

Use of Imaging Devices and Machine Learning Software to Assist in Autonomous Vehicle Path Planning

Final Report

Team 3

SmartAg: Client

Joseph Zambreno: Advisor

Souparni Agnihotri: Meeting Facilitator

Ashley Dvorsky: Document Manager

Eric Himmelblau: Webmaster

Fahmida Joyti: Test Master

John Orefice: Communications Lead

Bowen Zhang: Hardware Maintainer

Team Website: sdmay18-03@iastate.edu

Table of Contents

1. Introduction	3
1.1 Problem Statement	3
1.2 Solution	3
2. Requirements	3
2.1 Functional Requirements	3
2.2 Non-Functional Requirements	3
3. Project Design	4
3.1 Object Detection	4
3.2 Distance Determination	4
4. Implementation Details	6
4.1 Object Detection	6
4.2 Distance Determination	7
5. Testing Process and Results	8
5.1 Object Recognition	8
5.1.1 Implementation	8
5.1.2 Results	9
5.2 Distance Measurement	11
5.2.1 Implementation	11
5.2.2 Results	11
5.3 Integration Testing	12
5.3.1 Implementation	12
5.3.2 Results	12
6. Related Products and Literature	13
6.1 Autonomous tractor vehicles developed by others	13
6.2 Computer Vision Literature	14
Appendix I: Operation Manual	14
A1.1 Object Detection:	14
A1.1.1 To re-train the neural network:	14
A1.1.2 Run training and testing	16
A1.4 Undistorting Images	16
A1.3 Required Hardware	17
A1.4 Required Software	17
A1.5 Using the integrated system	17
	1

Appendix II - Alternative Designs	17
Appendix III - Other Considerations	18
Appendix IV - Code	19
A.4.1 PairXML.mat	19
A.4.2 xml_to_csv.py	20
References	21

1. Introduction

1.1 Problem Statement

Our client, SmartAg, is creating an autonomous tractor that works solely off of GPS coordinates and is guided by a preset path planning algorithm. The coordinates of obstacles must be set manually for every new farm and do not automatically update to changes in the environment. This can cause issues when new fences are added permanently to the fields, or when people, farm animals, or other farming equipment might be out in the territory.

1.2 Solution

We utilized a convolutional neural network in conjunction with stereo cameras to detect obstacles and measure their distance from the tractor in real time. Thus tractor's path can be adjusted in actual time to avoid possible collisions. We focused on fences, people, tractors, combines, and farm animals to detect these obstacles on fields and report how far away they assuming they are staying stagnant and not moving towards or away from us.

2. Requirements

2.1 Functional Requirements

1. The image processing system shall be able to detect objects such as fences, ditches, and terraces in real time using an appropriate Neural Network.
2. Use depth determination techniques to find how far away an object is so that the tractor can successfully circumvent them.
3. Return data that can be used to add object positions to the path planning map so that they can be avoided in the future.

2.2 Non-Functional Requirements

1. The speed of real-time object detection system should be greater than or equal to 15 frames per second
2. The system must fit into and be compatible with any late model tractor
3. In no way should the system prevent or inhibit manual driving, there should always be a safety override to stop the tractor.
4. The system must be powered by the tractor electrical system and not require any additional power than the system can provide for the whole operation.

5. This product needs to be easy to use by the target audience (farmers) who have little experience with autonomous machinery and neural networks.

3. Project Design

3.1 Object Detection

The object detection system has responsibilities of correctly localizing and identifying multiple objects of interest from a real time video feed and send the identified objects to the distance determination system. The core of the object detection system relies on the MobileNet SSD model chosen from the TensorFlow Object Detection API framework which is an open source framework built on top of TensorFlow that makes it easy to construct, train, and deploy object detection models.

As mentioned earlier, the system is built around the MobileNet SSD model, which has been re-trained to detect five objects of interest - fences, humans, farm animals, tractors and combines, as per the request of our client. MobileNet SSD V1 was chosen as our preferred model as its running time in milliseconds per 600X600 image was 30ms on a Nvidia GeForce GTX TitanX card. Further research showed that the detector performance of the model on the COCO dataset, as measured by the mAP score was 21.

The basic layout of our object detection system consists of reading in the real-time video feed using openCV and passing it into the trained neural network that detects the objects in each frame of the video. Once the bounding boxes are drawn on the objects detected, the bottom left and top right coordinates of the bounding boxes - xmin, ymin and xmax and ymax are then passed into the distance determination system, which computes the distance as described below.

3.2 Distance Determination

The final design of the distance determination system consists of two cameras working in tandem to calculate the distance and angle to an object of interest. A general flow of information through the system can be seen in **Figure 1** below.

