Oregon State University
College of Engineering

SENIOR DESIGN CAPSTONE SPRING MIDTERM PROGRESS REPORT

MAY 6, 2018

DEPTH SENSING WITH COMPUTER VISION AND LIDAR

PREPARED FOR

OREGON STATE UNIVERSITY

D. KEVIN MCGRATH

PREPARED BY

GROUP 69

KIN-HO LAM
LUCIEN-ARMAND T. TAMNO

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

2 Definitions

2.1 IR

IR refers to the infrared light spectrum.

2.2 IR Depth Sensor

A device that calculates distances by emitting infrared patterns.

2.3 lidar

Light Detection And Ranging - A method that uses lasers to measure distance

2.4 Microsoft Kinect

A product that uses an IR Depth sensor to measure distances. Referred as a benchmark.

2.5 Logitech Brio Webcam

Webcam made by Logitech. [?]

2.6 RPLidar A1

A budget lidar device made by Slamtec. [?]

2.7 Leddar M16

A solid-state lidar device made by Leddar. [?]

2.8 Computer Vision

The methods for acquiring, processing, analyzing, and classifying digital images and extracting information.

2.9 Python/C API:

[?]

API: application programming interface between Python and C programs

2.10 DSCVL:

Acronym that stands for Depth sensing with computer vision and lidar

3 Project Purpose

Infrared (IR) depth sensors such as the model used in Microsoft's Kinect 2.4 can quickly calculate distances in indoor scenarios. However, IR depth sensor readings can be obfuscated by other infrared emitting sources such as other IR depth sensors or natural sunlight. 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 technology 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 Leddar M16 provides accurate depth measurement in a horizontal dimension but lacks vertical perspective beyond a 40-degree spread. This project proposes bridging the utility of both devices by securing them in stationary positions, then using software to combine their outputs.

Figure 1 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.

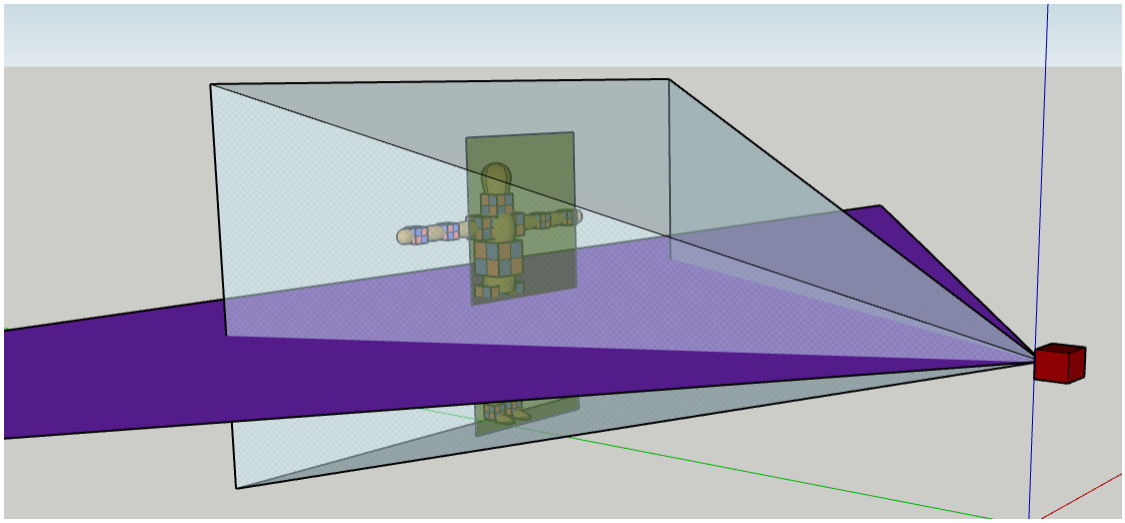


Fig. 1. Visualizing different dimensions measured by the M16 Lidar and Brio Webcam.

5 DEVICE MOUNT

Using some spare plywood, I created a mount for the Logitech Brio webcam and Leddar M16 as shown in Figure 2. This mount serves to stabilize the webcam and lidar devices in a stationary positions relative to each other so that accurate distance/visual calibrations can be performed. If the webcam and lidar devices are not placed in consistent positions, distance information will not be synchronized with object recognition. Writing an algorithm to compensate for automatic distance and object recognition calibration is an overly complex task and beyond the scope of this project.

Fig. 2. Part of the mount for .

