Andrew Willms
MTE 203 – Advanced Calculus
July 28, 2023

Using Gradient Descent to Locate Objects in LiDAR Point Clouds

Introduction

One sensor available for use in robotics automation is the LiDAR sensor. A LiDAR sensor can measure the distance between itself and an object in front of it. Using multiple adjacent LiDAR sensors or one moving sensor, a 2- or 3-dimensional representation of nearby objects can be constructed. These spatial representations are commonly known as LiDAR point clouds. The term "point cloud" is typically applied to a collection of points in $\mathbb{R}^3$, however this report will also apply the term to collections of points in $\mathbb{R}^2$. While it may be easy for a human to view a point cloud and identify and locate specific objects, such as a traffic cone, this can be a very difficult task for computers to accomplish. Mathematical descriptions may be used to directly identify simple objects, such as large flat surfaces or spheres, but this approach becomes vastly more challenging for complex and irregular shapes. This report will outline a method, based on the principle of gradient descent, for locating objects of arbitrary shape within a point cloud. [1]

Background

LiDAR – Light Detection and Ranging

LiDAR sensors are composed of two main components, a laser emitter and a detected. Most LiDAR sensors measure the unobstructed distance in front of them by firing a laser pulse forward and measuring the time until the reflection of that pulse is detected. This technique is commonly known as the Time of Flight, or ToF, principle. LiDAR sensors can produce precise, high resolution, and low latency distance measurements. This makes them ideal for many automation applications. [1]

The Gradient

The gradient of a function is a vector that points in the direction in which the function is changing the most. The formula for the gradient of a function, $f(x_1, \ \dots \ x_n)$, at point $\left(x_{1,0}, \ \dots \ x_{n,0}\right)$ is:

$$\nabla f(x_{1,0}, \dots \ x_{n,0}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1}(x_{1,0}, \dots \ x_{n,0}) \\ \dots \\ \dfrac{\partial f}{\partial x_n}(x_{1,0}, \dots \ x_{n,0}) \end{bmatrix}$$

Gradient Descent

Gradient descent is an iterative method that can be used to find the local minima of a function. The basic principle of gradient descent is that, starting at any point on a function, if many small steps are taken in the direction of the negative gradient, a local minimum of the function will eventually be reached [2] [3].

Gradient descent cannot be applied to all functions. For example, gradient descent is generally incompatible with functions that have saddle points and discontinuities in relevant locations [4].

One of the challenges of creating a robust implementation of a gradient descent algorithm is determining what size step, commonly called learning rate, to take in the direction of the negative gradient. Examples of a gradient descent algorithm with different learning rates are shown in Figure 1. As shown in Figure 1 choosing a smaller learning rate results in a more accurate result but requires more iterations to converge. In this way, choosing a balance or accuracy and speed is required for time sensitive gradient descent algorithms.

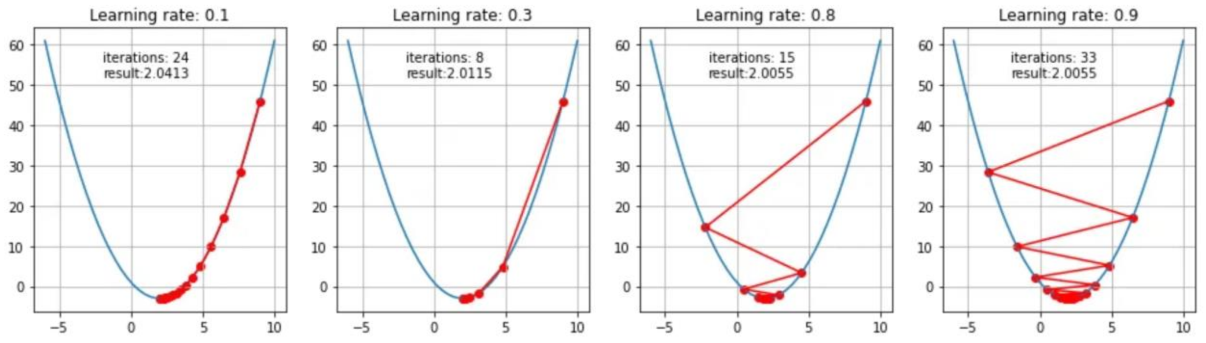

Figure 1: Examples of the output of a gradient descent-based algorithm with different learning rates [4].

