SENIOR DESIGN CAPSTONE FINAL REPORT

JUNE 5, 2018

DEPTH SENSING WITH COMPUTER VISION AND LIDAR

PREPARED FOR

OREGON STATE UNIVERSITY

D. KEVIN MCGRATH

PREPARED BY

GROUP 69

KIN-HO LAM

Abstract

Depth Sensing with Computer Vision and LIDAR proposes combining computer vision and LIDAR to create a reliable depth sensor. This document details its project member's progress toward a final design.

1 Table of Contents

Contents

REFERENCES

[1] "Logitech brio webcam with 4k ultra hd video & rightlight 3 with hdr." [Online]. Available: https://www.logitech.com/en-us/product/brio

[2] T. Huang, "Rplidar a1." [Online]. Available: https://www.slamtec.com/en/lidar/a1

[3] "Leddar solid-state lidar: M16 multi-element sensor module." [Online]. Available: https://leddartech.com/modules/m16-multi-element-sensor-module/

[4] M. A. Ibrahim, "How a depth sensor of kinect camera really works," Mar 2018. [Online]. Available: https://masterthesisiseeyouthere.wordpress.com/2014/03/27/how-a-depth-sensor-of-kinect-camera-really-works/

[5] "How radar works." [Online]. Available: http://www.bom.gov.au/australia/radar/about/what_is_radar.shtml

[6] "tensorflow/models." [Online]. Available: https://github.com/tensorflow/models

[7] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and et al., "Speed/accuracy trade-offs for modern convolutional object detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] "Coco - common objects in context." [Online]. Available: http://cocodataset.org/

[9] "openimages/dataset," Jan 2018. [Online]. Available: https://github.com/openimages/dataset

[10] [Online]. Available: http://www.cvlibs.net/datasets/kitti/

[11] "Tensorflow framework & gpu acceleration from nvidia data center." [Online]. Available: https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/tensorflow/

2 DEFINITIONS

IR	IR refers to the infrared light spectrum.
IR Depth Sensor	Device that calculates distances by emitting infrared patterns.
LIDAR	Light Detection And Ranging - Depth sensing technology that uses lasers to measure distance.
Microsoft Kinect	A product that uses an IR Depth sensor to measure distances.
Logitech Brio Webcam	A 4k Logitech web cam. [1]
RPLidar A1	A budget LIDAR device made by Slamtec. [2]
Leddar M16	A solid-state LIDAR device made by Leddar. [3]
Computer Vision	The methods for acquiring, processing, analyzing, and classifying digital images and extracting information.

3 INTRODUCTION

Depth Sensing with Computer Vision and LIDAR aims to create a reliable depth sensing system using computer vision and LIDAR. This project spanned 2018 Winter and Spring terms at Oregon State University. I, Kin-Ho Lam, researched, designed, and built this project in its entirety by myself. Much of this project would not have been possible without my project's sponsor, Kevin McGrath, who supplied equipment and technical guidance.

This project's motivation came from an annual robotics competition at Oregon State University, my client noted that all the robots had been designed and tested indoors. This later proved to be an issue because when the robots were brought outside for a competition, their guidance systems failed to work as the IR cameras were inherently subject to interference from the Sun's ambient light. As the prevalence of automation in our daily lives increases, such as autonomous cars or robots, a cheap and scalable depth sensing system that is resistant to common interferences is necessary. The goal of this project is to create a reliable depth sensing system using computer vision and LIDAR. Such a system that combines these two technologies will be more reliable than infrared depth sensing because cameras and LIDAR devices are less prone to interference from ambient light or anomalies.

3.1 Existing Depth Sensing Technology

3.1.1 Lidar

LIDAR stands for light detection and ranging. LIDAR is a distance finding system that uses a laser to determine distance and angle of objects relative to the sensor. Most LIDAR devices use a low powered 600-1000 nm laser which is mounted on a motor to sweep the laser in a circular motion. A receiver is used to measure the time it takes for the laser to bounce back to the sensor upon encountering an object. Commercial LIDAR systems are relatively low-cost. Their ability to work in varying lighting conditions and ability to detect small objects makes them well suited for use in self-driving cars or autonomous robots. [**?**]

3.1.2 Infrared Depth Sensors

IR, or infrared depth sensors, strobe an infrared pattern on objects in front of it. This infrared pattern is picked up by a receiving camera, which uses the pattern and some basic geometry to calculate distance. While very accurate and cheap, IR depth sensors are not suited for self-driving cars or autonomous robots because they are highly sensitive to light conditions. Two IR depth sensors pointing at the same subject will overlap and confuse each other's sensors. Natural sunlight will also blind the sensor by washing out the IR pattern. [4]

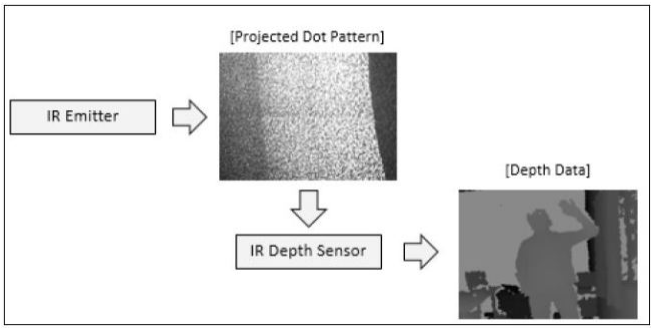


Fig. 1. Kinect depth sensor translating the IR-dot pattern into a 3D representation. [4]

3.1.3 Radar

Radar stands for radio detection and ranging. Radar is an object-detection system that uses radio waves to determine the distance, angle, and velocity of objects relative to the sensor. A radar system consists of a transmitter producing electromagnetic waves, a transmitting antenna, a receiving antenna, and a processing system to interpret the data and determine the locations of objects. Radar works by sending out electromagnetic waves, then measuring the intensity of the reflection. Radar systems are widely used for military applications, airplanes, and weather forecasting. While radar systems have been tested to work extremely reliably in extreme weather conditions, unlike LIDAR, radar cannot detect small objects as easily. This is especially problematic when trying to apply a radar system to an autonomous car system; a radar system may not detect a narrow pole or small object in front of it. [5]

3.2 Project Purpose

Infrared (IR) depth sensors such as the model used in Microsoft's Kinect 1 can quickly calculate distances in indoor scenarios. However, IR depth sensors can be confused by other infrared emitting sources such as other IR depth sensors or natural sunlight. For these reasons, IR depth sensors cannot be used in self-driving cars, outdoor robots, or any any device that requires high accuracy distance measurement in varying conditions. Depth Sensing with Computer Vision and LIDAR proposes combining computer-vision image classification with LIDAR to create a robust and reliable depth sensor.

