# Trilateration-Based Robot Localization with Learned Visual Landmarks

Valeria Salas and Alfredo Weitzenfeld

University of South Florida
Department of Computer Science & Engineering
4202 E Fowler Ave, Tampa, FL

salas@usf.edu, aweitzenfeld@usf.edu

## ABSTRACT

Many strategies for robot localization exist, such as trilateration and triangulation algorithms, that compute relative distance and orientation of robot to multiple landmarks. These algorithms require predefined knowledge of landmarks. In this paper we describe a trilateration algorithm for robot localization where new landmarks may be defined using deep learning algorithms. We use a single camera to learn new custom objects. This information is passed as input to the system together with the distance to the objects calculated by a previously calibrated distance detection algorithm. A deep learning algorithm is applied to the object detection model making use of TensorFlow's Object Detection API to identify custom objects in the environment. The information from the detected object in the camera image is used to calibrate the distance detection algorithm. The relative position of the objects is then used as input data to the trilateration-based localization algorithm. Examples of new objects used as landmarks in our system include a chair, sofa, fridge, shelf, and coffee table. To train the model with custom objects, a dataset of 100 images were collected by taking photos with a laptop camera. These images were randomly separated into two partitions: a train partition with 90 images and a test partition with 10 images. These objects were then tested in experimental work. The paper provides results and discusses shortcomings and future work.

## Keywords

Localization, Trilateration, Vision, Learning.

## 1. INTRODUCTION

The goal of this work is to develop a robot trilateration-based localization algorithm by applying deep learning for the recognition of new landmarks in an environment. The determination of robot location based on its distance from three landmarks is known as trilateration [1]. This work is part of the more general area of Simultaneous Localization and Mapping (SLAM) for mapping an unknown environment while simultaneously keeping track of the robot location within it [2]. SLAM problems analyze whether it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. Both the trajectory of the robot and the location of all landmarks are predicted with no prior knowledge of the location nor the environment. With the remarkable success of deep learning methods in computer vision, there is a growing inclination of using deep learning approaches to visual-based SLAM.

## 2. TRILATERATION

Trilateration methods compute (*x,y*) positions using distances to landmarks [3], as shown in Figure 1. Trilateration calculates robot location from measured distances to at least 3 landmarks at known positions by solving a system of equations:

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$
$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

where $r_1$, $r_2$ and $r_3$ correspond to the three distances from the camera to the reference points.

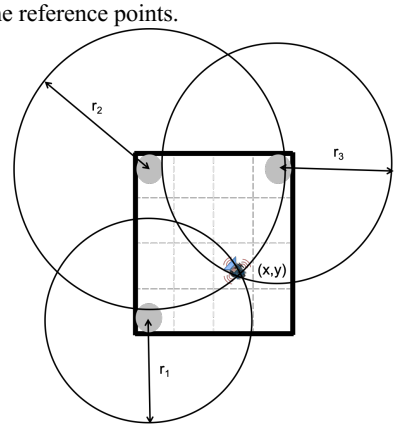


Figure 1. Trilateration algorithm based on the intersection of 3 circles formed by the distances $r_1$, $r_2$, $r_3$, of a camera to 3 known landmarks.

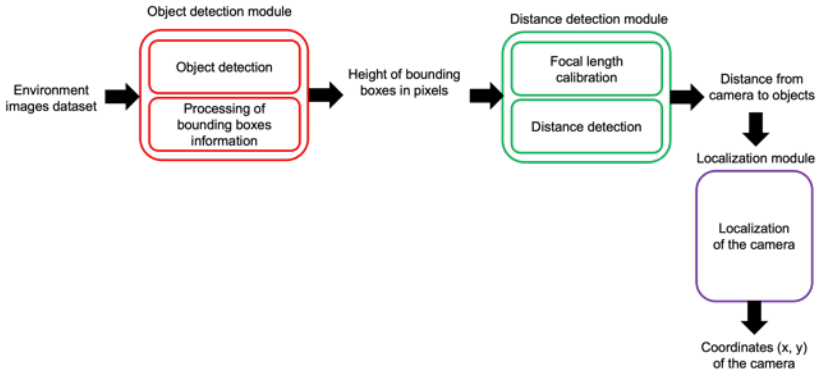


Figure 2. System Architecture containing an object detection module, distance detection module, and localization module.