Different optimizations techniques exist for minimizing the number of iterations required for conversion. One such of these techniques is known as Momentum. In areas of a where a function decreases more quickly in one direction than another, the negative gradient may largely point in the direction of steep change, even though the minimum is in the direction of more gradual change. Adding “momentum” to the motion can help the algorithm converge to the minimum faster. This is shown in Figure 2. [5]

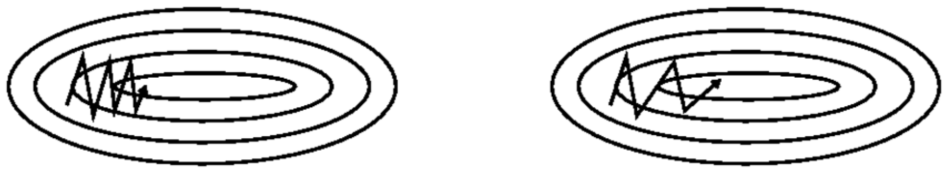

Figure 2: On the left: a representation of gradient descent without momentum. In this case the motion is largely in the direction of steepest change and much of it is waste. On the right: a representation of gradient descent with momentum. In this case the motion more strongly continues in the direction of consistent but gradual change and converges faster. [5]

Approach

Terminology and Notes

- **State**: For brevity, this report will use the term "state" will be used to refer to the position and orientation of an object.
- **Point Cloud**: The term "LiDAR point clouds" is typically used to refer to collections of points in $\mathbb{R}^3$, however this report will use the same term to refer to collections of points in $\mathbb{R}^2$.
- **Measurement and Simulation**: For this project no physical sensors were used. All "measurements" were taken using simulated sensors and any "real" states or values were simulated. When this report uses the term "measured" it is referring to quantities that were simulated. This choice was made to better distinguish "estimated" data from "simulated" data.

Overview

The goal of this project is to create a program that can determine the state of an object of a known shape within a LiDAR point cloud. The general approach is to make an initial estimate for the state of the object, determine what the point cloud would be if the object were in that state, and then compare the estimated point cloud to the measured point cloud. If the point clouds are very similar the estimate should be an accurate estimate for the state of the object. If the point clouds do not closely match, then the estimate should be a pour estimate for the state of the object. How well the point clouds match will be assessed using an error function. Starting from the initial estimate, gradient descent will be used to find a state at which the error function is at the minimum. This will correspond to a good approximation of the actual state of the object.

The Sensors

While many LiDAR sensor configurations are possible, this project will focus on a single configuration. The chosen configuration consists of 16 adjacent LiDAR sensors, all oriented in the same direction. This array of sensors will be used to scan for solid objects in a plane parallel to the ground. As such, an object positioned in the scanning plane of the LiDAR array will manifest as a horizontal cross section of the object at the height of the sensor plane. An example of this is shown in Figure 3.

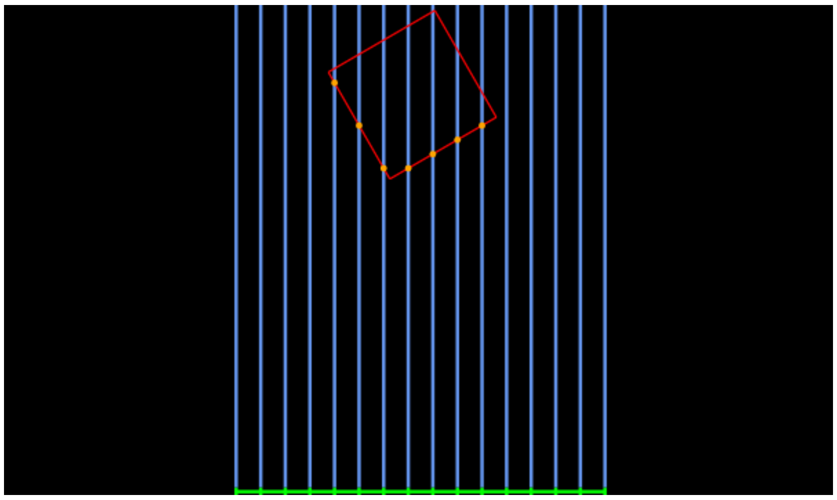
Figure 3: The 16 simulated LiDAR sensors (shown as green dots connected by a green line) firing lasers (visualized in blue) through the scanning place. The lasers reflect off the cross section of an object (show in red) in the scanning plane and return to the LiDAR sensors. The sensor readings are combined to form the LiDAR point cloud (shown in orange).