4 Design

The Logitech Brio webcam provides a high-resolution, two-dimensional image but lacks depth perception. The LIDAR provides accurate depth measurement in a horizontal dimension but lacks vertical perspective. This project proposes bridging the utility of both devices by securing them in stationary positions, then using software to combine their outputs. This involves using the M16 LIDAR to get depth sensing information and using computer vision to recognize objects. The result is a scalable and reliable depth sensor that will not be affected by natural light, and can be further improved by training a better computer vision model or adding more sensors. This project hopes to achieve a proof of concept design to be showcased in a live demo at Oregon State University's 2018 Undergraduate engineering expo. This live demo shall consist of the full system pointed at the project booth's audience.

Figure 2 illustrates different dimensions measured by the M16 LIDAR and Brio Webcam. The red cube represents the Logitech Brio webcam and M16 LIDAR secured in stationary positions. The flat purple triangle represents the M16 LIDAR's horizontal range detection. The transparent green rectangle in front of the person represents the computer vision model recognizing that there is a person in-front of the sensor. The transparent teal pyramid represents the Brio webcam's field-of-view.

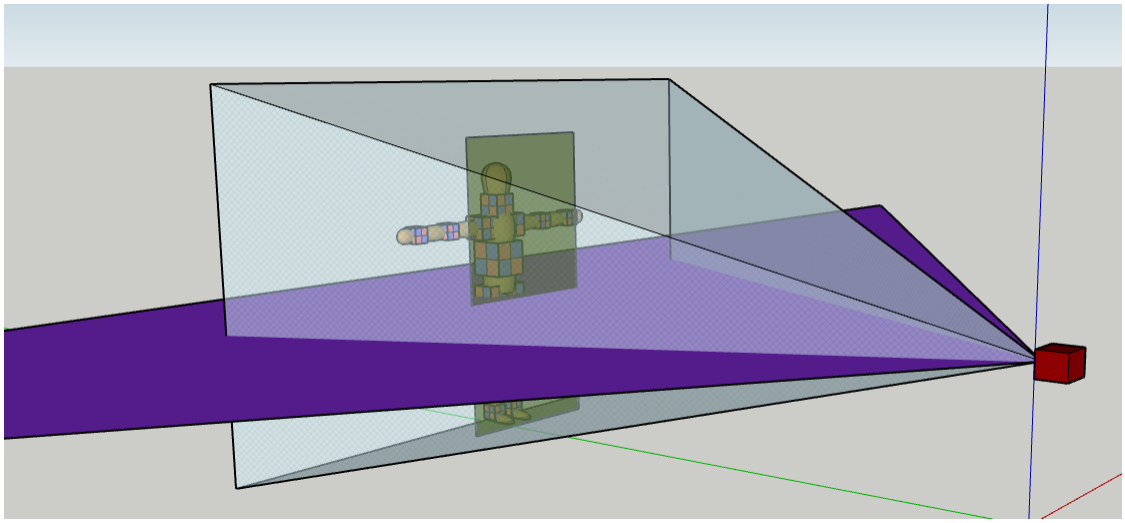


Fig. 2. Visualizing different dimensions measured by the LIDAR and Brio Webcam.

4.1 Physical Mount Design

A stationary mount simplifies the complexity of aligning the different dimensions measured by the LIDAR and webcam. I built a simple wooden mount for this project, joining them together at a 90-degree angle to make a stand as pictured in figure 4.

Fig. 3. The mount with LEDDAR M16.

Originally I built the mount to accommodate the LEDDAR M16 LIDAR, however when it became clear that it would not be ready for expo, I adapted the mount to fit the RPLIDAR A1. This was accomplished by drilling four small holes in the base of the mount to fit the RPLIDAR's plastic standoffs.

Fig. 4. The final mount design with the RPLIDAR A1.

4.2 Computer Vision

I started with OpenCV's pre-trained facial/pedestrian support-vector-machine (SVM) classifier. This SVM is a combination of several other SVMs that detect the upper body, eyes, mouths, and noses. The combined SVM is intended to detect faces with high accuracy. However, when applied to my design, I could not consistently replicate good results. This was due to several factors, namely the SVM used was meant to perform classification on still images where the camera's perspective is far from the subject.

My design specifications envision a system that quickly tracks multiple subjects in a crowded expo scenario. In an expo scenario, human subjects will be moving close or away from the camera, unpredictably shifting their positions, and moving in or out of the field-of-view. As seen in 5, the OpenCV SVM model does not perform to my specification. If the human subject were to turn their head or move too quickly, the SVM will have difficulty tracking their body. Additionally, the SVM performs intensive calculations on the computer's CPU, severely limiting the video output's frame-rate and resolution.