## 3. SYSTEM ARCHITECTURE

The system architecture is composed of three main modules: (a) object detection module, (b) distance detection module, and (c) the localization module, as shown in Figure 2. Figure 3 shows a flowchart to compute the trilateration-based localization algorithm.

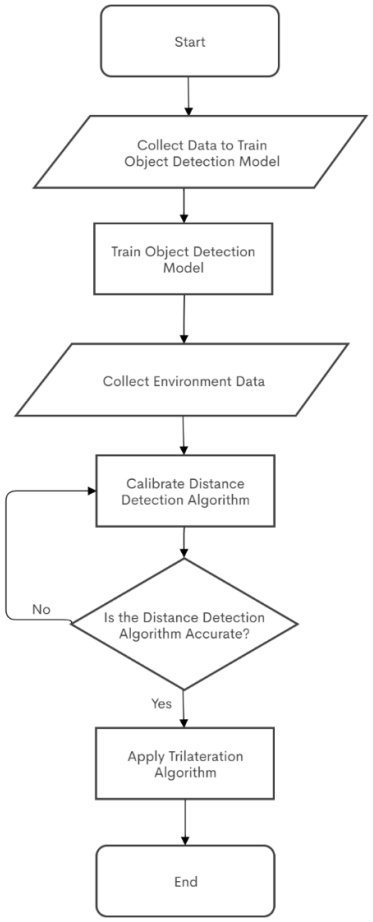


Figure 3. Trilateration-based localization algorithm.

### 3.1 Object Detection Module

The object detection module involves the following processes: (a) collection of data (image collection and annotation), (b) training and evaluation, and (c) object detection.

#### *3.1.1 Collection of data*

We use TensorFlow's Object Detection API [4] to construct, train and implement object detection. Model Zoo [5] was used, pre-trained on the COCO [6], KITTI [7], and Open Images [8] datasets to detect a number of generic objects. The architecture used for initializing the model was the SSD MobileNet V2 FPNLite 320x320 [9]. The images used for the training and tunning were collected and labeled, to distinguish between similar objects, such as two different chairs and two different tables.

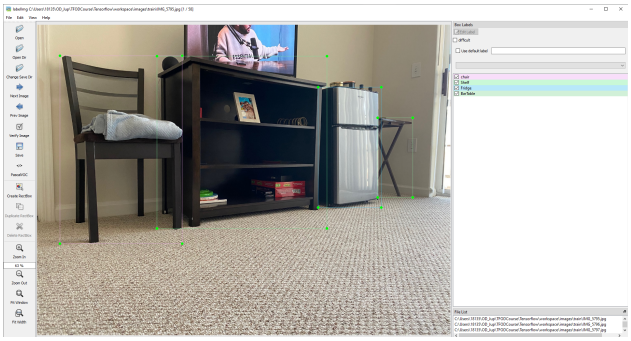

Figure 4. LabelImg's interface. An image of the train partition is being annotated using the LabelImg library.

Images were labeled using LabelImg [10], a graphical image annotation tool. The annotations were saved as XML files in PASCAL VOC format, to be able to train the model. Figure 4 shows LabelImg's interface of an image in the train partition being labeled.

To train the model for custom objects, a dataset of 100 images was collected by taking the photos with a laptop's camera, including pictures taken during the day and night, and to include pictures with different lighting in the room. These images were then manually and randomly separated into two partitions: a train partition with 90 images and a test partition with 10 images.

After testing the model with the 10 images in the test partition, approximately 5 images were taken with a cell phone's camera to continue testing the model under different conditions. This yielded

very imprecise results due to the difference between the laptop camera images and the images taken with the cell phone.

Figure 5 shows two different pictures of the environment taken with the two different cameras.

Figure 5. Comparison between pictures taken with different cameras. (left) shows a picture of the environment taken with a laptop's camera, and (right) shows a picture of the same environment at the same time of the day taken with a cellphone's camera.

### *3.1.2 Model Training and Evaluation*

After the data was collected, the model was trained making use of the images collected and the annotation files. Figure 6 shows sample output of the object detection model in the environment used for the tests.

