2024 IEEE/ACIS 22nd International Conference on Software Engineering Research, Management and Applications (SERA) | 979-8-3503-9134-3/24/$31.00 ©2024 IEEE | 10.1109/SERA61261.2024.10685604

A Spatial Data Pipeline for Streaming Smart City Data

Chase Carthen*, Araam Zaremehrjardi*, Vinh Le*,
Carlos Cardillo‡, Scotty Strachan†, Alireza Tavakkoli*, Sergiu M. Dascalu*, Frederick C. Harris* Jr.
**Computer Science and Engineering*, †*System Computing Services*, ‡*Nevada Center for Applied Research*
‡**University of Nevada, Reno*,†*Nevada System of Higher Education*
Reno, Nevada
Email: {ccarthen,azaremehrjardi,vle,ccardillo,tavakkol}@unr.edu,
sstrachan@nshe.nevada.edu, {dascalus, fred.harris}@cse.unr.edu

***Abstract*—Point cloud data in the form of LiDAR is often utilized for its spatial qualities, especially in smart city projects for tasks involving vehicles and pedestrians. However, the process in which LiDAR data is acquired can be cumbersome to setup and automate. In this paper, we introduce a streaming and an on-demand pipeline for capturing LiDAR data from Velodyne Ultra Pucks placed along northern Nevada intersections known as the Living Lab as part of a smart city project for the city of Reno. The data coming from these intersections consist of the following formats: ROS 2 bag file, PCD, LAZ, Google Draco, and PCAP. A streaming point cloud service with PCD, LAZ, and Draco was implemented to stream any of these formats, as well as to allow the user to capture the current monitored point cloud. Additionally, two on-demand web services were implemented for both the PCAP and ROS 2 bag file to enable a user to start and stop the acquisition of LiDAR data in these formats. Through our analysis, it was discovered that Draco provided the best processing time and had a wider range of options that affected the quality of the point cloud. To evaluate this pipeline, the features of existing software were compared and a discussion was provided with an analysis of the point cloud formats.**

***Index Terms*—MQTT, ROS, data streaming, IoT, data transfer, big data, smart city, LiDAR, data pipeline, edge computing**

I. Introduction

Smart cities have been an ever-growing concept that has created a knock-on effect across software research, leading into fields such as big data, embedded systems, cyberinfrastructure, and artificial intelligence. The promise of delivering a greener, safer, and more aware city infrastructure has captured the attention of researcher and city planner alike. Especially through the recent advancements in big data, the Internet of Things (IoT), and Artificial Intelligence, the vision of Smart Cities are becoming more and more real. However, the construction of a smart system governing something so obscenely massive like a city infrastructure comes with its own set of issues.

By now, it comes to no surprise that there are numerous challenges in the movement, storage, and analysis of massive, high velocity, and/or abstract data. The development of these software systems often times face considerable constraints imposed by either physical system restrictions such as limited broadband, or available computational resources. Typically, these constraints are imposed on systems that involve deployments of edge device(s), or endpoints of the larger system, for data acquisition. Additionally, these larger software systems are also expected to provide some measure of distributed computing environment for data processing purposes. It is because of these compounding constraints that researchers and other project personnel are forced to carefully fine-tune and develop a sustainable and scalable data pipeline.

In this paper, we present a multi-format point cloud data pipeline aimed at serving the smart city infrastructure within the city of Reno. The pipeline was designed to open up a gateway for smart city researchers to build software involving pedestrian, vehicle, and object detection. In our pipeline, we implemented both Google Draco and Laszip as point cloud formats with differing Lossy formats [1]. A variety of formats were included in the pipeline's design in order to enable multiple forms of point cloud compression and to allow for different levels of detail when considering a Lossy format. Finally, a specialized suite of customized applications and web services were developed, such as a metadata service that pulls from the infrastructure, as well as several web download services for various point clouds formats.

The rest of the paper is structured as follows. Section II covers the background and related works for this paper. The systems design and implementation are described in Section III. Results pertaining to the system's data services are presented in Section IV. Discussion of the results acquired for this paper is included in Section V. Finally, the overall impact of this paper is discussed in Section VI.