Fig. 5. SVM face classification (Left) fails when subject slightly turns their head (Right)

Recognizing the SVM's weaknesses, Tensorflow's open source object detection classifier presented a better computer-vision alternative. [6] The Tensorflow object recognition library is better suited for this project because its library has already been trained to recognize a large dataset of objects. [7] These pre-trained datasets in Tensorflow's library are sourced from other machine learning datasets including the COCO dataset, Kitti dataset, and the Open Images dataset. [8] [9] [10]

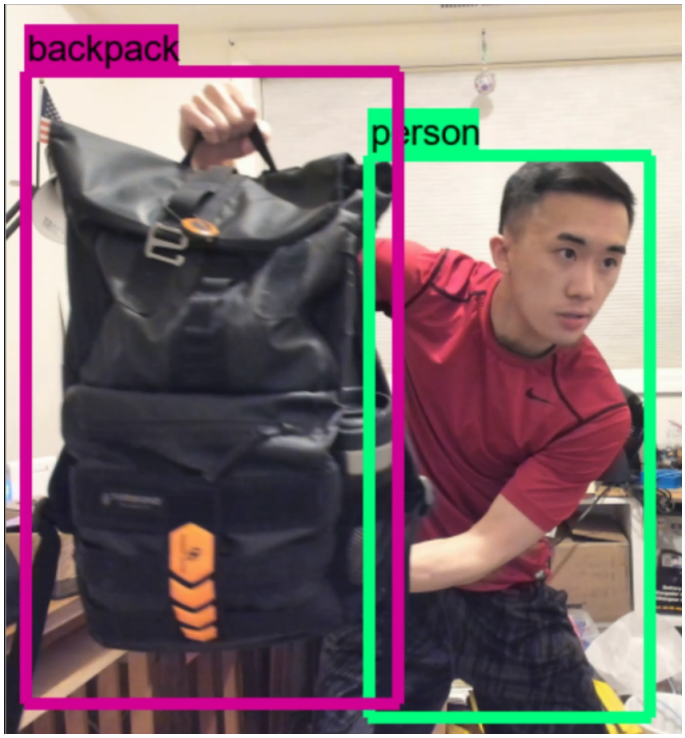


Fig. 6. Pre-trained Tensorflow model detecting multiple subjects.

Using this pre-trained Tensorflow model, my project is now able to accurately outline and label over 90 subjects as they come into view of the webcam. At the end of Winter Term, the state of the code will enable selectively editing the output video frames to draw bounding boxes on subjects as they move in and out of the camera's field of view.

Tensorflow also enables us to take advantage of NVIDIA CUDA, a driver that moves intensive calculations to the GPU. While this increases the list of material requisitions for demo, moving calculations to the GPU greatly improves the output video quality, frame rate, resolution, and classification speed. [11]

5 Final System Design

The final system consists of the RPLIDAR A1 and Logitech Brio web cam mounted as shown in figure 4. During development, it became clear that the RPLIDAR A1 is not an ideal LIDAR device. It is a budget hobbyist device whose cost savings significantly affect performance. This can be seen in the RPLIDAR's limited range. Its sensor/laser assembly fails to reliably detect subjects beyond two meters from the mount. This limits the overall system as subjects must stand within two meters of the system to be properly detected. Additionally, the laser experiences significant loss limiting the point aggregation necessary for proper classification. As a result, subjects that are broad or are holding large flat objects are more consistently detected.

5.1 LIDAR

Due to circumstances beyond my control, I was unable to use the LEDDAR M16 solid state LIDAR for this project and was forced to use the RPLIDAR A1. The M16 LIDAR is preferred over the RPLIDAR A1 because it's a commercial grade LIDAR device that is presumably faster and more versitile than the RPLIDAR A1. The RPLIDAR A1 is not an ideal LIDAR device, fortunatly, because this project is a proof of concept and not intended for commercial use, I was able to work around the RPLIDAR A1's shortcomings. The RPLIDAR A1 has a very limited range, the device consistantly fails to detect objects that are more than 2 meters away. Additionally, the device is fundamentally flawed because it relies on the mechanical rotation of the laser assembly. The Tensorflow computer vision model can read and classify all objects in a single video frame almost instantaniously. The RPLIDAR A1 needs to complete one full rotation before it returns a distance measurement, thus bottlenecking the entire system.

I developed a work-around to compensate for the slow LIDAR device by splitting the RPLIDAR A1 readings into a seperate thread that pushes data into a shared buffer. This allows the Tensorflow model to poll the buffer for new distance data as it needs it. However, this work around doesn't completely solve the rotational bottleneck issue. I observed the system's video output dropping to about 25 frames per second.

Aligning the RPLIDAR A1's laser sweep with the webcam's field of view was a challanging task. Upon detecting an object, the RPLIDAR A1 returns several points of incidence consisting of distance and the object's angle with respect to the tip of the device. Figure 7 illustrates the RPLIDAR A1's measurements as a polar plot. The tip of the device is considered 0^o, this is an important reference because it splits the usable areas into two hemispheres. Through trial and error, I discovered that while in the mount, the RPLIDAR A1's usable area ranged from $0^o - 55^o$ and $0^o - 305^o$, illustrated as a green triangle in figure 7.

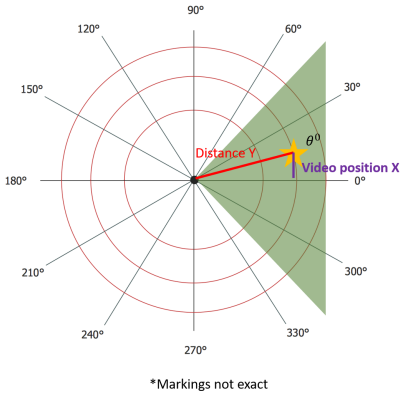


Fig. 7. Polar illustration of the RPLIDAR A1's detection.

One can now use this angle to transpose a point in X-Y dimensions into X-Z video dimensions via similar triangles geometry. Let the yellow star in figure 7 represents a point of incidence that the RPLIDAR A1 detects is at angle θ. One can now use the following equations to translate this point onto the video.

$$\theta \leq 55 : X = \frac{videowidth}{2} * sin(\theta) + \frac{videowidth}{2}$$
$$\theta \geq 305 : X = \frac{videowidth}{2} * sin(360 - \theta) \tag{1}$$

Adding an offset $\frac{videowidth}{2}$ is necessary because the video output is mirrored as shown in figure 8.

