PROGRESS TOWARDS LIDAR BASED BICYCLE DETECTION IN URBAN ENVIRONMENTS

A Thesis Presented

By

Antonio Antonellis Rufo

to

The Department of Electrical and Computer Engineering

In partial fulfillment of the requirements
for the degree of

Master of Science

in the field of

Computer Engineering

Northeastern University
Boston, Massachusetts

December 2017

1 ABSTRACT

The achievement of level 5 autonomous vehicles on urban roads requires performance equal to that of a human driver in every scenario. In order to achieve this level of autonomy many challenging obstacles must be overcome. In this paper, we will address the specific challenges bicycles pose for self-driving cars in urban environments. One of the most prevalent challenges is detection and tracking of bicycles. Their relatively transparent profile, which changes as the bicycle moves, and their slight frames make detection a difficult problem. Furthermore, their ability to quickly maneuver in cluttered urban environments can generate inaccurate tracking models and faulty prediction estimates. Significant work has been done in sensor and algorithm development to solve the bicycle detection, tracking, and prediction problem, yet problems remain as datasets and algorithm analysis are not accessible to academic researchers. This information is instead considered proprietary. Of the published work in this field, most approaches use idealistic datasets that do not accurately represent real world conditions in order to improve the quality of their results.

To further the development of LiDAR sensors and algorithms this thesis introduces the first open LiDAR dataset, collected in real world environments. Algorithms from various papers and publications are combined to create a unique implementation that performs in real world scenarios. The author presents realistic datasets taken with affordable sensors, along with qualitative performance results of leading algorithms. Easy access to this dataset and analysis allows researchers and developers to create systems and algorithms that perform in real world scenarios.

ACKNOWLEDGMENTS

I would like to thank Northeastern University and MIT Lincoln Labs for affording me the opportunity to complete my Master’s degree. I would like to express my sincere gratitude to my advisor, Professor Hanumant Singh, for the continuous support of my Master’s study and research, his patience, his motivation, his enthusiasm, and his immense knowledge. I could not have asked for a better advisor and mentor for my studies. My sincere thanks also goes to Dr. Alex Vasile. He continuously supported me in my work at MIT Lincoln Labs while I completed my Master’s degree.

Besides my advisors, I would like to thank the rest of my thesis committee: Professor Taskin Padir and Professor Robert Platt, for their insightful comments and hard questions.

I thank my fellow lab mates; Vikrant Shah, Pushyami Kaveti, Tarun Uday, Alek Razdan, Mithun Diddi, Lakshmi Ragupathy, and Abhishek Bajpayee, for their constant help, stimulating discussions, and all the fun we have had.

Last but not the least, I would like to thank my family; Josephine Antonellis, Tony Rufo, Catherine Hamilton, and Rachel Basso for their continuous support and advice.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

2 INTRODUCTION

As autonomous vehicles become increasingly prevalent in society, they need to be able to perform as equals to that of a human driver. Current autonomous driving technology is pushing past level 2 autonomy towards level 5 [1]. In level 2 autonomy, at least one critical driver assisted system is automated, however, the driver must always be ready to take over control of the vehicle. Alternatively, Level 5 autonomy requires driving performance equal to that of a human driver, while operating in all driving conditions. A description of levels zero through five can be seen in figure 1. Presently, many car manufacturers have implemented level 2 autonomy as cruise control and lane-centering; where steering, acceleration, and deceleration are automated using information about the driving environment.

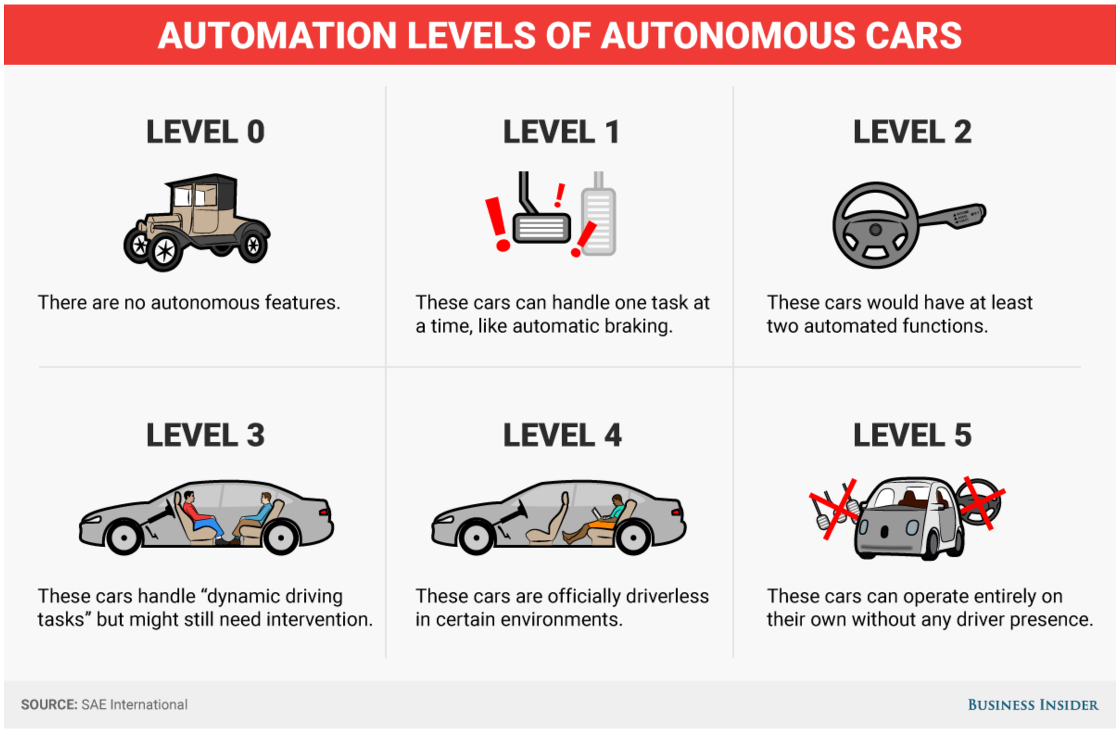
AUTOMATION LEVELS OF AUTONOMOUS CARS
LEVEL 0
There are no autonomous features.
LEVEL 1
These cars can handle one task at a time, like automatic braking.
LEVEL 2
These cars would have at least two automated functions.
LEVEL 3
These cars handle “dynamic driving tasks” but might still need intervention.
LEVEL 4
These cars are officially driverless in certain environments.
LEVEL 5
These cars can operate entirely on their own without any driver presence.
SOURCE: SAE International
BUSINESS INSIDER

Figure 1. Descriptions of levels 0 through 5 of autonomy.

However, there are companies, such as Tesla, that are already expanding automation to include complete steering control and parking assistance [2]. Most of these level 2 autonomy implementations are done using cheap sensors, such as cameras and radars, as seen in figure 2. While these systems greatly decrease the amount of intervention required by the driver, they still fail in many non-ideal driving conditions including urban environments.

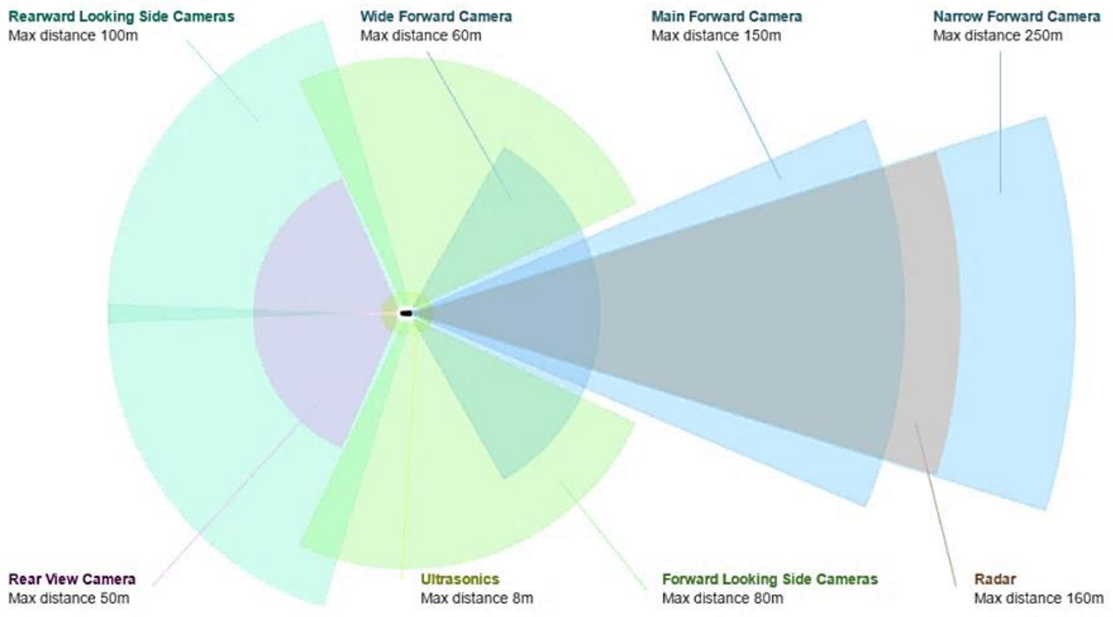

Figure 2. Each sensor and its maximum sensing distance used in Tesla’s autopilot hardware.

Many obstacles have been overcome towards the goal of achieving level 4 or 5 autonomy, however the challenges presented by bicycles in urban environments still pose a significant problem [3]. A bicycle’s relatively transparent profile, small frame, and quick maneuverability pose challenges to the relatively cheap sensors used in most level 2 autonomous cars. Because of this, industry and researchers are investigating other potential sensors that could be used to help accomplish the task.

Figure 3. Bicycles traveling on an urban road in downtown Boston.

One sensor identified as desirable in helping to accomplish this goal is a laser light distance and ranging sensor, known as LiDAR. The current leading LiDAR sensor is the Velodyne HDL-64E which can collect 1,300,000 points per second in a 360-degree field of view [4]. In recent years data collected by this sensor has been the go-to for object detection, classification, and tracking algorithm development across all stakeholders, including research institutes, universities, and industry.

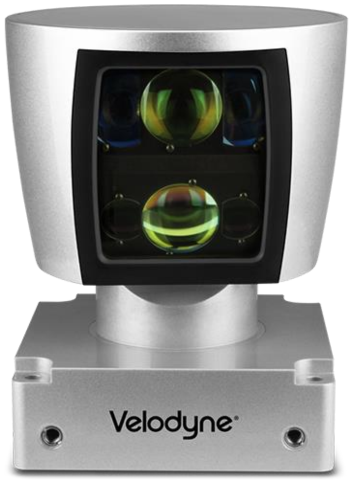

Figure 4. Velodyne HDL-64E LiDAR Sensor.

While these sensors collect high resolution data, the ability to use them for research and in consumer markets is challenging, due to high costs (approximately $75,000), power (60 watts), and size/weight requirements (28 pounds) [4]. With limited access to these sensors, the engineering development cycle that would normally increase sensor performance and reduce the cost of LiDAR has been slowed. This becomes a cyclical problem, as until LiDAR becomes commonly used it will retain its high cost, which is what currently keeps it from being commonly used, Because of this many autonomous car companies are beginning to move towards employing multiple lower priced and poorer performing sensors, in place of a single expensive sensor.

Additionally, due to the competitiveness of the autonomous robotics field, a majority of the datasets and algorithms developed using this sensor are not accessible to the general public but are instead developed as proprietary assets. For example, companies like Google have claimed that they can identify and track multiple bicycles traveling around a vehicle simultaneously however, they only provide images like figure 5 but no software or datasets to back up their claims [5]. This lack of knowledge sharing coupled with the lack of technology access creates a significant problem for researchers trying to further sensor, system, and algorithm development.

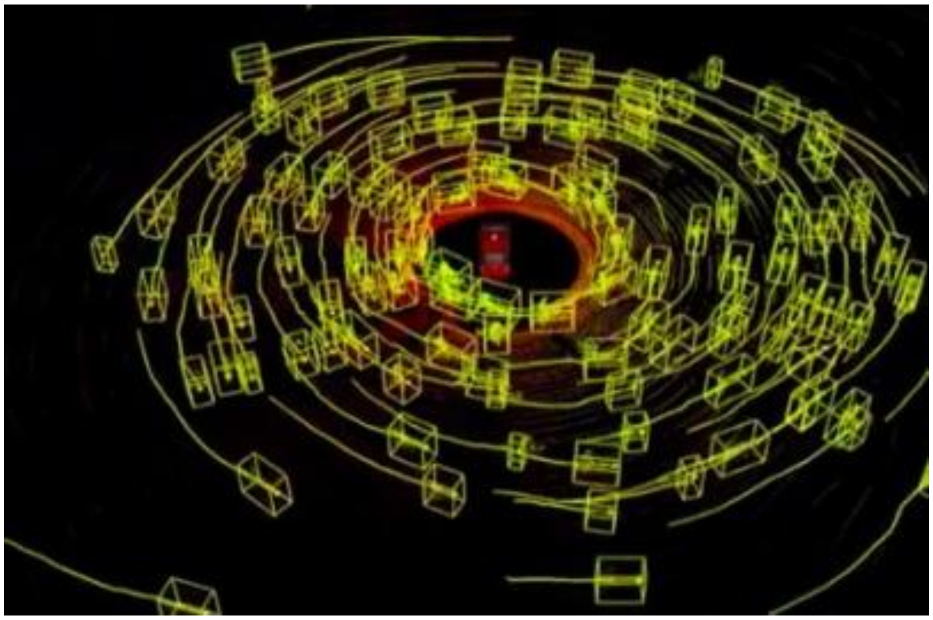

Figure 5. Google's claim towards bicycle and pedestrian detection and tracking.

Although most implementations of LiDAR in autonomous systems are proprietary there are a few published sources for datasets and algorithms [6] [7] [8] [9] [10] [11] [12]. This work exists mainly in the university research space, and while this work does contain insight into the challenges of autonomous driving and sensing, it has significant issues. All of the existing published and public data is taken in idealistic environments, where multiple variables of the test setup are controlled. These variables include background, clutter, number of objects, sensor states, etc. While these idealistic datasets can be useful for theoretical insight into LiDAR and algorithm development, they do not represent an accurate depiction of real world environments. In order for true level 5 autonomy to be reached, datasets and algorithms containing real world environments and applied algorithms must be collaboratively shared.

To further the development of LiDAR sensors and algorithms this paper introduces the first open LiDAR dataset collected in real world environments. The author presents realistic datasets drawn from affordable sensors, along with qualitative performance results of leading algorithms. We first discuss the hardware and describe the scenarios in which data was collected. Next, we analyze a custom variation of leading object detection, classification, and tracking algorithms on simplistic data. We then implement the discussed algorithms on multiple real-world scenarios to qualitatively judge the performance. Lastly, we review the strengths and weaknesses of current LiDAR system and discuss future work to improve them.

3 DATA COLLECTION

The Robotic Operating System (ROS) was used for integration as well as data collection for all the sensors [13]. Descriptions of these different sensors and the collection environment setup are described in the following sections. Additionally, samples of each dataset are displayed. All of the datasets are freely available at the GitHub link provided in the appendix.

3.1 Method of Collection

For the following data collections, a sensor setup incorporating Velodyne LiDAR sensors, cameras, and DataSpeed Incorporated advanced driver assistance systems (ADAS) development kit are utilized. The final vehicle setup can be seen in figure 6. Detailed descriptions of the LiDAR sensor as well as the vehicle are present in the following section. Additionally, various mechanical setups where experimented with to find the optimal mounting location for the LiDAR sensor. Descriptions of these mechanical setups are present in the following sections.

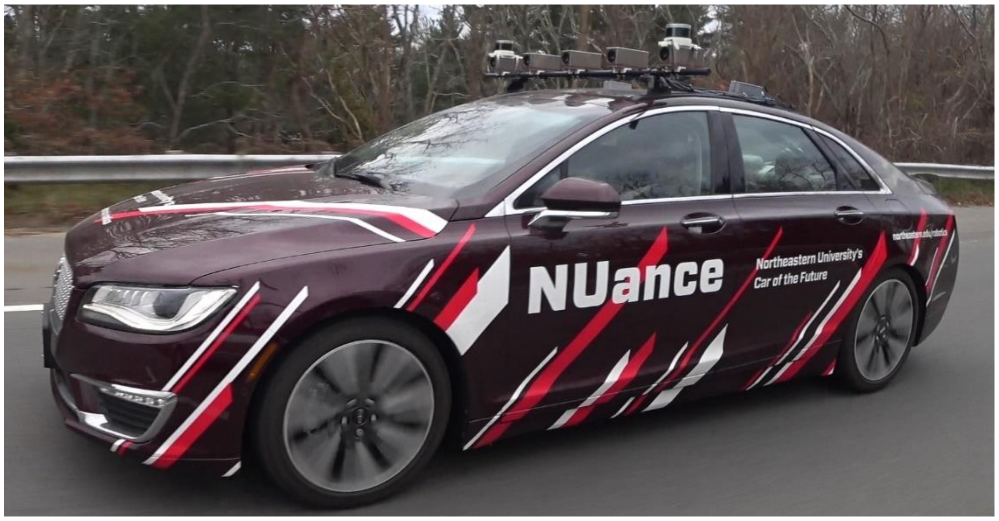

Figure 6. Northeastern University’s Autonomous Car. A Lincoln MKZ with an ADAS kit modified by DataSpeed Inc.