II. Background and Related Works

A. Infrastructure Environment

The hardware infrastructure for this project was implemented across nine intersections in Reno, NV as part of the Living Lab at the University of Nevada, Reno (UNR), a portion of which is shown in Fig. 1. Each intersection was deployed with one edge computer and two Velodyne VLP-32c sensor pucks placed diagonally across one another on opposite traffic poles. A cross-connection exists in-between the edge deployments for communication between the edge deployment network and data center network. Regarding the

Fig. 1. A subset of the intersections implemented in the Living Lab with the red circles outlining the deployed intersections.

distributed computing needs, the implementation uses a university operated on-campus computing cluster for the operation of message brokers, web services, and orchestration software. Furthermore, the needs for long-term storage for captured point clouds is met by an off-campus computing cluster, named Pronghorn.

B. Fog Computing

One of the popular topics in Smart Cities and Internet of Things research is the transmission of data from one point to another point efficiently while allowing the receiving point to use the data effectively. One popular solution lies a paradigm known as Fog computing, where computation and data services are down at the edge rather than the cloud in order to improve latency and closeness of computation to the source [2]. Fog computing is becoming more popular in the Smart Cities and Internet of Things research especially involving various sub-domains such as: healthcare, smart conveyance, smart waste management, smart energy handling, and digital twin [3], [4]. The creation of a fog computing architecture or paradigm requires special attention to specific types of software, network protocols, and how data is handled at any given point. This paper in particular covers the creation of a streaming pipeline for spatial data in the form of point clouds.

C. Data Processing

There are a number of different ways of enhancing the streaming of spatial data that includes improving the hardware, compression of the data between receiver and sender, selection of network protocol, and handling of the data between receiver and sender. The more common hardware adopted by spatial data pipelines are often times low power and low resource usage due to the power, cost, and network constraints. Raspberry Pi's and Nvidia Jetson devices have been for several different Smart City applications regarding point cloud data [5]–[7]. To further optimize these pipelines, data can be transformed prior to transmission through Lossy compression algorithms such as draco, PCL's compression implementation, or using lossless compression software like lasZIP and zStandard [1], [8]–[10]. Several protocols within the internet of things space have been developed to lower the overhead of known internet protocols at the software level. Some examples of these protocols are zigbee, MQTT, and LoRaWAN. How the data is handled by the sender and receiver can be a source overhead and is something that must be considered especially when adding compression to the transmission of data. This paper looks specifically at how to improve the sending and receiving of point cloud data.

D. MQTT

MQTT is a communication protocol specification and a key player in our pipeline's design. MQTT was first introduced in 1999 with subsequent updates to the protocol with versions 3.1.1 and 5.0 [11]. MQTT is developed on the application layer using a publisher-subscriber methodology for message transfer across nodes [12]. A typical MQTT setup contains a series of clients with one or more brokers. Brokers within a MQTT setup are used to facilitate the transfer of messages amongst clients through a series of topics, or streams of data. Clients can subscribe to one or many topics thus allowing clients to "listen" for incoming messages from other clients publishing onto a topic. The protocol offers a level of configuration defined as "Quality of Service" (QoS) for operating within constrained environments, such as those limited in bandwidth or exhibiting high-latency conditions. MQTT comprises three levels of QoS in which controls the guarantee of message delivery between senders and recipients in various network conditions. MQTT has been used for a variety of IoT applications that call for the movement of data from edge devices to either fog or cloud computing environments. Within the area of monitoring systems, MQTT has been used for facilitating the transfer of data within a conceptual air pollution monitoring systems [13], [14].