The Error Function

The error function must represent how similar the estimated point cloud is to the measured point cloud in a single number. The most intuitive way to do this is to make the error function the average distance from each point in the estimated cloud to the nearest point in the measured cloud. Using this function, if the estimated state is exactly correct, each point in the estimated cloud will correspond to an identical point in the measured cloud and the error function will be 0, its lowest possible value. Similarly, if the estimated state varies greatly from the real state, the average minimum distance from the estimated points to the measured points will be high, and thus the error function will have a large value.

A potential problem with this approach might occur when many solid objects are detected in the scanning plane. This could result in the error function having many false minima – states where the average minimum distance is low, but the state is not close to the real state. This will be address as necessary.

The error function will take the following inputs:

- The LiDAR point cloud – a collection of points in $\mathbb{R}^2$.
- A polygon defining the cross section of the object.
- The current estimate for the state, (x, y, r), of the object. x represents the polygon offset in the x direction, y represents the polygon offset in the y direction, and r represents the clockwise rotation of the polygon around the point $(0, 0)$ in degrees.
- The polygon and the current state estimate will be used to construct the estimated point cloud.

Multiple Starting Points

Since gradient descent may lead towards any local minimum, and since the error function may have many local minima, executing gradient descent from a single initial guess does not guarantee convergence to the absolute. In order to account for this, gradient descent will be performed many times starting from initial states evenly distributed throughout the relevant domain.

Implementation

Choice of Tools

The program was created in the C# programming language due to the author's familiarity with the language and the plethora of C# desktop GUI frameworks available. The source code is available at github.com/Andrew-Willms/Lidar-Object-Detection.

Numerical Differentiation

Determining the gradient requires calculating the derivative. Since the error function is non-trivial to differentiate, a numerical approximation of the gradient is used instead. This approximation was obtained using the following formula:

$$\nabla f(x_{\mathrm{n}}, y_n, r_{\mathrm{n}}) = \begin{bmatrix} \dfrac{f(x_{\mathrm{n}} + h, y_n, r_{\mathrm{n}}) - f(x_{\mathrm{n}}, y_n, r_{\mathrm{n}})}{h} \\ \dfrac{f(x_{\mathrm{n}}, y_n + h, r_{\mathrm{n}}) - f(x_{\mathrm{n}}, y_n, r_{\mathrm{n}})}{h} \\ \dfrac{f(x_{\mathrm{n}}, y_n, r_{\mathrm{n}} + h) - f(x_{\mathrm{n}}, y_n, r_{\mathrm{n}})}{h} \end{bmatrix}$$

Where h was chosen to be 10^{-8} $[\mathrm{m}]$ or $[°]$ respectively. Comments on the choice for the value of h can be found in the Discussion section.

Initial Conversion Criteria

The initial implementation used rudimentary conversion criteria were used. If the change in error value and the change in state were sufficiently small between subsequent iterations algorithm would be deemed to have converged. The maximum change allowed for convergence was chosen to be 10^{-3}. Since the units for $(x_{\mathrm{n}}, y_n, r_{\mathrm{n}})$ are meters, meters, and degrees, respectively, this corresponds to a change of 1 $[\mathrm{mm}]$ in the x or y direction and a change of 0.001 $[°]$ in the r direction. These values were chosen because an error of 1 $[\mathrm{mm}]$ or 0.001 $[°]$ degrees was deemed acceptable for most robotics applications.

Preventing Overshoot

During the initial tests of the algorithm the program would frequently get stuck in an infinite loop because the convergence criteria were never met. After studying the output of each iteration of gradient descent the problem was identified. The algorithm would step in the direction of the negative gradient but would frequently step too far and the error at the new state would be higher than the error at the previous state. To prevent this from occurring an additional check was put in place. If the change in state would result in a higher error value, the size of the step would be halved. This prevented the algorithm from overshooting local minima.