3.1.1 LIDAR

Data was collected using Velodyne Puck (VLP16) LiDAR sensors, as shown in Table 1. It consists of a column of 16 single lasers, covering a pitch range of approximately 30 degrees. It rotates at a rate of 10 Hz, sweeping the complete horizontal ground plane and producing approximately 30000 points per turn [14]. These are currently the most commonly used low cost LiDAR sensors. These sensors are currently present in many self-driving car systems, including Ford’s, Apple’s, Nutonomy’s, GM’s, and various other autonomous platforms. This sensor costs significantly less than the HDL-64E model, priced at $8000 compared to $75,000, while maintaining similar performance quality. A table of the sensor’s performance can be seen below in Table 1 [14].

Table 1. Velodyne Puck (VLP16) LiDAR Sensor and Performance.

	VLP16	
	Range	Up to 100 m
	Accuracy	±3 cm
	Horizontal/Azimuth Angular Resolution	0.1° - 0.4°
	Vertical Angular Resolution	2.0°
	Vertical Field of View (FOV)	±15°
	Horizontal Field of View (FOV)	360°
	Laser Channels	16
	Points per Second	300,000
	Update Rate	5 Hz – 20 Hz

During all of our data recordings the update rate of the VLP16 was set to 10 Hz and the horizontal and azimuth angular resolution were set to 0.2 degrees. The data was recorded in ROS using Velodyne's built in drivers. The ROS topic recorded for the sensor was "velodyne_points". For the initial data collection, we incorporated a single Velodyne Puck (VLP16) LiDAR sensor on a tripod directly connected to a laptop running ROS. This configuration allowed for easy setup and fine tuning of experiments.

3.1.2 Autonomous Car

For our field data collections, we used a VLP16 LiDAR sensor mounted on the roof bar of a Lincoln MKZ Hybrid advanced driver-assistance systems (ADAS) kit provided by DataSpeed Inc [15]. The ADAS kit provides a software development kit for integrating the car's various sensor with ROS (IMU, GPS, etc.). However, this mounting scheme posed geometrical issues. With the sensor mounted in the middle of the roof bar the bar itself along with other cameras interfered with the field of view of the LiDAR sensor.

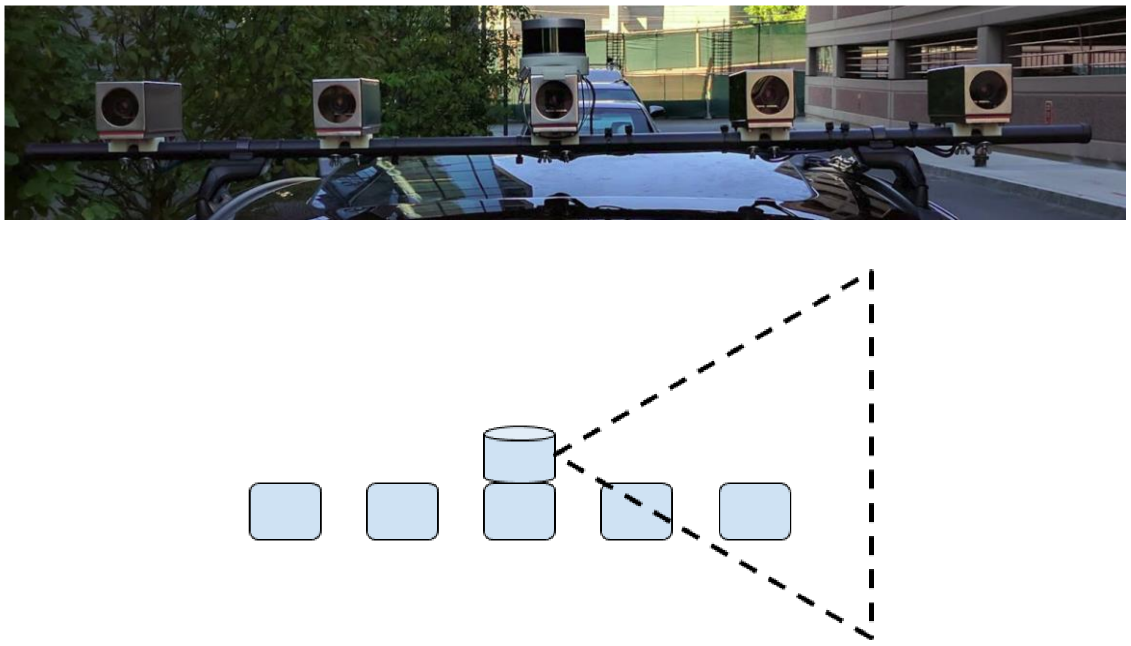

Figure 7. Vehicle Camera and LiDAR Sensor Setup v1.0. The top image is a photograph of the physical setup. The bottom image is a depiction of the blocking caused by the cameras.

The location of the LiDAR sensor in this setup caused the field of view to be blocked by the neighboring cameras. Because of this, objects directly to the left and right of the sensor were occluded in the data. In order to work around this issue, we utilized two Velodyne Puck (VLP16) LiDAR sensors. This setup allows the overlapping data from the two sensors to completely cover the 360-degree field of view around the car. In this configuration the sensors are blocking each other's respective view, however the area is covered by the blocking sensor.

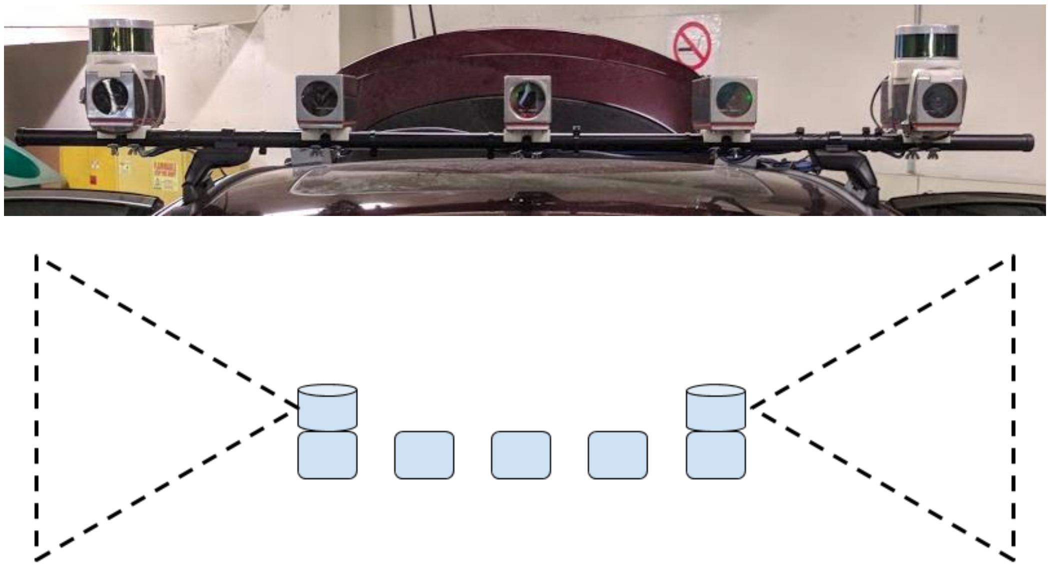

Figure 8. Vehicle Camera and LiDAR Sensor Setup v2.0. The top image is a photograph of the physical setup. The bottom image is a depiction of the blocking caused by the cameras.

3.2 Single Bicycle Data

The first set of data was taken with a stationary VLP16 mounted on a tripod in an empty parking lot. The sensor and laptop were connected and powered by a car which is approximately 1.6 meters from the sensor. For this dataset, a single bicycle traveled in a circle around the sensor at various distances and directions while maintaining an approximately constant velocity. First, clockwise and counterclockwise routes were traversed with the sensor centered in the circle at approximately 1.6 meters in distance.

Second, the bicycle traveled clockwise and counterclockwise with the sensor off of the circle’s center point. This made the bicycle appear close (approximately 1.6 meters away) at one end of the circle and 10 meters away at the other end.

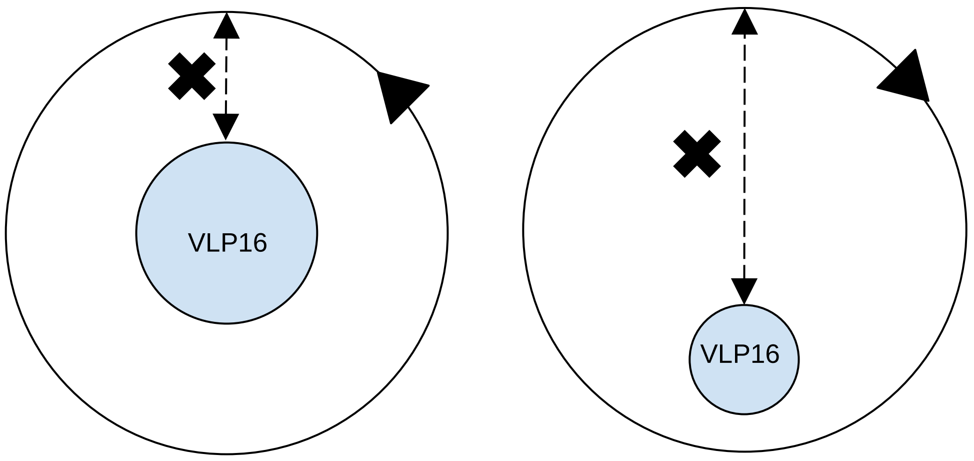

Figure 9. Bicycle Paths (Left: 1.6-meter circles around center, Right: circle with max distance of 10 meters and minimum of 1.6 meters). Not to Scale.

Additionally, data was taken with a stationary bicycle placed in multiple orientations relative to the LiDAR. Left, right, front, and rear viewpoints of the bicycle were recorded at 1.6 meters with a person seated on the bicycle. These datasets were used in the early stages of our data processing to better identify and track the bicycle before it moved relatively longer ranges. We can see the overall point cloud as well as side profiles of the bicycle in both the long distance and close-range datasets in figure 10 and figure 11.

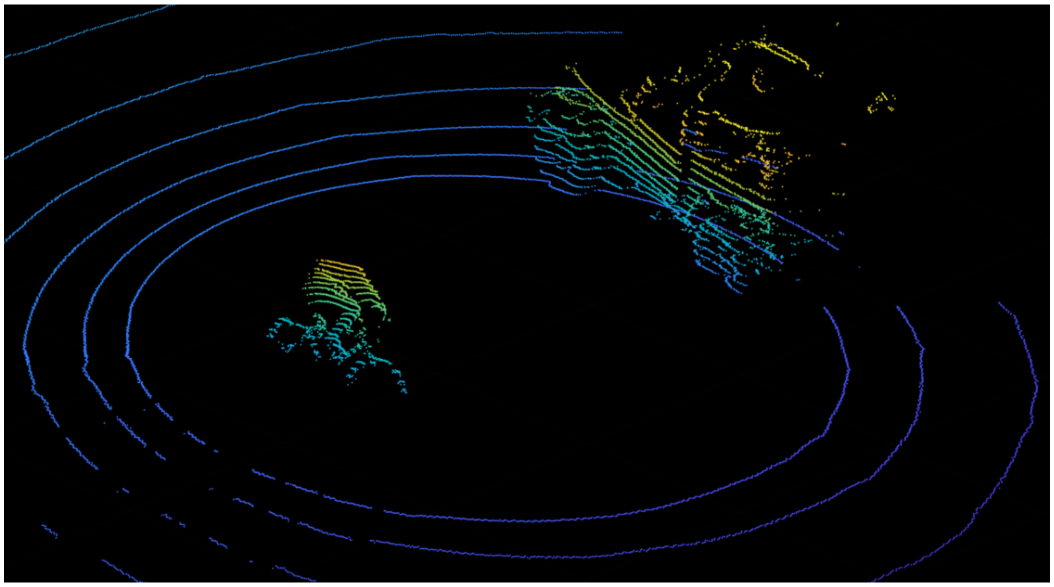

Figure 10. Single bicycle data collection in an empty parking lot.

In the left image below, we can see that when the bicycle is five meters away, all 16 lines of the Velodyne sensor are reflected by the bicycle. However, we also notice that due to the close proximity of the target, some of the desired profile is missing. At this specific viewpoint we are missing the head of the bicycler and the bottom half of the bicycle.

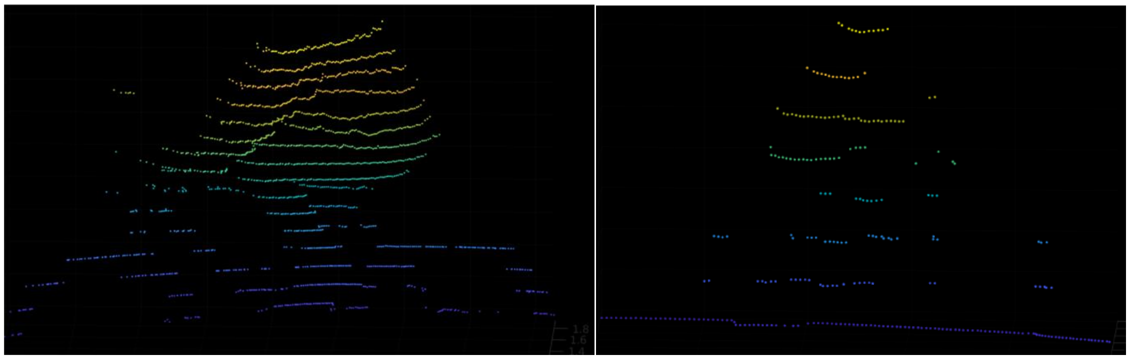

Figure 11. Bicycle Profiles – The left image shows the bicycle traveling in a 1.6 meters counter clockwise circle around the sensor. The right image shows the bicycle traveling at 6.2 meters clockwise around the circle.

Conversely, the data collected with the bicycle at longer ranges (shown in the right image), has significantly less points. Only 8 of the 16 total lines are present on the bicycle at a distance of approximately 6.2 meters. Although the density of the point cloud is significantly lower the entire profile of the bicycle is seen, including the rider's head and the bottom of the bicycle. Links to videos of the data collected along with raw datasets are detailed in the appendix.

3.3 Multi-Bicycle Data

The two-sensor setup on the Lincoln MKZ was utilized for this data collection. The purpose of this data collection was to test the ability to identify and track multiple bicycles in a cluttered environment. The car was parked in the center of a parking lot and 5 bicycles were driven around the car in various directions, distances, and speeds. The bicycles would also periodically synchronize their routes around the car so as to test the ability to identify individual bicycles clustered in a group.

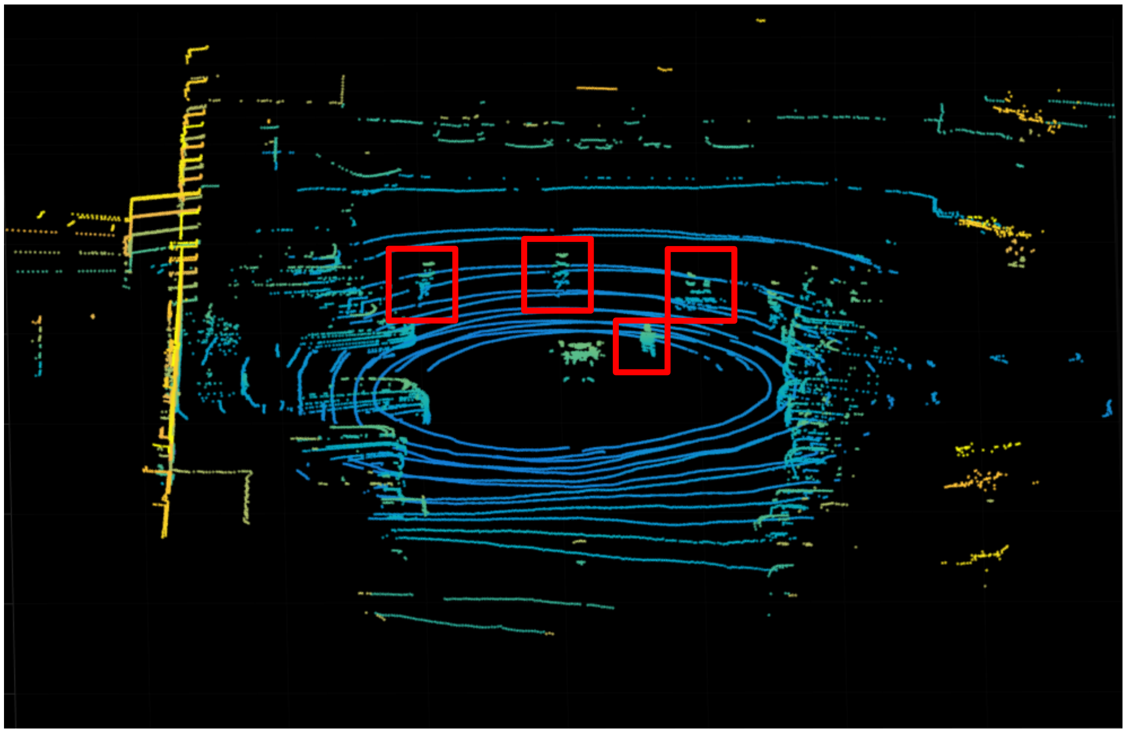

Figure 12. Multi-bicycle data collection. A single frame shows the bicycles traveling in various directions at various distances from the sensor.

