Robust Decision Making via Cooperative Estimation: Creating Data Saturated, Autonomously Generated, Simulation Environments in Near Real-Time

Israel Toledo-Lopez[1,2], Dylan Pasley[1,2](✉), Raul Ortiz[1,3], and Ahmet Soylemezoglu[1,2]

[1] Construction Engineering Research Lab, Champaign, IL 61801, USA
Dylan.a.pasley@erdc.dren.mil
[2] University of Illinois, Champaign, IL 61801, USA
[3] University of Puerto Rico – Mayaguez Campus, Boulevard Alfonso Valdes, 00680 Mayaguez, Puerto Rico

Abstract. Every branch of the U.S Military, as well as foreign military agents, have a vested interest in the broad applications and development of robotic systems. Advancements in data collection and storage capabilities has exposed an opportunity to increase the utility of simulated environments. At the most basic level, operations that involve robotic systems require detailed simulation environments to test algorithms and edge cases. The wealth of information collected from robotic platforms can be utilized to autonomously generate simulation environments, which can provide a robust platform for enhanced decision-making capabilities.

Current industry standards depend on labor intensive post processing methods which generate static simulation environments. These simulation environments lack much utility beyond controlled testing. To address this gap, we introduce the foundational research for an intelligent simulation module, a system that utilizes sensory data, collected from semi-autonomous robotic mapping platforms, to generate in near-real time high fidelity digital twin simulation environments of real-world locations. With this system, end-users will be provided with the details they need to make operational decisions without the delay of post processing.

Our system bridges the ROS platform with Unity3D game engine to achieve the generation of its simulated environments. Combat Engineer operations that rely on autonomous robotic platforms will benefit from having a system that can generate high fidelity digital twin simulation environments to aid testing research, mission planning, and robotics control. In general, the intelligent simulation system will allow for robust decision making in autonomous mobile robots, by improving navigation, path planning, coordination between agents, and task planning.

This research has the potential of being utilized in hardware in the loop scenarios where multi-agent control and coordination is required to complete a mission thus advancing the field of cooperative estimation. Further, with the use of virtual reality technology, an operator could potentially be inserted into an operation site virtually; the virtual environment and agents operating within it would be parallel to the physical site, and the operator can then possibly supervise, control, and coordinate both virtual and real hardware robotic systems remotely.

J. Mazal et al. (Eds.): MESAS 2021, LNCS 13207, pp. 273–289, 2022.
https://doi.org/10.1007/978-3-030-98260-7_17

Keywords: Near real time · Autonomous platforms · Digital twin · No post processing · Unity3D · ROS · Multi-agent control and monitoring · Virtual reality

1 Introduction

This paper introduces the research, including the challenges and successes that we have encountered while attempting to automate the process of generating a 3D simulated environment which can be extended and further utilized with some additional research.

1.1 The Problem Space

Military engineers are presented with a broad scope of responsibilities that have a direct impact on the built environment. One factor that is unique to the military engineer and sets them apart from their civilian peers is the expectation to operate within austere environments. The austere nature of these environments presents a significant amount of risk to the field engineers assigned to analyze the potential usability of specified areas of interest. To better carry out their mission, military engineers need to be provided with the appropriate tools as well as accurate and robust sets of data. Current methods of data collection provide a significant increase, both in the volume, and fidelity of data, that engineering units have at their disposal. However, the process of converting this data into a usable information platform is still a time consuming and resource intensive process. The shortcoming of the current process sets the foundation of this research effort. A concrete example is as follows; a military engineer is tasked with overseeing a construction effort in an unknown environment. Data is collected on the unknown environment, to provide the engineer with enough information for the planning and execution of the mission. The system this paper proposes takes this data and automatically generates a simulated environment; this simulated environment provides, to the engineer, the collected information in a manner that is now easy to digest and actionable. The engineer can interact with this simulation and plan out all the individual tasks for the mission. The system this paper proposes also has the ability to display simulated vehicles that can be synced to their physical counterparts, this means that engineers can also monitor the whole operation in real time utilizing our system.

1.2 Interdisciplinary Topics

The sections below outline the approaches we took in analyzing modern processes of collecting, storing, processing, and displaying data which include the use of robotic and autonomous platforms. Our decision to explore Unity 3d was informed by related research which conducted a comparative analysis of Gazebo and Unity [5, 16, 17] and found that it offers a degree of extensibility that can be utilized in robotics applications. Additionally, there has been a growing interest in connecting Unity to ROS [3, 4, 6] in recent years. Even with recent developments in the field it is important to note that the generation of a 3d simulation environments is inherently complicated, time consuming, and resource intensive task.

1.3 Primary Contributions

The primary contribution of this research is to create an enhanced 3D simulation environment which can be generated, updated, and displayed without relying on post processing. An additional requirement of this research mandates that this environment should be generated with minimal input from the end users. By addressing the issue stated above with an automated system we would enable the rapid utilization of data collected from the field in near real time. The development of such a system carries significant implications in the development and application of multi robot systems [6, 7], training machine learning algorithms [5], human machine interfacing via virtual and augmented reality [8–10] and cooperative estimation. Our own internal research confirms that there is a growing need for the capability to control and monitor multiple platforms in a safe and efficient manner. Although the research detailed in this passage does not fully meet this need, it begins to create a foundation that is robust and flexible enough to incorporate these capabilities in future research efforts.

By the end of this effort, we have been able to generate a 3D simulated environment without the extensive hassle of post processing. We were able to cut down the total time and effort required to create a simulated environment while ensuring that the environment accurately represents the real-world landscape where the data was collected. The findings of this research are promising, however there remains a significant amount of work to be done in order to optimize and further automate this process. This will be elaborated on in the results section of this paper.

