover potential letter candidates, and group letters into regions of text. We used the Tesseract[2] OCR library to convert these regions into digital text.

2 Design Decision

Our major design decision was the choice to focus on localization over Optical Charactar Recognition (OCR). Because a lot of advanced OCR research has been accomplished and made openly available, we chose Google's Tesseract OCR library. This choice allowed us explore the current research in the space of Text Localization, an area that interested both of us. After considering many research papers, we chose one that leveraged Stroke Width Transforms and Connected Components. This made for a challenging and interesting pronect.

3 Code Structure

The code for this project can be found on GitHub[3][4]. We chose to structure our code into a test module and a lib module. Within the lib module, we broke down the problem into 4 main files, one for each of the major algorithms. swt.py is responsible for generating a Stroke Width Transform (SWT) of the image. connected_components.py analyzes the stroke width transform and divides it into discrete components based on similarity of stroke width. Each of these regions are considered as possible letters, and the most likely letter candidates are sorted out. textLocalizer.py manages the entire localization process, and groups the letter candidate into regions of text. letterCombinator.py contains letter, pair, and letter chain classes are used by textLocalizer in the identification of regions based on letter candidates. In the test directory, we have tests for each of these files which include functions to output an image at that step in the process for visual inspection Finally, we have Incorporated this set of algorithms into a ROS node which translates the field of view of the Neato's camera.

[1]http://digital.cs.usu.edu/ vkulyukin/vkweb/teaching/cs7900/Paper1.pdf
[2]https://code.google.com/p/tesseract-ocr/
[3]https://github.com/nlintz/comprobo2014/tree/master/src/text_localizer
[4]https://github.com/nlintz/StrokeWidthTransform

4 Challenges

One major challenge we faced was speed. If a Neato is 'reading' text while driving, and the information on signs is affecting its' choices, it needs to be able to read quickly. The algorithms we worked with were fairly computationally intensive. In order to get the speed we needed, we used Cython, a python transpiler that converts some parts of python computation into C. This gave us a 10x overall increase in speed.

Our other main challenge was in working with Tesseract. We expected the library to be more accurate. In actuality, OCR is a hard problem and while Tesseract seems to be the best option available, there were many examples in which we located text within a world accurately, but were unable to get any useful output from Tesseract.

5 Future Improvements

If we had extra time on this project, there are a few areas we've identified for improvement. First, we'd like to experiment with variants of the letter filtering and letter connecting algorithms outlined in the Microsoft research paper, in order to improve upon their work. Secondly, we'd like to move into the OCR space and implement a recognition algorithm tailored to processing text on signs. Lastly, we'd like to explore standard improvements such as processing larger images more quickly.

6 Lessons

Choosing a well segmented project in order to increase personal ownership over the project ended up being very effective for us. We will work to scope projects similarly in the future. Test more and test more frequently. We did a lot of testing in this project. This helped us reduce error and maintain a deep understanding of the project. When we ran into issues, it was generally because we didn't test enough. Explore external dependencies earlier. We decided to use Tesseract early on but didn't test it until the end of the project. Had we better understood it's strengths and weaknesses earlier, we could have build the project so as to use the library more effectively.