Journal of UAB ECTC
Volume 15, 2016
Department of Mechanical Engineering
The University of Alabama, Birmingham
Birmingham, Alabama USA

APPLICATIONS OF LIDAR IN AUTONOMOUS VEHICLES

Allen J. Stewart, Jonathan Burden, Kevin McFall
Kennesaw State University
Marietta, Georgia, United States

ABSTRACT

This paper explores the applications of multidirectional LIDAR in autonomous vehicles. Such devices map their field of vision in a two dimensional outline. It summarizes the use of LIDAR in current autonomous vehicles in industry, such as the Google self driving projects. This paper also reviews the functionality of autonomous vehicles that could be enhanced with the inclusion of LIDAR systems. The scope of this project involves investigating how existing functionality could be replicated with similar performance given less expensive hardware, and how these developments can impact future applications within autonomous vehicles. A major focus of this project is to ensure that the analysis software will be compatible with as many platforms as possible. To this end, Python, an open source scripting language with a major focus in data analysis, is used as the programming language for the research and development. A major focus of this project was to use Python to reverse engineer the serial communication protocol between the UBG-05LN and a computer.

INTRODUCTION

The National Oceanic and Atmospheric Administration (NOAA) defines LIDAR, Light Detection and Ranging, as a range perception system that uses pulsed laser light to map environments [8]. LIDAR is prevalent in autonomous vehicles as a result of its unique ability to produce a 1:1 scaled 3-D model of the environments surrounding the sensor [3]. This functionality comes at a price. As stated by Fisher, “at $75,000 to $85,000 each, Google's lidar costs more than every other component in the self-driving car combined, including the car itself.” While the utility that these LIDAR units provide is valuable, a cheaper alternative is necessary if the general public is to ever see autonomous vehicles on a large scale [3].

Thus far, Google has set the standard when it comes to self-driving technologies [3]. The vehicles that Google is capable of producing do not require any input from their user. They are able to accelerate, brake, and steer themselves completely autonomously [4]. However, by Google’s own admission “The project is still far from becoming commercially viable” [5]. While many systems are working in conjunction to make this a reality, the vehicles get a non-trivial amount of their information from the LIDAR sensory systems (Hillel et al). As a result, reducing the cost to implement LIDAR into autonomous vehicles, for a wide range of applications, is a top priority [3].

LIDAR serves a plethora of functions in autonomous vehicles. As discussed previously, LIDAR allows for a vehicle to generate a detailed map of its surroundings [3]. The Google vehicles use this map to do all of the following: detect non-vehicle objects that may cause a disturbance to navigation, work in conjunction with radar and sonar devices to assist in adaptive or intelligent cruise control, assist in lane keeping and station keeping calculations by observing and documenting the moment of other vehicles, and interrupt the various other systems of the vehicle in the event of an emergency [3].

Current systems leverage high cost LIDAR hardware for accurate mapping. This project aims to investigate the feasibility of a low cost LIDAR system that is capable of replicating the functionality of existing systems by means of more sophisticated software.

PROCEDURE

The LIDAR sensor for this project is the UBG-05LN manufactured by Hokuyo. This device is intended for use case scenarios that require only a binary value indicating whether an object is within a certain region of the sensor’s view, such as for use in an automatic door [6]. The binary sensor regions are to be programmed with the included software. This is the intended use of the hardware by the manufacturer [6].

Work began by establishing a physical connection between the sensor and a Windows based computer via a DB-9 cable using the RS-232 communication standard in order to interface with the first-party software. As it was shipped, the sensor did not come with a DB-9 serial connection installed, so a makeshift connector was constructed to correctly interface the data lines of the sensor with the serial port of the computer system. This was done by jumping the rx, tx, and ground connections of the sensor to a standard serial cable based upon the data sheet specifications. The first-party software recognized and began to communicate with the sensor. It then produced a continually refreshing point cloud map of all the data in its 180° field of view. An example of this is shown in Figure 1.

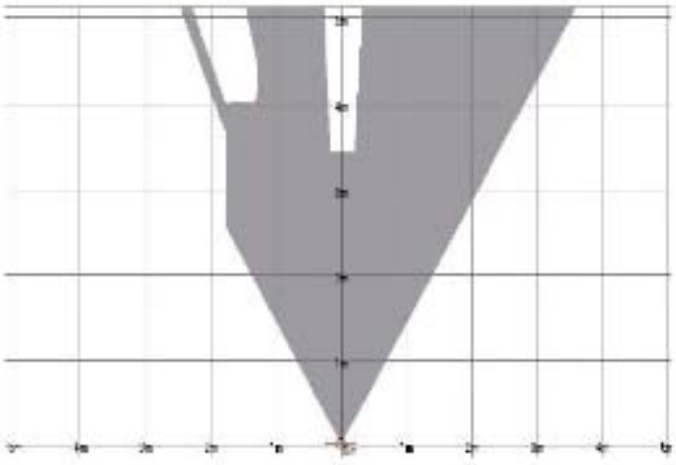

Figure 1. Sensor Demo Software

It was noted from the data that was presented by the example software that the sensor hardware was capable of far more functionality than the manufacturer specified. The software was able to return relatively high resolution range measurements over its serial link. However, out of the box, the sensor did not provide ready access to this numeric data.