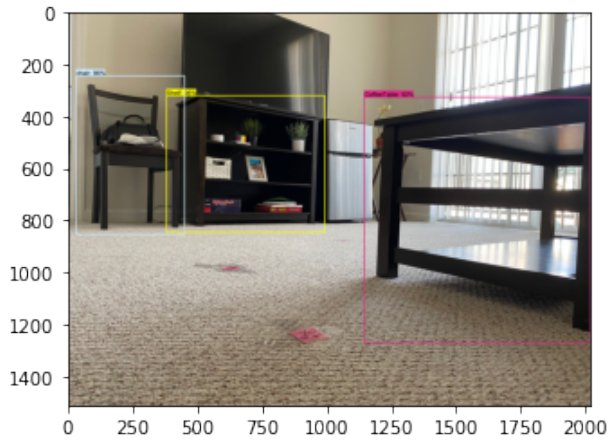


Figure 6. The image shows sample results of the object detection system used to detect different objects.

Initially, the model was tested with 10 images in the test partition, which were taken with the same camera as the images in the training partition and showed satisfactory results. However, when the model was tested with new pictures taken from a different camera, it revealed inadequate results, showing both false negative and false positive results in some of the pictures. Figure 7 shows an example of both the case of false positives and false negatives. The yellow bounding box on the right of the image shows how a false positive would look in an image tested with the object detection model. On the left of the picture, there are two objects (a fridge and a table) that the model did not recognize. The model was trained for 2000 steps using the first 90 images with their respective annotations.

Performance tuning was performed on the model by adding 25 new pictures taken from a different camera and the model was trained for another 2000 steps to improve average precision and recall and to have a more generalized model. This newly trained model showed better results in terms of the number of false negatives and false positives. The results of the evaluation yielded an average precision of 82% and an average recall of 73%.

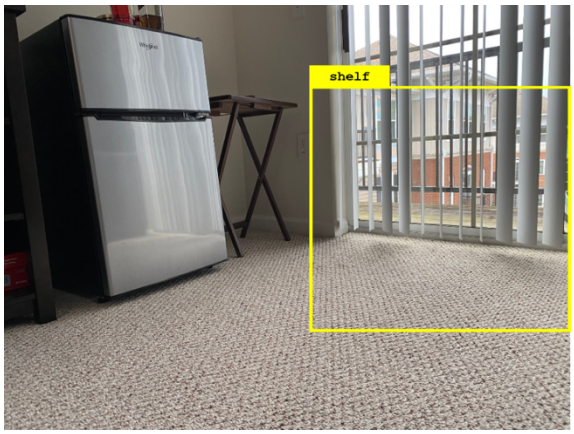


Figure 7. Example of a false positive and two false negative results. On the left of the image, two objects (a fridge and a small table) that the model should recognize. On the right of the image, a false positive highlighted by the yellow bounding box.

### *3.1.3 Bounding Box Information*

During object detection, an important factor to consider is the precision of the size in pixels of the bounding boxes of the objects detected in the images. The accuracy of the bounding box size with respect to the size of the detected object is important since the size of the bounding box, in particular its height in pixels, will be passed as input information to the distance detection algorithm.

The object bounding box is described by four coordinates: *xmin, ymin, xmax, ymax.* These coordinates represent the bottom-left and top-right coordinates in the image, as shown in Figure 8.

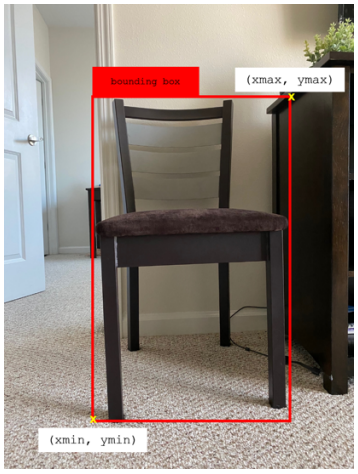


Figure 8. Bounding box showing coordinates *(xmin, ymin)* and *(xmax, ymax).*

Only the height (or width) of an object is needed to calibrate the distance detection algorithm. We decided that the height was going to be used since it is perspective independent as opposed to the width, that can change significantly from one point of view to another. We compute the height in pixels of the bounding box of object $i$ by:

$$p_i = (ymax_i * image\ height\ in\ pixels) - (ymin_i * image\ height\ in\ pixels)$$

where $ymax_i$ is the maximum $y$ of object $i$, $ymin_i$ is the minimum $y$ of object $i$, and $p_i$ is the height in pixels of the bounding box that surrounds object $i$.