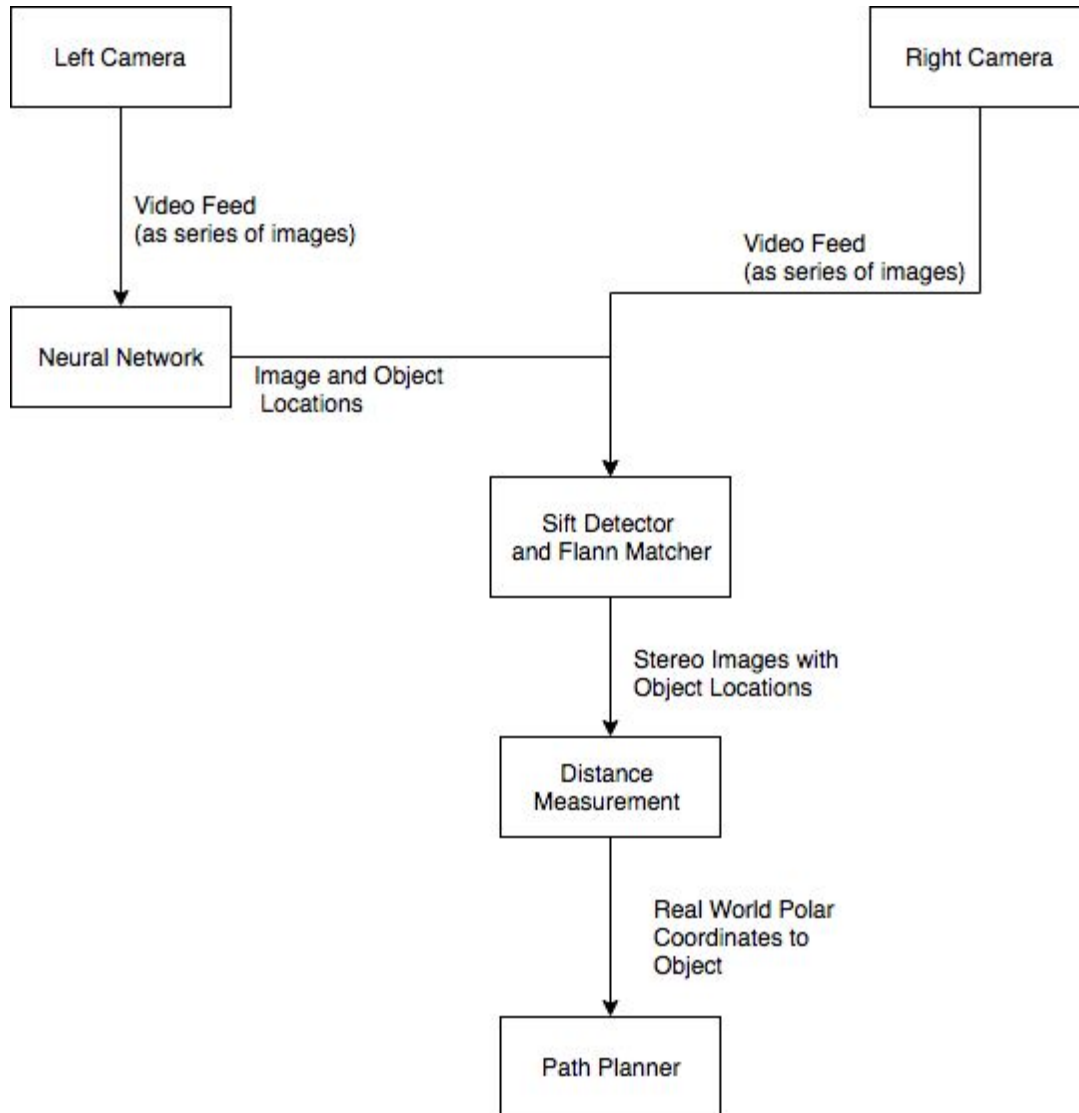


Figure 1:
Data Flow through Distance Measurement System

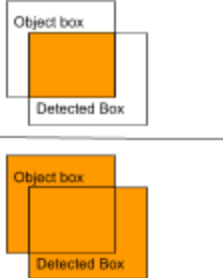
The neural network provides initial object detection one of the video feeds, and feature matching is used to find the matching objects in the second video feed. From here, camera intrinsics and the disparity between the two images are used to calculate the distance and angle to the object. This information, along with the object's class as detected by the neural net, are provided to the path planning algorithm.

4. Implementation Details

4.1 Object Detection

To intelligently detect objects in an image we opted to use Mobilenet SSD v1 from the list of COCO trained models in Tensorflow's Model Zoo [1]. This was a tradeoff between the speed of the network and its mean average precision (mAP). Mean average precision for COCO is defined below.