1.4 Paper Outline

Section 2 will give a brief overview of the technology and research that provide the foundation for the efforts discussed in this paper. Section 3 will elaborate further on related research before we delve into the technical details that address the primary scope of this paper. Section 4 will provide the technical details that outline how we were able to create our simulation environment. The related research section (Sect. 3) will make use of the terms defined in the background section (Sect. 1) and will prepare the reader to better understand the methods section (Sect. 4). Finally, Sect. 5 concludes this paper by providing the results, limitations, and future efforts of our research.

2 Background and Rational

This section will briefly introduce the Robotics for Engineer Operations (REO) research project that preceded the efforts outlined in this paper. By the end of this section there should be a clear understanding of where our data sets come from, how they are processed, and where they are stored prior to map creation. To do this we briefly introduce the platform hardware, as well as the software dependencies that exist in our system. Additionally, this section should provide an explanation of our objectives which will later tie into our justification and the impact of applying this research.

2.1 Project Background

The effort to explore autonomously generated simulation environments is an expansion of the REO research effort being conducted at the Construction Engineering Research Center. REO seeks to extend the capabilities of military engineer units into the modern era by providing a semi-autonomous robotic platform which is capable of surveying and accurately depicting real-world operating environments to expose any challenges or obstacles that the warfighter might encounter. One outcome of this research to date is the development of a large data storage system, which we refer to as the Site Model Database. Proper utilization of the Site Model Database has allowed us to create a data visualization system that is generated in near real time, without the need of post processing, that provides the end user with a robust information system that can be used to guide them in their decision-making process.

An in-depth description of the REO system is outside of the scope of this manuscript. However, in order to explain our data acquisition strategy, it is necessary to provide a high-level overview of the REO system. This REO overview can be found in the system hardware section below. Additionally, the software section will cover the typical workflow that is followed during data collection and processing.

2.2 System Hardware

The data collection portion of this research is carried out on a semi-autonomous robotic platform which is designed to function in a completely offline environment by utilizing standard simultaneous localization and mapping algorithms. The robotic platform that we utilize can be retrofitted with sensor payloads that are specific to its individual mission. For the purpose of this paper the platform was equipped with a 16 channel Velodyne sensor which provides us with our primary point cloud dataset. In order to collect enough information to enable the user to make robust decisions we also use stereo and mono cameras to collect colorized image data. This data is combined with the lidar data in order to create a depth image of the operating environment. In order to facilitate on board processing the payload includes at least one Karbon 700 computers. In order to conduct our tests we ran the program on Razor's Blade 15 studio edition laptop.

The configuration presented above provides the general hardware pieces that are needed to gather and process the data that we use to create our simulated environment. To explain the software components of this research the next section will provide a brief workflow and address the individual software pieces that are responsible for each section of the workflow.

2.3 Software

Initially, our platform is introduced to the environment where it will begin collecting data and building a map using iterative closest point (ICP). In an effort to increase the ability for the platform to localize, our system ingests an initial set of a-priori data. This a-prioi data is data which has already been previously collected and made available for use. In our case the data sets we utilize as a-priori data are 2D GeoTiffs (Fig. 1).

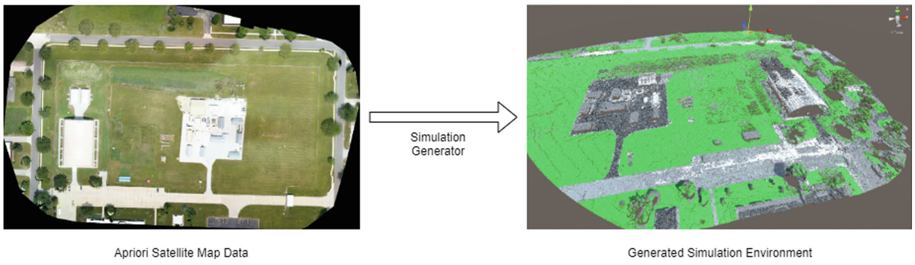


Fig. 1. A-prioi input and 3D mapped output

In order to facilitate the interchange of data between internal subsystems of our platform we rely on ROS. ROS is a set of open-source software libraries and tools which facilitate many common robotic operations [13]. Our data is sent through a series of ROS nodes where it is translated and voxelated before it finally reaches our internal database which we refer to as the Site Model Database. The voxel becomes a core data structure, which is addressed below in the methods section. In order to work with the large quantity of data that is collected in the map building process we chose to use a non-relational database scheme which is currently facilitated by MongoDB. Once the data has gone through some initial processing and is stored in the Site Model Database it can be used to generate display outputs for 2D, 3D, and AR/VR systems. At this point, the initial processing of data is complete, and it is now stored in a format that we can work with to create a simulated environment inside of Unity 3D.

It is important to highlight some of the significant software dependencies that influenced our design decisions of this system's architecture. Perhaps the most significant requirement belongs to ROS and Unity. ROS is compatible with Linux operating systems, whereas Unity 3D is currently most stable on Windows. This set of requirements presents a significant challenge in the attempt to create a bridge between the two operating systems. Fortunately, this issue has been addressed and has seen significant development in previous years [1]. Some related research suggests that in the coming years and the further development of ROS 2 will eliminate this dependency on a ROS-Unity bridge. For now, this connection is made using previously explored socket technology and ROS # [11]. For the purpose of comparability, it is important to note that this project used the HTC Vive system for our virtual reality display. This system operates Steam VR which is the software component for the HTC Vive. Any other dependencies and limitations of this research will be elaborated on in the results and conclusion section of this paper.