Progress with the Leddar M16 is slow and I do not foresee it to be working and integrated in time for expo. Fortunately, I was able to read distance information with the RPlidar A1. I adjusted the mount to accommodate the RPlidar A1 by drilling a few shallow holes in the base of the mount. This allows the RPlidar A1's four plastic standoffs to fit into the base of the mount as shown in Figure 3.

Fig. 3. Visualizing different dimensions measured by the M16 Lidar and Brio Webcam.

The device mounting system is complete. Our project's components can be fixed in stable positions to ensure

consistent readings in different environments. This mounting system is important to our final design because it ensures operational consistency and simplifies the overall problem.

6 Computer Vision

During Winter Term, I started experimenting 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 our 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.

Our design specifications envision a system that quickly tracks multiple subjects in a crowded expo scenario. In a such a scenario, human subjects will be unpredictably shifting their positions and moving in or out of the field-of-view. As seen in Figure 4, the OpenCV SVM model does not perform to our 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. 4. SVM face classification (Left) fails when subject slightly turns their head (Right)

Tensorflow's open source object detection classifier presented a better computer-vision alternative. [**?**] 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. [**?**] These pre-trained datasets in Tensorflow's library are sourced from other machine learning datasets and employ advanced algorithms such as Single Shot Multibox Detection and Region-Based Fully Convolutional Networks. [**?**] [**?**] [**?**]

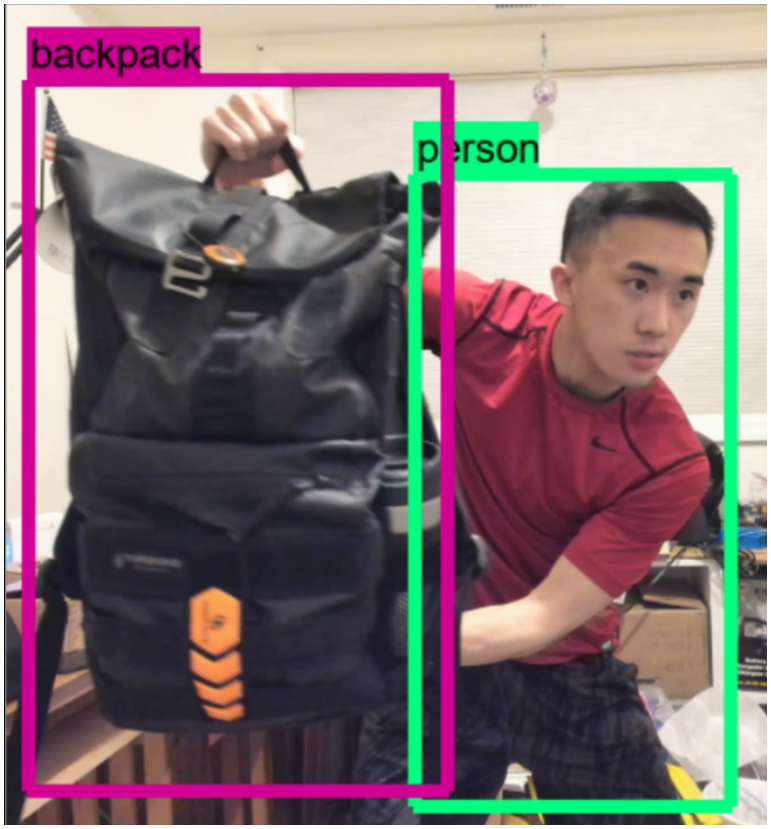


Fig. 5. Our pre-trained Tensorflow model can detect multiple subjects in near real-time

Using this pre-trained Tensorflow model, our project can now accurately outline and label over 90 subjects as they come into view of the webcam. The current state of the code will enable us to selectively edit 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 our list of material requisitions for our physical expo demo, moving calculations to the GPU greatly improves the output video quality, frame rate, resolution, and classification speed. [**?**]

The computer vision aspect of our project is now complete. Combined with a stable mount, we now have a versatile system that can recognize over 80 distinct models such as humans, bags, or animals in near real-time.

7 Current Problems

7.1 RPlidar A1 and Computer Vision Integration

To ensure our project will be in a presentable state by expo, I am modifying my design to use the RPlidar A1. Progress with the M16 lidar device has been slow, I do not foresee it being and integrated in time for expo. I have successfully tested the RPlidar A1 and have achieved reading some point-distance angle information. However, the python library used by the device calls a generator function to store its distance data. This type of data structure is hard to adapt for our purposes because the device's buffer will overfill and overwrite readings if it is not called fast enough. Figure 6 demonstrates this issue. To remedy this, I will experiment adjusting the loop calling the Tensorflow image recognition functions. I will also experiment running the RPlidar A1 in parallel with the Tensorflow image recognition.