When training, we look at the known bounding box for a training image and the bounding box for a class detection on that image. It is common to assume that an object prediction is correct (a true positive) if it's area of intersection over its area of union (IoU) is greater than 0.5.

$$IoU = \frac{\text{Area of intersection}}{\text{Area of union}} = \frac{\text{Object box} \cap \text{Detected Box}}{\text{Object box} \cup \text{Detected Box}}$$


$$TP(c) = \text{Number of true positives for each class} = \text{detections with } IoU > 0.5$$
$$FP(c) = \text{Number of false positives for each class} = \text{detections with } IoU \leq 0.5$$

$$Precision = \frac{TP}{TP + FP}$$

$$\text{Mean Average Precision (mAP)} = \frac{1}{|\text{classes}|} \sum_{c \in \text{classes}} \frac{TP(c)}{TP(c) + FP(c)}$$

Equation 1: Mean Average Precision

For COCO trained models, mean average precision (mAP) is the number of true positives divided by total positives averaged over each class. This is then augmented by finding the average of mAP over 10 IoU thresholds from 0.5 to 0.95. This gives a good estimate of model accuracy in a single statistic. For MobilenetSSD v1, mAP is 21. [2][3] This is lower than other models in the zoo, but it is also among the fastest of the models. When dealing with a live video feed, speed was very important to our project goals as a proof of concept. Should our client decide to develop on our work, we recommend a network with a higher mAP, as accuracy when controlling a 35,000 lb tractor is of vital importance.

Once our neural network model was settled upon, we were tasked with finding enough images to train the model with our objects of interest. Fortunately, the ImageNet database had both images and annotations for the objects we wanted to find: fences, farm animals, humans, tractors, and combines. We used 2,200 images to train the set of 5 classes.. 20 images from each class were used as a training set to calculate our class accuracy.

Once the neural network was properly trained and we obtained a loss of approximately 0.8, we decided to test the neural network on real time video feed. There seemed to be some misclassifications and some overfitting but overall, the system seemed to perform pretty well. The object detection code was then integrated with the distance determination, which will be described below.

4.2 Distance Determination

For stereoscopic distance calculations, the location of an object must be known in both images. Unfortunately, the aforementioned steps are very computationally taxing, meaning that it is infeasible to have the neural network detecting objects for both video feeds if we hope to have real time performance. Additionally, the neural network approach we are only provided with a bounding box, rather than accurate pixel coordinates. This is problematic since the distance calculation is heavily reliant on the disparity between pixels of interest between the two camera feeds.

Fortunately, OpenCV provides a single solution to both of these problems. The SIFT^[4] (Scale Invariant Feature Transform) algorithm identifies useful features in a grayscale image. Paired with a FLANN (Fast Library for Approximating Nearest Neighbors) matcher, we can efficiently compare and match the features in two images. Two regions which have a high number of matching features are likely to be matching sections of stereo images.

While this is an improvement over running the neural network two separate times, there are still improvements that can be made thanks to the use of the neural network on the first video feed and the fact that we are working with stereo images. A useful feature of stereo images is that one image is simply a horizontal shift of the other. Additionally, the neural network will provide us an AxB pixel window that encapsulates the object of interest. Using these together, we are able to perform a horizontal search across the second image using the method explained above. By finding the window with the most matches, we have not only found the object in the second picture, but we have also obtained numerous matching features in both images which will provide more accuracy than simply comparing the bounding boxes. An example of this matching process can be seen in **Image 1** below.

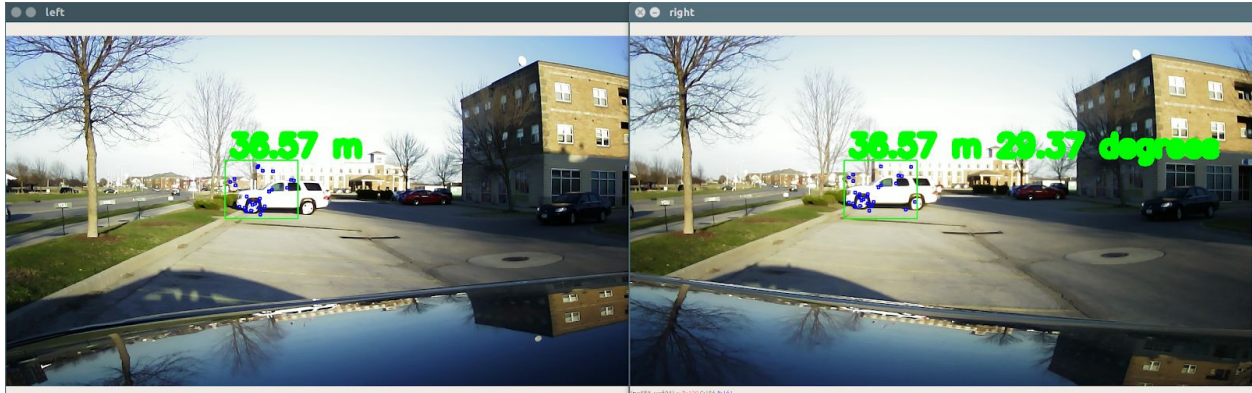


Image 1: SIFT/FLANN Matching

Now that a set of matching features between the two images has been obtained, we are able to combine this with the camera intrinsics to calculate the polar location (distance and angle) of the object using **Equation 2** [5][6][7] and **Equation 3** [5][6] below.