2.4 Objectives

After conducting this research, it has become clear that there are multiple objectives that can be obtained through continued advancement of this foundational research. First and foremost, the objective of this research is to establish an applied framework for automating the creation of simulated environments which are identical to real world locations. In essence, we aim to create digital twins of real-world operating environments in an autonomous fashion. This research establishes a proof of concept that carries significant implications to future work which will be addressed below. With these implications in mind, it is also an objective of this research to create an environment using software components that are flexible and extensible. The intention of this environment is to enhance the end user's ability to view and interact with the data collected.

2.5 Justifying the Research

By creating a 3dimensional simulation environment that is accurate and reflects the real-world operating environment we provide the end user with a data rich, responsive, and integrated environment that is created for, tasks that involve the use of cooperative estimation. There are two functional scales at which we operate at for this research. The first is at the sensor-to-sensor level, where a suite of sensor work together to accurately locate objects as they appear in the real world. Once this research is expanded the capability to use and track multiple machines in the same operating environment can be utilized to perform cooperative estimation as a means of confirmation of the systems accuracy.

3 State of the Art

Past work in the area of modeling and simulation in the fields of robotics has focused mostly on generating an accurate representation of a robotic agent that can be observed and utilized in a simulated environment. This paper focuses on the simulated environment itself; generating a simulated environment near-real time, that is as close to a real location as possible so that robotic agents can be studied in this simulated space. In this section, we will discuss some of the past works that are related to this research.

Babaians et al. make two contributions with their work. First, they propose a method to interface ROS from the Unity engine. Second, they simulate various robotic sensors such as LIDAR, RGBD and Monocular cameras in Unity. Although our work does not share their objectives, it does shed some light on the advantages and shortcomings of ROS# as ROS to Unity interface (which is the interface we use in our research). Their work also highlights the tools and techniques available in Unity to simulate robotic sensors, which is something that will be added to our system in future efforts. This paper also highlights that: "Simulators cannot guarantee the final result for industrial or mobile robotic applications since the success of off-line programming depends on how similar the real environment of the robot is to the simulated environment" [1]. This statement supports the need for a system like ours, that generates a simulated environment that is analogous to the real environment the robot operates in.

Codd-Downey et al. develop in their work a virtual reality-based teleoperation interface for autonomous systems by bridging ROS and Unity [2]. Utilizing their system, a user can take control and teleoperate a robotic ground vehicle through a virtual reality platform. The simulated environment that the user can drive the robotic vehicle through is mapped by the robot utilizing a standard SLAM algorithm and afterwards it is prestored and loaded into Unity as a particle system. Thus, the simulated environment in this system is static and offers little details to the user; in comparison, the system proposed in our work generates dynamic maps that offer detailed information of the robotic vehicle's surroundings.

Hu et al. present in their work a real-time three-dimensional simulation system, ROSUnitySim, for local planning by miniature unmanned aerial vehicles (UAVs) in cluttered environments [3]. In this work the authors focus on sensor modeling (mainly LIDAR), the interface between ROS and Unity and multiplatform control. This last feature is something that we would like to add to our system in future efforts; the ability to control and observer several robotic vehicles at the same time. The simulated environment that was used to fly the simulated UAVs in this work was static. handcrafted and did not necessarily resemble any real-life location. Handcrafting a map requires time and effort from developers and it is not guaranteed to resemble the target real-life location, thus here we see an example where our system would've been a better environment choice for the simulated UAV flights.

4 Methodology

The following sections present the architecture of the Environment Simulation System developed in this work. This research was conducted in order to generate a near-real time simulated environment that is dynamic, immersive, and interactive. This section concludes with a discussion of the separate components that comprise our system. Using the processes outlined below we were able to successfully establish an automated approach to displaying large quantities of data into an interactive simulated environment. The foundational platform that was established with these methods can be used to improve the level of detail provided to the end user and ultimately better inform their decision-making process.

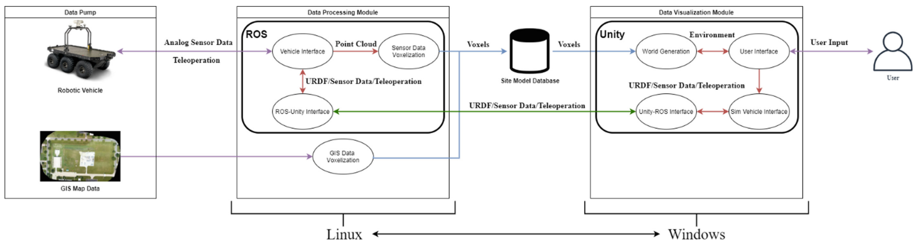


Fig. 2. Architecture for the environment simulation system

4.1 Architecture

An overview of the proposed architecture can be seen in Fig. 2. As shown in the figure, there are four main components that make up this system: the data pump, data processing module, database, and data visualization module. The general workflow of this system is as follows; first, the data pump component introduces real-world data into the system. This data is then used to generate the simulated environment. Next, the data processing module takes in the real-world data, provided by the data pump, generates voxel objects, and stores them in the database; it also handles one end of the interface between the simulated environment and the robotic vehicle.

The Site Model database houses the voxel objects generated by the data processing module. This database is where the voxel objects will be stored until they are called upon for visualization. Finally, the data visualization module retrieves the voxels from the database, generates the simulated environment, and renders the environment for the user. Additionally, the data visualization module handles user input and facilitates interactions between the simulated environment and the simulated vehicle. it also takes care of user input and the other end of the interface between the simulated environment and the robotic vehicle.

