Camera Localization and Pose Estimation Utilizing Features of Downtown Toronto's Urban Landscape.

Report Written and Prepared by: Graham Rigby

Group Members: Graham Rigby (1002090211), Paul LoBuglio (1002161741)

Introduction and Related Work:

Our project estimates camera location and pose given an outdoor photo taken in downtown Toronto's scenic urban landscape. We are able to do this by utilizing key geometric features that are visible from various locations with little to no building obstruction. Specifically, we take advantage of the symmetricity of the CN Tower and the Aura Tower, as well as the parallel lines, formed by Toronto's street grid like layout, present at street level. In order to do this, we realized we would need to create some sort of object detector to detect key landmarks or buildings of known location and use this information to give a rough estimation of where the photo was taken. In addition, we would need some known parallel lines in world space formed by buildings or streets and find vanishing points of those lines to give an idea of the camera pose or orientation that took that photo.

Since I have moved to Toronto, I have been astounded by the size of the CN Tower (553m). The CN Tower is an iconic landmark in Toronto and visible throughout most of Toronto when not obstructed by buildings. Similar to Polaris, The North Star, the CN Tower gives Torontonians a rough idea as to where they are. This idea of using the CN Tower as a reference point was the primary inspiration for the project. In addition, when I realized I could apply some of my favorite topics from CSC420, object detection using CNN's and camera models and homographies, I was ever so more excited to work on this project.

Another camera location estimation done in a study by Marcus Brubaker, Andreas Geiger and Raquel Urtasun (Brubaker et al). This location estimation leveraged Gaussian mixture models and vehicle trajectory obtained by two cameras on the roof of a car. From there, a graph representation of street maps was used to derive the most likely position of a vehicle on a street map given the vehicle's trajectory. This approach however is not bounded by the same constraints of a single camera and a single image. In addition, the photo was not necessarily taken from a vehicle with a computable trajectory.

Given:

(Fig 1)

Numerous methods to determine camera orientation estimates using vanishing point detection have been performed (Jo et al). Although these estimations typically record orientation relative to some landmark. These orientation estimators use video recordings; however, our approach attempts to compute orientation using a single image taking advantage of parallel lines in world space.

Methods:

In attempt to estimate the location of a camera I first used a convolutional neural network to identify notable towers in the Toronto skyline. The neural network I used was a TensorFlow implementation of the Faster Region Based Convolutional Neural Network, (Faster RCNN). This network structure was chosen for object detection because of its high accuracy and speed: scored 73.2% mean average precision on Pascal VOC 2007 test set and detected object regions 3x faster than using selective search methods (Ren et al). In addition, I chose this network because it leverages a region proposal network. Region proposal networks output bounding boxes, or anchors, around objects of interest as well as respective scores that are indicative of how likely an object of interest is inside drawn box.

The dataset used to train this network included about 150 photos of the CN tower and 150 photos of the Aura Tower. Images were scraped from google images and encompassed views of the CN Tower or Aura Tower from a variety of viewpoints and changes in lighting. The images were then split into training and validation sets at an 80:20 ratio respectively. A program called LabelImg was used to draw ground truth bounding boxes around the CN Tower or Aura Tower in each photo. We then modified a python script to convert these images and bounding boxes into a file format compatible with TensorFlow's faster rcnn model. Prior to training, a few modifications of the batch size and prefetch capacities had to be made to run on my system. I ran the neural network locally on a Nvidia GTX 1050ti. The network training took roughly three hours using TensorFlow GPU packages. Training was terminated when the network achieved a total loss of under 0.01 consistently.

The bounding boxes obtained from the neural network were then used to build a regression model to estimate how far away the camera was from the detected object. In order to build this model accurately we needed to satisfy two requirements. First, all the photos needed to be taken from the same camera or same camera calibration matrix and resolution. Second, the detected objects need to be approximately symmetrical. This is because the model takes pixel width of the bounding box of a detected object, and distance between the camera and the actual object in real world coordinates. The idea here is that the width of the bounding box will be larger when the actual object is closer and vice versa. Other methods to compute distance between a point in world space and camera location such as disparity with stereo cameras are not as accurate for points very far away. The dataset that was used was pictures of the CN Tower and Aura tower taken from my cell phone camera from various locations across Toronto. I recorded the world coordinates of my camera at the location of each photo to compute distance between myself and the world coordinates of the detected tower. To account for the earth's natural curvature, simply calculating Euclidean distance between coordinates would not suffice as a distance metric. For this reason, I used a python library called GeoPy and used an algorithm that does account for earths natural curvature to compute the distances between coordinates of camera and tower in kilometers. Two regression models were built using my cellphone photo dataset, one for each tower. Each model passed in the width of the respective detected tower's bounding box, drawn by the Faster RCNN model, as well as the distance between camera coordinates and tower coordinates.