Figure 9 shows a comparison between (a) an image (purple rectangle) whose bounding box is accurate with respect to the size of the object, and (b) an image whose bounding box is of an imprecise size with respect to the detected object.

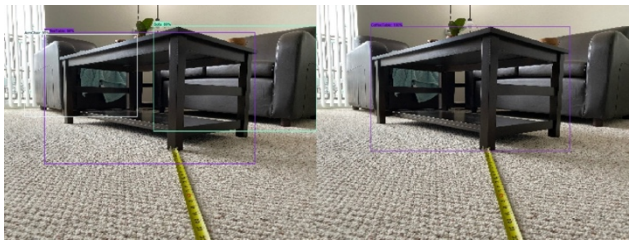

Figure 9. Comparison between two images and their bounding boxes. (Left) the bounding box of the sofa table (purple color) is imprecise with respect to the size of the object. (Right) the bounding box surrounds the sofa table more precisely.

Altogether, the trained objects in the environment for the tests included: (a) armchair, (b) shelf, (c) mini fridge, (d) small table, (e) sofa table, (f) small sofa, and (g) large sofa, as shown in Figure 10.


Figure 10. Detected objects for the environment.

## 3.2 Distance Detection Module

We used for analysis a closed indoor environment of 161" by 169". The coordinates for the objects in the environment were specified using the center of each object. These coordinates were later used as input data to the trilateration-based localization algorithm together with the output distances from the distance detection algorithm. These objects were then used as landmarks or reference points for the localization algorithm. Figure 11 depicts an approximate 3D model of the configuration of the environment. Figure 12 shows the corresponding 2D configuration.

Figure 11. A 3D model of the environment where the localization algorithm is tested.

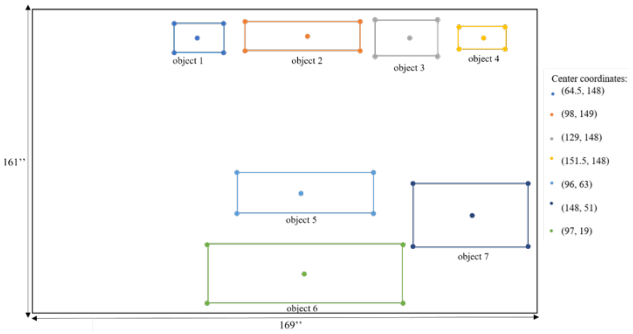


Figure 12. A 2D diagram of the environment. The points inside the objects represent the center coordinate of each object that was collected for the localization algorithm.

The focal lengths to calibrate the distance detection algorithm with respect to each object in the environment were calculated. We used two parameters, the height (or width) of the object and the distance to the camera, as shown in Figure 13. The focal length of object $i$ was defined by:

$$f_i = \frac{(p_i * d_i)}{h_i}$$

where $h_i$ is the object $i$ known height in inches, $p_i$ is the height in pixels of the bounding boxes that surrounds the object $i$, and $d_i$ is the internal distance from the camera to the object $i$.


Figure 13. Focal length for each object $f_i$ is computed from object height $h_i$, pixel height $p_i$, and distance $d_i$ to the laptop camera.

The focal lengths of the objects were calculated multiple times using several images from different perspectives and different distances to get compute a more generalized focal length for each

object. Then the average of all the focal lengths for an object was calculated. The focal lengths of all objects were then adjusted, and a unique perceived focal length $f$ of the camera was computed:

$$f = \frac{\sum_{i=1}^{N} ver \frac{(p_i * d_i)}{h_i}}{N}$$

The new focal length was then used to determine the distance to the object from the camera. The distance to object $i$ is computed by:

$$d_i = \frac{(h_i * f)}{p_i}$$

# 4. TESTS AND RESULTS

Test were carried out to analyze the accuracy of distance detection and the localization algorithm. All software modules were written in Python and gathered in Jupyter notebooks. CUDA and cuDNN software were used for GPU acceleration, using an NVIDIA GeForce GTX 1660 Ti, training at an approximate of 0.132 seconds per step.

## 4.1 Distance Detection

Distance detection tests were performed between the camera and the multiple objects. Figure 14 shows the results of the distance measurement of the camera moving from 40 to 100 inches to each one of the objects in the environment in ranges of 5 inches, for a total of 13 images for each object.