III. Design and Implementation

A. Design

In designing the streaming data pipeline for this paper, there was a significant emphasis on serving spatial data in a scalable and performance-conscious manner. To this effect, we designed a system that makes use of existing software,

979-8-3503-9134-3/24/$31.00 ©2024 IEEE

SERA 2024, May 30-June 1, 2024, Honolulu, USA

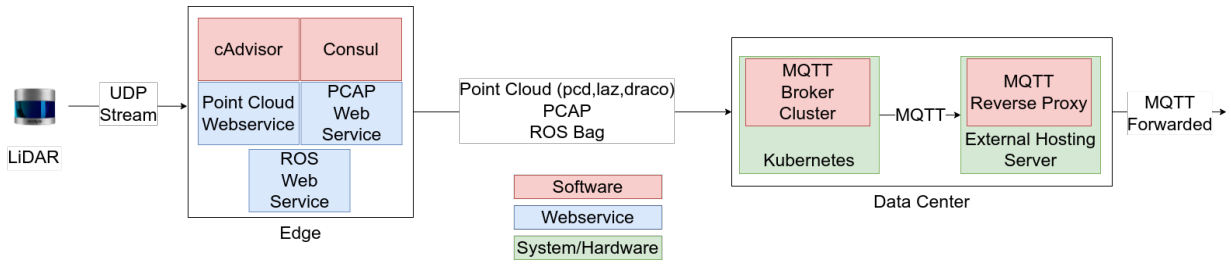

Fig. 2. High-level system components and data flow.

while at the same time, implemented significant portions of it by scratch to strike that fine line between standardization and meeting project goals. As such, we adhered to the following requirements:

- The system shall make point clouds available in near real-time with a resolution within seconds to minutes.
- The system shall allow for PCAP and ROS data to be available for recording on demand.
- The system shall make it so that the storage, service availability, and system resource metadata are monitored.
- The system shall provide point cloud data over MQTT and on demand download over a RESTful API.
- The system shall keep a portion of data available for download in the form of PCAP and ROS 2 data.
- The system shall perform analytics of the traffic flow through the Living Lab.

These requirements cover a streaming data pipeline with on-demand capabilities to record and stream LiDAR in different formats. These requirements encapsulate a good majority of the functionality within the system.

In developing the streaming data pipeline for this project, the following use cases heavily influenced the usability:

- The user is able to start and stop recording of PCAP and ROS data.
- The user is able to receive or download from a live stream or from a web service.
- The user is able to get information on the health and state of edge devices and services.
- The user is able to see traffic flow along the Living Lab.

Each use case is meant to account for the requirements described previously. The goal is for users to record data on demand and watch point clouds in real time for developers to be able to make apps against existing data streams, and for traffic engineers to use the existing data flow for smart city applications.

B. Infrastructure of the Implementation

Fig. 2 illustrates the overall flow of data starting from the edge devices as it travels to the data center then finally reaching the external hosting server. The edge machine receives series of UDP packets from a LiDAR sensor which is a VLP-32c in this case. These packets are then processed capture by a UDP packet replicator application called Samplicator that allows for the ROS web service to record the data into a ROS 2 file, and also allows the Point Cloud web service to send out point cloud data over either a RESTful API or a MQTT broker. Additionally, a PCAP web service is able to grab the data from the LiDAR as it is received and store it into a Packet Capture (PCAP) file.

All of these data services are orchestrated and started using docker-compose and are set to restart if any of the services go down. The MQTT data stream from the point cloud web service is pushed to a three node VerneMQ broker cluster running in Kubernetes at the data center. This data is then captured by a external hosting server that forwards traffic through Nginx to the internet. A software called cAdvisor is used to monitor the edge computer and the docker containers for disk and CPU usage. Consul is used both at the edge and in the Kubernetes workflow in order to monitor the liveliness of the data services that are present at the edge and Kubernetes. Apache MiNiFi is used to send any file data, such as ROS and PCAP files, to Apache NiFi running in Kubernetes. The collection of these data services and software make up the infrastructure for the Living Lab.

Nginx was chosen for the external hosting server because it allowed for the current data services and any future data services to be served with ease. To be clear, the external hosting server was meant for developers and researchers to be near real-time applications for smart cities. This approach was never meant for critical operations, such as allowing pedestrian in real-time within milliseconds. However, it was chosen as it allows for network protocols such as MQTT to be forwarded with relative ease. For the setup used in this paper, both a web socket and TCP MQTT were forwarded over TCP to Nginx.