Symmetrical objects are useful for making distance estimations between an object and camera because the width of the object in the image plane is the same on all sides making the model more accurate. However, there are no way to tell what side of a symmetrical tower that the camera was when the photo was taken. This results in a range of possibilities around the edge of some circle with a radius estimated by our regression model. The solution to this is a requirement that two towers be present in the image. Given we have two detected towers in an image with known world coordinates as well as estimated distance between each tower we end up with a simple trigonometry problem for estimating camera coordinates. (**Fig 2**)

$$\text{A} = \text{Distance between CN Tower and Aura Tower}$$
$$\text{B} = \text{Distance between camera and CN Tower}$$
$$\text{C} = \text{Distance between camera and Aura Tower}$$
$$\text{Latitude} = \frac{A^2+B^2-C^2}{2AC}$$
$$\text{Longitude} = \pm\sqrt{B^2 - Latitude^2}$$

(Fig 2.)

Camera pose/orientation estimation was performed by computing the vanishing points of parallel lines in world space. This requires a photo with parallel lines in world space present that aren't necessarily parallel in image space to compute vanishing points. This was used to find the camera's absolute orientation. Because Toronto's street architecture is grid shaped, we can compute the bearing of view relative to one of the street's directional axis (North/South, East, West). The only requirement to do this is to use a photo that contains two parallel lines in world space. The intersection of two lines can be represented in homogenous coordinates, for example, the intersection in the direction of south: [0,0,1,0]. A camera's calibration matrix can then be used to project the vanishing point into image space. (**Fig 3**)

$$\text{Vanishing Point} = \text{v}$$
$$\text{Homogeneous directional coordinates: s Camera callibration matrix: K}$$
$$\text{Change of basis from street space to camera space: R} = [\text{r1, r2, r3}]$$

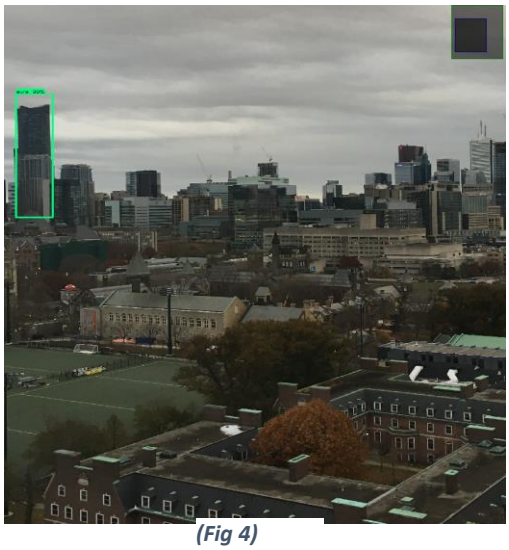
$$\text{Then } v = K[R]s^T$$
$$\text{Then } v = Kr_3$$

(Fig 3.)

The inverse of the camera calibration matrix can be utilized to find where the image plane was projected from. In other words, looking at the inverse of the camera calibration matrix and a vector of points set at a bearing of how it was projected. Then taking the other parallel lines from the street space that are visible in the image space, we can take the cross product r3 * r1 = r2. From here a simple change in basis from world space (true north, true south, true east, true west) is all that is necessary to compute the bearing of the camera. This is because we have a matrix that transforms street space to camera space, and we already know where the vanishing point is in image space.

Results and Discussion:

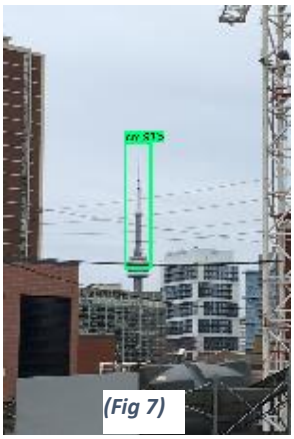
The Faster RCNN training on the CN Tower and Aura Tower Image datasets took approximately 3 hours and was stopped when total loss was below 0.01 consistently calculated with bounding box regression. The resulting Neural Network results were fairly accurate. Tight bounding boxes hugging the detected Towers can be seen in (**Fig 4,5,6,7**). This is especially useful for calculating distance since we take the width of the bounding box as an input for our regression model. The results are not as accurate when there is any building obstruction present, resulting in lower scores and poorer boxes.



(Fig 4)

(Fig 5)

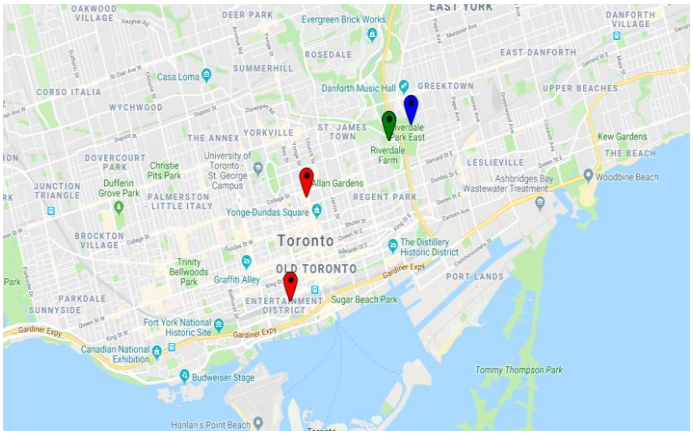
(Fig 6)