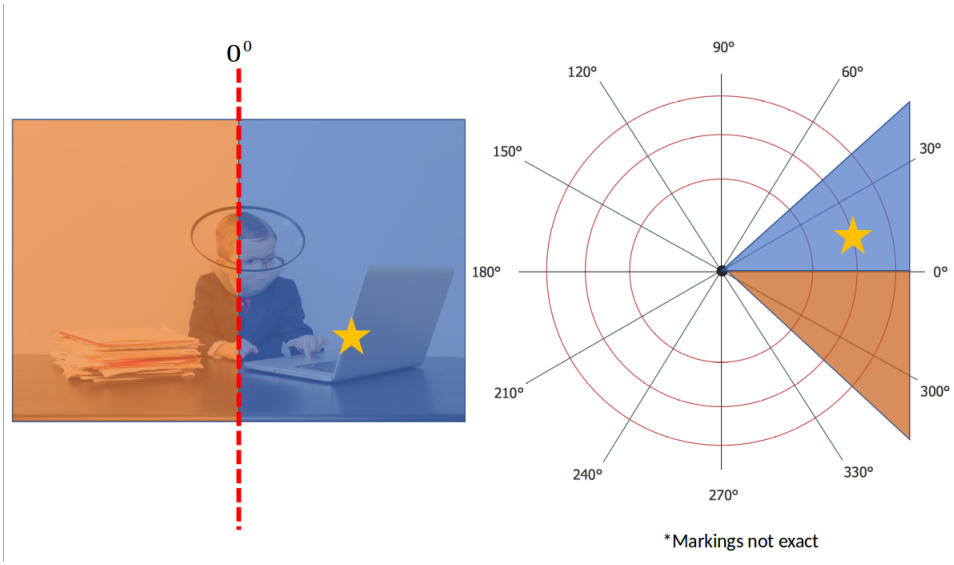


Fig. 8. Flipping the point over 0^o so that it appears on the correct side.

Figures 9, 10, and 11 illustrate a live output of the completed system. Points are read in from the RPLIDAR A1, plotted in a polar graph to visualize what's being detected, then the angles are translated into coordinates on the video and sent to the visualization to display the distances. Occasionally the computer vision model may mis-classify objects as shown in figure 11 where the notebook I am holding is misread to be a chair.

Fig. 9. Two-subject full system test. Successful classification but unsuccessful distance measurement on one subject.

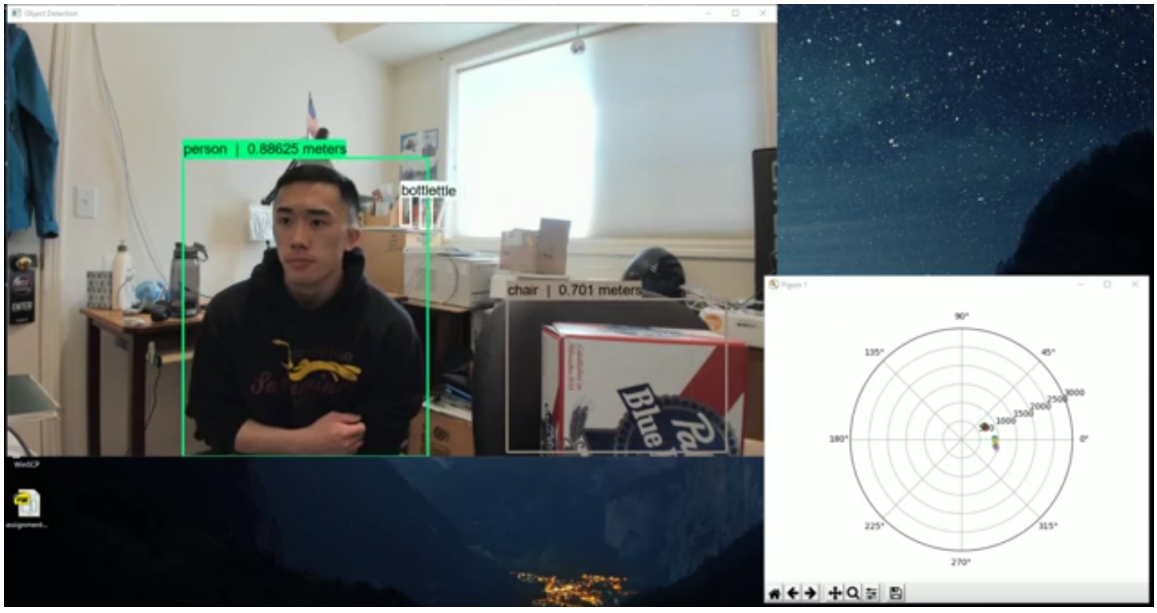


Fig. 10. Two-subject full system test with two stationary subjects.

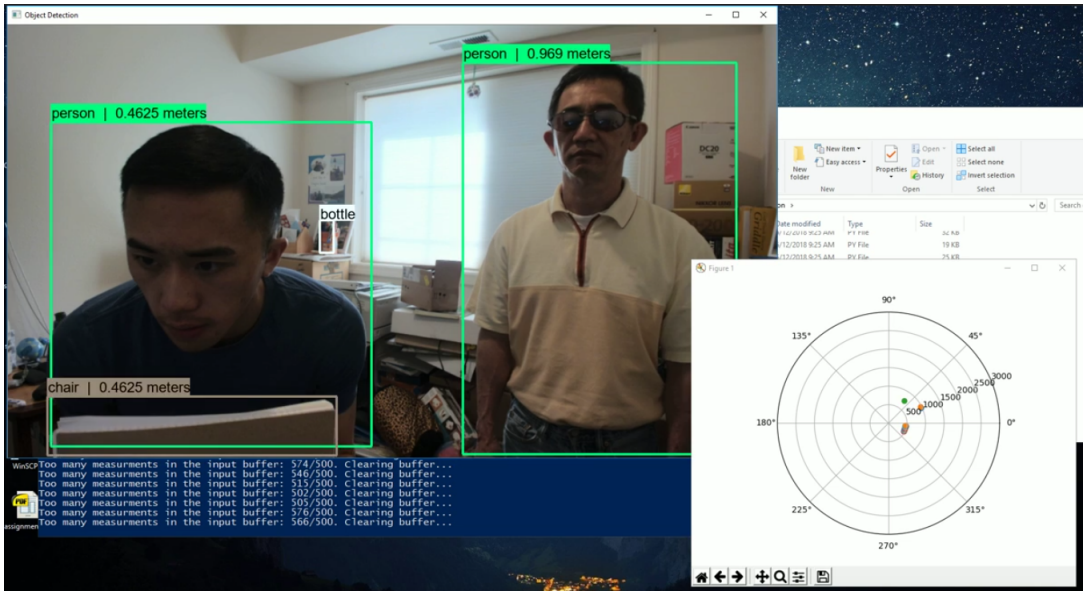


Fig. 11. Two-subject full system test with one stationary subject and one moving subject. One misclassification.