In Figure 12, we can clearly see 4 bicycles, outlined in red, within the field of view of the two LiDAR sensors (circled in red). In actuality there are 5 bicycles traversing around the sensor. One bicycle is located just to the left of the car and its proximity is so close that only the head of the bicycler is visible. Figure 13 shows another frame from the same dataset. In this point cloud frame the bicycles have traversed around the car and their direction has changed as well.

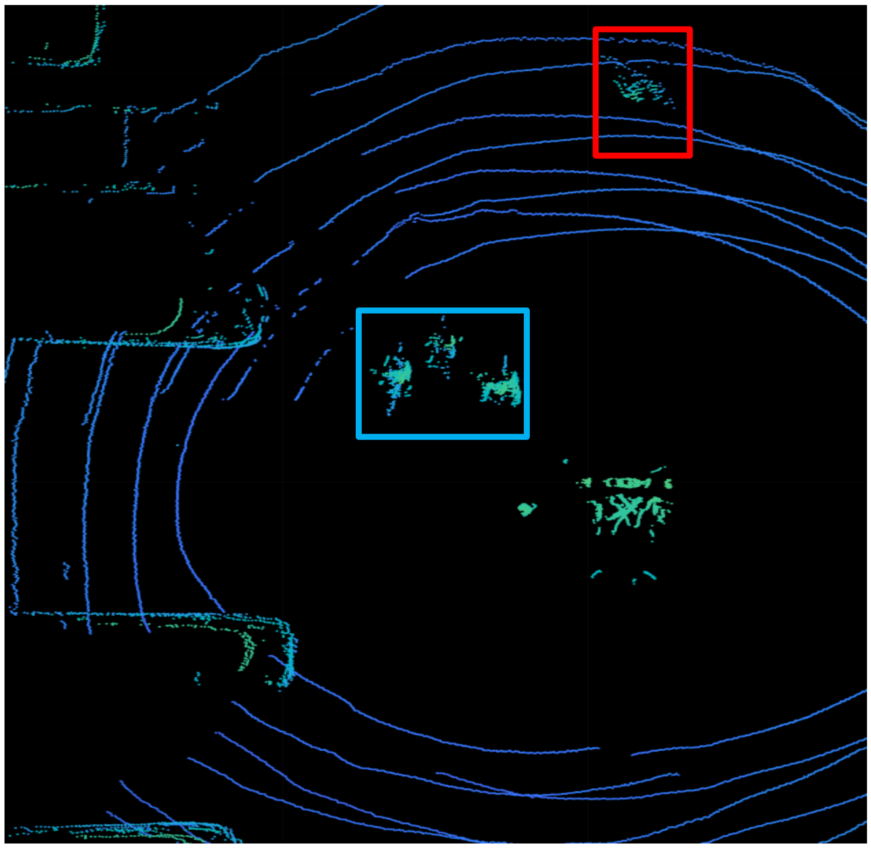

Figure 13. Multi-bicycle data collection. A single frame shows three bicycles traveling together as a group (cyan) around the vehicle and a single bicycle (red) traveling alone.

In this frame three of the bicycles (highlighted in cyan) are significantly closer together while one of the bicycle (highlighted in red) is traveling by itself. If we observe the same frame from a more acute viewing angle we can see the difference in the profiles of each bicycle.

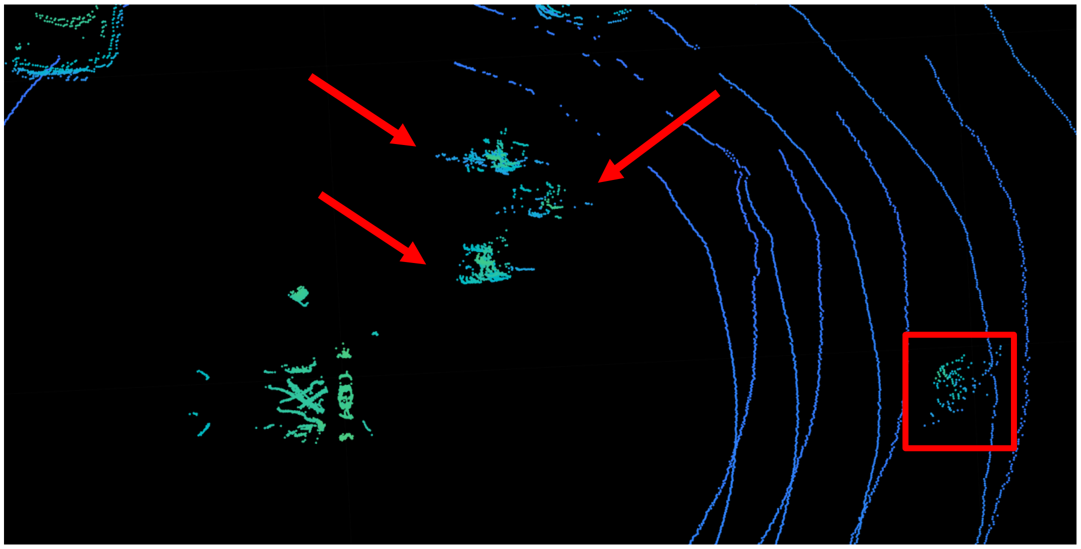

Figure 14. Multi-bicycle data collection. A zoomed in single frame from LiDAR data shows 4 bicycles traveling in various direction around the car. The group of bicycles are individually labeled with red arrows. The single bicycle is outlined in a red square. The effect of distance from the sensor to the number of points on target is evident when comparing the bicycles.

The bicycle that is traveling alone (furthest from the car) has a limited number of data points. This coincides with figure 11 from the single bicycle data collection. Additionally, we can see the effects of bicycles that travel close together in the data product. Of the 3 bicycles traveling together, the first shows up with the densest collection of points. However, since that bicycle is closest to the car it also has the smallest visible profile. Looking at the second bicycle in the group we can see that its entire profile is visible with points distributed evenly. While the third bicycle in the group is farthest from the car it is still missing a significant amount of points from its profile. This is due to the first bicycle in the group blocking the LiDAR sensors ability to see the entirety of the third bicycle. This situation is extremely common in LiDAR sensors and creates difficulties in distinguishing groups of targets from their individual components. Links to videos of the data collected along with raw datasets are detailed in the appendix.

3.4 Driving Data

The two-sensor setup on the Lincoln MKZ was utilized for this data collection. The purpose of this data collection set was to collect data in real world urban environments. Data was recorded during commuter hours (5-7pm) in the Roxbury and South End communities of Boston.

The first route was 0.8 miles, starting near Ruggles Station and ending on Botolph Street. The drive started going East down Columbus Avenue. This route has 2 traffic lanes in each direction with street parking on both sides as well as bicycle lanes. There is also a considerable amount of foot traffic across the road. The route then took a left through the major intersection of Columbus Avenue and continued North down Massachusetts Avenue. This road is also a two-lane street with street parking and bicycle lanes. The drive then ended after taking a left going west on Botolph Street.

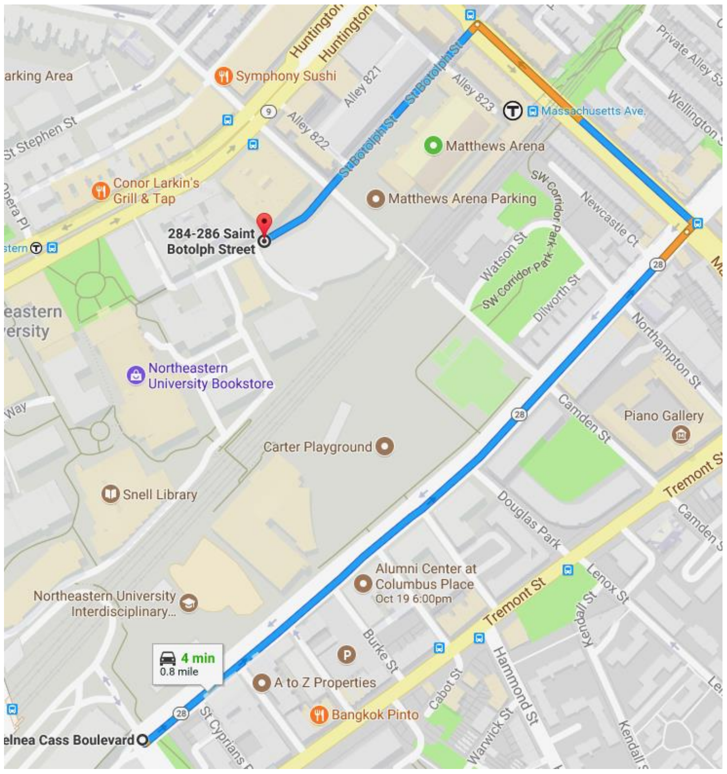

Figure 15. Driving Route A. Recorded between 5-7pm. Total length of 0.8 miles. Starting location at 805 Columbus Ave and ending at 284 Botoloph Street.

The second route again started near Ruggles station and continued east down Columbus Avenue, turning around at the intersection of Broadway street and returning Westbound on Columbus Avenue towards Ruggles station. This route was 1.2 miles long, containing 2 traffic lanes in each direction with street parking as well as bicycle lanes on both sides of the road.

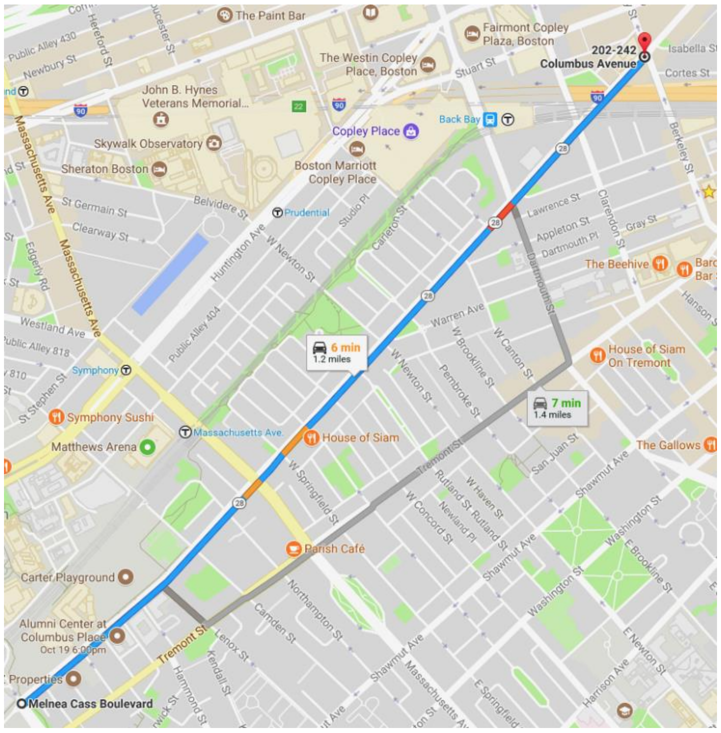

Figure 16. Driving Route B. Recorded from 5-7pm. Total length of 1.2 miles. Starting location at 805 Columbus Ave and ending at 242 Columbus Ave. This route was repeated multiple times while traveling in both directions.

If we look at a random frame from the second route we can see parked cars on both sides of the road, cars traveling in our lane, cars traveling in the opposite direction, and a bicyclist (highlighted in red).

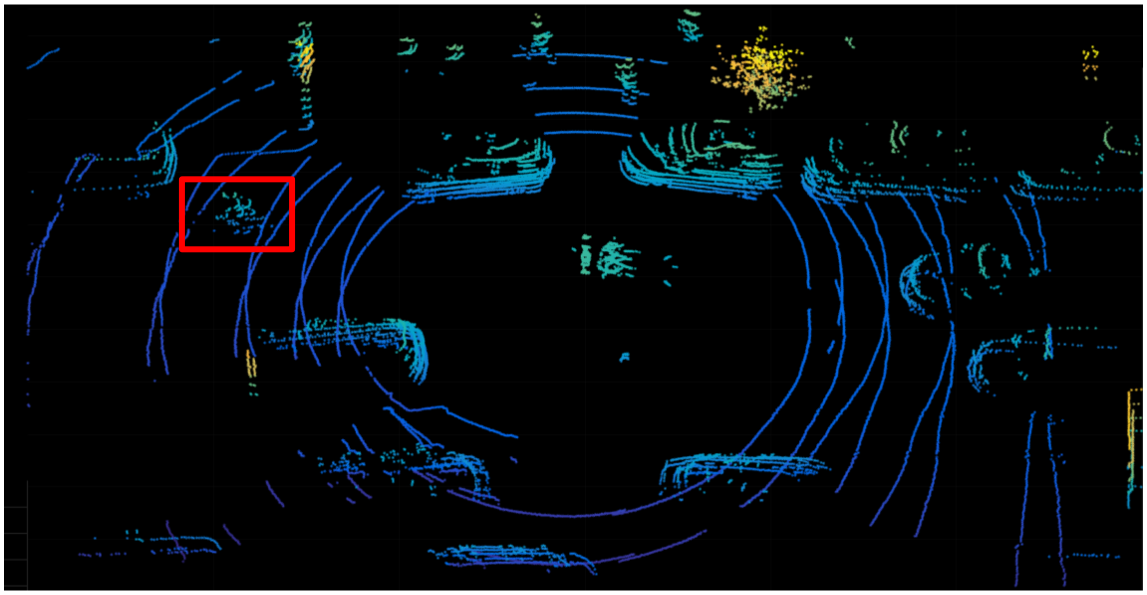

Figure 17. Driving route B data collection. The passing bicycle is highlighted in red.

If we take a closer look at the bicycle in this frame we can see that most of the points fall on its rear profile. There are only 5 lines present on the bicycle in figure 18 and figure 19. Links to videos of the data collected along with raw datasets are detailed in the appendix.

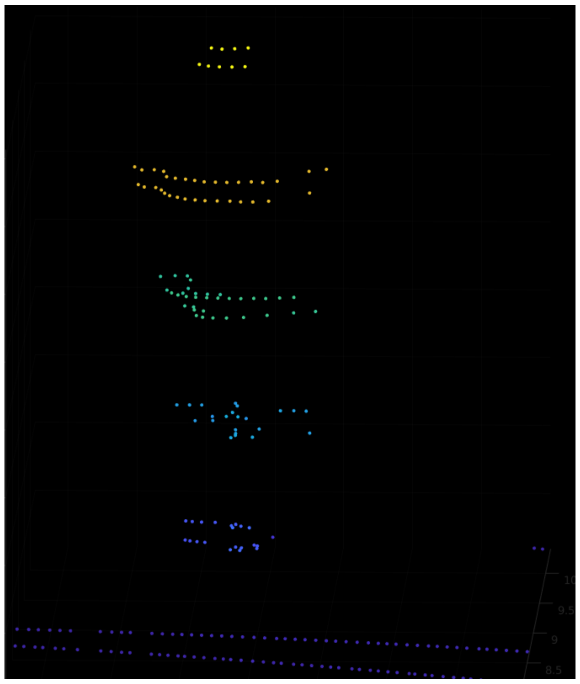

Figure 18. Rear view profile of a bicycle from driving route B. Only 5 lines of data points fall on the bicycle.