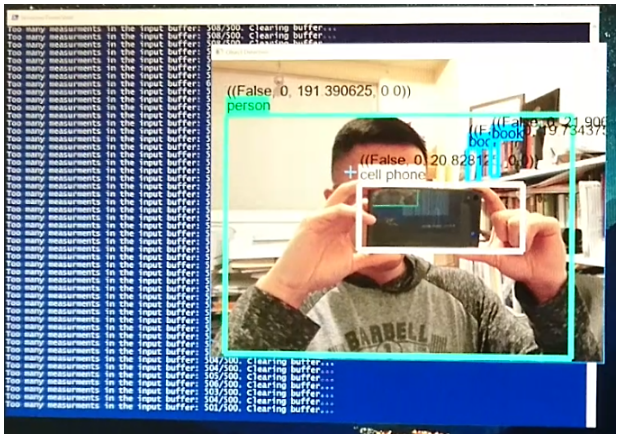


Fig. 6. Due to the overfilling buffer my current implementation does not calculate distances correctly.

8 LEDDAR M16 AND PYTHON/C API- [CONSTRUCTS]———LUCIEN

9 PROJECT PURPOSE

The purpose of the DSCVL2.10 project is to come up with an effective solution of painting on a screen an image form a camera with a single point signal from another source technology called Lidar. And for the sake of DSCVL final solution accuracy and better performance,the project concept is to look for any technological tool or system that helps to build an ultimate robust software.

In fact, on the source technology used in this project, namely the lidar technology is known to be reliable in outdoor environment in terms of objects distance measurements, and mostly when that lidar technology uses its 16-Segment SOLID-STATE LIDAR equipment also known as the Leddar M16 **??**. One the other hand, a webcam specifically the Logitech Brio Webcam 2.5 would help to get an image that is located right at the exact same distance than the point signal provided by the new adopted Leddar M16. All this to render with precision the object of interest whether it be a person or a liquid.

Moreover, the fact that the lidar technology (Leddar M16) is much more accurate to provide better objects detection than the IR technology 2.1 in outdoor, prompts its de facto adoption in this DSCVL project. However, it is important to know how to use this type of device, how this kind of material would fit into the entire to be built application, and ultimately how to connect the code run by the M16 to other components of the DSCVL project.

10 CURRENT STATE OF THE DSCVL

Though, the past winter term progress report described the RPLidar A1 capability 2.6 of outputting useful information but the reality is that the DSCVL project needed some changes for the sake of better accuracy. some changes took place at different level of the project. And the most important being the replacemnt of RPLiadr A1 by the 16 segment solid state M16 that provides better detection, localization, and distance measurements when is to compares to the RPLidar A1.

Additionally, those differences can even be more tangible at the code level, because the M16 model has its code literally written in C program while the RPLidar code is closely related to the python environment. And this M16 factor code-based caused me to conduct researches to figure out how can I make the C program code work in python environment. The result of that research was to discover the python **??** functionality to play the intermediary role of having the C program code runs in a python environment.With the same idea, I dug a little bit more to finally find out that Visual Studio is the suitable Windows application to implement the python/C API.

10.1 Preliminaries and the Python/C API implementation

10.1.1 The default M16-Windows setting

To stick with the Leddar M16 technology, I have to implement the python/c API2.9 using the C-code that was shipped with the M16 device. And to do so, I have to make sure first that the M16 can properly run in a Windows environment by default. And for me to test that functionality, I installed and configured the default M16 settings, as shown in figure 7 below.