6 Weekly Blog Posts

6.1 Winter 2018

Week	Positives	Deltas	Actions
1	-	Kim says we're moving too slowly compared to previous teams In my opinion making a website is a far simpler task than writing code that performs analysis and aggregates data in a system that is unproven and has never been done before Kim don't want us to work on filtering acceptable images even through that is what we stated we would do in our problem statement All problem statement drafts focused on filtering acceptable images None of our problem statements alluded to focusing on image analysis to interpret aerosol content	OUR TWO OPTIONS PRESENTED TO US: 1) Kim wants us to focus on a system to interpret aerosol content in the atmosphere based on images of the horizon, EXIF data from those images, and data from wunderground 2) (In her view) continue on this image filtering path we have started, overall this current project goal of a filter is not applicable to Aerolyzer I believe we have not gone astray. What I have written is relevant in either option. I still believe we can accomplish 1). Seems like she gave up on us (in my opinion) we are a lower priority given our pace and low confidence in our abilities The code I wrote is still relevant. It will still pick out the sky which is still very necessary, it will still need to be used no matter the option we choose. I am the only one on Github discussing anything about the project. Neither Logan or Daniel have communicated to me that the direction we were going was wrong I don't think Kim understands the scope of the code that is necessary for a filter or why a filter is necessary I don't think she is reading what I'm documenting on our github or how my code is used.
2	Meeting with kevin over webex last Friday transcribed notes May or may not have a partner Usable up to 50 meters, 450 spread Take feed from normal webcam and overlay Leddar point cloud on video With minimum range and maximum range Device: Leddar Lydar M16 Webcam: Logitech Brio webcam Most depth cameras don't work well outdoors, light sensors are overwhelmed by ambient light, overlaying point cloud on webcam video may be a way to resolve this problem	-	Research Leddar Lydar M16. 2D Lydar unit, see if I can overlay point cloud on webcam image. Ideas to try: Research dept/distance sensing fpr

3	Talked with Lucian over email Unfortunately this time won't work for him, I want to keep this slot so we can at least talk, I'll leave it to him to find time to talk with you Gave him a quick rundown of what I understood the project to be He mentioned he was going to write up a problem statement and design document.	Not sure exactly how to help Lucian writing the documents.	Don't need ROS (Robot operating system, don't need to use it) Camera depth sensor doesnt work in sunlight, accompany with LIDAR unit Use realsense as point of comparison
4	Borrowed Logitech brio from Kevin Got it working with VLC on Windows 10, set it as capture device.	Alpha stage of project = visually complete interface Responsive code with placeholders for functionality	Experimenting with Brio webcam and opencv vision models.
5	Interface design? Suggestion: To start, draw red dot on video feed and output Construct internal pipeline first	Lucian and Kevin met in person for our weekly meeting, did not answer webex call so I'm not sure what they discussed.	Sent Kevin my update + screenshots and sketches of my proposed design.
6	Created opencv python script to overlay green box on webcam video feed Successfully overlaid pixels on camera output Output is stuck on 4:3 resolution, going to work with it now and write code to be scalable, I want to increase ratio to 16:9 sometime, see if it's because I'm running on Linux, try with Lucian's computer later today	So the algorithmic dilemma I'm struggling with: We want to transfer what the lydar detects in physical 3 dimensional space into a 2D representation overlaid on a video. Let this white bar I've drawn here represent an offset in the y dimension (2D space in video) and z dimension (3D physical space). As we move our point of interest to specified distances, this angle needs to remain constant and our y/z offset needs to grow or shrink accordingly. This means our camera and lydar need to be stationary to define some constant angle, once that is done we can just apply some pretty simple math.	Experiment with background subtraction and opencv SVM to capture human faces.
7	Lucian was talking about using some kind of equation in opencv to get distance, I think we need to get a better illustration to see how Kevin envisions us connecting the webcam feed to the data from the lydar	Sometimes I'm not sure what Lucian means.	Per my conversation with Kevin, I successfully used a trained SVM cascading front-face model to detect faces, not extremely accurate right now, thinking about using this to get face detection Specifically, this is needed because the Lydar will capture objects in an X-Y plane, we need the face detection to have some frame of reference in the Z direction. Our midterm progress report due date got pushed back, I drafted and finalized a few sections. Lucian wrote the other half.
8	Accidently missed our poster critique session, we're going to attend the extra credit session for credit.	-	Successfuly implimented a pretrained Tensorflow object detection model. This detects full or partial bodies with very high accuracy. After minimal tuning, the machine learning part is effectively done.

9	Poster Critique session had the following suggestions: Make sure correct poster format, put figure and captions, put x-y-z scale on main image, change layout, ”visually connecting” layout to have presentation flow Doesnt need period at tagline Include contact Include Client	-	Made adjustments to poster design.
10	Dead week, we’re busy writing the Winter report.	Lucian accidentally wiped the contents and history of our documents repo. Fortunatly I had it cloned on my laptop so nothing was lost.	Need to discuss work schedule with Kevin and Lucian.

6.2 Spring 2018

Week	Positives	Deltas	Actions
1		No idea if Lucian is working on the M16, sounds like he has trouble with it. Has yet to show me any working code.	I’m going to work on the Rplidar A1 as backup RPLIDAR A1 Driver: https://www.silabs.com/products/developr tools/software/usb-to-uart-bridge-vcp-drivers
2		I’m not convinced Lucian is working on his part of the project. He keeps promising he’ll get it done that week but when we meet to work on it he claims hes blocked trying to fix some environment issue.	I’m going to continue working on Rplidar A1 and designing the mount.
3	I’ve finished building the mount, I’m using two pieces of plywood joined in an L-shape. It’s made out of wood, the logitech brio webcam will just sit ontop and I’ve marked out where the LIDAR m16 will screw in. Kevin lent me some go-pro mounts to mount everything but I can’t use them because the Brio’s usb-c cable sticks out the back and gets in the way.	Still no progress from Lucian	
4		No progress from Lucian.	We need to work on the midterm progress report and poster.
5	All of my stuff is done, the computer vision stuff is completed.	No progress from Lucian.	I’m going to get the RPLIDAR A1 and start integrating that. Unfortunately Lucian has it and he hasn’t given me any code to use it so I guess I’ll just figure it out.