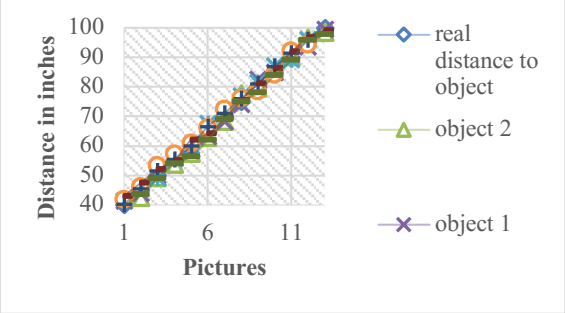


Figure 14. Results of the distance detection algorithm.

## 4.2 Localization

To test the results of the localization algorithm, images were taken from several previously selected points, without the camera following any specific path. If the object detection model did not detect at least 3 known objects, it was not possible to perform the localization. In such case, the camera was rotated, and a new image was taken from the same point until the image detected at least 3 known objects. In the case that more than 3 known objects were detected in the image, the first three objects, ordered from largest image size to smallest image size with respect to pixel height, were selected to perform the localization.

The results of the localization algorithm varied depending on the setup of the environment and the specific objects and positions detected by the algorithm. Large trilateration errors arose, as shown in Figure 15, due to the inaccuracy in measuring distances and mapping reference points, influenced by the geometrical arrangement of the reference points and the objects. Additionally, the solution provided for the trilateration was an approximation of the intersection of the three circles. The results had an approximate average localization error of 72.37782 inches, corresponding to an average error of 2.25%.

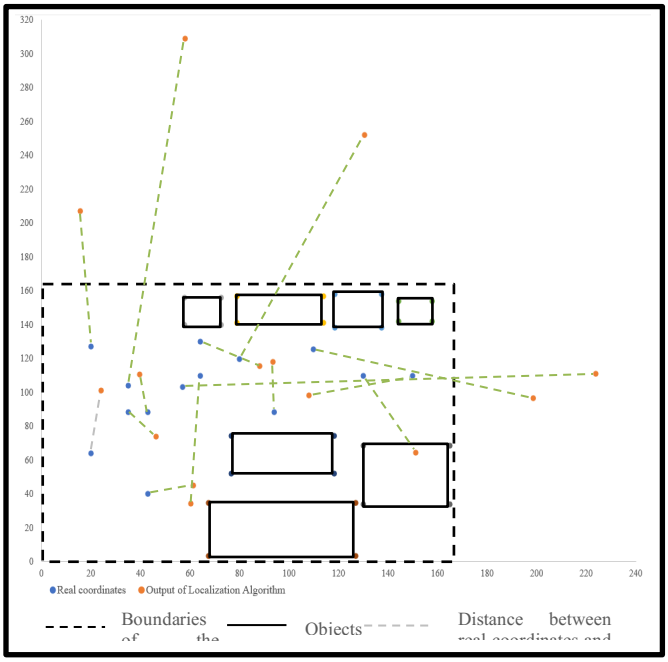


Figure 15. Results of trilateration algorithm before distance modification.

To improve the results, we added to the calculated distances the average of the sum of the distance from the center of the object to a horizontal point, a vertical point, and a diagonal point, as shown in Figure 16. This resulted in the calculated distances being more accurate.

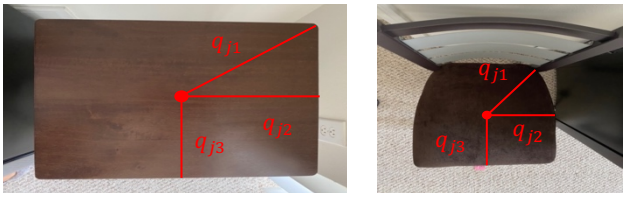


Figure 16. Modified distance calculation, where $q_{j1}$, $q_{j2}$ and $q_{j3}$ represent the distances from the center point to points in the outside object, one point in the vertical line, parallel to the center of the object and one point in the horizontal line, parallel to the center of the object.

Twenty-one $q_j$ values were measured, three for each of the seven objects. The value added to each distance is given by:

$$d'_j = d_j + q_j$$

where $d'_j$ represents the new distance, $d_j$ refers the original distance, and $q_j$ corresponds to the $q$ value of the respective object,

$$q_j = \frac{(q_{j1} + q_{j2} + q_{j3})}{3}$$

where the values of $q_{j1}$, $q_{j2}$ and $q_{j3}$ are measured beforehand in each one of the objects and added to the distance measurements coming from the distance detection algorithm.