$$z = \frac{fb}{x_L - x_R}$$

Equation 2: Stereo Depth Measurement with Focal Length

$$z = \frac{b * x_0}{2 \tan\left(\frac{\varphi}{2}\right)(x_L - x_R)}$$

Equation 3: Stereo Depth Measurement with View Angle

$$\theta = \arctan\left(\frac{2x_1 \tan(\varphi)}{x_0}\right)$$

Equation 4: Stereo Angle Measurement

$$f = \frac{x_0}{2 \tan\left(\frac{\varphi}{2}\right)}$$

Where z represents the distance to the object, f is the focal length of the camera, b is the distance between the two cameras, x_0 is the image width in pixels, x_L and x_R are the x coordinate of the feature in the left and right images respectively, φ is the viewing angle of the camera, and x_1 is the pixel distance of the feature from the center of the image.

5. Testing Process and Results

5.1 Object Recognition

5.1.1 Implementation

To test the neural network we followed common machine learning practice and divided our 2,300 image data set into a testing and training data set. This was divided into a 2,200 image training set and 100 image testing set. The testing images were completely withheld from the training process, so there would not be bias in the test results. We used mAP (Mean Average Precision) as an accuracy metric.

5.1.2 Results

- Training Loss (five objects):

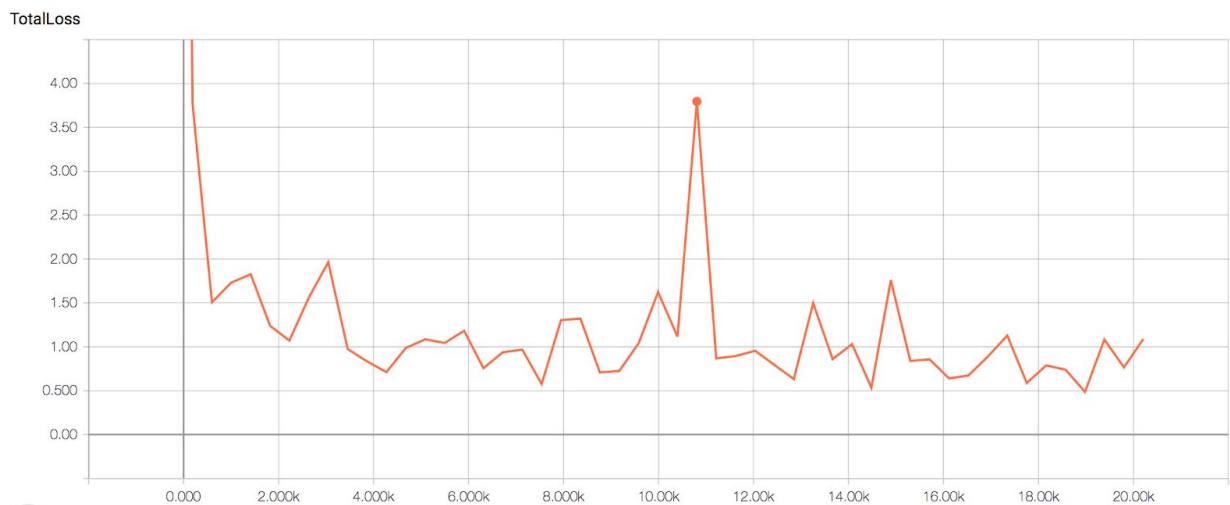


Figure 1: Neural Network Loss

Training loss is the error on the training set of data. In the range of 500~1000 steps, the loss dropped fast. However, after 4000 steps, the loss is almost stays the same.

- Output images:



Image 2: Horses



Image 3: Fence



Image 4: Tractor



Image 5: Combine

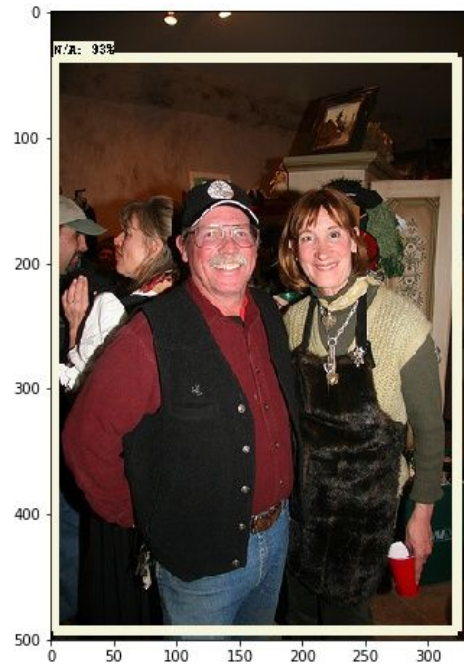


Image 6: Human

TensorFlow uses a dataflow graph to represent the computation in terms of the dependencies between individual operations. It has inputs, different layers, and finally training weights.