Kubernetes was chosen in order to deploy any number of applications and to scale the applications up with relative ease. VerneMQ was chosen to run Kubernetes as it has support for being scaled up as an MQTT broker, which may be needed as more intersections are continuously added to this existing infrastructure. Additionally, Kubernetes is also supporting other existing infrastructure within the project to transport PCAP and ROS data over Apache MiNiFi to Apache NiFi. This infrastructure sends PCAP files zipped as gzip and that are then compressed to XZ to a storage server in the data center [15]. Within the Kubernetes cluster, a metadata information service is active and contains information about the different deployed LiDAR sensors and their subsequent edge computers.

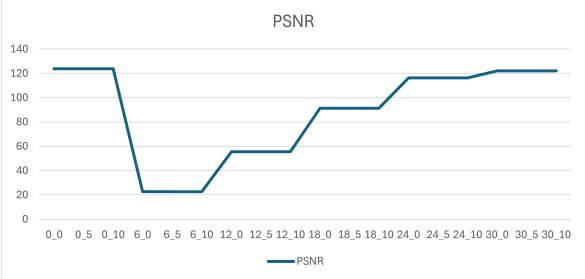

Fig. 3. The PSNR for different levels of compression and quantization. In the horizontal axis the first number is the quantization level and the second is the compression level.

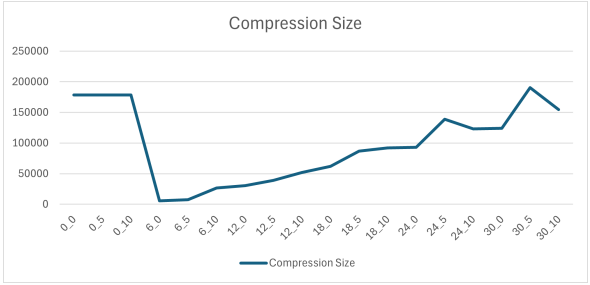

Fig. 4. The compression size for different levels of compression and quantization. On the horizontal axis, the first number is the quantization level and the second is the compression level.

C. Streaming and Data Services

Both the PCAP and ROS services were designed with FastAPI, a framework for deploying RESTful APIs. FastAPI was chosen for these endpoints due to their robust set of helper features, such as providing automated documentation of your API endpoints by default. Internally, the web services are designed similar in that they have a start, stop, and status endpoint. Both the start and stop endpoints start the process for recording. In the case of the ROS service, it utilizes the ROS 2 record command. For the PCAP service, it will start a tcpdump command for UDP on ports specific to the LiDAR. The status endpoint will tell whether a recording is in progress, what files are currently in the directory, and how much time has elapsed in the recording.

The point cloud capture service was implemented with several C++ libraries providing different types of support for point cloud formats. The most prominent libraries used in this service is the Point Cloud Library (PCL), Google Draco, and lasZIP. This web service is able to read from PCAP files and LiDAR streams from Velodyne sensors. Naturally, it is also able to write the data from LiDAR streams in several different formats, namely PCD, Draco, and LAZ. These formats can be written to a flat file, streamed over MQTT, or downloaded through the REST API within the application. Additionally, this service also provides rudimentary machine learning algorithm meant to capture moving vehicles and pedestrians. The underlying machine learning algorithm uses a voxel map to filter for the least updating areas of the LiDAR along with the built-in k-nearest neighbor algorithm implemented in PCL to detect potential vehicles and pedestrians [16].

A metadata service, also implemented through FastAPI, was created to help keep track of what hardware is available at each intersection, along with what files are currently in storage at the data center. The database used to track this information was a Mongo database, since the collected data did not constitute much in terms of relational needs, and to ensure some level of scalability in the future. A GraphQL API was added to this data service in order to interact with the ROS and PCAP service to get information about their status, along with the edge machine's storage from the cAdvisor API. This web service runs on the Kubernetes cluster and is meant to provide information about the existing hardware, data consistency across web services, and information regarding the status of the web services.