4.2 Data Storage and Utilization Components

This section will present an in-depth description of the backend of the developed system. The components discussed in this section are responsible for gathering real-world data, generating voxel objects, and storing said voxels.

Data Pump. The Data Pump gathers and introduces real-world data into the system through its two main methods: the GIS maps and the robotic vehicle. In the first method, we utilize GIS maps, which can be collected from either satellite imagery, or unmanned aerial vehicle (UAV) imagery, to provide an accurate geographic description of the area to be modeled by the system. All the pixels in these maps are geographic points tied to real coordinates, therefore they introduce to the system a list of points with latitude, longitude, altitude and RGBA color information that is then used to generate the voxels.

The second method of data collection utilizes our robotic platform. The J8 Atlas Xtreme Terrain Robot robotic vehicle, is a ROS based electric, eight-wheeled, amphibious, and all-terrain mobile unmanned ground vehicle (UGV) [14]. Once this platform is fitted with an array of sensors it can capture a detailed map of its surroundings and inject this map into the simulation generator. Our J8 load-out is capable of gathering lidar data, providing camera feedback, reporting its live GPS location, as well as other important feedback data. Although the different data collected by the J8 are important, this system is primarily interested in the lidar data. Using the lidar data, the Data Processing Module can generate the voxel objects that make up the simulated world environment.

Data Processing Module. The Data Processing Module is responsible for managing the interface between the system and the robotic vehicle. This module is also responsible for generating and storing voxels, as well as managing the Linux end of the interface between the robotic vehicle and the simulated environment. This module is comprised of four components: the vehicle interface, the sensor data voxelization service, the ROS-Unity interface, and the GIS data voxelization service.

Due to the current limitations of ROS that are outlined in the related research section above, each of these components operate within the Linux operating system. In order to interact with the robotic platform directly, we placed the vehicle interface, ROS-Unity interface, and the Sensor Data Voxelization service inside the ROS platform. This design decision can be seen in Fig. 2. The Vehicle Interface is a series of ROS scripts that enable communication and data transfer between the on-board computer(s) in the robotic vehicle and the rest of the system. The ROS-Unity Interface handles communication between the robotic vehicle and the simulated environment on the Linux side of things. This component is essentially a bridge between Linux and Windows, specifically between ROS and Unity. The ROS-Unity Interface mainly makes use of ROS#, a set of open-source software libraries and tools in C# for communicating with ROS from.NET applications, in particular Unity [12].

Within the ROS# libraries, the ROS-Unity Interface utilizes the ROS package *file_server*, which allows the system to transfer files between platforms. With this package the system can transfer the robotic vehicle URDF files (XML format that represents a robot model) to Unity where the model is rendered for the user. Another important ROS package used by the ROS-Unity Interface is the *rosbridge_server*, this package creates a WebSocket connection that allows outside platforms to interact with ROS through JSON messages. The *rosbridge_server* package can receive JSON strings and convert them into ROS calls, as well as convert ROS responses into JSON string. The ROS-Unity Interface utilizes this package to transfer sensor data and teleoperation commands between the two platforms. The GIS Data Voxelization service is a series of scripts that take GIS map data and generate voxel objects with it. The service iterates through the pixels of a GIS map and retrieves the following information from each pixel: geographical coordinates (latitude, longitude, altitude), local pose (xyz coordinates based on the origin of the map and x y z w orientation), and RGBA color information (red, green, blue, alpha).

With this information the service generates a voxel object that corresponds to each pixel and stores it in the Site Model Database. The voxel object contains an id, the type of material, its geographical coordinates, and its local pose. The type of material is determined by the service utilizing a simple classifier based on the color information of the pixel. Figure 3 shows the structure of a voxel object with details on the information it holds. Once all the voxels are generated, they are inserted into the Site Model database. The generation of our simulated environment happens in two phases. First the Voxels that are created from GIS map data form are added into the environment. Once the GIS generated voxels are added, the system ingests the data that was collected from the J8's sensors. After the J8 data is added, our simulated environment is complete.