5.2 Distance Measurement

5.2.1 Implementation

To test the distance measurement subsystem we first needed to simulate the output of the neural network. To do so, we chose an object of interest and manually created a bounding box around it. In addition, we also manually measured the distance to the object. With this

information, we could then calculate the distance to the object using stereo properties and compare this to the manually measured distance. This process was repeated with various objects and differing distances and with different baseline distances between the cameras.

5.2.2 Results

Baseline Distance (cm)	Actual Distance (m)	Calculated Distance Eq 2 (m)	Calculated Distance Eq 3 (m)	Percent Error Eq 1	Percent Error Eq 2
30	27.43	3.69	3.37	86.5	87.7
30	35.66	3.45	3.15	90.3	91.2
30	46.63	3.27	3.62	93.0	92.2
60	27.43	18.4	16.81	32.9	38.7
60	35.66	28.15	25.73	21.1	27.8
60	46.63	19.91	22.07	57.3	52.7
90	27.43	7.68	7.02	72.0	74.4
90	35.66	5.21	5.78	85.4	83.8
90	46.63	5.33	5.90	88.6	87.4

Table 1: Distance Measurement Test Results

These results are certainly not ideal, however we were not able to find the cause of the large error rates. We have however noticed the trend that a baseline distance of 60 cm seems to be optimal for our test case, yielding an average error of 40%. Additionally, all combinations of equation and baseline distance saw their worst performance at a distance of ~45 m, meaning this is around the feasible limit of our distance calculation.

5.3 Integration Testing

5.3.1 Implementation

Once the two systems had been independently tested, we had to test how well they worked together. To do so, we prerecorded stereo video of our clients tractor driving across a field. We recorded the tractor driving both in a straight line and at an angle with two different baseline distances between the cameras. Using this video data we were able to pass the video feed from the left camera through the neural network to perform the initial object detection, and then calculate the distance and angle to the object.

5.3.2 Results

The testing for integration is not working perfectly because of the qualities of videos we recorded. When recording, we put the cameras on the top of the car, so the hood is inside video, and it reflected another tractor. Also, because the tractor in our video was too small, the object detection system had a hard time recognizing the tractor. But it worked well when the object detection system recognized the tractor, and the system put the the distance in meter in **Image 7**.



Image 7: Distance for the tractor.

6. Related Products and Literature

In developing our object recognition and distance measurement system we had to research into variety of fields such as Computer Vision, Deep Learning, Computational Geometry and other areas. This research can be divided into the categories specified below.

6.1 Autonomous tractor vehicles developed by others

When we researched other products we found some similarities in the Autonomous tractors developed by Case IH and New Holland in the problem being solved and the solutions being implemented[20][21]. The autonomous tractors developed by these two companies use object recognition along with sensors such as radar, LiDar and cameras to detect both stationary and moving farming obstacles in its path. Both our image recognition system and the autonomous tractors developed by these companies rely on live video feed to assist the tractor in detecting

obstacles and assisting in its navigation. However, our project primarily relies on detecting obstacles by using Neural Network while the others mentioned above primarily rely on various sensors to identify obstacles. Hence, our solution potentially could be more cost-effective in the long run. Although our project aims to assist a tractor navigate by recognizing obstacles and sending polar coordinates to its path planner, it could be used irrespective of the tractor model being used.

Likewise, John Deere is collaborating with NASA to design a completely autonomous tractor. This collaboration has been in the works since 2011 and is not yet in production. John Deere is still working on this technology, but is also marketing an auto-steer product. This keeps farmers driving their tractors in a straight line down the fields, but does nothing for object detection. If you were to see a new fence, you would have to manually stop and manually add that fence into the tractor's field database.

There are also some similar products currently in the works outside of the agricultural field. General Motors, Tesla and Waymo (Google) are all currently in the testing phase for fully autonomous motor vehicles with real time obstacle detection. These products will primarily be driven on roads and not used for agricultural harvesting and production. Tesla currently has cars on the market that are semi-autonomous where the car can control steering and speed under certain conditions, but the driver is still in charge of driving the vehicle. These related products show that we are tackling a very complex and difficult problem as some high tech companies are still working on the idea of autonomous vehicles.