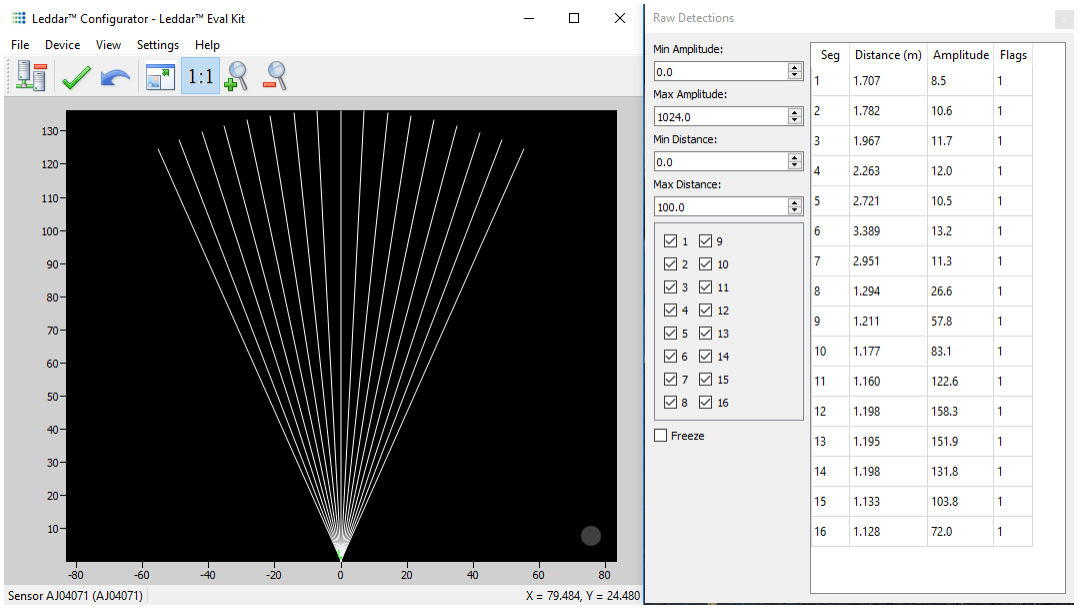


Fig. 7. figure
Leddar M16 outputs distance measurements using the default windows installation [right] and default M16 screen [left].

That figure 7 displays a sample distance outputs as it is supposed to be visualized by a user at the final stage of the DSCVL project. Though, differences may be seen in term of data disposition on the screen,because the M16 code has to run in conjunction with webcam technology to overlay images form the python environment. Hence, the implementation of the DSCVL application has an additional technological constraint to be done via an API.

10.1.2 The Python/C API implementation, related issues and attempted solutions

To get the written C language code to python, the Visual studio 2017 application appeared to be the adequate tool to use. Therefore, I installed visual Studio in Windows 10 system, imported any necessary library to connect the visual Studio to the Leddar M16 C-code,using the appropriate [**?**] documentation related to the topic. The remaining bulk of work is to write and debug the API implementation code to make sure the M16 interacts properly from the visual studio perspective and secondly, to integrate in a single package the python and the M16 C codes, as shown in figure 8.