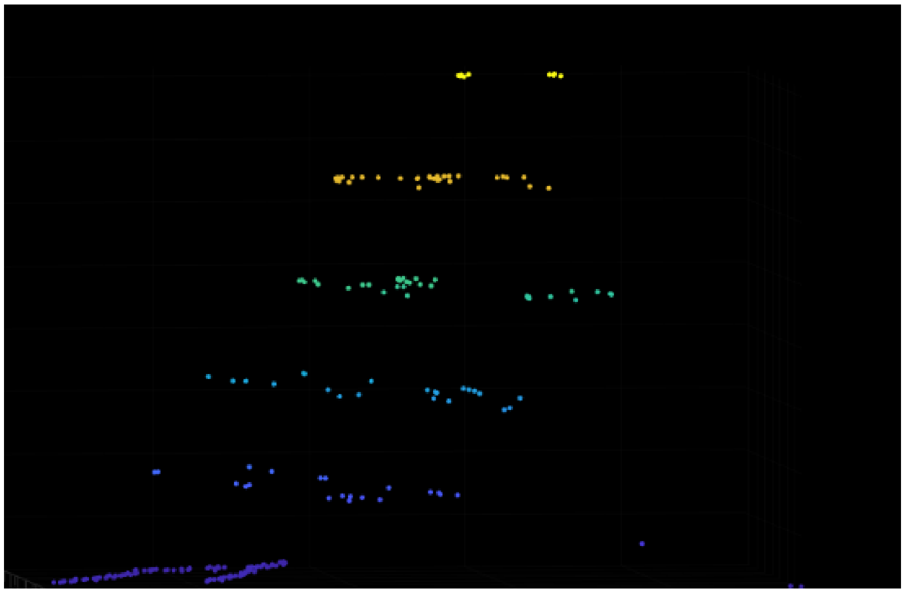

Figure 19. Side view profile of a bicycle from driving route B. Only 5 lines of data points fall on the bicycle.

4 BICYCLE DETECTION (METHODS)

For processing of the data, we utilized a general detection, classification, and tracking approach proposed by Azim [6]. We combine this approach with other detection techniques and a unique implementation to support our sensor setup, development environment, and real-world scenarios. We first take the point clouds from each sensor and calibrate them to align them to each other. We then break down the point cloud into voxels using an Octree data structure. Once the tree structure is created the volume and occupancy of each voxel is calculated to find movers. The moving voxels are then merged to better represent the points the encompass. These voxels can then be classified based on various properties. Lastly, we track the location of these voxels over time. Each processing step is described in detail in the sections below. Links to the software developed for this thesis as well as documentation are detailed in the appendix.

4.1 Calibration

Once data from both sensors is recorded, the next step is fusing the data. In order to register the two-point clouds, we use a translational and rotational transform, referenced from the base-link of the car. In the ROS environment a three-dimensional model of the car and its components are represented (including the doors, wheel, etc.) in reference to a base-link. The base-link of the car is fixed to the center of the rear wheel axle. From this reference point, we measured the translation and rotation offsets to the two Velodyne sensors and inputted them into a static transform function in ROS. The resulting product is a single fused point cloud that encompasses all the points of the two separate LiDAR sensors.

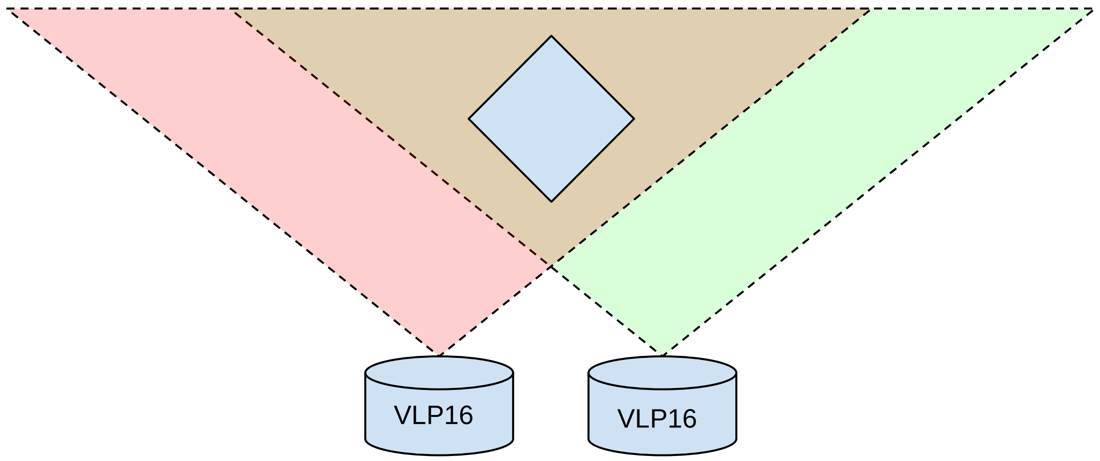

Figure 20. Calibration setup. The field of view of both LiDAR sensors overlap with the calibration target. Port sensor is red and starboard sensor is green.

In order to calibrate the effectiveness of our transform, we set up a man-made object that would allow us to accurately align the two point clouds. The object used was a tall rectangular box with one of its corner edges pointed directly between the sensors. We would then use this edge to accurately fuse the point clouds in X and Y dimensions while using the top of the box to fuse the data in the Z dimension.

For clarity, the point clouds for the port and starboard LiDAR sensors have been colored red and green respectively. In the left image we can see the combination of the two points clouds with no transform. Both sensor’s point clouds have their data points originating from the 0,0,0 coordinate in XYZ space. Because of this the calibration target (box) is seen twice in XYZ space; once for the port sensor (red) and once for the starboard sensor (green).

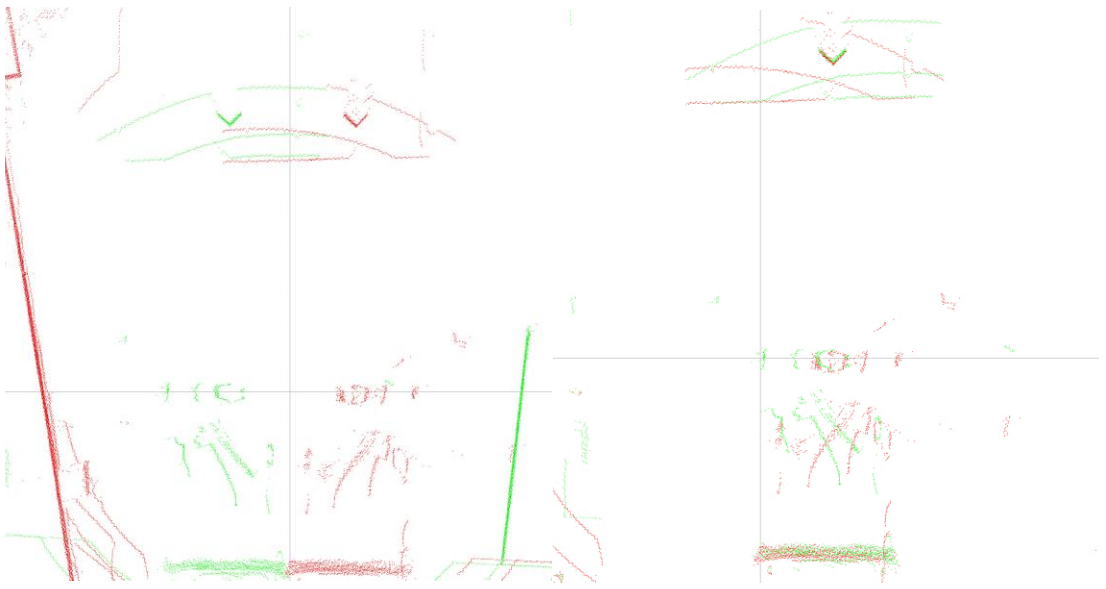

Figure 21. Calibration applied to LiDAR sensor data (Left: Unregistered or transformed point clouds, Right: point clouds after calibration transformation and registration). The transformations computed by the calibration processes align the data in both point clouds.

In the right image we can see the product of fusing the two point clouds using the measured translational and rotational transforms. The centers of each sensor have now moved to their respective sides and the calibration target is only visible in one place in XYZ space. In order to perfect the transformation, we iteratively adjusted the values of the transform so that the points reflected from the calibration target from both LiDAR sensors is reduced to less than the accuracy of the sensor’s themselves. This iterative process gives us a transform that produces nearly accurate fused point clouds. After this rough transform has been computed we use an iterative closest point algorithm to calculate any further transformations to align the point clouds [16] [17] [18].

4.2 Octrees

After the data is fused, we break the point cloud into smaller section for simpler manipulation. In order to efficiently break down the point cloud we use a method called octrees. An octree is a data structure in which each internal node has eight children and is recursively divided into eight children [19]. This three-dimensional partitioning is continued through the data structure until a minimum voxel size or point density is achieved as seen in figure 22. These smaller cubic volumes are called voxels and will be referred to as such for the remainder of this paper.

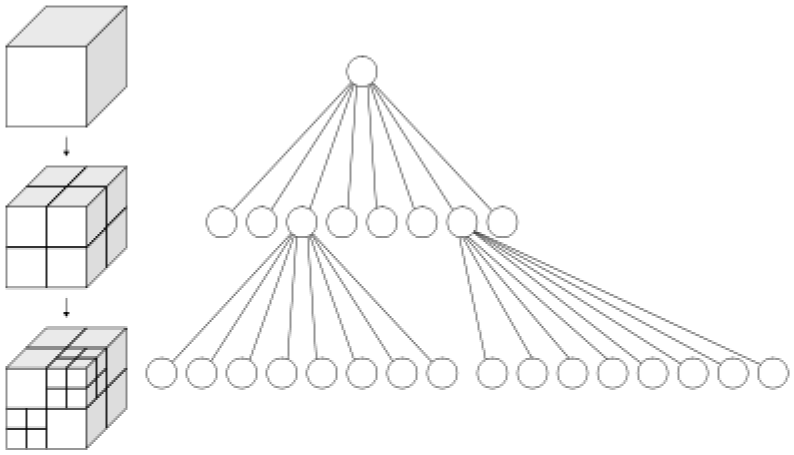

Figure 22. Generic standard Octree structure representation.

Additionally, these tree structures can be broken down equally or with weights. If broken down equally, each voxel will have the same cubic volume as the other children within its subtree. Conversely, if the weighted division is used, then the size of the voxel within one subtree can vary in cubic volume [19]. One section of the cubic volume could contain mode points and therefore be a larger voxel than its neighboring children, while other areas could have less points and therefore contain smaller voxels. We can see an example of this in figure 23. Even though one voxel is significantly larger than its neighbors the division still equates to eight total voxels in the subtree.

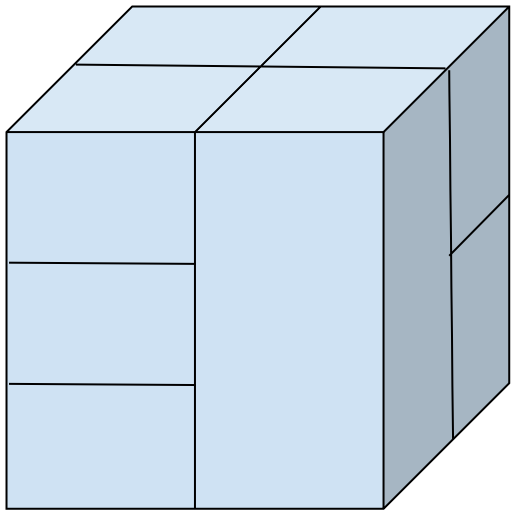

Figure 23. Generic weighted Octree structure representation. The subdivision no longer maintains the same size.

After the voxels are created we perform a shrinking function. This reduces the size of the bounding box for each voxel to tightly encompass the points within it [20]. This operation creates gaps in the final space not covered by a voxel as seen in figure 24. However, this is advantageous, as it removes areas with either zero or an insignificant amount of points. Additionally, this reduces the overall size of the octree structure, making computation easier.

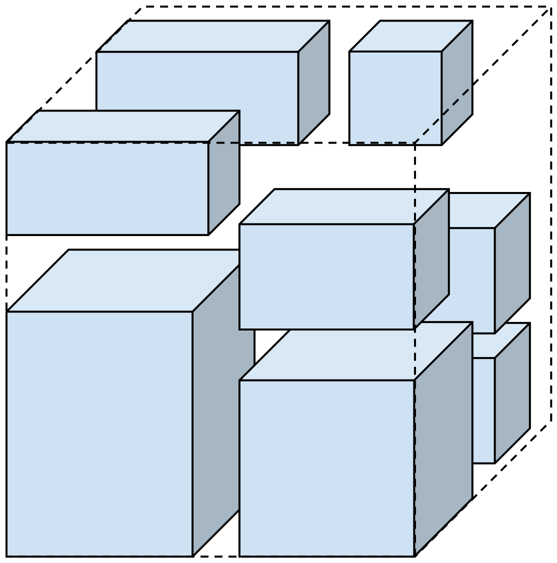

Figure 24. Generic weighted and shrunk Octree structure representation. Empty spaces are present in the three-dimensional structure.

We can look at a real octree as an example of the equally weighted mode as seen in figure 25. We view the 3-dimensional voxel plot as a 2-dimensional plot for simplification. In this example all of the voxels within a subtree are of equal size. In some areas the voxel is not broken down into sub-voxels; this is because the algorithm determines that the voxel has either reached its minimum size or it does not have enough data to justify breaking down. Additionally, the voxels line up perfectly across any single axis because they are weighted equally.

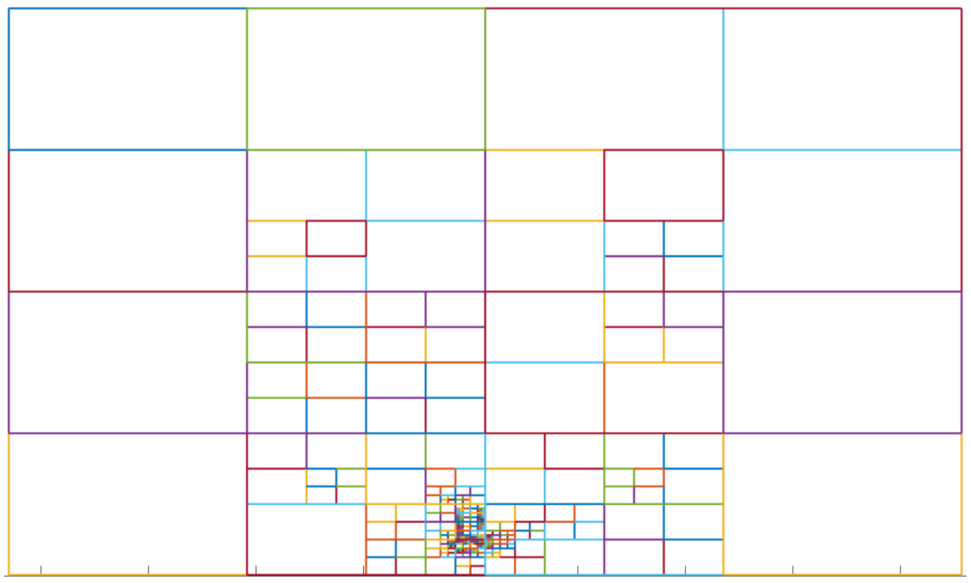

Figure 25. Standard Octree of single bicycle dataset. Two-dimensional representation of dataset.

If we then weight the octree based on the data within it, the resulting graph is significantly different, as seen in figure 26. From this we can denote that some voxels are significantly larger than those in the equally weighted tree, while other voxels are significantly smaller. Additionally, voxels look to be overlapping or crossing in this figure. This is due to the fact that the size of the voxels in X, Y, and Z can be different and therefore the voxels might not line up when observing them from a specific angle.

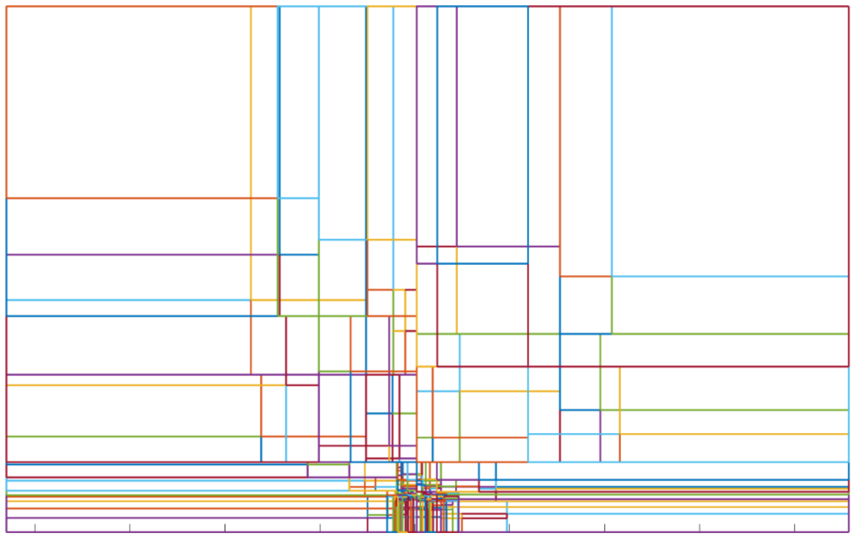

Figure 26. Weighted Octree of single bicycle dataset. Two-dimensional representation of dataset.