6.2 Computer Vision Literature

Both object detection and distance measurement are open areas of study and research in computer science. Thanks to this, there was a wealth of literature available to assist us in this project. At the beginning of our research phase we read a plethora of papers [13][14][15][16][17][18] comparing various neural networks together. Ultimately, combined with our own testing, this information led us to the final decision of using Mobilenet SSD over its competitors. In addition to literature comparing neural networks, we took advantage of Stanford's free course, CS 231n [19] to gain an understanding of the fundamentals of neural networks which was the source of a definite knowledge gap on our team.

In regard to distance measurement, we read numerous research papers documenting the concept of both mono[10][11][12] and stereo [6][7][8] vision solutions to distance measurement. Although there was literature to support both methods as viable solutions, the stereo method was uniform across different sources while the monovision methods varied across sources. Due to this, we felt that the stereo method seemed to be more generally accepted, so we opted to use stereo calculations in our project. Fortunately, most of these papers are written in the context of robotics and autonomous systems, so the concepts could be directly translated to our project.

Appendix I: Operation Manual

A1.1 Object Detection:

A1.1.1 To re-train the neural network:

The project works well on iOS or linux operating systems. The machine should have the anaconda environment installed as well as tensorflow, openCV and pip packages which can be done on the conda environment.

We will need a dataset of images and corresponding annotations. The resource we used for acquiring the data is the imageNet dataset [\[8\]](#).

Part 1:

Download the object detection code from github [\[9\]](#)

Part 2:

After downloading the images and the annotations follow these steps:

1. Match images to xls using the MATLAB code (provided in appendix 4 - pairxml.mat)
2. Separate the images into training and testing folder. A 80-20 approach can be used where 80% of the images and their corresponding annotations in the dataset will be training and the other 20% will be testing.
3. Run command- python xml_to_csv.py (python file is provided in appendix 4)

Part 3:

Navigate to the research folder in the object detection code downloaded - ../models/research

1. Activate tensorflow using this command - source activate tensorflow (on mac or linux) or activate tensorflow (on windows)
2. Export the path - `export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim`
3. Navigate to the object_detection folder
4. Run generate_tfrecord.py
5. Modify:
 - a. `ssd_mobilenet_v1_coco.config` //or the name of your model
 - i. Change the number of classes trained to the number of classes you have
 - ii. The number of examples should be equal to the number of testing image

- iii. Change the number of steps in training to your required number of training steps.
- b. Object-detection.pbtxt
 - i. Add your class labels in the following format depending on your classes like given below:

```
1  item {
2      id: 1
3      name: 'fence'
4  }
5
6  item {
7      id: 2
8      name: 'Animal'
9  }
10
11 item {
12     id: 3
13     name: 'Tractor'
14 }
15
16 item {
17     id: 4
18     name: 'Combine'
19 }
20
```

Part 4:

1. Move all of these files into the object_detection folder:
 - a. From the data folder:
 - i. Test_labels.csv
 - ii. Test.record
 - iii. Train_labels.csv
 - iv. Train.record
 - b. ssd_mobilenet_v1_coco_2017_11_17 //or the corresponding model you trained
 - c. ssd_mobilenet_v1_coco.config
 - d. From the training folder:
 - i. Object-detection.pbtxt
 - e. Folder containing all of your images

Miscellaneous:

1. Run this command if you need to reconfigure protoc
`/local/seniorDesign/models/protoc_3.3/bin/protoc
object_detection/protos/*.proto --python_out=.`

2. Change checkpoint number to restart training from a specific checkpoint.

```
python3 export_inference_graph.py \  
    --input_type image_tensor \  
    --pipeline_config_path training/ssd_mobilenet_v1_coco.config \  
    --trained_checkpoint_prefix training/model.ckpt-8746 \  
    --output_directory training_inference_graph
```

A1.1.2 Run training and testing

1. To start the training run this command from ../models/research/object_detection.

```
python3 train.py --logtostderr --train_dir=training/  
--pipeline_config_path=training/ssd_mobilenet_v1_coco.config
```

Change the .config file to the config file of the model you are using.

2. To test the trained model run the command - python object_detection_tutorial.py

A1.4 Undistorting Images

1. Open MATLAB and select the Image Calibration app
2. Load in at least 15 images of a standard calibration checkerboard for each camera.
3. Export camera intrinsics to text file
 - a. Steps 2-3 are handled by a single MATLAB script we have included
4. Load in camera intrinsics using Python script and apply undistort method
 - a. Handled by 2 Python methods

Note: Distance calculation accuracy decreased when undistortion was applied, and we say little effect on object detection, so we have decided against undistorting images.

A1.3 Required Hardware

- Two identical USB cameras
- Dedicated GPU

A1.4 Required Software

- Python 3
- Pip
- Anaconda environment
- Opencv-Contrib-Python
- Tensorflow
- Tensorflow Object Detection API

A1.5 Using the integrated system

The integrated system will run on any pre-recorded video or live video feed. The system provides distance estimates to obstacles of interest. Knowing the tractor's position from its onboard GPS, and the location of the cameras on the tractor, the coordinates of the obstacle can be calculated. These coordinates can then be used to update SmartAg's path planning algorithm.