IV. Results

A. Streaming Pipeline Comparison

The handling of LiDAR and point cloud is not necessarily uncommon and a fair amount of software platforms tailored to these technologies have been developed over the years, and each have a wide array of features. Throughout the last decade, many of these platforms utilized specialized data pipelines with designs that use software such as Point Data Abstraction Library (PDAL) that has several plugins to transform point cloud data or even implementing custom algorithms to classify/filter the data from the ground up. Another similar software designed to work with Velodyne LiDAR sensors is VeloView, which has the capability for visualizing LiDAR data, slam data analysis, and export LiDAR data out in various different formats. Finally, Geographic Information Systems (GIS), such as QGIS or ArcGIS, are a common staple among many envirosensing groups and the tool of choice when visualizing LiDAR geospatially, possessing a fair variety of processing algorithms built-in.

Regarding the streaming pipeline presented in this paper, the main focus was on making data available for researchers and without much pre-processing. Like VeloView or PDAL, this pipeline offers a number of different point cloud formats such as PCD, LAZ, and draco. Additionally, the pipeline also provides on-demand formats are either PCAP or ROS 2 file. Regarding analytics, while this streaming pipeline does not provide any real-time analytics on the point cloud data, it does provides a voxel classification machine learning service implemented in the streaming point cloud service. Finally, although the machine learning algorithm was designed to capture moving pedestrians and vehicles, it outputs trajectories of the pedestrians and vehicles as a JSON.

B. Point Cloud Format Comparison

When developing the pipeline presented in this paper, heavy consideration was put into the point cloud formats to make as close as possible a near real-time transmission rate. All three formats were analyzed by looking over their average sizes over time. It was found that a PCD file gathered from the

979-8-3503-9134-3/24/$31.00 ©2024 IEEE

SERA 2024, May 30-June 1, 2024, Honolulu, USA

intersection deployments had an average size of 669293 bytes. Similarly, Draco files with settings of compression level set at 5 and quantization level of 5 was 74360.7 bytes. Interestingly enough, LAZ files on average had a size of 126642 bytes.

On average, a PCD file took about 6.1 milliseconds to process since PCL can convert directly into this format. LAZ had a average processing time of 24 milliseconds and Draco had a average processing time of 11 milliseconds. Draco was faster than LAZ, but was either a tad shorter or comparative to PCD's processing time. Both Draco and Laz formats are smaller in size than the PCD due to both being formats with compression. LAZ is a Loseless format that is prevalent in many GIS softwares and was implemented to allow for future integration with GIS software. Draco itself is a Lossy format where it uses several different parameters along with quantization to reduce the size of the point cloud. By setting the compression level parameter at 5 and quantization at 5, we found that Draco was a superior choice as the format to be streamed in real-time, due to it being the smallest in size, but still being rather quick to encode and decode.

In order to evaluate the effectiveness and quality of the Draco format the peak signal to noise ratio (PSNR), encoding, and decoding were recorded then computed at different levels of compression and quantization. The peak signal to noise ratio was computed with the following formula:

$$PSNR(x,y) = 10\log_{10}\frac{MAX^2}{MSE}[dB] \tag{1}$$

The mean squared error was computed with the following formula:

$$MSE(x,y) = \frac{1}{N_x}\sum_{i=1}^{N_x}(x_i - y_{nearest})^2 \tag{2}$$

The MSE is the mean squared error of two point clouds with $y_{nearest}$ being the nearest point to the point x_i and N_x being the number of points in point cloud x. The MAX is defined as the maximum of the nearest neighbor point pairs in relation to point cloud x, but that requires finding the average point density of point cloud x first. These definitions are based on Sato et al [1].