Lastly, we can look at the effects of shrinking the octree so that each voxel tightly encompasses the data points within it. From figure 27 we can see that voxels are now “floating” in space instead of bordering their neighbor voxels. This drastically reduces the time required to analyze each voxel, but it also creates gaps in the data where no voxel can reach. While this might be a detriment in some cases, for our scenario this removes most if not all the empty space of our point cloud.

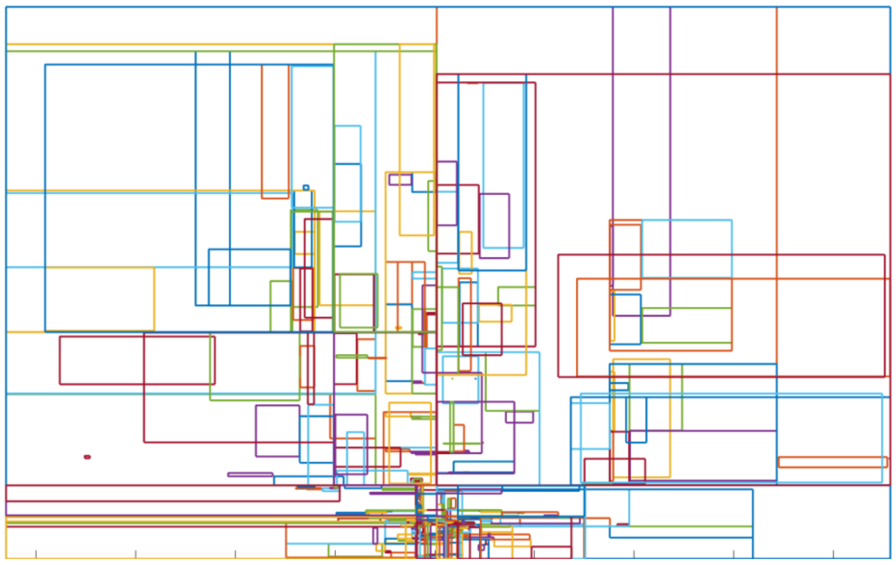

Figure 27. Weighted and shrunk Octree structure for single bicycle dataset. Two-dimensional representation of dataset.

4.3 State Estimation

Once all the voxels have been calculated for the current frame we compute the point volume for each of these voxels. We then compute the point volume of those voxel locations in the previous LiDAR frame. If the volume of the voxel is above a threshold then we mark that voxel as occupied, otherwise we mark the voxel as free. This builds a basic SLAM map for the current frame [21].

$$S_{t,(x,y,z)} = \begin{cases} Free \\ Occupied \end{cases}$$

Next, we compare the point volumes of the two corresponding voxels from each frame and determine whether or not the voxel is considered a moving object [6]. Let us denote the current state as S_t and the previous voxel state S_{t-1}. If the previous state of a voxel is marked as free and the current state is marked as occupied then this is the case where a moving object is possibly present. If the previous voxel state S_{t-1} is considered occupied and the current voxel state S_t is free then we can't determine anything about the presence of a moving object. If the voxel state between any two frames is continuous then we can assume the area is either free of movers or the object is not moving. This method generates a large number of false positives, but they are filtered out and removed in the following sections.

Once the voxels are calculated, we run them through a density and state estimation algorithm using the previous frame [6] [21]. Figure 28 is the output of that step. Red points correspond to movement and grey points are the background.

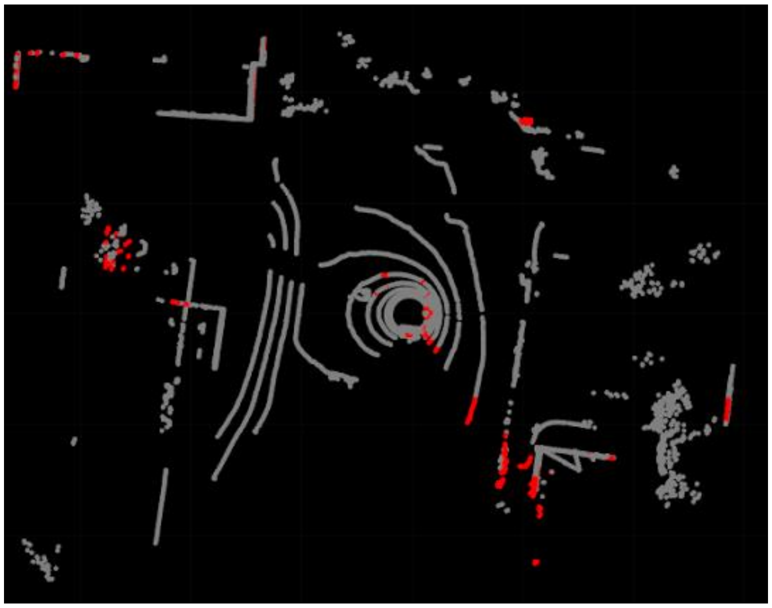

Figure 28. State estimation map for single bicycle dataset. Grey identifies non-occupied stationary points. Red identifies moving occupied points.

4.4 Merging Voxels

The next step is to reduce the massive number of voxels determined to be movers by ruling out the false positives. We accomplish this by combining moving voxels that belong to the same object [6]. In order to determine if the moving voxels belong to the same object we calculate their Euclidean distance from each other as well as and overlap between the areas in three-dimensional space in which they occupy. The Euclidean distance between the voxels is computed using their centers as seen in the following equation.

$$D_{i,i+1} = \sqrt{\left(V_{i+1,X} - V_{i,X}\right)^2 + \left(V_{i+1,Y} - V_{i,Y}\right)^2 + \left(V_{i+1,Z} - V_{i,Z}\right)^2}$$

If the distance calculated is less than a threshold then the moving voxels are combined into a larger voxel encompassing the entirety of the smaller voxels. While this works well for most cases there is a failure case when two voxels encompass each other but the center of one of the voxels is farther away than the threshold.

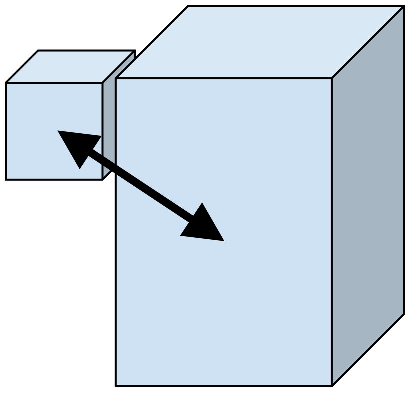

Figure 29. Euclidean Distance of the centers of two neighboring voxels.

To compensate for this, we calculate if the voxel has overlapping areas. If their areas overlap then we check to ensure that their distances are not at an unrealistic distance and combine them as well.

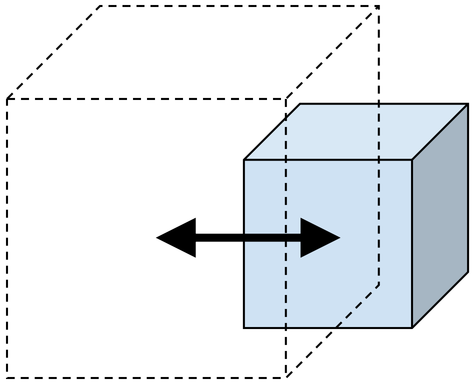

Figure 30. Overlapping voxels merged if the distance between there centers is within a threshold.

This process of merging voxels is repeated until the nearest voxel is outside of the determined threshold. We then move to the next voxel and check its neighboring voxels. The merging process is repeated until all voxels have been combined appropriately. If we now look at the voxels that encompass these points we can see that some sets of data points which look to be from the same object are actually split into different voxels as seen in figure 31.

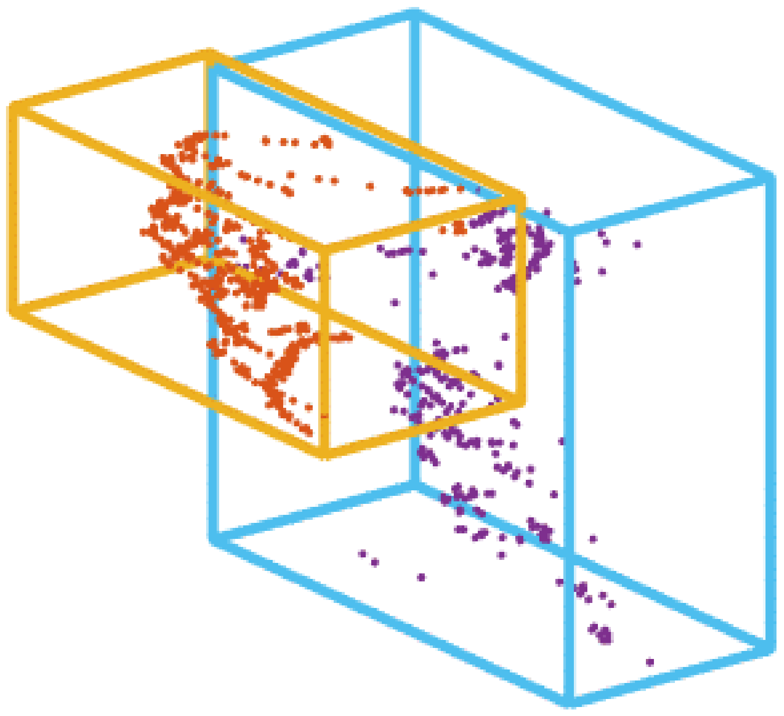

Figure 31. Unmerged voxels of a single bicycle.

A real-world dataset of a similar situation can be seen below in figure 32. On the left we see a car with multiple voxels covering front. On the right we can see a bicycle cover in a large number of voxels that are both neighboring or intersecting.

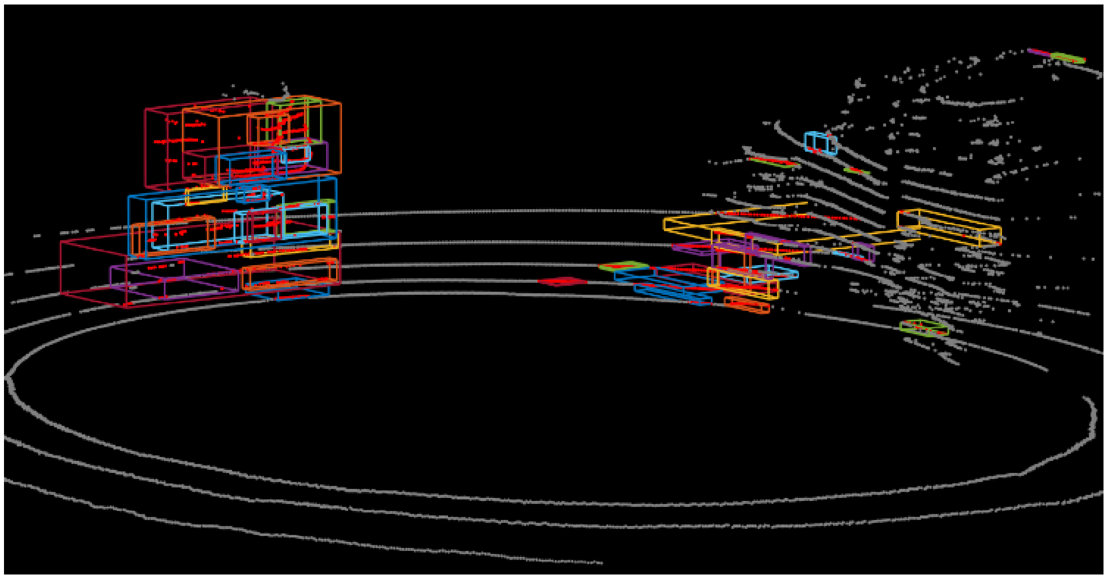

Figure 32. Unmerged voxels from single bicycle dataset.

The points within these neighboring and intersecting voxels are most likely from the same object and therefore their respective voxels should be combined. Figure 33 shows the merging of two voxels from figure 31. After the voxels have been merged a bicycle profile becomes more apparent than in either of the individual voxels shown above. This is important when we move to the classification step in the next section.

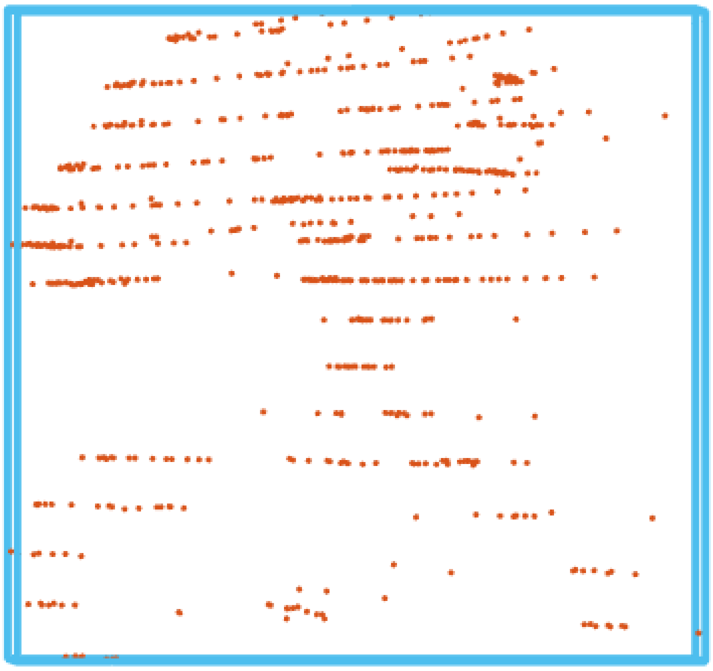

Figure 33. Merged voxels of a single bicycle.

In figure 34 we can see that the voxels that composed the bicycle have been combined into one voxel that accurately represents the bicycle inside it. Additionally, the voxels that encompassed the car have been combined into two voxels. The yellow voxel was not combined with the green voxel because it contained points relatively far from the center of the green voxel and therefore did not meet the merging criteria. Additionally, these voxels are need to be removed in the classification step as they don’t represent a moving object.

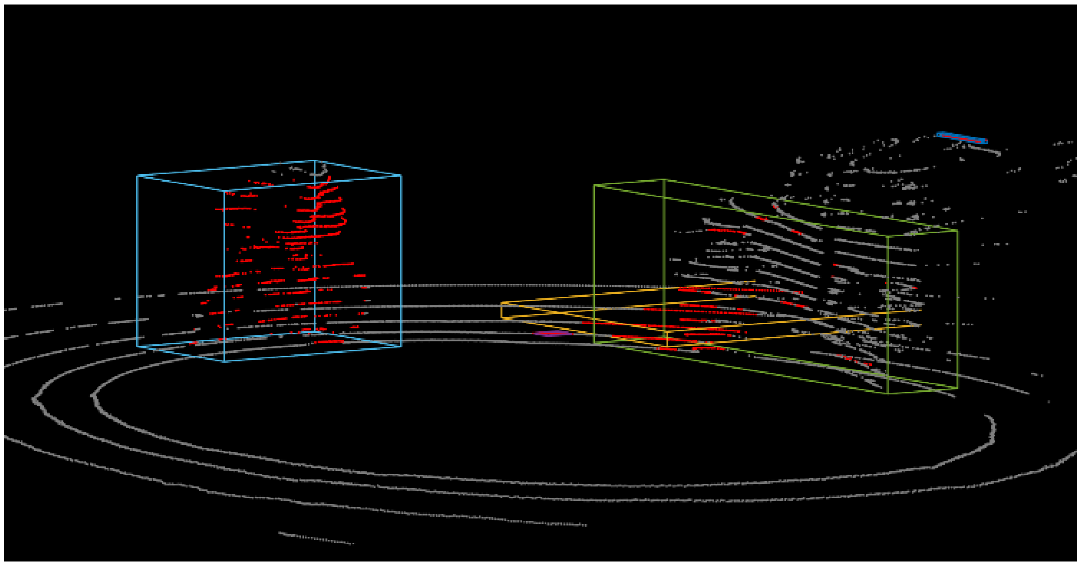

Figure 34. Merged voxels from single bicycle dataset.

4.5 Classification

After the moving voxels have been merged appropriately we classify the voxels based on a set of parameters. We first remove any voxels that are too large or too small to be real objects of interest. This removes a majority of the noise produced by the state estimation and voxel merging sections of the algorithm.

Classification is then performed on the properties of the bounding box of each remaining moving object. Labels are determined based on the width and height of the bounding box and the ratio between them as an indicator for its class [6]. We have defined these parameters for cars, buses/trucks, bicycles/motorbikes, pedestrian, and groups. The classification designated as groups is specifically used for multiple bicycles or pedestrians traveling together or in proximity of each other. An example of a bicycle, car, and group class are visible below.