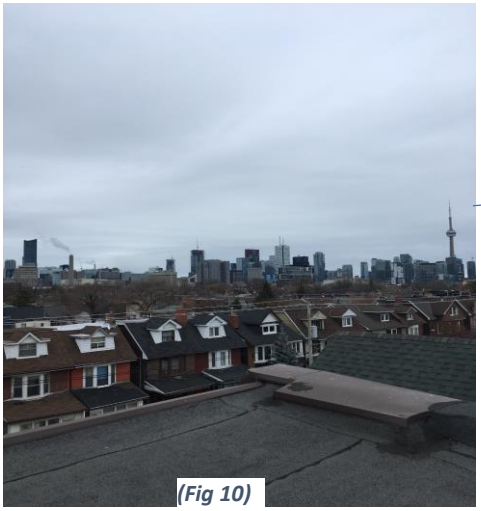
(Fig 7)

The Regression model used to estimate distances was only accurate to within a couple of blocks. In our two test images our model predicted the CN Tower and Aura Tower were closer to the camera than they were by roughly 100-150 meters. Using these models our overall estimation of the camera using the calculations described in the methods section gave somewhat accurate estimations of camera location. Figure 8 and Figure 10 correspond to the test images of the CN and Aura Towers and Figures 9 and 11 correspond to estimations of the camera location: Legend (Red marker = Cn and Aura Tower coordinates, Blue marker = actual camera coordinates, Green marker = estimated camera coordinates).

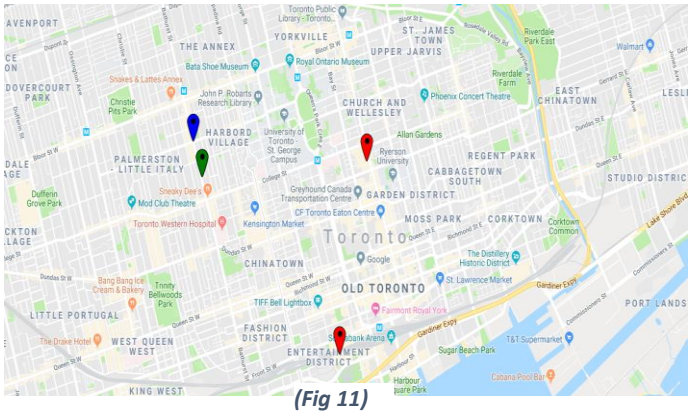
(Fig 8)



(Fig 9)



(Fig 10)



(Fig 11)

The orientation estimation field of view appear very accurate however we did not have access to a compass to verify the true underlying field of view from the camera. The photo in **(fig 1)** was taken at U of T Varsity stadium with the resultant field of view and was a nice way to demonstrate all the objectives from our project.

Main Challenges:

The main challenge in terms of getting a good object detector was picking a good model. TensorFlow offers several models with trade-offs for speed and accuracy. Initially the object detector used an SSD MobileNet network model, a neural network that is revolutionary for its ability to compile on cell phone chips and offers reasonable performance. This method did not work as well for detecting our objects because it required image sizes to be at most 300x300 pixels. This seriously lowered the performance of our regression model for estimating distances between object and camera. We had to make a trade-off for improved accuracy over speed in order to get somewhat accurate results for camera estimation.

TensorFlow GPU libraries were a bit of a pain to set up on my system and running the models caused my video card to run out of memory before slightly resizing images that were super high resolution. In addition, I had to lower the batch size and prefetch queue sizes for network training.

For computing distances between coordinates, I was using Euclidean distance for countless hours getting poor results. I assumed that the earths natural curvature could hardly affect the distance between two coordinates. I was very wrong, GeoPy distance estimation between two coordinates was the last fix that was made that gave us reasonable results that we were quite proud of.

In term of getting accurate camera orientation boiled down to hoping that street lines were reasonably parallel in most locations, this proved to be only a minor worry as we believe the results were somewhat accurate based on the photo and the output FOV map.

Conclusion and future work:

Our final project utilized a faster RCNN model implemented on TensorFlow and was trained on dataset that contained a wide selection of photos of the CN Tower and the Aura Tower in downtown Toronto. Photos were taken from a variety of viewpoints and contained various changes in lighting. From here two regression models were trained to estimate distance between camera coordinates and tower coordinates using the width of the bounding box of the detected tower as inputs. From here basic trigonometry was used to estimate the camera location. Next, we took advantage of Toronto street grid architecture to find the vanishing points of two parallel lines in world space. We were able to apply a projection using the camera's internal calibration matrix and deduce the camera orientation and plot the resultant field of view on google maps. We are planning to continue development to this project following its due date. We believe that we can improve our regression model and get more accurate results by providing more training data. In addition, using higher resolution images we could get more accurate bounding boxes further improving the regression model's prediction accuracy.

References:

Marcus Brubaker, Andreas Geiger and Raquel Urtasun. Lost! Leveraging the Crowd for Probabilistic Visual Self-Localization. www.cs.toronto.edu/~mbrubake/projects/CVPR13.pdf

Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

Youngran Jo, Jinbeum Jang, Joonki Paik. Camera orientation estimation using motion based vanishing point detection for automatic driving assistance system. https://ieeexplore.ieee.org/document/8326303