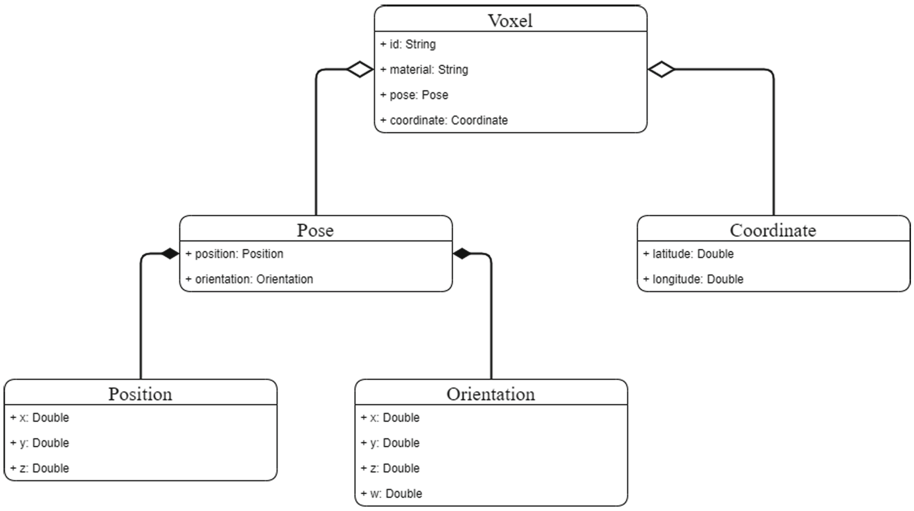
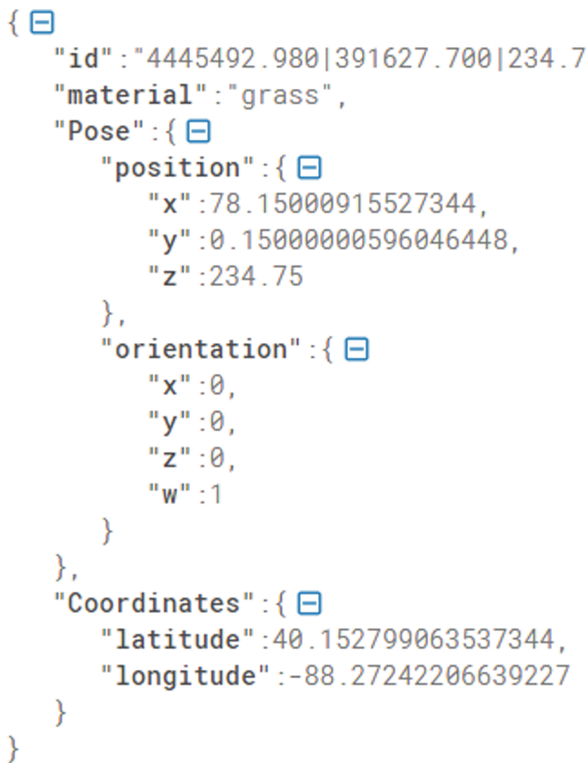


Fig. 3. Voxel object UML diagram

The last component of the Data Processing Module, the Sensor Data Voxelization service, is a series of scripts that take in colorized point cloud data from the Vehicle Interface and generate voxel objects with it. The service iterates through each point in the point cloud data, extracts relevant information (abovementioned in the GIS Data Voxelization service description), generates a voxel object for each point and inserts them into the Site Model Database. This service is responsible for consistently updating the simulated environment, thus any changes in the real-life environment captured by the robotic vehicle will be reflected in the simulated one. Also, this service fills the information gaps in the environment left by the GIS map data; for example, a GIS map may have information on the roof of a building, but no information on any of its walls, with the Sensor Data Voxelization service, it is possible to generate that missing information and introduce it to the simulated environment.

Site Model Database. The Site Model Database component contains the data elements required to create the simulated environment as a list of voxel objects. These voxel objects are inserted by the Data Processing module and retrieved over the network by the Data Visualization module. This database was set up utilizing MongoDB, a document database that stores data in JSON-like documents. The advantages of using MongoDB are its data flexibility (data structure can be changed over time), ad hoc queries, it is a distributed database, and it is free to use. Figure 4 shows an example of a voxel object stored in the Site Model database.



```
{
   "id":"4445492.980|391627.700|234.75|16",
   "material":"grass",
   "Pose":{
      "position":{
         "x":78.1500091552734 4,
         "y":0.15000000596046448,
         "z":234.75
      },
      "orientation":{
         "x":0,
         "y":0,
         "z":0,
         "w":1
      }
   },
   "Coordinates":{
      "latitude":40.152799063537344,
      "longitude":-88.27242206639227
   }
}
```

Fig. 4. Voxel object example in the database

4.3 Data Visualization and Interaction

This section presents a detailed description for the frontend of the developed system. The component discussed in this section is responsible for generating and rendering the simulated environment, as well as managing the user interface.

Data Visualization Module. The Data Visualization module is responsible of managing the user interface, managing the simulated vehicle interface, generating and rendering the simulated environment with voxel objects, and managing the Windows end of the interface between the robotic vehicle and the simulated environment. This module is made up of four components: the World Generation service, the Unity-ROS interface, User Interface, and Sim Vehicle Interface. These components operate in the Windows operating system and within the Unity game engine as seen in Fig. 2.

The World Generation service is a series of scripts that generate and render the simulated environment based on the voxel objects available in the Site Model database. The simulated environment is made up of quads (Unity primitive plane that has edges of one unit long and its surface is oriented in the XY plane of the local coordinate space [15]) that are meshed to form blocks (simple cubes with one-unit long sides), these blocks are then meshed to form chunks and the chunks are lined up to form the world or simulated environment. Figure 5 shows the structure of the simulated environment.

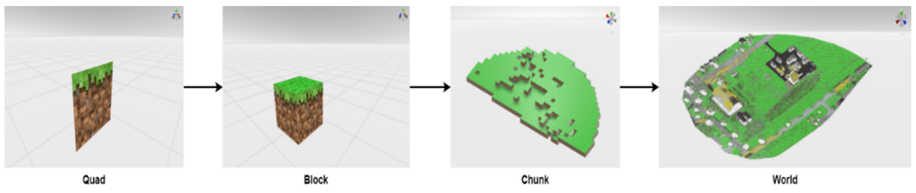


Fig. 5. Structure of the simulated environment

The steps taken to achieve the simulated environment generation and rendering can be seen in the flowchart in Fig. 6 and 7. The Unity-ROS Interface makes use of ROS#. Specifically, the following ROS# plugins are utilized: *RosBridgeClient* and *UrdfImporter*. The *RosBridgeClient* plugin is the.NET API that interacts with ROS through the *rosbridge_server* package and allows communication between the platforms from the Unity side. The *UrdfImporter* plugin imports the robot's URDF files to Unity, parses them, and generates a model of the robot in the Unity environment (Fig. 8).