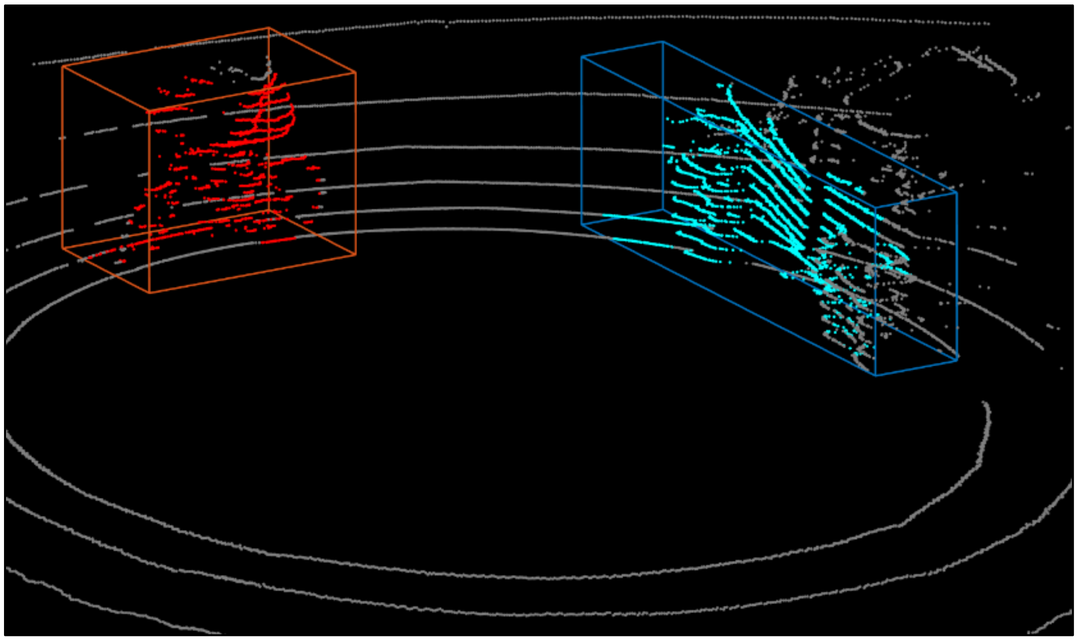

Figure 35. Object classification from single bicycle dataset. Red represents bicycles and cyan represents groups.

Once the classification occurs most of the unwanted boxes are removed and only objects of interest remain. Red is used to depict bicycles, blue for cars, green for people, and cyan for groups or unknown objects. In figure 35 we see that the majority of the bicycle is classified correctly however there are two errors. First, part of the rear wheel and cyclists head are not included in the classification. Secondly, the car not moving and is incorrectly classified as a group.

4.6 Tracking

The final step is to track all moving objects over multiple frames. To accomplish this, we implemented a Hungarian particle tracking filter [22]. The filter operates on the same principle as a global nearest neighbor filter however, it utilizes combinatorial optimization to solve the assignment problem. The algorithm creates a matching score matrix S for each tracker and detection (denoted tr and d respectively) [22]. This matching score is calculated from function s(tr,d) which evaluates the distance between detection d and each particle p of tracker tr [22]. This function employs a classifier $c_{tr}(d)$, a gating function $g(tr, d)$, and a Normal distribution evaluated for the distance between d and p represented by $pN(d - p)$ described in [22].

$$s(tr, d) = g(tr, d) * \left(c_{tr}(d) + \alpha * \sum_{p \in tr}^{N} pN(d - p) \right)$$

The pair (tr,d) with the highest score is iteratively selected until every detection is assigned to a tracker. The associated detections are then filtered for a matching score above a threshold, ensuring that a selected detection actually is a good match to a target [22]. In our implementation the detection is actually a set of points X, Y, and Z corresponding to the center point of a particular voxel. The filter takes in the center points of the moving voxels as X, Y, and Z coordinates and outputs the corresponding track ID. A two-dimensional example of the Hungarian particle filter is given in figure 36.

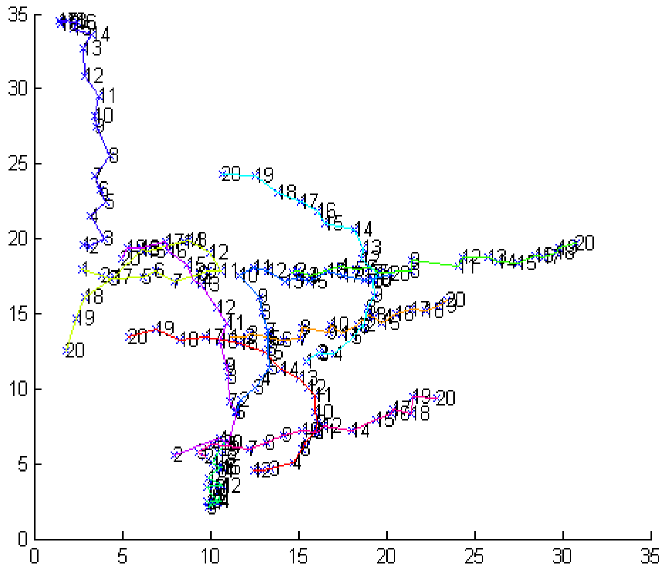

Figure 36. Two-dimensional simulation of Hungarian particle tracking filter.

After the all of the points have been assigned to tracks we filter them to remove any tracks shorter than a predetermined length. This helps remove any tracks that belong to false positive moving voxels. Additionally, as the list of tracks increases frame to frame our filter removes tracks that cross between objects, ensuring all points on a track correspond to the same voxel object. Additionally, we hold on to the tracks for a predetermined amount of time to help accurately predict the next location of the track. An example of this operating on our datasets can be seen in figure 37.

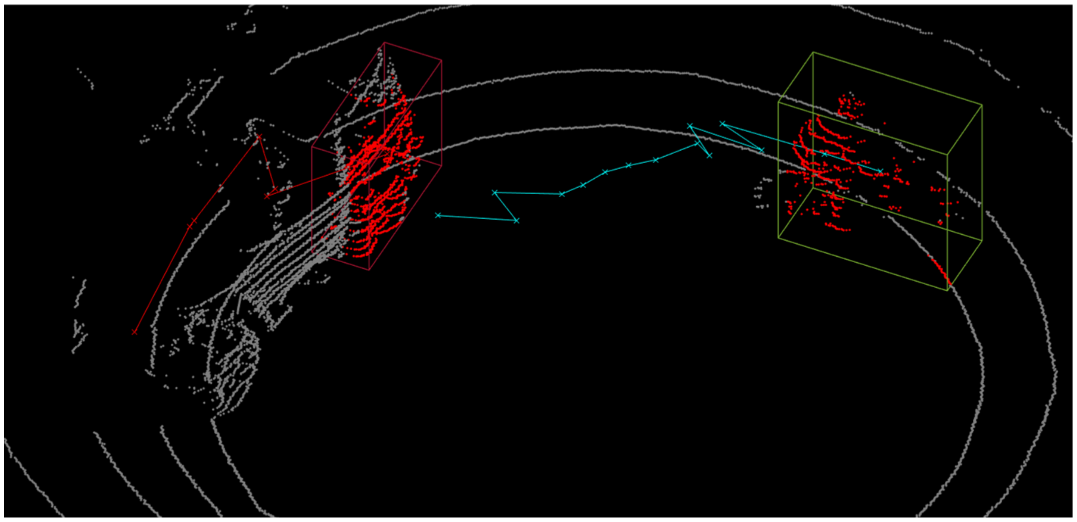

Figure 37. Hungarian particle tracking on a single bicycle, without smoothing.

5 RESULTS

We then analyze the effectiveness of the method proposed in the previous chapter. We tested these methods on all of the real data collected by a Velodyne VLP-16 Lidar scanner mounted on top of an experimental vehicle as described in section 2. There is no ground truth information available, therefore we have performed a qualitative evaluation of the performance. The results of this method on our various data collections are shown in the following sections.

5.1 Single Bicycle Data

We first applied our method to the single bicycle scenario to test the results of our methods without confusers. The first dataset we will look at is a single bicycle traveling clockwise around the Velodyne LiDAR sensor at a maximum distance of 10 meters and a minimum of 1.6 meters for 45.5 seconds. If we grab a random frame from this dataset we can see that the algorithm accurately acquired the bicycle, as seen in figure 38. We can also see that the algorithm has been tracking the bicycle for multiple frames, as depicted by the cyan line.

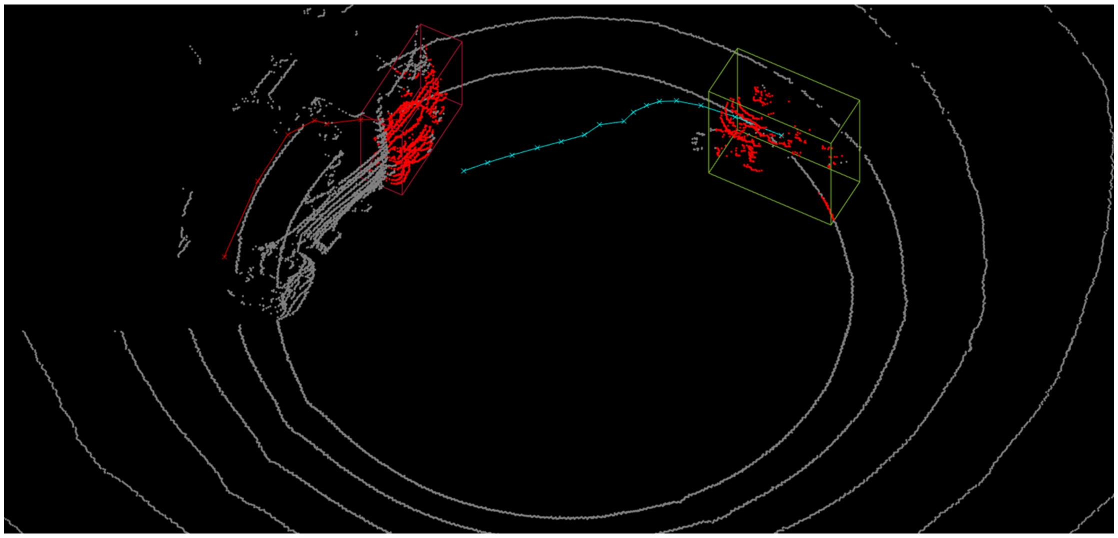

Figure 38. Hungarian particle tracking on a single bicycle, with smoothing.

However, the output still has errors. The car which is not moving has been identified as a moving object and incorrectly classifies it as a bicycle. To analyze how these errors, propagate over the entire dataset we can plot all of the tracks from this dataset onto the point cloud as seen in figure 39.

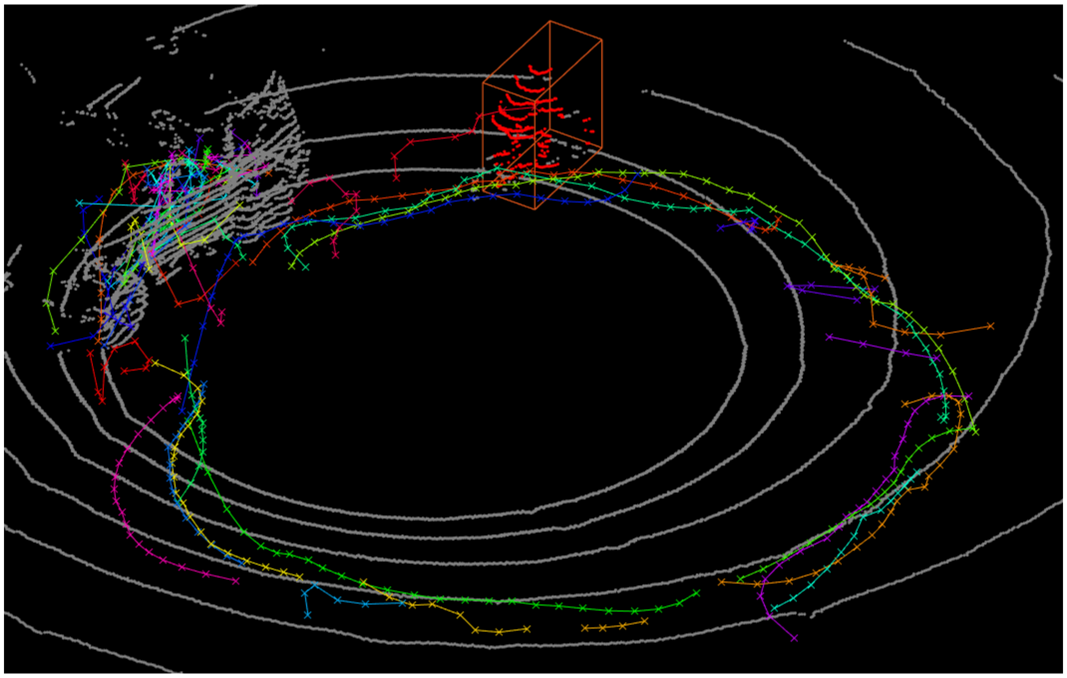

Figure 39. Single bicycle with persistent plot of tracks. Clockwise direction.

From this we can see that the tracks follow the path of the bicycle around the sensor very well. However, there are a significant number of tracks corresponding to the stationary car. This means that the algorithm is detecting moving objects at the location of the car and those objects are consistent over multiple frames. If we break this down into individual frames we can see what is causing the tracks and boxes near the parked car.

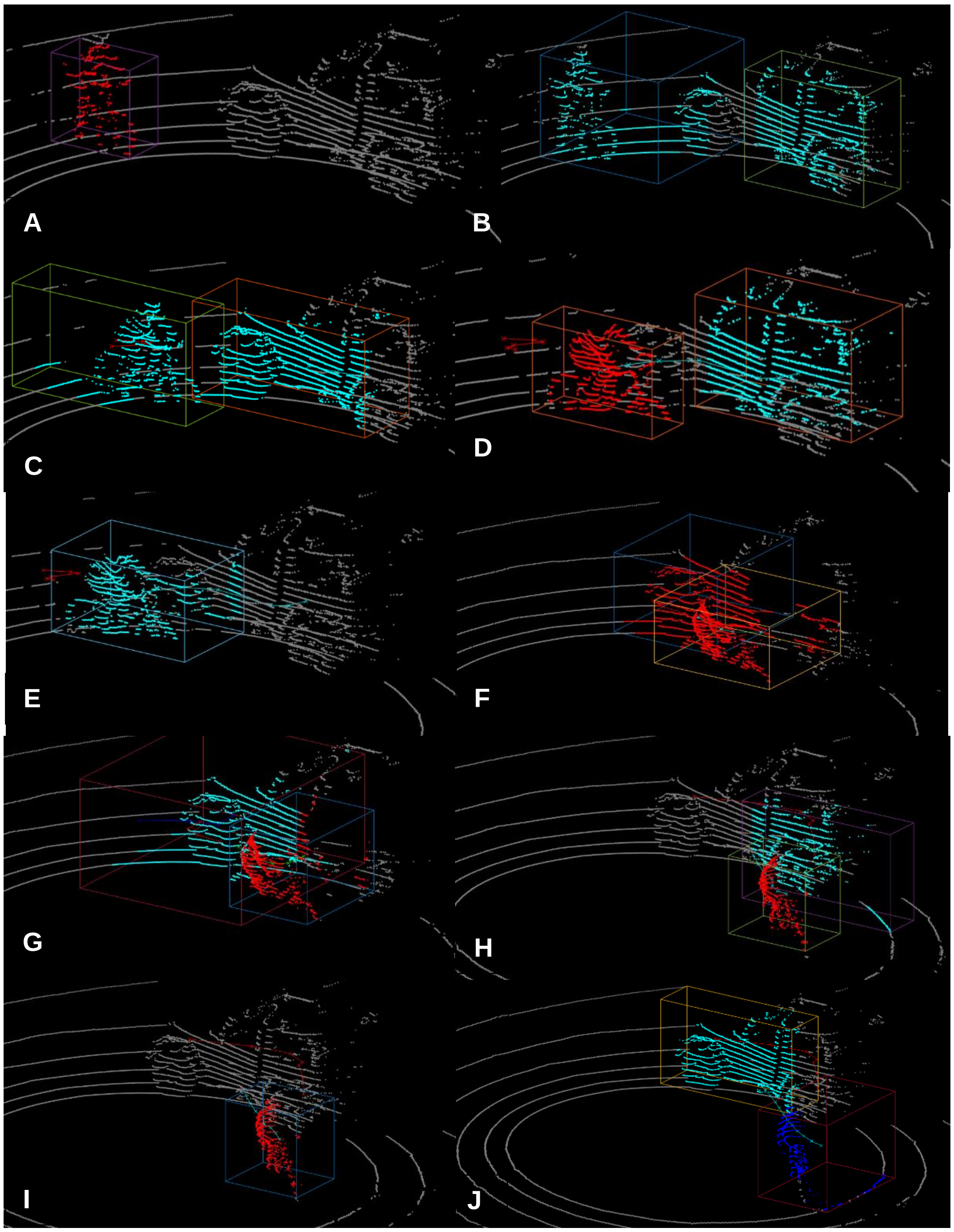

Figure 40. Clockwise single bicycle frames A to J. The bicycles pass directly in front of a parked car.