At this point, the method by which the sensor communicated with the software was unknown. In order to better understand the communication protocol, a serial port monitoring program was used. This program logs all data traffic passing through a specified serial port. By observing the data traffic in the logs, the control codes and data stream structure could be determined. All communication done by the sensor is via ASCII encoded characters. The sensor responds to control commands, which start with the "$" character, a command character or characters, then is terminated by a linefeed or newline character. The control command used primarily for this research was "$G". When this command was received by the sensor, the sensor sent back a stream of 1548 bytes. Inside this 1548 byte "datastream" was all the range data for the current "frame" the sensor was seeing.

In order to access this data, a Python script was written to automate the jobs of connecting to the sensor, requesting and receiving data, as well as decoding the data into a human usable form. The general flow of the script's function is as follows:

1. The script binds to the COM port of the computer.
2. A "request data" packet ($G) is sent over the serial port.
3. The data stream returned over the COM port is loaded one byte at a time into a list object
4. The data list object is decoded using the appropriate encoding scheme
5. The decoded data is graphed onto a plot window
6. Steps 2 through 5 are repeated until the operator stops the script.

The sensor uses a slightly odd 18-bit encoding scheme to represent the range data from the sensor. Only the 6 least significant bits in each byte are used. Every point measurement from the sensor is represented by 3 ASCII characters in the datastream. The first 6 characters were dropped from the datastream, as they contained an echo of the command that was received, then 3 status bytes. The remaining 1542 characters were converted from ASCII into the representative integer numbers. The next step in decoding was to subtract 48 (30 in hexadecimal notation) from each number, then convert the integers to binary form. After that, the six least significant bits in each character were concatenated, three at a time to form one 18-bit binary number.

For example, a three character measurement might be received in the datastream as "0Wk"

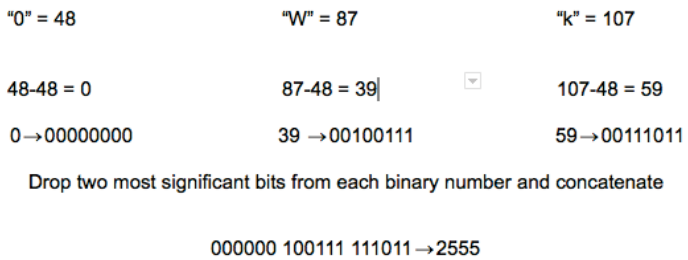

Figure 2. Measurement Example (Jonathan Burden)

This three-character set represents a range of 2,555 to the sensor.

That number, converted back into an integer was the range for a single measurement point from the sensor, given in millimeters. This process was repeated for the remainder of the datastream to form a single "frame". Each frame is made up of 513 point measurements, starting at the sensor's 3 o'clock position and sweeping leftwards across its field of vision, ending at the sensor's 9 o'clock position. The script plots these data points onto a graph and displays them, updating as new data is available.

CONCLUSIONS

Comparing the collected data both in real time, and offline, allows for a variety of LIDAR based applications. A particular application relevant to autonomous vehicles, and achievable with this data, is station-keeping. With the information collected by the sensor, the vehicle can monitor its surroundings and constantly check for changes in the environment and react accordingly in order to maintain a desired driving path.

Figures 3 displays the results of simulating an obstacle approaching a vehicle from the front, while the obstacles to the sides remain relatively unchanged. This behavior is achieved by placing the sensor inside a box. The sensor begins at the back of the box facing the front. It is then moved toward the front of the box, stopping at the midpoint. The range information is

represented in millimeters. The box has physical outer dimensions of about 400mm deep and about 450mm wide. Note that since the sensor is within the box, the physical size of the sensor affects the measured values, taking roughly 30 mm off of the effective depth and 20mm off of the width (10 on each side). Making the expected measured value from the back of the box about 370 mm deep and about 430 wide. From these measurements, the distance from the sensor to the either corner, from the back of the box, should be roughly 427mm.