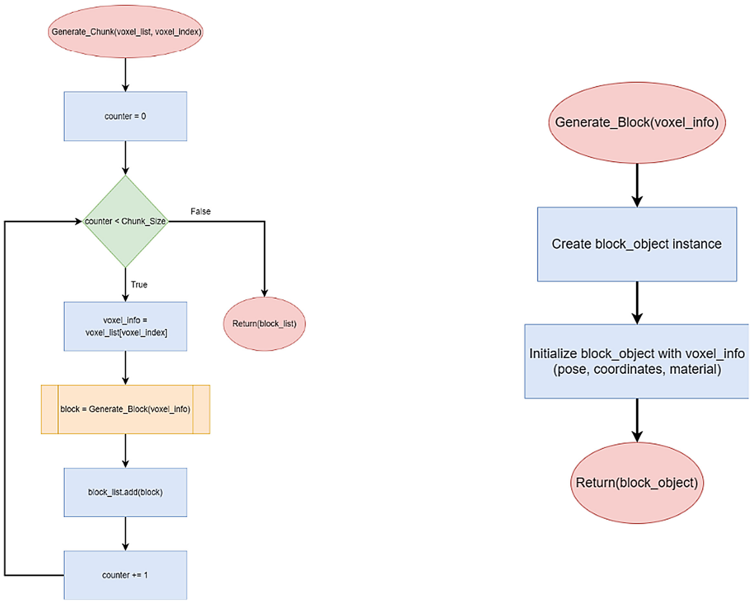

Generate_Chunk(voxel_list, voxel_index)
counter = 0
counter < Chunk_Size
False
True
Return(block_list)
voxel_info = voxel_list[voxel_index]
block = Generate_Block(voxel_info)
block_list.add(block)
counter += 1
Generate_Block(voxel_info)
Create block_object instance
Initialize block_object with voxel_info (pose, coordinates, material)
Return(block_object)

Fig. 6. Left to right: generate chunk, generate block

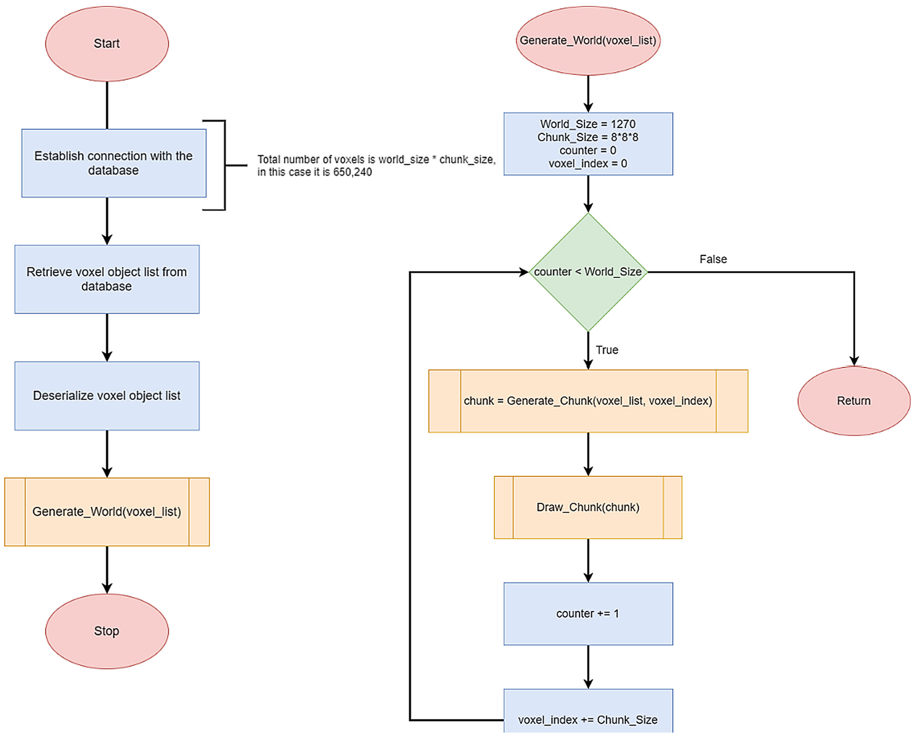

Start
Establish connection with the database
Total number of voxels is world_size * chunk_size, in this case it is 650,240
Retrieve voxel object list from database
Deserialize voxel object list
Generate_World(voxel_list)
Stop
Generate_World(voxel_list)
World_Size = 1270
Chunk_Size = 8*8*8
counter = 0
voxel_index = 0
counter < World_Size
False
True
chunk = Generate_Chunk(voxel_list, voxel_index)
Return
Draw_Chunk(chunk)
counter += 1
voxel_index += Chunk_Size

Fig. 7. Left to right: overview, generate world, generate chunk, generate block workflows.

The Sim Vehicle Interface is a series of scripts tied to the robotic vehicle simulated model that allows the user to interact with the simulated robotic vehicle and with the actual robotic vehicle. With this interface, and using the ROS# plugins, the simulated robot could receive sensor feedback from the physical platform and display it for the user. Additionally, it could receive user input (like teleoperation commands from Unity) and execute them in the physical robotic vehicle. Thus, this interface permits a synchronization between the simulated robotic model and the physical platform; it also gives the user control over the simulated robotic vehicle in the simulated environment.