As we move down the individual frames in figure 40 we can see the bicycle passing by the car on its path. We can see in image A the bicycle is accurately identified. As we look from images B through E we can see that as the bicycle comes closer to the parked car the algorithm still identifies that the object is moving but is unable to correctly classify it. In images F through I we can see that the bicycle moves away from the car and is again correctly identified. In the last image J, we can see the bicycle is still being tracked but is incorrectly identified as a car. This is due to data points from the ground being included as the moving object. We can also see that as the bicycle passes past the car there are points that become occluded and then reappear in the point cloud. The movement of these points from occlusion is what causes the algorithm to incorrectly identify and classify them as moving objects.

5.2 Multi-Bicycle Data

After we applied these algorithms to the single bicycle data we moved to the multi-bicycle datasets. This dataset included five bicycles traveling around the parked vehicle in multiple directions, speeds, and distance for 202 seconds. In addition, the bicycles randomly stop and start their movement throughout the data collection. The bicycles would also periodically synchronize their routes around the car as to test the ability to identify individual bicycles clustered in a group. If we grab a random frame from this dataset we can see that a group of bicycles and a single bicycle are correctly identified, as seen in figure 41. We can also see that the algorithm has been tracking the bicycles for multiple frames, as depicted by the red line.

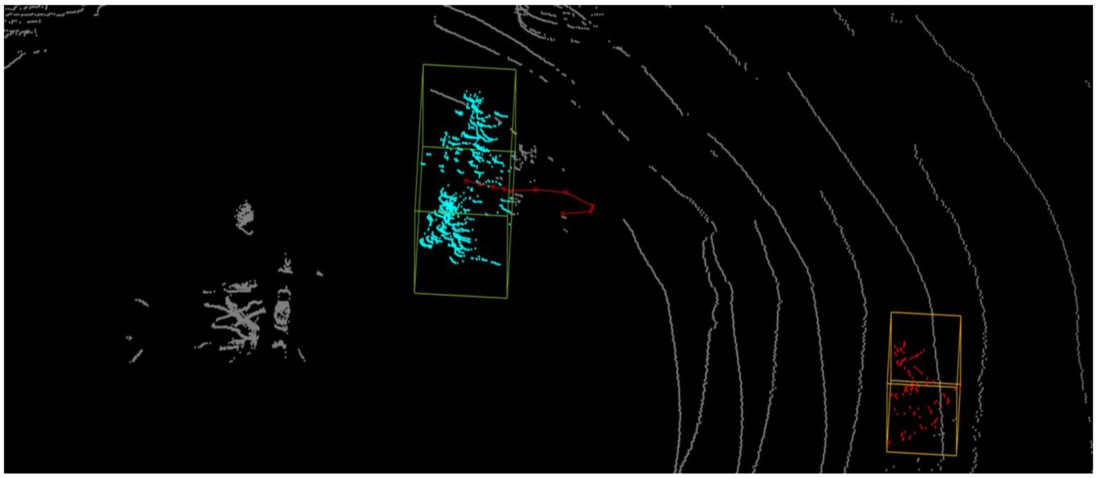

Figure 41. Detection, classification, and tracking on multiple bicycles. Three bicycles are classified correctly as a group (cyan) in the center of the image. A single bicycle is also identified (red) on the far-right side of the image.

Three bicycles are present in the group (cyan) but the algorithm is unable to differentiate them into single bicycles. This is most likely due to the limited number of points on each of the bicycles.

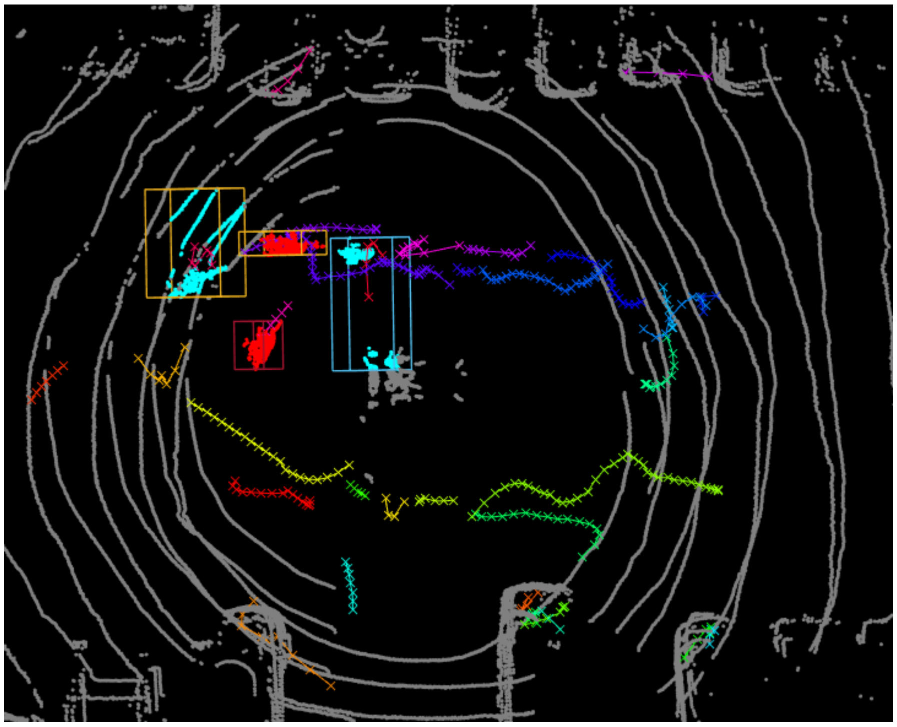

Figure 42. Detection, classification, and tracking on multiple bicycles with persistent plot of tracks. Four bicycles have been classified. Two correctly as individual bicycles labeled in red and two incorrectly labeled as groups in cyan.

From this we can see that the tracks follow the path of the bicycles around the sensor very well. However, there are a significant number of tracks displayed around the location of the parked cars (bottom center). These are most likely caused by obscuration. When a bicycle passes in front of a parked car the group of points directly behind the bicycle are blocked. When the bicycles move away from that area the points appear again. When the algorithm is computing the change in voxel states it may see this appearance of points are an object. We can see this effect more clearly by looking at multiple frames.

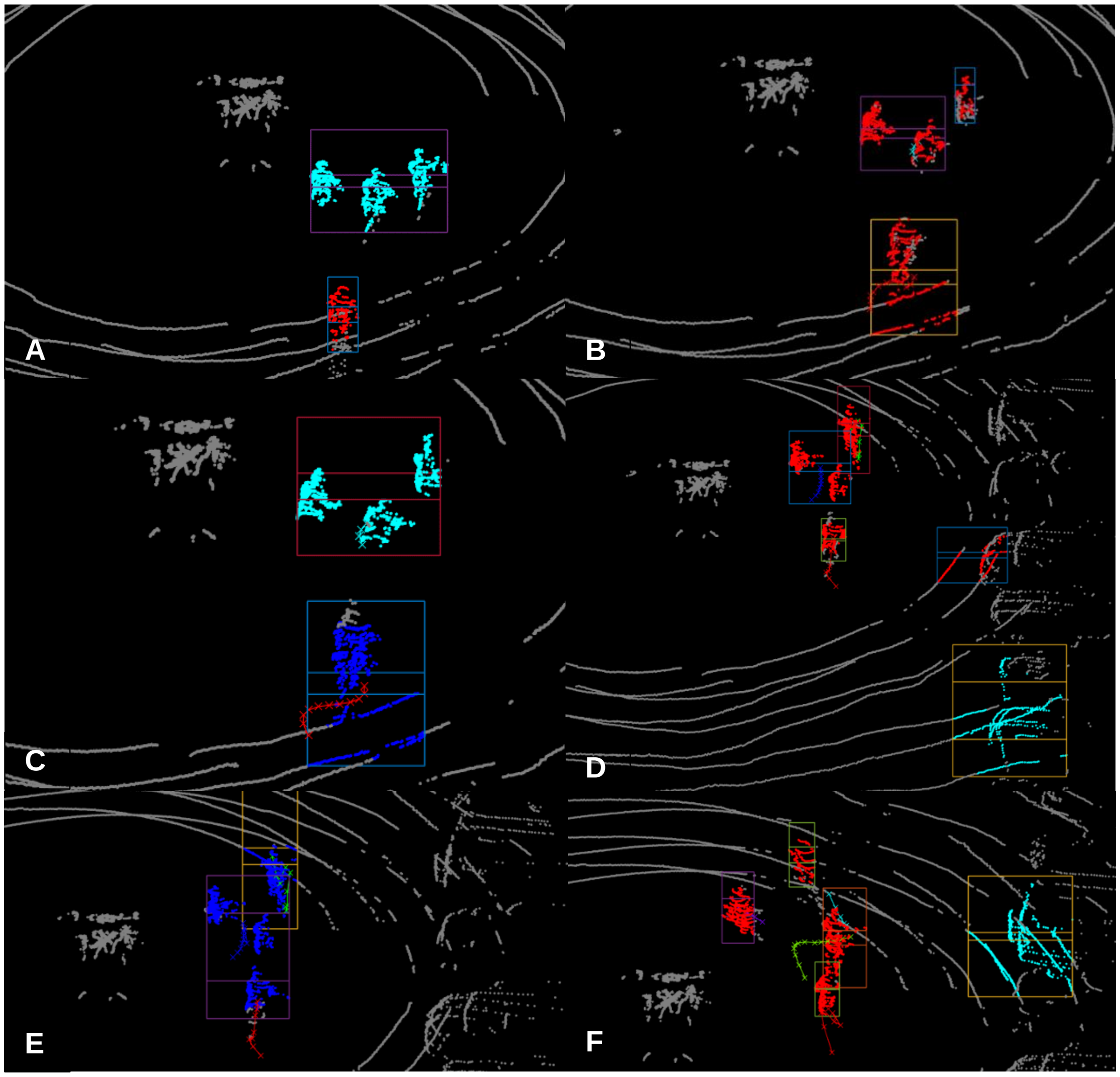

Figure 43. Multi-Bicycle Frames A to F. Four bicycles are tracked over time.

As we move down the individual frames in figure 43 we can see the bicycles passing by the parked cars. In image A, a group and a single bicycle are correctly classified. As the bicycles start to separate from each other in image B they are classified as individual bicycles. However, this is only temporary; in image C we can see that the algorithm has regrouped the bicycles and also incorrectly classified the single bicycle as a car. As we saw in the previous section the inclusion of points from the ground bias the classification process. As we move to image D we can see that the algorithm correctly identified two of the bicycles however, the two center bicycles are grouped into the same voxel and incorrectly identified as a single bicycle. In image E the bicycles are grouped together but since they are slightly further away from each other than normal they are classified as a car. The singular bicycle classified as a car in this image is also due to the inclusion of points on the ground. In frame F we can see that the bicycles are once again correctly classified. In images D and F, we can see that a voxel appears on the parked cars as the bicycles travel past them. Similar to the previous section this false movement effect is due to occlusion.

5.3 Driving Data

Lastly, we applied these algorithms to driving dataset. This dataset included over 20 bicycles that traveling perpendicular and parallel to the car while traveling down Columbus Avenue for 16 minutes and 47 seconds. In the dataset the same bicycle is seen multiple times as it passes the car and then the car passes the bicycle. Additionally, there are other moving cars traveling both directions down the road as well as pedestrians walking on the sidewalk and crossing the street. If we grab a random frame from this dataset we can see a single bicycle passing the car on the right.

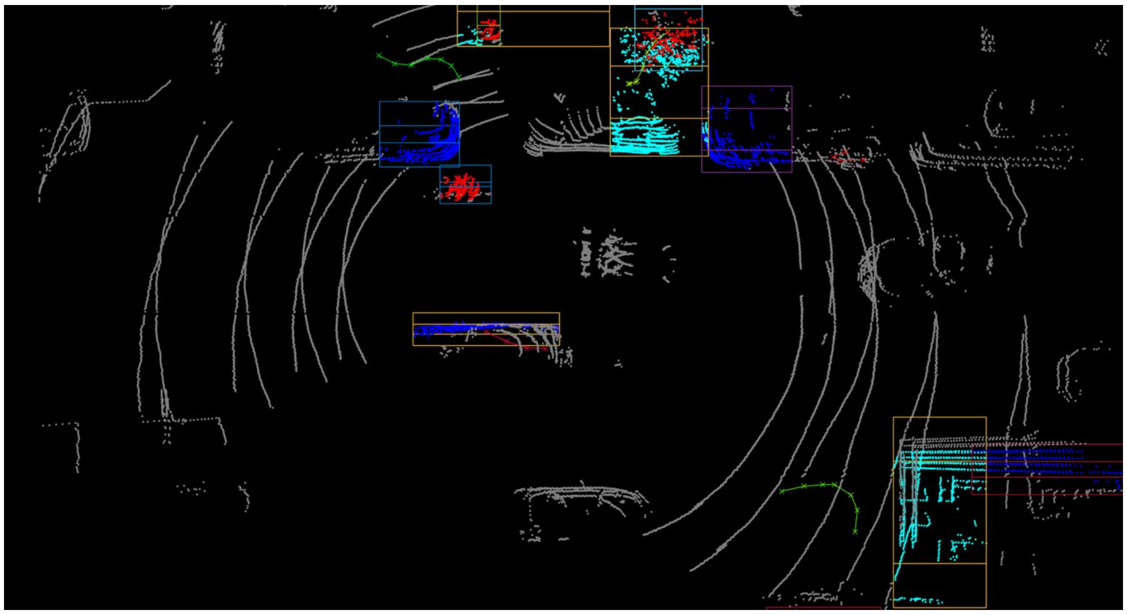

Figure 44. Detection, classification, and tracking on all moving objects in driving route B. A single bicycle is correctly identified in red. Three cars are correctly identified in blue.

The passing bicycle has been correctly classified as a bicycle. In addition, the cars to the right of our test vehicle as well as the car passing on the other side of the street are correctly identified. However, the output still has errors. One of the cars (center top) is identified as a group instead of a car. This object is also grouped with the tree above it. Additionally, a truck that is passing on the left of our test vehicle is incorrectly classified as two objects, a car and a group.

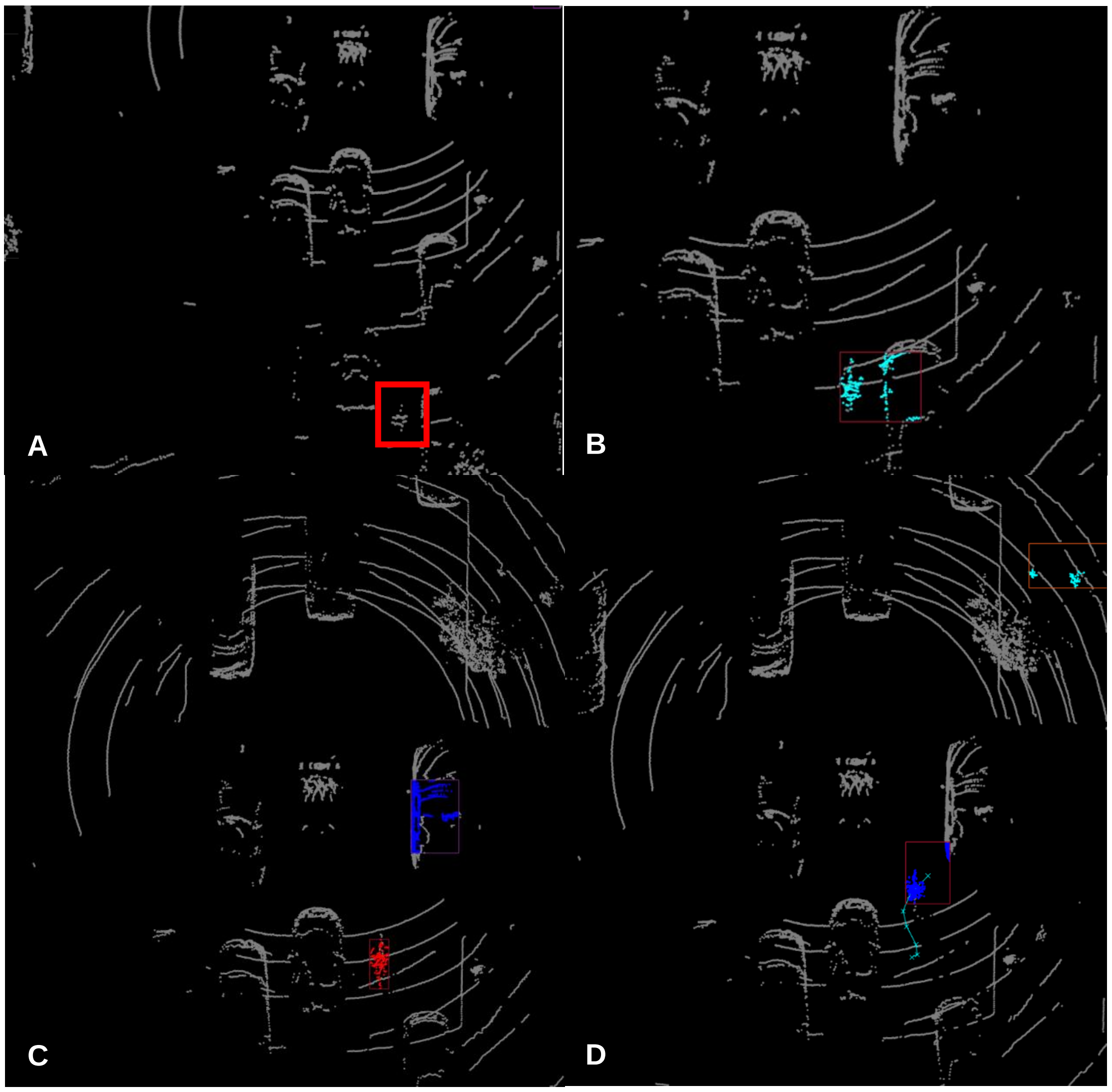

Figure 45. Driving route B - Frames A to D. The bicycle approaches the car from the rear.