Incorporating the Second Derivative

During further testing of the algorithm the program would sometimes converge at local maxima. It was determined that this would occur because near local maxima the change in error and in state were small enough to meet the conversion criteria. To prevent this from occurring, an additional condition was added to the convergence criteria. With this change, for the algorithm to be considered converged, the second derivative (approximated in the same way as the first derivative) was required to be positive. This restriction was added because at a local minimum the function must be concave up and so the second derivative must be positive. With this restriction in place no further false convergences of the algorithm were observed.

Results

Figure 4 shows the results of three different tests of the gradient descent program. The data from the same three tests is shown in Table 1. Table 1 and Figure 4 to show that the program determines the position of the object very well, and the orientation of the object slightly less well. Theories as to the reason for this will be presented in the Discussion section.

Table 1: The true position and estimated position of the object in tests 1, 2, and 3.

	True x	True y	True r	Estimated x	Estimated y	Estimated r
Test 1	1.45	0.5	30	1.453	0.4995	31.50
Test 2	1.55	0.3	45	1.540	0.2993	40.50
Test 3	1.3	0.2	65	1.306	0.2001	67.50

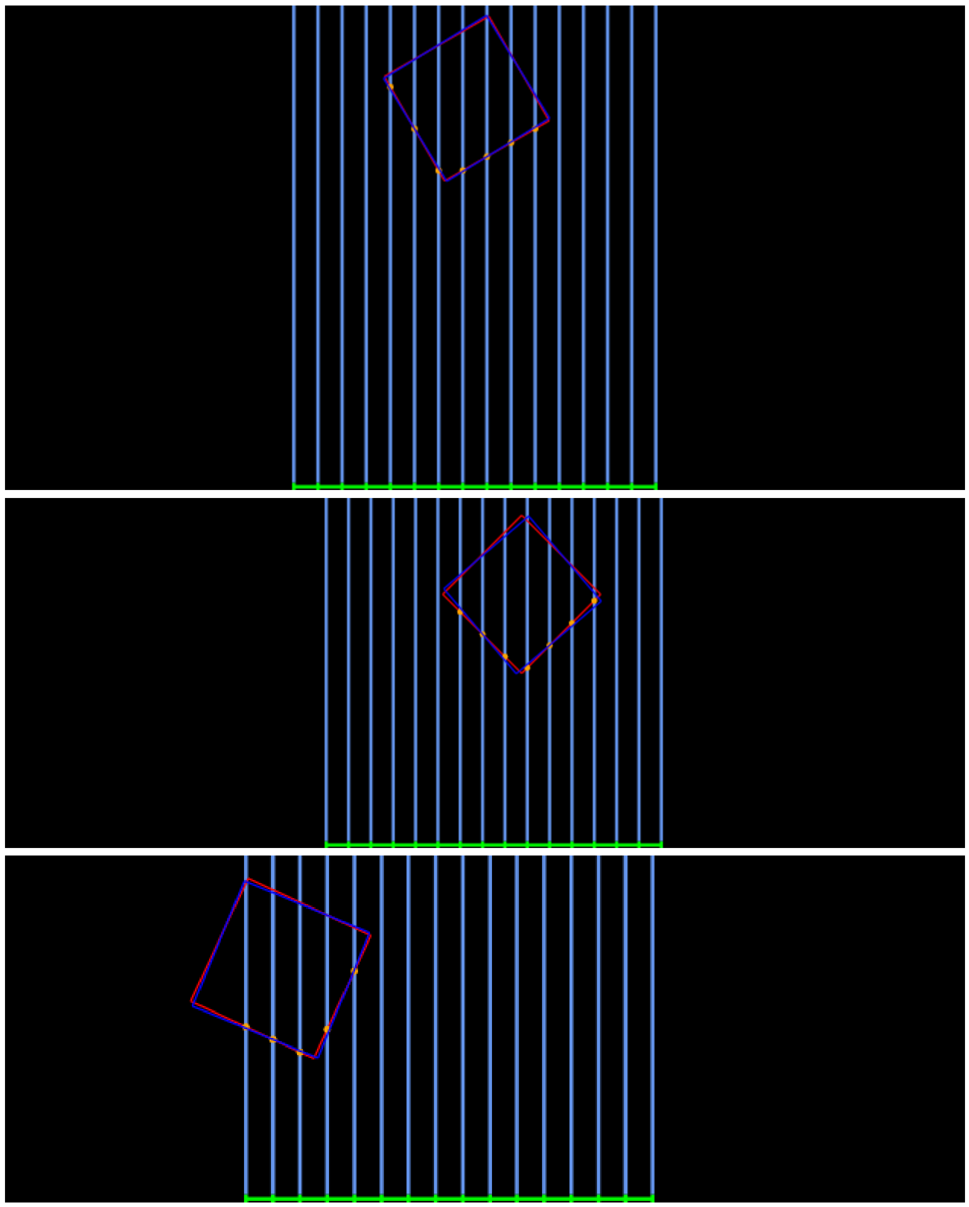

Figure 4: The results of three different tests of the program. The tests are numbered 1, 2, 3 from top to bottom. The original object is shown in red. The position and orientation estimated by the program is shown in dark blue.

Discussion

Inaccurate angle

While the algorithm was very good at determining the position of the object it was less proficient at determining the angle. This is likely because of the units used and the size of step taken when calculating the gradient. When calculating the gradient, the same size step was taken in the x, y, and r directions, even though a one-meter change in the x or y direction would likely have a much larger effect on the error than a one-degree change in the angle. Because of this, the partial derivative in the r direction would likely have been overwhelmed by the other two partial derivatives. This would result in very little change in the r direction when updating the estimated state. To fix this issue, different sized steps should be used when calculating the x and y and r partial derivatives.

LiDAR Error

The LiDAR sensors simulated in this project had no error in their measurements. A valuable extension of this project would be to study effects of using LiDAR sensors with a realistic amount of error.

Performance Optimizations

The algorithm took between 30 and 90 seconds to complete on average. This is woefully inadequate for real time applications such as automation and robotics. Several potential performance optimizations for future exploration are discussed below:

- **Programming Language**: C# was not chosen for high performance or well optimized math libraries. A language designed for high performance math would likely greatly improve performance.
- **Not Retreading Steps**: Since the gradient descent algorithm used many different starting values it is likely that many of these starting values resulted in quick convergence on similar paths. It is possible that considerable performance gains could be achieved be discarding iterations that converge to states other iterations have already tested.
- **Smarter Error Function**: The current implementation of the error function determines the distance to the closest point by testing the distance to every point in the measured point cloud. Using spatial hashing, quad trees, or some other data structure, it is likely that the time required to compute the value of the error function can be meaningfully reduced.
- **Faster Convergence**: While the gradient descent algorithm sometimes converged in 10 iterations or less, other times it took over 100 iterations in order to converge. These very high iteration counts greatly increased the time required to complete the algorithm. Using gradient descent optimizations methods, such as the momentum concept discussed in the Background section, could decrease the number of iterations required to reach convergence.
- **Parallelization**: The current implementation of the algorithm only uses a single thread, and it is likely that the time to completion could be reduced by parallelizing the work. Implementing this should be relatively straight-forward since having multiple starting points provides natural lines of division for the work.

Conclusion

This report outlined a method, based on the principle of gradient descent, for locating objects of arbitrary shape within a point cloud. However, while the basic goal was achieved, vastly more work is required to create a robust gradient descent algorithm than can determine the state of an object at 50 or 100 Hz.

References

[1] M. Ninad and N. Srushti, "Review on Lidar Technology," 12 June 2020. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3604309. [Accessed 28 July 2023].

[2] IBM, "What is gradient descent?," [Online]. Available: https://www.ibm.com/topics/gradient-descent. [Accessed 28 July 2023].

[3] S. H. Haji and A. M. Abdulazeez, "COMPARISON OF OPTIMIZATION TECHNIQUES BASED ON GRADIENT DESCENT ALGORITHM: A REVIEW," *PalArch's Journal of Archaeology of Egypt,* vol. 18, no. 4, pp. 2715-2743, 2021.

[4] R. Kwiatkowski, "Gradient Descent Algorithm — a deep dive," 22 May 2021. [Online]. Available: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21. [Accessed 28 July 2023].

[5] S. Ruder, "arxiv," 15 June 2017. [Online]. Available: https://arxiv.org/abs/1609.04747. [Accessed 28 July 2023].