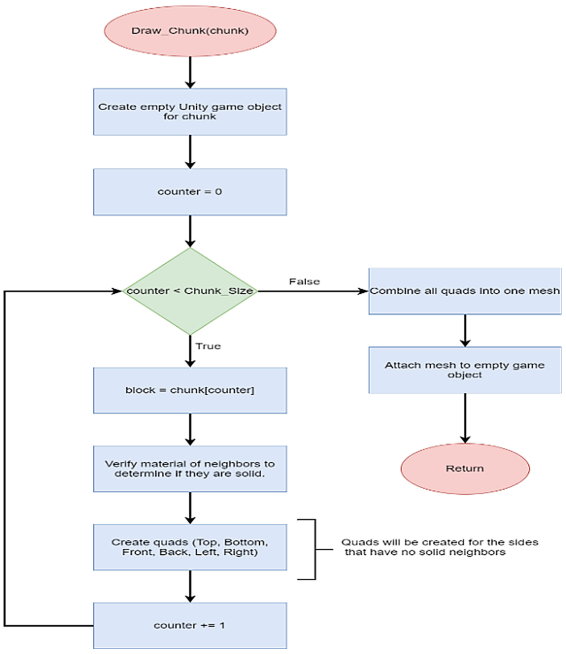

Draw_Chunk(chunk)
Create empty Unity game object for chunk
counter = 0
counter < Chunk_Size
False
True
Combine all quads into one mesh
Attach mesh to empty game object
Return
block = chunk[counter]
Verify material of neighbors to determine if they are solid.
Create quads (Top, Bottom, Front, Back, Left, Right)
Quads will be created for the sides that have no solid neighbors
counter += 1

Fig. 8. Workflow for drawing a chunk

The final component in the Data Visualization Module is the User Interface, which is a series of scripts that allow the user input commands to interact with the simulated environment and simulated robotic vehicle. The User Interface gives the user a Virtual Reality (VR) and a First-Person control over the whole system. The VR controls were achieved utilizing Unity's XR Interaction Toolkit plugin. This plugin tracks the user's VR headset movement (in this case the HTC Vive Pro) to control the in-game camera and the hand controllers to move around the simulated environment. The First-Person controls were achieved with a series of Unity scripts, and they allow the user to control the simulated robotic vehicle and move around the simulated environment with the use of regular joystick controllers or with a mouse and keyboard.

5 Conclusion

5.1 Results

Through the efforts of this research, we have been able to connect the Site Model Database system hosted in Linux to the Unity 3d platform hosted in Windows. The large technical challenge here is presented by bridging these separate systems and allowing them to properly communicate with each other without introducing significant lag and without relying on postprocessing. This has been an issue that was thoroughly explored in previous research. Once this was resolved we were able to pull large amounts of voxelated data (around 650,00 voxels that cover a radius of 250 m around the GIS map's origin) into the unity environment in approximately 6 min and 26 s. Figure 9 shows the render time of the environment in relation to the number of voxels.

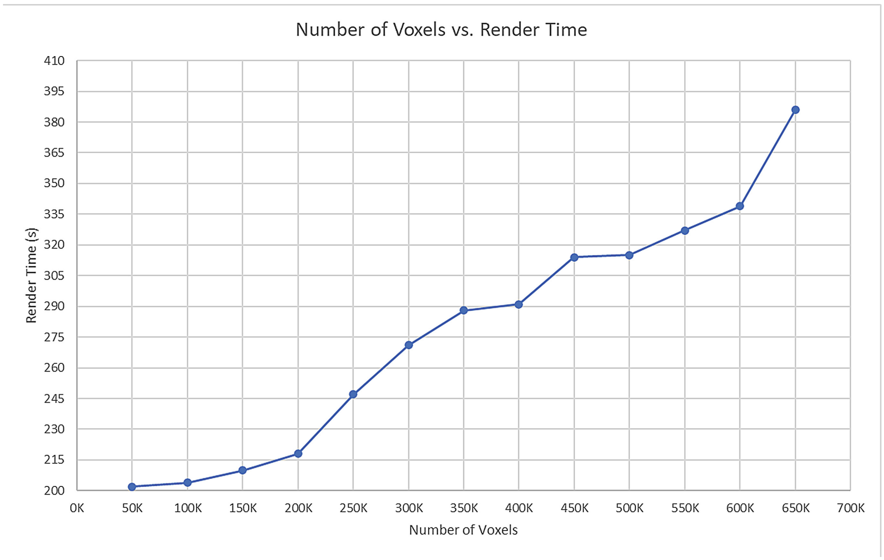


Fig. 9. Graph of the number of voxels vs. the render time (in seconds)

This time expands in an inconsistent manner once we integrated the virtual reality interface into the system. When conducting this exercise, we have experienced rendering times of around 7 to 7:30 min. Table 1 shows the render time for the system in different scenarios and with different combinations of elements integrated into the system, like for example the virtual reality controller. As for fps (frames per seconds) performance, we did not observe any change in performance in relation to the number of voxels, it maintained a steady rate between 900 and 1000 fps. We did observe fps drops when we integrated other elements into the system, for example by integrating the virtual reality controller we observed the frames per second drop to around 42 fps. Table 1 shows the frames per second observed in the different test scenarios. At this point the ability to automate the production of digital environments has been established.

With the current state of the research, we have been able to establish Unity 3d as a flexible environment. Using Unity, we have been able to establish control of a camera avatar allowing a user to navigate the world freely. Additionally, we have been able to establish rudimentary control of a simulated robotic platform and drive it around the simulated world. Additional work is needed to improve this system to add more features and increase its utility. These improvements are discussed in the future work section. Screenshots of the product can be seen in Fig. 10.