Fig. 3, 4, and 5 were recorded and computed with quantization levels at 0, 6, 12, 18, 24, 30 and with compression levels at 0, 5, 10 with collected values being collected three times and averaged on the same point cloud. Fig. 3 shows that as quantization is increased, the quality of the point cloud that is decompressed is higher when looking at the PSNR due to more bits being used to represent the original point cloud. The level of compression was altered from 0, to 5, and then to 10 to show that the PSNR is not affected by the compression level. At the zero quantization level, the original point cloud was used for compression. It can be seen that at a quantization level of 30 the quality of the point is essentially the same as the original. Fig. 4 shows that the compression size at compression levels 0 and 10 were comparable, with compression level 5 being the worst at almost all recorded quantization levels. In Fig. 5, the decompression time was remotely the same

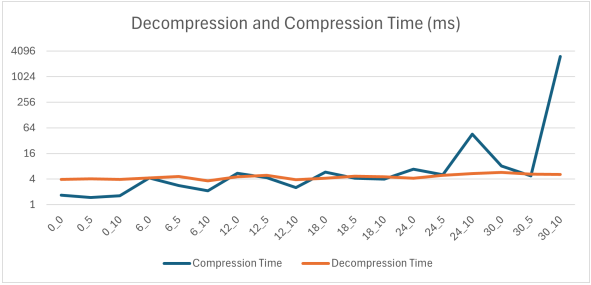

Fig. 5. The compression and decompression time for different levels of compression and quantization. In the horizontal axis, the first number is the quantization number and the second is the compression level.

across all quantization and compression levels. However, the compression time seems to be much higher at level 0 for lower quantization and higher at level 10 for higher quantization.

V. Discussion

At present, docker-compose was used at the edge and within the Kubernetes instance deployed at the data center so that services may be spun up or spun down with ease. This setup enables for our pipeline a level of robustness in that most errors can be resolved with restarting a container. Nginx was used due to its scalability, to host the various data streaming services to external clients. VerneMQ was implemented within the Kubernetes cluster and can scale if the number of MQTT clients were to increase and by default, was set to retain the last message for any topics sent to VerneMQ.

Regarding the streaming point cloud service, Draco was set as the main point cloud format for streaming over MQTT, as it was the superior means to save bandwidth and processing time. Aside from Draco, LAZ was added for its compatibility with existing GIS applications to enable future integration with the streaming pipeline. LAZ by itself has a number of configurations that affect the compression amount and rate. While we have LAZ configured for only one of these options, additional configurations would allow for improved flexibility and versatility. Ultimately, both LAZ and Draco were supported in this pipeline due to their popularity as formats and the functionality that both carry.

A comparison of PSNR was included to show that a point cloud will lose quality at different levels depending specification of compression and quantization levels. At present, Draco had the best compression and processing time in comparison to LAZ, while PCD had the best time due to PCL being able to convert to the format without any extra processing. Breaking down the total operation time, the prototype machine learning algorithm that classified pedestrians took around 6 milliseconds, and the most slowest format's (LAZ) processing time was 24 milliseconds. This means that the machine learning in combination with point cloud format generation is greater than 30 milliseconds. Unfortunately along with the bandwidth over MQTT, this workflow may not be suitable for realtime applications, but it still works rather well for near-realtime applications.

VI. Conclusion and Future Work

In this paper, we presented a data streaming data pipeline that supported: handling and exportation of point cloud data in multiple different formats, on-demand recording of PCAP files and ROS bags, and handling of infrastructure from edge deployment to/from data center. Primarily, these services were created to provide a level of cyberinfrastructure to researchers to build applications against the available data. Draco, after being tested among the other formats was set as the default streaming format due to its considerable compression potential and having the availability to configure the quality of the point cloud. Both Kubernetes and docker-compose made it so that the pipeline could scale horizontally and recover from failure with ease. Finally, an early prototype of a machine learning algorithm was implemented to test the viability of sending trajectory data over MQTT along with the point cloud data.