Appendix II - Alternative Designs

1. Initially, we considered using other neural network models such as Darkflow and Darknet. However, further research proved that MobileNet SSD provided faster and more accurate results.
2. The Caffe framework was initially used as a backend for training the neural network model. However, due to the various dependencies that Caffe used, we had trouble installing and working with it. We found an alternative version of Mobile Net SSD that worked with Tensorflow as a backend and we were able to implement that.
3. There was always a tradeoff associated with the speed and accuracy of the neural networks used. We tried training the two different neural network models - Faster RCNN inception and Mobile Net SSD. According to the papers we referred, Faster RCNN inception was supposed to be faster and more accurate. However, it proved to be a challenge to set up and run on our computers and resulted in multiple errors and exceptions.
4. We calibrated our cameras to account for distortion in the video feed. However, we get less accurate results for distance measurement when we do the camera calibration. Hence, we opted not to undistort the video feed before performing the distance calculation.
5. For finding matching features in both images we attempted to use MSER(Maximally Stable Extremal Regions). However, the results we obtained were inconclusive and we opted to use SIFT detector to find matching features in both the images.
6. We have considered using Neural Network models such as DepthNet and W-net for doing the distance measurement. Since running two Neural Networks with a single Nvidia TX2 GPU would be very computationally intensive, we opted to use stereo vision instead to perform distance calculation.
7. An alternate design that we considered early on was to calculate the distance off of data from a single camera. To make up for the lack of stereo image data, this requires either using rotational (two images taken at different angles) or translational (two pictures taken as the tractor drives) data to measure distance. Since this is a more dynamic platform we felt that there was a higher likelihood for error with this design.

Appendix III - Other Considerations

As a team we studied training and testing a neural network model, and found that it involves a lot of trial and error. Our group spent a significant time collecting and annotating images in order to train our object recognition model, but we fell short of obtaining good accuracy results most of the time. We had to go back, gather more data and retrain the model multiple times to see how the accuracy gets affected. Fortunately, the model we used for doing the object detection performed data augmentation on the images we passed in and we were eventually able to get the neural network to identify our objects of interest correctly. To prevent overfitting we also tweaked parameters such as the dropout rate which allows a percentage of nodes from a given layer to be removed and “drop out” from the network. We then saw an improvement in terms of how well our model performed on the testing set.

We learnt a lot about TensorFlow and OpenCV while working on this project, and for most of us this was our first experience with the technology. Also, we had a lot of practice with collecting and labelling images. This was also the first time we worked with AWS (Amazon Web Services) and we found that service to be helpful in training our object recognition model when we did not have access to any GPU. Most of the underlying concepts we used to accomplish this project such as Convolutional Neural Network and Scale Invariant Feature Transform were fairly new to us and it took us a significant time to gain an in-depth understanding of these concepts necessary to complete the project. However, once we understood them it was very rewarding to put those concepts into practice.

All in all, we have learnt that it takes a lot of time and effort to make any project truly successful. This project has taught us the value of patience and perseverance and has enforced the essential skills of dedication and optimism to see the project work.

Appendix IV - Code

A.4.1 PairXML.mat

```
1 import os
2 import glob
3 import pandas as pd
4 import xml.etree.ElementTree as ET
5
6
7 def xml_to_csv(path):
8     xml_list = []
9     for xml_file in glob.glob(path + '/*.xml'):
10         tree = ET.parse(xml_file)
11         root = tree.getroot()
12         for member in root.findall('object'):
13             a = root.find('filename').text + '.JPEG' # the filenames need to have extensions.
14             value = (a,
15                     int(root.find('size')[0].text),
16                     int(root.find('size')[1].text),
17                     member[0].text,
18                     int(member[4][0].text),
19                     int(member[4][1].text),
20                     int(member[4][2].text),
21                     int(member[4][3].text)
22                    )
23             # print(root.find('filename').text)
24             xml_list.append(value)
25     column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
26     xml_df = pd.DataFrame(xml_list, columns=column_name)
27     return xml_df
28
29
30 def main():
31     for directory in ['train', 'test']:
32         image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
33         xml_df = xml_to_csv(image_path)
34         xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
35         print('Successfully converted xml to csv.')
36
37
38 main()
```