Table 1. Scenario based performance comparisons based on render time

Sim objects rendered	Render time (minutes/seconds)	Frames per seconds
World	6:26	900
World & sim. robotic vehicle	6:27	700
World & first-person controller	6:26	84
World & VR controller	7:30	45
World & VR controller & sim robotic vehicle	7:31	42

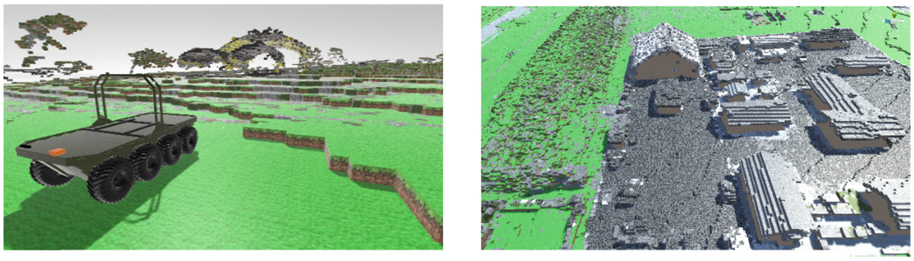


Fig. 10. Left: J8 rendered in simulation. Right: Simulated environment output

5.2 Future Work

The previous section established that we have successfully created an automated workflow for generating a 3D Simulated environment inside of Unity. Additionally, we have established the ability to navigate a viewport around the environment as well as control a simulated robot. This research confirmed that Unity is collaborative and flexible enough to handle additional capabilities with further research.

Short term work will focus on optimizing the display and generation of the world environment. The goal of this effort will be to establish a consistent and reliable load time while increasing the resolution of displayed data. Additional work efforts will focus on utilizing the connection between the simulated platform and the real world platform, to simulate several different robotic sensors, increase the level of control the user can exhibit within Unity, add user interface options that take provide better situational awareness to the end user, increase the level of automation in this process, add the ability to send world modification updates back to the site model database, and explore this platform's potential to control multiple real world robots. Once this final effort is complete this system would be able to utilize cooperative estimation algorithms as a means of redundant verification of real-world object placement.

References

1. Babaians, E., Tamiz, M., Sarfi, Y., Mogoei, A., Mehrabi, E.: ROS2Unity3d; High-performance plugin to interface ROS with Unity3D engine. In: 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium 2018, pp. 59–64. IEEE (2019)
2. Codd-Downey, R., Mojiri, P., Speers, A., Wang, H., Jenkin, M.: From ROS to unity: leveraging robot and virtual environment middleware for immersive teleoperation. In: Proceeding of the IEEE International Conference on Information and Automation, pp. 932–936. IEEE, China (2014)
3. Hu, Y., Meng, W.: ROSUnitySim: development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning. Simulation **92**(10), 931–944 (2016)
4. Hussein, A., García, F., Olaverri-Monreal, C.: ROS and unity based framework for intelligent vehicles control and simulation. In: IEEE International Conference on Vehicular Electronics and Safety (ICVES), pp. 1–6. IEEE, Spain (2018)
5. Konrad, A.: Simulation of mobile robots with unity and ROS- a case-study and a comparison with Gazebo. In the Department of Engineering Science University West (2019). diva2:1334348
6. Meng, W., Hu, Y., Lin, J., Lin, F., Teo, R.: ROS+Unity: an efficient high-fidelity 3D multi UAV navigation and control simulator in GPS-denied environments. In: IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, pp. 002562–002567. IEEE, Japan (2015)
7. Reid, R., Cann, A., Meiklejohn, C., Poli, L., Boeing, A., Braunl, T.: Cooperative multi-robot navigation, exploration, mapping and object detection with ROS. In: IEEE Intelligent Vehicles Symposium (IV), pp. 1083–1088. IEEE, Australia (2013)
8. Roldán, J.J., et al.: Multi-robot systems, virtual reality and ROS: developing a new generation of operator interfaces. In: Koubaa, A. (ed.) Robot Operating System (ROS). SCI, vol. 778, pp. 29–64. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91590-6_2
9. Rosen, E., Whitney, D., Phillips, E., Ullman, D.: Testing robot teleoperation using a virtual reality interface with ROS reality. In: Proceedings of the 1st International Workshop on Virtual, Augmented, and Mixed Reality for HRI, Illinois, USA (2018)
10. Sidaoui, A., Elhajj, I., Asmar, D.: Human-in-the-loop augmented mapping. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3190–3195. IEEE, Spain (2018)
11. Whitney, D., Rosen, E., Ullman, D., Phillips, E., Tellex, S.: ROS reality: a virtual reality framework using consumer-grade hardware for ROS-enabled robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Spain (2018)
12. ROS-Sharp GitHub. https://github.com/siemens/ros-sharp/wiki/. Accessed 25 July 2021
13. ROS Home Webpage. https://www.ros.org/. Accessed 25 July 2021
14. Army Technology Webpage. https://www.army-technology.com/projects/j8-atlas-xtreme-terrain-robot-xtr/. Accessed 14 July 2021
15. Esri Webpage. https://www.esri.com/en-us/what-is-gis/overview. Accessed 27 July 2021
16. Marian, M., Stîngă, F., Georgescu, M.-T., Roibu, H., Popescu, D., Manta, F.: A ROS-based control application for a robotic platform using the gazebo 3D simulator. In: 21st International Carpathian Control Conference (ICCC), Slovakia (2020)
17. de Melo, M.S.P., da Silva Neto, J.G., da Silva, P.J.L., Natario Teixeira, J.M.X., Teichrieb, V.: Analysis and Comparison of Robotics 3D Simulators. In: 21st Symposium on Virtual and Augmented Reality (SVR), Brazil (2019)