At present, the different settings of Draco are being evaluated in order to find the best trade-off for quality versus size. Draco enabled the compressing of point cloud data to a low size, but at the cost of different analytics algorithms running poorly. Experimenting with optimizations in the infrastructure or even trying out another format may alleviate this issue. Another improvement that could be made to our pipeline is to use other network protocols such as gRPC or use DDS (the underlying message protocol of ROS 2). In Particular, gRPC has the ability to reduce size of messages through flatbuffers. It may be possible to even cache the contents of messages in Redis and/or store messages for later use in a database. DDS would allow for focus on the pipeline to better integrate with robotics research, such as the various smart vehicles being developed in Reno.

Acknowledgment

The authors would like to acknowledge the City of Reno, as well as the Cyberinfrastructure Team in the Office of Information Technology at the UNR for all of their support. This material is based in part upon work supported by Washoe County Regional Transportation Commission (RTC) under grant number AWD-01-00002406. It is also based in part upon work supported by the National Science Foundation (NSF) under grants OAC-2209806, OIA-2019609, and OIA-2148788. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Washoe County RTC or NSF.

References

[1] K. Sato, R. Shinkuma, T. Sato, *et al.*, "Prioritized transmission control of point cloud data obtained by lidar devices," *IEEE Access*, vol. 8, pp. 113 779–113 789, 2020.

[2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15, Hangzhou, China: Association for Computing Machinery, 2015, pp. 37–42.

[3] S. Jain, S. Gupta, K. K. Sreelakshmi, and J. J. P. C. Rodrigues, "Fog computing in enabling 5g-driven emerging technologies for development of sustainable smart city infrastructures," *Cluster Computing*, vol. 25, no. 2, pp. 1111–1154, Jan. 2022.

[4] P. Bründl, A. Scheck, H. G. Nguyen, and J. Franke, "Towards a circular economy for electrical products: A systematic literature review and research agenda for automated recycling," *Robotics and Computer-Integrated Manufacturing*, vol. 87, p. 102 693, 2024.

[5] A. P. Ajay Kumar Mohammad Amir Khusru Akhtar and R. P. Srivastava, "Smart city vehicle accident monitoring and detection system using (mems, gsm, gps) raspberry pi 4," *IETE Journal of Research*, vol. 0, no. 0, pp. 1–9, 2022.

[6] C. Mai, H. Chen, L. Zeng, *et al.*, "A smart cane based on 2d lidar and rgb-d camera sensor-realizing navigation and obstacle recognition," *Sensors*, vol. 24, no. 3, 2024.

[7] T. Azfar, J. Li, H. Yu, R. L. Cheu, Y. Lv, and R. Ke, *Deep learning-based computer vision methods for complex traffic environments perception: A review*, en, Jan. 2024.

[8] *Zstandard - Real-time data compression algorithm*.

[9] M. Isenburg, "Laszip: Lossless compression of lidar data," *Photogrammetric engineering and remote sensing*, vol. 79, no. 2, pp. 209–217, 2013.

[10] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[11] B. Mishra and A. Kertesz, "The use of MQTT in m2m and iot systems: A survey," *IEEE Access*, vol. 8, pp. 201 071–201 086, 2020.

[12] MQTT. "MQTT - the standard for iot messaging." (), [Online]. Available: https://mqtt.org (visited on 03/24/2024).

[13] R. A. Atmoko, R. Riantini, and M. K. Hasin, "IoT real time data acquisition using MQTT protocol," *Journal of Physics: Conference Series*, vol. 853, no. 1, p. 012 003, May 2017.

[14] R. K. Kodali and B. S. Sarjerao, "MQTT based air quality monitoring," in *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, 2017, pp. 742–745.

[15] C. Carthen, A. Zaremehrjardi, V. Le, *et al.*, "Orchestrating apache nifi/minifi within a spatial data pipeline," in *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, 2023, pp. 366–371.

[16] A. Muñoz, C. Carthen, V. Le, S. D. Strachan, S. M. Dascalu, and F. C. Harris, "LDAT: A LIDAR data analysis and visualization tool," in *ITNG 2022 19th International Conference on Information Technology-New Generations*, S. Latifi, Ed., Cham: Springer International Publishing, 2022, pp. 293–301.