6	Windows update 1803 messed up the RPLIDAR A1 driver. I found a fix on their forums Steps to fix: 1) Download the Windows 10 Universal driver from here:https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers 2) Edit the .inf file and replace all 13 with 12. 3) Open admin cmd 4) bcdedit /set testsigning on I have successfully integrated the RPLIDAR A1 into my work. This involved writing a small python program to deal with the RPLIDAR's poorly implemented generator function in the python driver.	No progress from Lucian.	
7	Meeting with Kevin - Kevin lent me the following for expo • GTX 1080 ti • 125 GB SSD Tested new components in demo PC, everything works. Framerate improvement over GTX 750 ti, 1080 is probably unnecessary because the demo PC's i5 is a bottleneck but the error "running out of video memory" is gone. Expo was very successful, a lot of interest in my project.	At the beginning of the week it's clear Lucian still hasn't done anything. He messed up the repo organization but whatever, nothing important was in that particular one. I think it's pretty sad I had to do this whole project by myself.	
8			Worked on final report
9			Worked on final report
10			Worked on final report

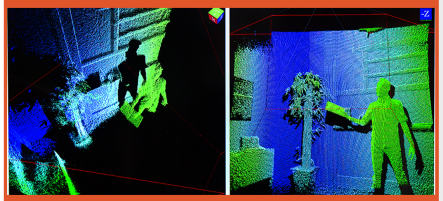
COLLEGE OF ENGINEERING | Electrical Engineering and Computer Science | CS69

DEPTH SENSING WITH COMPUTER VISION AND LIDAR

Combining computer vision and lidar to create a reliable and accurate depth sensor.

INFRARED DEPTH SENSING

- Commercial depth sensors such as the Microsoft Kinect strobe an IR pattern to detect objects.
- This method is relatively inexpensive and very accurate in certain conditions

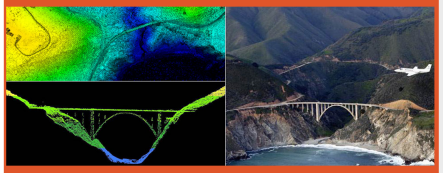


Above - Microsoft's visualization demonstration.
Source: developer.microsoft.com/windows/kinect

- The Sun emits natural IR that blinds depth sensors.
- Multiple IR sensors will overlap and confuse each other.

LIDAR DEPTH SENSING

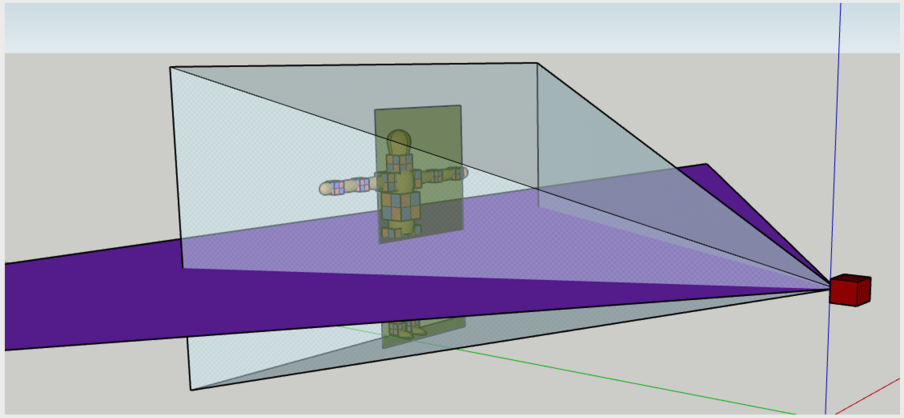
- LIDAR – Light Detection And Ranging, this method pulses a laser to accurately measure variable distances



Above – NOAA survey aircraft's LIDAR data on Bixby Bridge in Big Sur, California.
Source: oceanservice.noaa.gov/facts/lidar.html

- Common ambient interferences do not affect LIDAR
- Direct object-point cloud data extraction from LIDAR has not been achieved

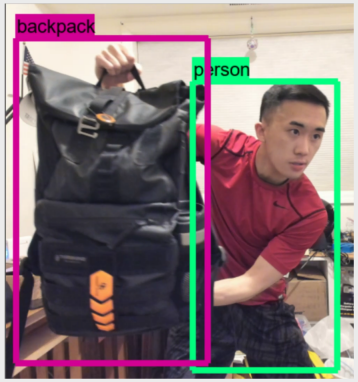
Oregon State University



Above – Visualization of planes captured by our project
- Red cube - Represents the webcam and lidar device secured in stationary positions
- Flat purple triangle - represents the lidar's horizontal range detection.
- Transparent green rectangle - Computer vision model recognizing that there is a person in-front of the sensor.
- Transparent teal pyramid - represents the webcam's field-of-view.

COMPUTER VISION

- Built using pre-trained TensorFlow computer vision models
- GPU Acceleration facilitates object recognition calculation and near real-time rendering



LIDAR DEPTH SENSING

- LIDAR sends data to computer vision system which is calibrated to ignore LIDAR data points outside of the camera's FOV
- Relevant LIDAR distance data points are matched to classified objects based on their XY positions

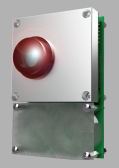
PROJECT MEMBERS

Kin-Ho Lam (Left)
Sponsor: D. Kevin McGrath (Right)

Project source code available at github.com/DSCVL

DEVICES & SOFTWARE USED

- TensorFlow pre-trained models
- Logitech BRIO 4K Webcam
- RPLIDAR A1 (Left)
- LEDDAR M16 (Right)



COMMERCIAL APPLICATIONS

- Component to an autonomous driving system in cars and robots
- Proof of concept demonstrates this system is relatively cheap and effective alternative to IR depth sensing.