In image A the bicycle is still over six meters from the test vehicle. In this frame the bicycle is not detected by the algorithm. This is due to the fact that there are not enough points on target in order to detect a moving object. As the bicycle approaches closer in image B the algorithm detects and incorrectly classifies it as a group with the car to its right. As the bicycle continues to move in image C it is correctly classified as a bike and the car is also classified correctly. When we move to image D the bicycle is again classified incorrectly, but this time as a car. However, the algorithm still holds the tracks from the previous correct classifications as shown by the cyan line in image D.

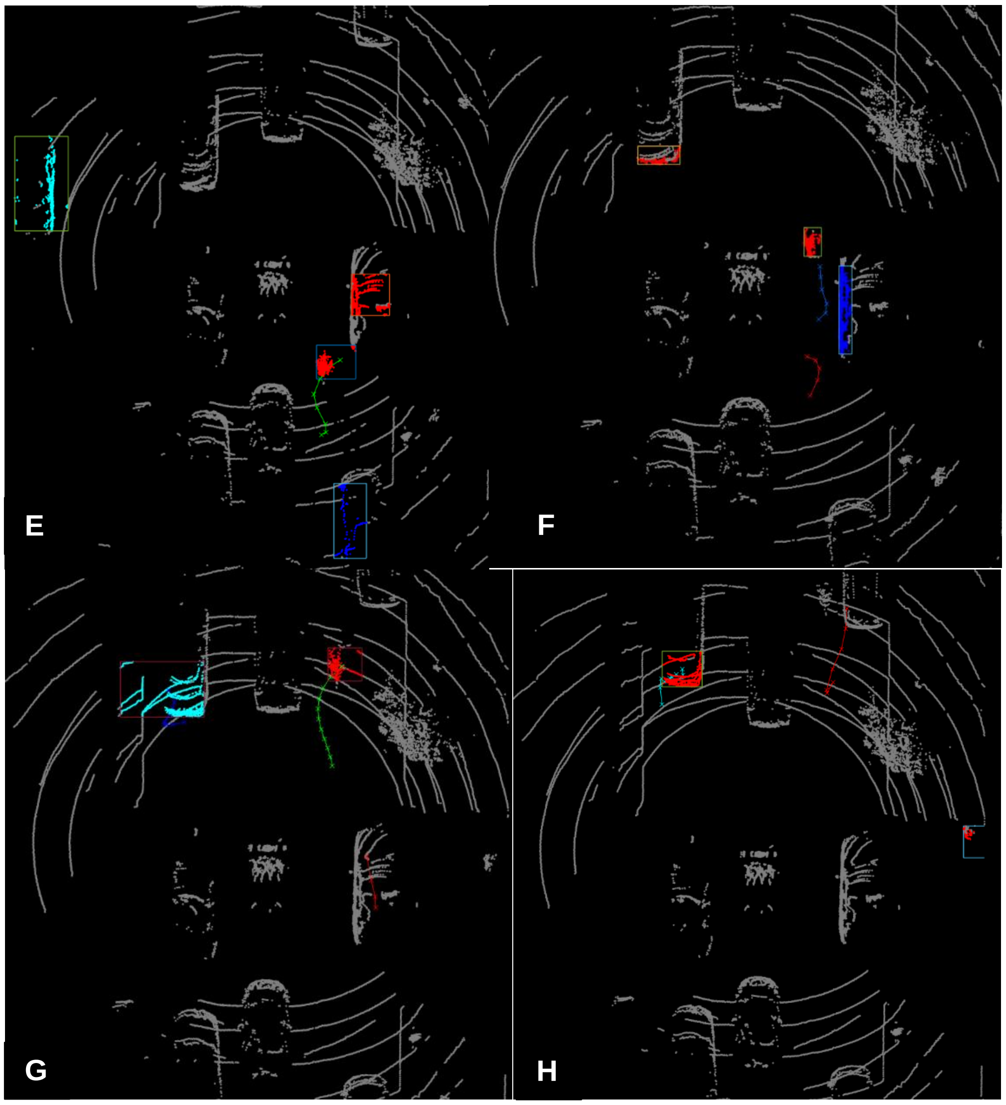

Figure 46. Driving route B - Frames E to H. The bicycles passed directly by the car.

If we continue through the next few frames we can see the bicycle approach and pass the test vehicle. In image E the bicycle is reclassified correctly as a bicycle, but the car to the right is now incorrectly classified as a bike. In image F, the bicycle and car are again classified correctly. However, there is a break in the tracks. This is due to the moving object not being identified for two frames. In image G the bicycle has moved significantly further away from the test vehicle while being correctly identified and tracked. In image H the bicycle is over four meters away from the test vehicle and the object can no longer be identified.

6 CONCLUSION AND FUTURE WORK

The challenges created by a bicycle's relatively transparent profile, small frame, and quick maneuverability present a serious issue for autonomous cars in urban environments. However, with further development of LiDAR sensors and algorithms these obstacles might be overcome. We have presented a realistic dataset that accurately depicts the problematic challenges with detecting, tracking, and classification of moving objects in LiDAR data. To further the development of LiDAR sensors and algorithms this paper introduces an open LiDAR dataset. The authors present realistic datasets from affordable sensors along with qualitative performance results of leading algorithms. This dataset and analysis allow easier access for researchers and developers to create systems and algorithms that perform in real world scenarios.

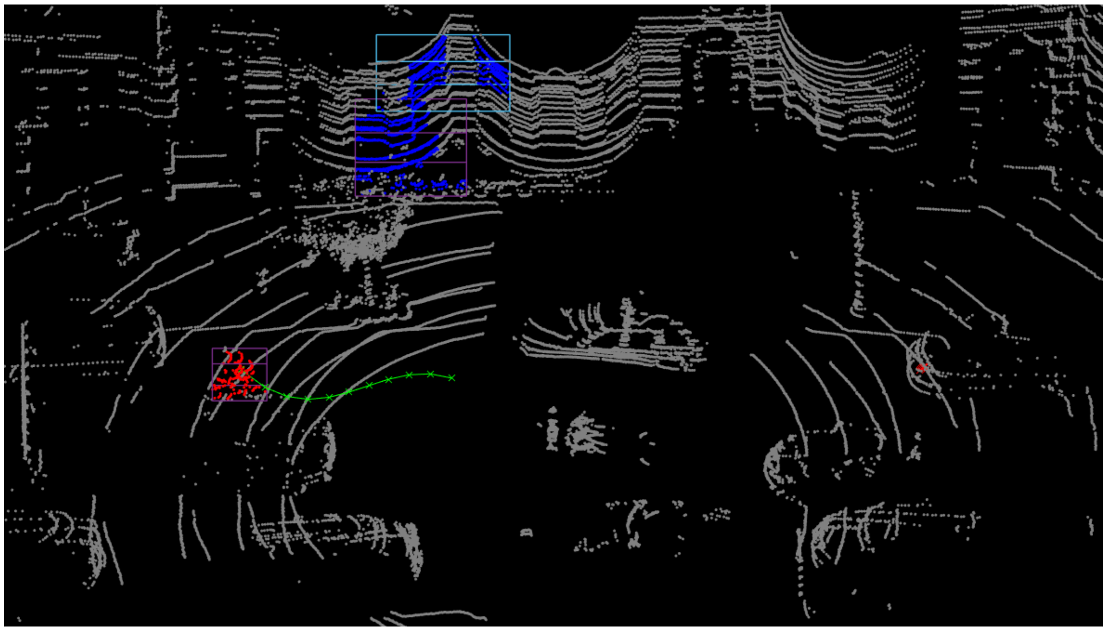

Figure 47. Driving route bicycle track. A single bicycle is accurately classified and tracked across multiple frames.

We have implemented a variation of one of the leading algorithms in object detection, classification, and tracking of moving objects, qualitatively showing its performance as seen in figure 47. Experimental results have shown that downgrading to an affordable LiDAR sensor can greatly decrease the performance of these algorithms. In order to make LiDAR truly useful a sensor that can perform on par with expensive ones like the Velodyne HDL-64E. Cheaper sensor simply cannot produce dense enough point clouds for algorithms to perform well. In addition to improving the sensor itself significant advances can be made in algorithms. In our implementation one of the leading causes of errors was incorrectly classifying objects. Upgrading from a simplistic model to a machine learning technique for feature-based classification can prove better. Methods that fuses multiple sensors together might increase the classification success rate as well [23]. Another flaw in our algorithm implementation was the movement of points behind an object. Using a simplistic algorithm leveraging ray tracing we may be able to predict points that move behind objects and eliminate them from being tracked. Lastly, we need to implement a technique for compensating with motion. A simplistic version of this would be using an inertial measurement unit (IMU) for motion detection and offsetting that motion in the raw point clouds. This would remove a majority of buildings and parked cars detected as objects, however it would also eliminate any object traveling at near identical speeds to the sensor. Some of the errors that propagate through the algorithm are caused by the rings on the ground being included as moving objects. Removing these errors would increase classification accuracy as well [24]. Implementing the algorithm described in this thesis in python or C++ using the Point Cloud Library would serve useful for implementation in real time systems [25].

Lastly, the combination of the algorithms described in this thesis with light field arrays from [26] can create a system to greatly increase an autonomous car’s ability to detect and classify targets of interest. Light field processing can be used to remove occlusion from scenes. However, in order to accurately do so, the surface along which to render a light field image must be known. In figure 48, we see an example of image reconstruction on an occluded sign.

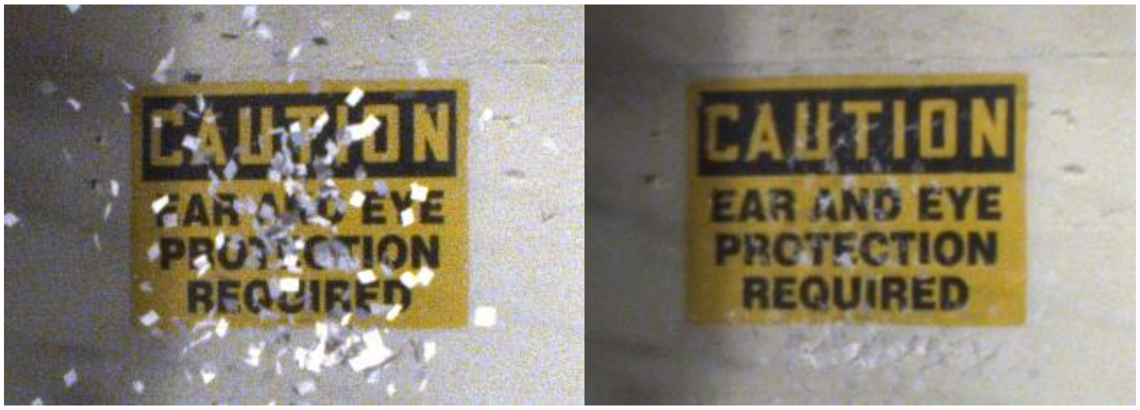

Figure 48. Light field based occlusion removal. The left is a raw image from one camera of an occluded sign. The right is a rendered using a light field array and median ray color selection.

In order for the light field algorithms to perform this operation the desired surface must be fully parameterized. For this example, the distance and orientation of the surface are measured and recorded manually. However, in real world scenarios this information is unknown beforehand. Trying to process all possible surfaces would be computationally intensive and therefore occlusions such as this would never be removed. In order to increase the usability of this technology we propose the combination of light fields and LiDAR. Using the algorithms described in this thesis distance and orientation of surfaces corresponding to objects of interest can be fed into light field algorithms in order to remove occlusions and obtain better classification of objects.

7 REFERENCES

[1] J. Pokrzywa, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems.," *SAE Standard J3016,* pp. 01-16, 2017.

[2] CleanTechnica, "Tesla & Google Disagree About LIDAR — Which Is Right?," 29 July 2016. [Online]. [Accessed 20 November 2017].

[3] P. Fairley, "Self-driving cars have a bicycle problem [News]," *IEE Spectrum 54.3,* pp. 12-13, 2017.

[4] V. LiDAR, "HDL-64E.," 2014. [Online].

[5] D. Coldewey, "Google talks up its self-Driving cars' cyclist-Detection algorithms," 1 July 2016. [Online]. [Accessed 20 November 2017].

[6] A. Azim and O. Aycard, "Detection, classification and tracking of moving objects in a 3D environment.," in *Intelligent Vehicles Symposium (IV), 2012 IEEE*, 2012.

[7] S. Kato, "An open approach to autonomous vehicles.," *IEEE Micro 35.6,* pp. 60-68, 2015.

[8] C. Mertz, "Moving object detection with laser scanners," *Journal of Field Robotics 30.1,* pp. 17-43, 2013.

[9] A. Mukhtar, X. Likun and T. B. Tang, "Vehicle detection techniques for collision avoidance systems: A review," *IEEE Transactions on Intelligent Transportation Systems 16.5,* pp. 2318-2338, 2015.

[10] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Autonomous Robots 26.2-3,* pp. 123-139, 2009.

[11] C. Premebida, "A lidar and vision-based approach for pedestrian and vehicle detection and tracking," in *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE.*, 2007.

[12] M. Szarvas, S. Utsushi and J. Ogata, "Real-time pedestrian detection using LIDAR and convolutional neural networks," in *Intelligent Vehicles Symposium, 2006 IEEE*, 2006.

[13] M. Quigley, "ROS: an open-source Robot Operating System," *ICRA workshop on open source software. Vol. 3. No. 3.2.,* 2009.

[14] V. LiDAR, "VLP-16.," 2016. [Online].

[15] D. Inc., "We Make Robots Move and Cars Drive Themselves.," [Online]. [Accessed 20 August 2017].

[16] J. Han, "Enhanced ICP for the registration of large-scale 3D environment models: An experimental study.," *Sensors 16.2,* p. 228, 2016.

[17] Y. Chen and G. Medioni, "Object Modelling by Registration of Multiple Range Images," *Image Vision Computing. Butterworth-Heinemann,* vol. 10, no. 3, pp. 145-155, April 1992.

[18] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on pattern analysis and machine intelligence 14.2,* pp. 239-256, 1992.

[19] M. Wang and Y.-H. Tseng, "Lidar data segmentation and classification based on octree structure.," *Parameters 1,* p. 1, 204.

[20] S. Laine and T. Karras, "Efficient sparse voxel octrees–analysis, extensions, and implementation.," *NVIDIA Corporation 2,* 2010.

[21] D. M. Cole and P. M. Newman, "Using laser range data for 3D SLAM in outdoor environments.," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.*, 2006.

[22] M. D. Breitenstein, "Robust tracking-by-detection using a detector confidence particle filter," in *Computer Vision, 2009 IEEE 12th International Conference on.*, 2009.

[23] H. Cho, "A multi-sensor fusion system for moving object detection and tracking in urban driving environments," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[24] M. Velas, "CNN for Very Fast Ground Segmentation in Velodyne LiDAR Data," *arXiv preprint arXiv:1709.02128,* 2017.

[25] R. R. Bogdan and S. Cousins, "3d is here: Point cloud library (pcl).," in *Robotics and automation (ICRA), 2011*, 2011.

[26] A. Bajpayee, A. H. Techet and H. Singh, "Towards Real-time Life Field Processing for Autonomous Robotics," Submited for review.

[27] T. D. Vu, J. Burlet and O. Aycard, "Grid-based localization and local mapping with moving object detection and tracking.," *Information Fusion,* pp. 12:58-69, 2011.

8 APPENDIX

Software and documentation can be found online at the link below.

https://github.com/arufo/LiDARUrbanBikeDetection.git

The contents of this link are described below.

1. Code – All MATLAB code is stored in this directory.
2. Videos – Videos of raw data and output products are stored and labeled by scenario.
3. Images – Images of raw data and products are stores and labeled by scenario.