The results improved the accuracy of the localization algorithm to an approximate average error 4.075916 inches, reducing the average error to 0.13%.

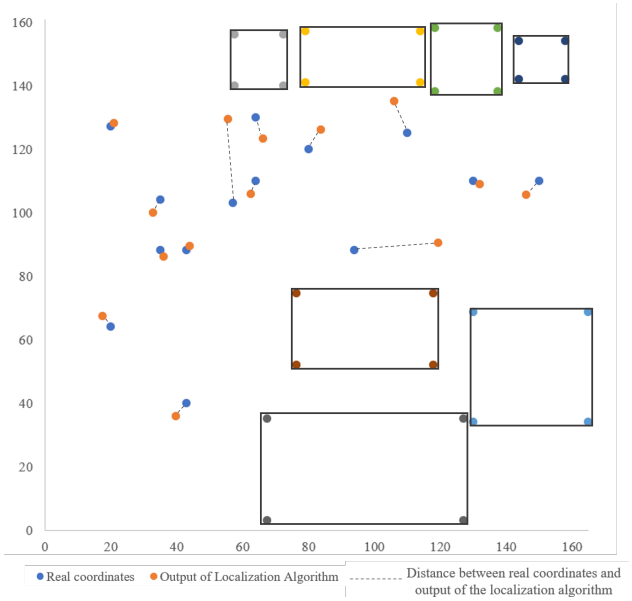


Figure 17. Results of trilateration algorithm after distance modification.

## 5. CONCLUSIONS AND DISCUSSION

The paper presented a trilateration-based localization algorithm based on visual information from a single camera to a set of new 7 objects in the environment with known positions. These objects were trained into the object detection system, requiring at least 3 objects to be detected for localization. The trilateration algorithm was tested with a laptop moved around the physical environment.

Results showed that the algorithm is particularly sensitive to distance measurement. To improve localization, modifications to object distance computations were included, resulting in more accurate results.

In the future we plan to test the algorithm using one of our Raspberry Pi robots available in the BioRobotics Lab at USF. We also plan to use a stereo camara to avoid calibrating the distance detection algorithm and compare results with other algorithms such as Particle Filters and the Extended Kalman Filters.

## 6. ACKNOWLEDGMENTS

This work was funded in part by NSF IIS Robust Intelligence research collaboration grant #1703225 at the University of South Florida entitled “Experimental and Robotics Investigations of Multi-Scale Spatial Memory Consolidation in Complex Environments”.

## 7. REFERENCES

[1] Wenfei Liu and Yu Zhou, "Recovering the position and orientation of a mobile robot from a single image of identified landmarks," 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (2007) pp. 1065-1070, https://doi.org/10.1109/IROS.2007.4399059.

[2] Stachniss C., Leonard J.J., Thrun S. “Simultaneous Localization and Mapping”. In: Siciliano B., Khatib O. (eds) Springer Handbook of Robotics. Springer Handbooks. Springer, Cham. (2016). https://doi.org/10.1007/978-3-319-32552-1_46.

[3] Zhou Yu, Liu Wenfei, Lu Xionghui and Zhong Xu. “Single-Camera Trilateration”. Applied Sciences. (2019): 9(24):5374. https://doi.org/10.3390/app9245374.

[4] TensorFlow. “Models/Research/Object_detection at Master Tensorflow/Models.” GitHub, Google, 10 May 2021, github.com/tensorflow/models/tree/master/research/object_detection.

[5] Model Zoo (2022). https://modelzoo.co/

[6] Lin, Tsung-Yi, et al. "Microsoft COCO: Common objects in context." European conference on computer vision. Springer, Cham, 2014.

[7] Geiger, Andreas, et al. "Vision meets robotics: The KITTI dataset." The International Journal of Robotics Research 32.11 (2013): 1231-1237.

[8] Kuznetsova, Alina et al. “The Open Images Dataset.” International Journal of Computer Vision 128 (2020): 1956-1981.

[9] H, Yu, et. al. TensorFlow Model Garden (2020). https://github.com/tensorflow/models.

[10] Tzutalin. LabelImg (2015). https://github.com/tzutalin/labelImg.