Above – Completed system concept output

8 Conclusions and Reflections

1) What technical information did you learn?
   I learned how to use Google Tensorflow's image recognition dataset, and I gained experience in working with a hobbyist LIDAR device.
2) What non-technical information did you learn?
   I practiced communication, work delegation, and pre-planning for unexpected events.
3) What have you learned about project work?
   I learned that project work is very demanding especially if group members are not technically skilled or equipped to accomplish their tasks.
4) What have you learned about project management?
   I learned that project management is very difficult when you neglect to select the best people for your project. Dealing with random individuals with no real work experience or motivation to excel is very challenging, and it falls to the group leader to provide motivation and enforce the importance of accountability.
5) What have you learned about working in teams?
   I learned that working in teams that you do not select yourself is very challenging. In other group projects, I was able to work with individuals who are highly motivated and very knowledgeable. Unfortunately during this senior capstone project, I lacked a good team and consequently had to do the entire project myself.
6) If you could do it all over, what would you do differently?
   If I had to do this entire project all over with the foreknowledge of this past year, I would have not used the RPLIDAR A1 and focused entirely on getting the LEDDAR M16 to work.

Appendices

8.1 Installation Instructions

The original Tensorflow model can be viewed here https://goo.gl/Hkcrf3 Tested, developed, and executed on Windows 10, build 1803

8.1.1 Dependencies

1) Python 3.5.2 - https://www.python.org/downloads/release/python-352/
2) I recommend installing Tensorflow-gpu to run visualization
   ```
   pip3 install --upgrade tensorflow-gpu
   ```
3) RPlidar A1 - https://www.slamtec.com/en/lidar/a1
   ```
   pip3 install rplidar
   ```
4) OpenCV 3.4.0.12
   ```
   pip3 install opencv-python==3.4.0.12
   ```
5) CP210x USB to UART Bridge VCP Driver (If on Windows 10, build 1803, please see below)
   https://goo.gl/oiywRV

8.1.2 Tensorflow-gpu

If you are using Tensorflow-gpu, I used the following CUDA and CUDnn versions. This project was developed and tested on an Nvidia GTX 1080, but any CUDA-enabled GPU should work too. Memory issues were noted when running on a GTX 750 Ti.

1) Nvidia CUDA Toolkit 9.0.176
2) Nvidia CUDnn v7.0.4 for CUDA 9.0

8.1.3 Windows 10, build 1803 Driver Fix

The CP210x USB to UART Bridge VCP Driver, required for RPlidar A1, does not meet Windows driver INF requirements and will not install.

Read more here: https://goo.gl/5erDwL
Instructions to enable unsigned driver installation - https://goo.gl/EzrGUx
My modified driver download- https://drive.google.com/file/d/1jKxpAml_JoFPP2b4fXIjJWZ2_udghgj9/view?u

8.1.4 Run Visualization

1) Install all dependencies
2) Clone repository
   ```
   git clone https://github.com/DSCVL/DSCVL-Vision.git
   ```
3) In file `lam_helper.py`, replace line 71 `PORT_NAME = 'COM15'` with the COM port that the RPlidar A1 is connected to.
4) Ensure web cam and RPlidar are plugged in, execute with the following command.
   ```
   python main.py
   ```

8.2 Essential Code Definitions

8.2.1 main.py

The following code consists of the main function that is executed at system start.

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

import math
import cv2

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

sys.path.append("..")

from utils import label_map_util
from utils import visualization_utils as vis_util

from lam_helper import * # My code

import matplotlib.pyplot as plt



VIDEO_WIDTH = 1290
VIDEO_HEIGHT = 720
DMAX = 3000          # Max distance (meters)
IMIN = 0
IMAX = 100


# # Model preparation

# ## Variables
#
# Any model exported using the 'export_inference_graph.py' tool can be loaded here
    simply by changing 'PATH_TO_CKPT' to point to a new .pb file.
#
# By default we use an "SSD with Mobilenet" model here. See the [detection model zoo](
    https://github.com/tensorflow/models/blob/master/object_detection/g3doc/
    detection_model_zoo.md) for a list of other models that can be run out-of-the-box
    with varying speeds and accuracies.

# In[4]:

# What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the object
    detection.
```

```
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90


# ## Download Model

# In[5]:

opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())


# ## Load a (frozen) Tensorflow model into memory.

# In[6]:

detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')


# ## Loading label map
# Label maps map indices to category names, so that when our convolution network
    predicts `5`, we know that this corresponds to `airplane`.  Here we use internal
    utility functions, but anything that returns a dictionary mapping integers to
    appropriate string labels would be fine

# In[7]:

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=
   NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)


print("\tMain viz start...")
cap = cv2.VideoCapture(0)    # Start camera
cap.set(3,VIDEO_WIDTH)       # Set Resolution
cap.set(4,VIDEO_HEIGHT)

print("\tStarting polar plot")
ax = plt.subplot(111, projection='polar')
ax.set_rmax(DMAX)
ax.grid(True)

with detection_graph.as_default():
```

```
with tf.Session(graph=detection_graph) as sess:
  for lidar_read in AsynchronousGenerator(function=gen(), maxsize=1):
      skip = False
      readings = []
      for read in lidar_read:
          theta = read[1]
          distance = read[2]
          if (theta >= 0) and (theta <= 55):
              x_c = (VIDEO_WIDTH / 2) + math.sin(math.radians(theta)) * (VIDEO_WIDTH /
                  2)
          elif (theta >= 305):
              x_c = math.sin(math.radians(360 - theta)) * (VIDEO_WIDTH / 2)
          else:
              continue
          readings.append( (x_c, distance) ) # Append x coordinate and distance
          ax.scatter(math.radians(theta), distance) # Plot
          ax.set_rmax(DMAX)
          ax.grid(True)

      plt.pause(0.0000001)
      plt.cla()

      ret, image_np = cap.read()