A.4.2 xml_to_csv.py

```
1 import os
2 import glob
3 import pandas as pd
4 import xml.etree.ElementTree as ET
5
6
7 def xml_to_csv(path):
8     xml_list = []
9     for xml_file in glob.glob(path + '/*.xml'):
10        tree = ET.parse(xml_file)
11        root = tree.getroot()
12        for member in root.findall('object'):
13            a = root.find('filename').text + '.JPEG' # the filenames need to have extensions.
14            value = (a,
15                    int(root.find('size')[0].text),
16                    int(root.find('size')[1].text),
17                    member[0].text,
18                    int(member[4][0].text),
19                    int(member[4][1].text),
20                    int(member[4][2].text),
21                    int(member[4][3].text)
22                )
23            # print(root.find('filename').text)
24            xml_list.append(value)
25        column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
26        xml_df = pd.DataFrame(xml_list, columns=column_name)
27        return xml_df
28
29
30 def main():
31     for directory in ['train', 'test']:
32         image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
33         xml_df = xml_to_csv(image_path)
34         xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
35         print('Successfully converted xml to csv.')
36
37
38 main()
```

References

- [1] rathod , vivek, et al. "Tensorflow/Models." *GitHub*, 3 Mar. 2018
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.
- [2] Arlen, Timothy C. "Understanding the MAP Evaluation Metric for Object Detection." *Medium*, Medium, 1 Mar. 2018,
<https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3>.
- [3] Hui, Jonathan. "MAP (Mean Average Precision) for Object Detection – Jonathan Hui – Medium." *Medium*, Medium, 7 Mar. 2018,
https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.
- [4] "Introduction to SIFT (Scale-Invariant Feature Transform)." *OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)*, 4 Aug. 2017, 04:46:33,
https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html.
- [5] Mahammed, Manaf A, et al. "Object Distance Measurement by Stereo VISION." *International Journal of Science and Applied Information Technology*, vol. 2, no. 2, 11 Mar. 2013, pp. 05–08.,
<https://pdfs.semanticscholar.org/5c62/ab7ff1aa72bd04df10336665d0f7f52f6cd7.pdf>.
- [6] MROVLJE, Jernej, VRANČIĆ, Damir. Distance measuring based on stereoscopic pictures. V: GAŠPERIN, Matej (ur.), PREGELJ, Boštjan (ur.). *Proceedings of the 9th International PhD Workshop on Systems and Control, October 1-3, 2008, Izola, Simonov zaliv, Slovenia : young generation viewpoint*, Ljubljana: Institut Jožef Stefan, 2008, 6 str.
<http://dsc.ijs.si/files/papers/s101%20mrovlje.pdf>
- [7] Rao, Rajesh. "Stereo and 3D Vision." 5 Mar 2009., Computer Vision, University of Washington. *Microsoft PowerPoint* presentation.
<https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf>
- [8] ImageNet. (n.d.). Retrieved from <http://www.image-net.org/synset?wnid=n07942152>
- [9] pkulzc. "Tensorflow Object Detection API", GitHub, 20 Mar. 2018
https://github.com/tensorflow/models/tree/master/research/object_detection

- [10] Alizadeh, Peyman. "Object Distance Measurement Using a Single Camera for Robotic Applications." *Laurentian University Sudbury, Ontario, Canada*, 2015.
https://zone.biblio.laurentian.ca/bitstream/10219/2458/1/Peyman%20Alizadeh%20MSc.%20The%20Corrected_2_2.pdf
- [11] Saxena, Ashutosh, et al. "3-D Depth Reconstruction from a Single Still Image." *International Journal of Computer Vision*, vol. 76, no. 1, 2007, pp. 53–69., doi:10.1007/s11263-007-0071-y.
- [12] Rosebrock, Adrian. "Find Distance from Camera to Object Using Python and OpenCV." *PyImageSearch*, 19 Jan. 2015,
www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/.
- [13] Chuanqi. "MobileNet-SSD, GitHub, 5 Aug. 2017.
<https://github.com/chuanqi305/MobileNet-SSD>
- [14] Pjreddie. "Darknet". GitHub. 1 Oct. 2017.
<https://github.com/pjreddie/darknet>
- [15] Thtrieu. "Darkflow." GitHub, 24 Sept. 2017,
<https://github.com/thtrieu/darkflow>
- [16] Balancap. "Balancap/SSD-Tensorflow." GitHub, 10 Apr. 2017,
<https://github.com/balancap/SSD-Tensorflow>.
- [17] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2016.91
- [18] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2017.6
- [19] Johnson. "Convolutional Neural Networks". Stanford University. 10 May. 2017
- [20]"The New Holland NHDrive Concept Autonomous Tractor Shows a Vision into the Future of Agriculture." NHAG, 23 April 2018
<http://agriculture1.newholland.com/nar/en-us/about-us/whats-up/news-events/2016/new-holland-nh-drive-new-concept-autonomous-tractor>
- [21]"Case IH Unveils New Tagline, New Focus and a New Autonomous Concept Vehicle." BigAg, 23 April 2018
www.bigag.com/topics/equipment/case-ih-unveils-new-autonomous-concept-vehicle/.