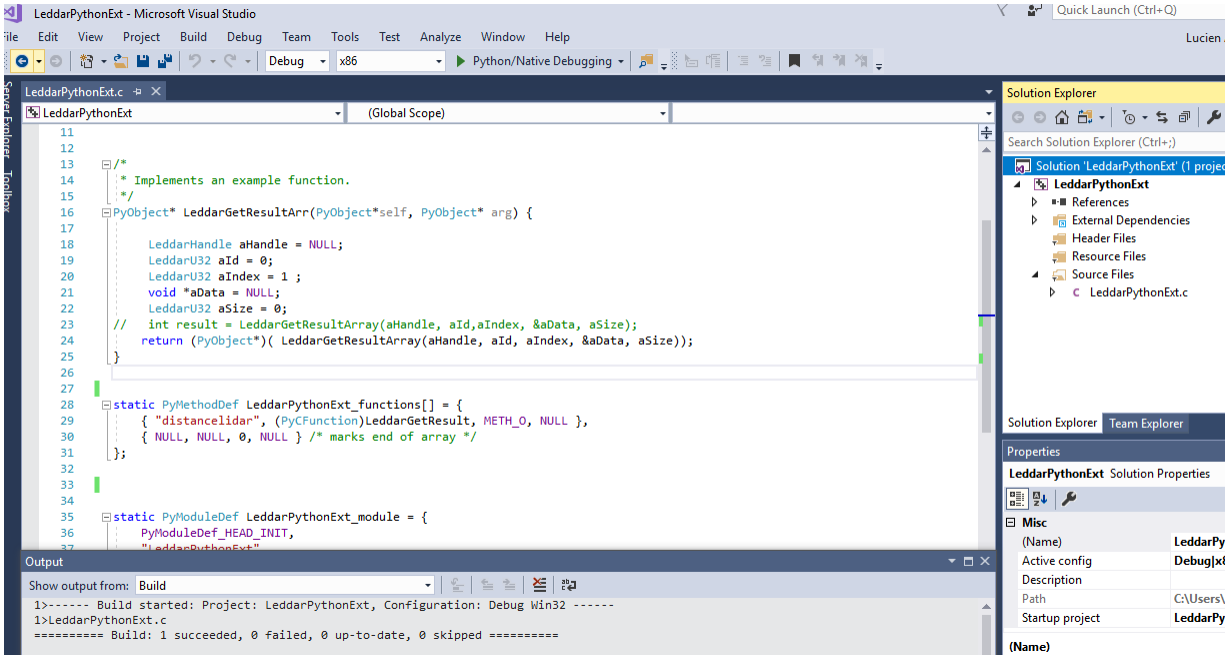


Fig. 8. figure
Python/C API built in Visual Studio 2017.

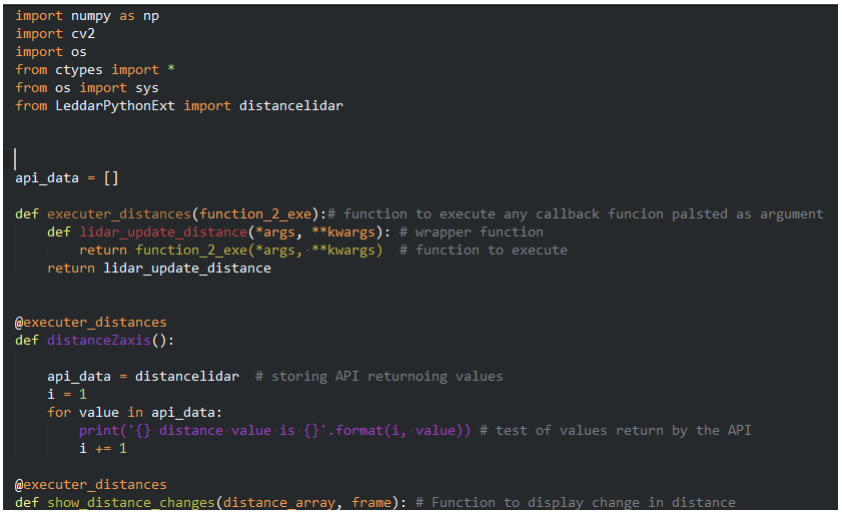
However, when working on the above API, I encountered multiple setbacks.among others, the two most important issues are:the slight Leddar M16 differences seen in the the M16 files, for instance I had to change one default header, as shown in figure 9 below.

```
#pragma once

#ifdef WIN32
  #ifdef LEDDARC_LIBRARY
  #define LEDDARC_DLL __declspec( dllexport )
  #else
  #define LEDDARC_DLL __declspec( dllimport )
  #endif
#else
  //#define LEDDARC_DLL __attribute__ ((visibility ("default")))
#define LEDDARC_DLL __declspec( dllimport )
#endif

#ifdef __cplusplus
extern "C"
{
#endif
```

Fig. 9. figure
Snippet code changed in the Leddar M16 header file.

And the worst being found in the current debugging phase that presents a successful result in code checker but generates several linkage errors.

Finally, I also wrote an additional decorator function in the python module to speed up the python execution module as shown in figure 9here underneath.



```python
import numpy as np
import cv2
import os
from ctypes import *
from os import sys
from LeddarPythonExt import distancelidar


api_data = []

def executer_distances(function_2_exe):# function to execute any callback funcion palsted as argument
    def lidar_update_distance(*args, **kwargs): # wrapper function
        return function_2_exe(*args, **kwargs)  # function to execute
    return lidar_update_distance


@executer_distances
def distanceZaxis():

    api_data = distancelidar  # storing API returnoing values
    i = 1
    for value in api_data:
        print('{} distance value is {}'.format(i, value)) # test of values return by the API
        i += 1

@executer_distances
def show_distance_changes(distance_array, frame): # Function to display change in distance
```

Fig. 10. figure

Python decorators code.

10.2 Legacy: GUI Interface

In order to prioritize the DSCVL backend, the Frontend was relegated because the backend is more important and represent the project core functionality as envisioned in the final stage. The GUI downgrade, yet still provides the basic services it has before, but it is not clear whether the final DSCVL system would make use of the legacy graphical user interface.

11 Conclusion

This midterm progress report provides a succinct description of the work done in about a month, presents the focal point of the bulk of work that is to implement the python/C API and ultimate, problems encountered during that implementation process and finally some attempted solutions as remedy to solve those issues.