      # Expand dimensions since the model expects images to have shape: [1, None, None
          , 3]
      image_np_expanded = np.expand_dims(image_np, axis=0)
      image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
      # Each box represents a part of the image where a particular object was detected
          .
      boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
      # Each score represent how level of confidence for each of the objects.
      # Score is shown on the result image, together with the class label.
      scores = detection_graph.get_tensor_by_name('detection_scores:0')
      classes = detection_graph.get_tensor_by_name('detection_classes:0')
      num_detections = detection_graph.get_tensor_by_name('num_detections:0')
      # Actual detection.
      (boxes, scores, classes, num_detections) = sess.run(
          [boxes, scores, classes, num_detections],
          feed_dict={image_tensor: image_np_expanded})
      # Visualization of the results of a detection.

      vis_util.visualize_boxes_and_labels_on_image_array(
          image_np,
          np.squeeze(boxes),
          np.squeeze(classes).astype(np.int32),
          np.squeeze(scores),
          category_index,
          use_normalized_coordinates=True,
          skip_scores=True,
          line_thickness=4,
          lidar_m = readings)

      # Display webcam output
      cv2.imshow('Object␣Detection', image_np)

      if cv2.waitKey(25) & 0xFF == ord('q'):
          cv2.destroyAllWindows()
          break
```

```
plt.show()
```

8.2.2 lam_helper.py

The following code synchronously reads data from the RPLIDAR A1's python generator driver and stores it in a buffer to be called as needed.

```
import threading
import collections
from rplidar import RPLidar

try:
    import Queue
except ImportError:
    import queue as Queue

class AsynchronousGenerator:
    """
    The AsynchronousGenerator class is used to buffer output of a
    generator between iterable.__next__ or iterable.next calls. This
    allows the generator to continue producing output even if the
    previous output has not yet been consumed. The buffered structure is
    particularly useful when a process that consumes data from a
    generator is unable to finish its task at a rate comparable to which
    data is produced such as writing a large amount of data to a
    low-bandwidth I/O stream at the same time the data is produced.

        >>> for chunk in AsynchronousGenerator(function=makes_lots_of_data):
        ...     really_slow_iostream.write(chunk)
    source: https://www.reddit.com/r/Python/comments/ew9is/
        buffered_asynchronous_generators_for_parallel/
    """
    def __init__(self, function, args=(), kwargs={}, start=True, maxsize=0):
        self.generator = iter(function(*args, **kwargs))
        self.thread = threading.Thread(target=self._generatorcall)
        self.q = Queue.Queue(maxsize=maxsize)
        self.next = self.__next__
        if start:
            self.thread.start()

    def __iter__(self):
        return self

    def __next__(self):
        done, item = self.q.get()
        if done:
            raise StopIteration
        else:
            return item

    def _generatorcall(self):
        try:
            for output in self.generator:
                self.q.put((False, output))
        finally:
            self.q.put((True, None))

class wrapper(object):

    def __init__(self, generator):
        self.__gen = generator()
```

```
    def __iter__(self):
        return self

    def __next__(self):
        self.current = None
        while self.current == None:
            try:
                self.current = next(self.__gen)
            except:
                print("ERROR: Lidar init failed. Please restart.")
                quit()
        return self.current

    def __call__(self):
        return self

PORT_NAME = 'COM15'
@wrapper
def gen():
    lidar = RPLidar(PORT_NAME)
    return lidar.iter_scans()
```

8.3 Senior Design Expo

The demo system was completed in time for senior design expo. I built a computer from some spare parts to run the demo system. The computer's specs are as follows:

1) Intel i5 2400
2) 8GB RAM DDR3 @ 2400Mhz
3) GTX 1080Ti FE (Borrowed from sponsor)

The completed system was able to run in stable, uninterrupted operation during the entire 8 hour expo without catastrophically crashing. It was also able to successfully classify a large amount of subjects and pedestrians passing by the booth.

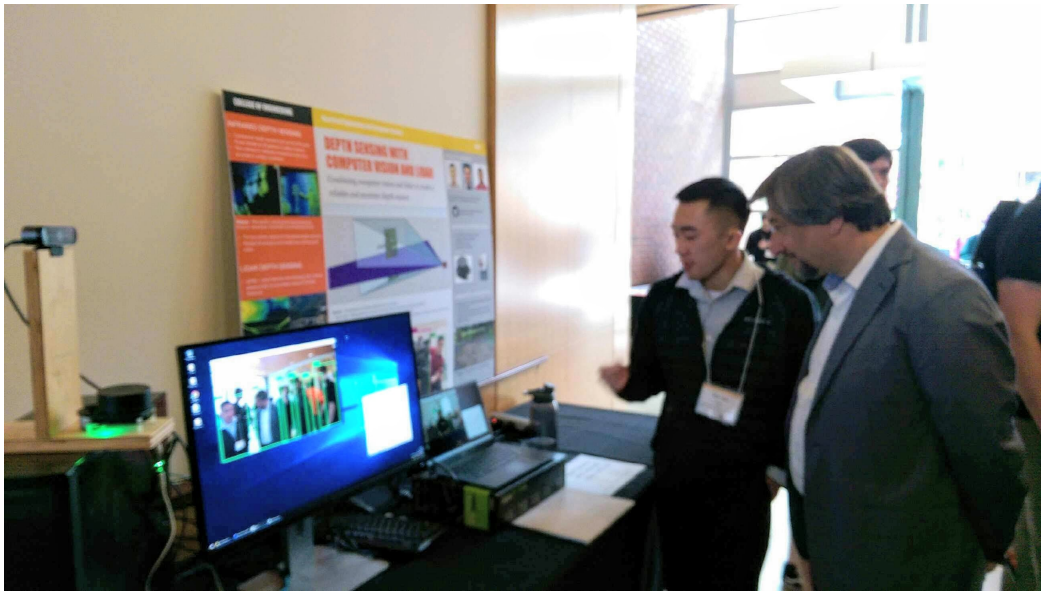


Fig. 12. Senior design expo.

Fig. 13. Senior design expo, photo by Dr.Kirsten Winters