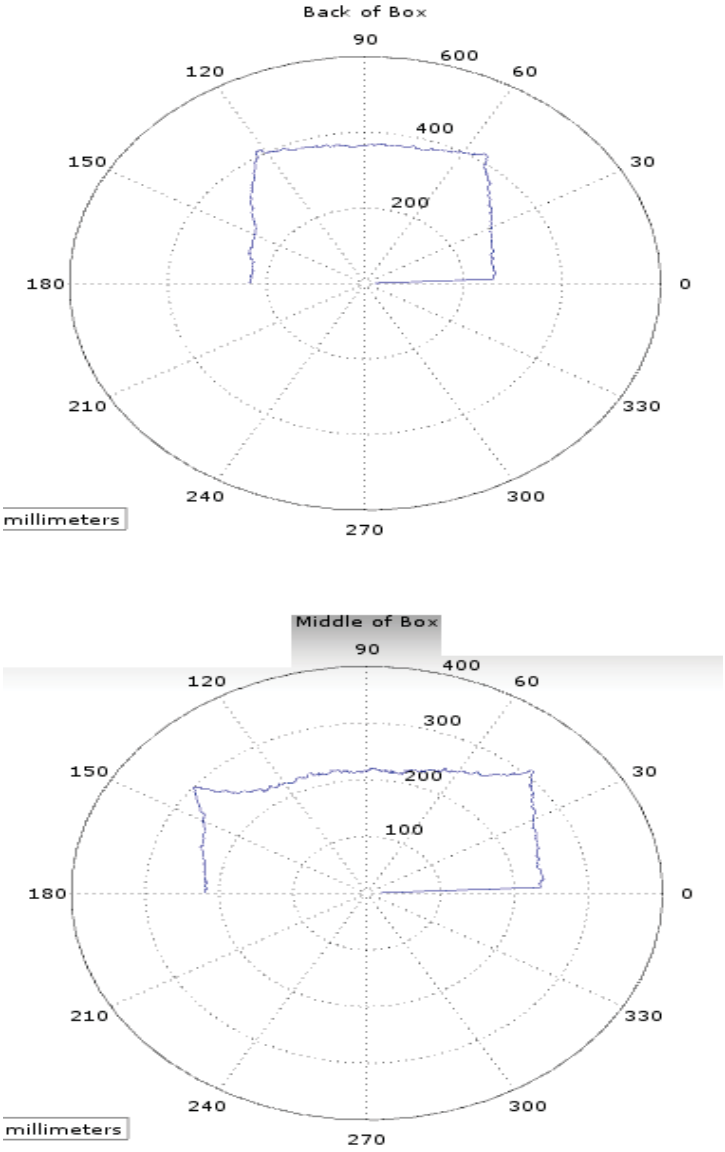

Figure 3. Results from simulating an obstacle approaching a vehicle with the obstacle first farther away (top) and then moving closer (bottom). Note: distance scales on top and bottom images are not the same.

The measured valued are about 6.32% lower than the expected values. This error likely comes from the sensor not being completely flush with the back of the box. It is also worth noting that the sensor is intended for use up to 5 meters, in which case an error of about 30 mm would be completely acceptable for applications in autonomous vehicles.

This kind of analysis is directly applicable to adaptive cruise control algorithms where the distances between a vehicle and the vehicle in front of it must be constantly monitored, while proper lateral distance is maintained with vehicles in adjacent lanes.

This project demonstrates the importance of LIDAR in self-driving applications. It set out to show that a less sophisticated LIDAR sensor is capable of collecting ample data to recreate some of the functionality of much higher priced systems. The software developed for this project is capable of poling new data from the sensor at a rate of about 15 times a second. Solutions like these are vital to the success of autonomous vehicles. Steps such as these are what will make autonomous vehicles an achievable goal.

REFERENCES

[1] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine Vision and Applications*, vol. 25, no. 3, pp. 727–745, Feb. 2012.

[2] Distance Measuring Type Obstacle Detection Sensor UBG-05LN. (2006, May 18). Retrieved from http://www.robotshop.com/media/files/PDF/UBG-05LN-Specs.pdf

[3] Fisher, A. (2013, September 18). Inside Google's quest to popularize self-driving cars. In *Popular Science*. Retrieved from http://www.popsci.com/cars/article/2013-09/google-self-driving-car

[4] Gannnes, L. (2014, May 14). Google's new self-driving car ditches the steering wheel. In *re/code*. Retrieved from http://recode.net/2014/05/27/googles-new-self-driving-car-ditches-the-steering-wheel/

[5] G. Erico, "How Google's Self-Driving Car Works," *IEEE Spectrum*, vol. 18, 2013.

[6] Scanning Range Finder UBG-05LN. (2009, August 3). Retrieved from https://www.hokuyo-aut.jp/02sensor/07scanner/ubg_05ln.html

[7] S. Yenikaya, G. Yenikaya, and E. Düven, "Keeping the Vehicle on the Road - A Survey on On-Road Lane Detection Systems," *ACM Computing Surveys*, vol. 46, no. 1, p. 2, Oct. 2013.

[8] What is LIDAR. (2015, May 29). In *National Ocean Service*. Retrieved from http://oceanservice.noaa.gov/facts/lidar.html