USING UNITY FOR 3D OBJECT ORIENTATION IN A VIRTUAL ENVIRONMENT

Hassam Khan Wazir and Fawaz Yahya Annaz

Institut Teknologi Brunei, Department of Electrical & Electronic Engineering, Faculty of Engineering, Jalan Tungku Link, Gadong, BE 1410, Bandar Seri Begawan, Brunei Darussalam, *Email:* hassamkhanwazir@yahoo.com, fawaz.annaz@itb.edu.bn

Keywords: UAV, virtual environment, robot navigation

Abstract

Although Unmanned Aerial Vehicles (UAVs) have gained a lot of attention by researchers in the past couple of decades, the development of UAV safe and efficient test platforms still needs much attention. The main reason for this being the unstable nature of the platforms and the potential risks associated with multi-rotor UAVs. To address and understand the above problems, this paper introduces UAV navigation in a virtually generated environment that is coupled to the real hardware platform. The virtual environment (VE) is developed using the Unity game engine, providing users with sufficient tools to build 3 dimensional, multi-level maps. The VE parses the orientation data (received from the hardware) and depicts instances of a UAV inside the created VE maps. Thus, this approach provides a suitable environment to examine different geographical scenarios and to test control and navigation algorithms to their limits without the need to physically manoeuvre the UAV and risk damaging it while doing so.

1 Introduction

Virtual environments (VEs) play a very important part in research and development by providing researchers with a way to dynamically alter environments to test different scenarios without the need to physically rebuild environment every time a scenario is changed. Thus it offers flexibility, reduces cost, eliminates crash risks, and reduces effort to develop, fly and control UAV systems. This approach has been mainly used in studies that are related to autonomous and remote controlled helicopter and multi-copter tracking. The studies range from obtaining real-time orientation and position data from the UAV to building more accurate UAV models in order to create learning tools and VEs for UAV platform testing and flight training.

2 2D Versus 3D Virtual Environment

Virtually created 2D environments have been used in the past to test navigation algorithms and examine the performance and efficiency of various types of ground-based robots if physically building the environment proved to be very costly or time consuming. 2D VEs provided a reliable way to gather data, monitor and control various types of Unmanned Ground Vehicles (UGVs), [1], [2] and [3]. 2D-VEs do a decent job of when UGVs are considered, provided that flat ground is assumed, however, they are at a disadvantage when multiple degrees of freedom visualising is considered in cases such as articulated robots, humanoids and UAVs, where movements of these robots can only be fully explained and visualised in a 3D space.

Although a number of researchers in the past have created 3D-VEs using game engines, there were major differences and limitations related to cost and portability [4] [5] [6]. A number of researchers nowadays are using sophisticated equipment [7] and [8], for example, VICON Motion Capture System was used to measure UAV orientation and position to obtain very precise data to perform complex UAV manoeuvres and control multiple UAVs simultaneously [9] and [10]. However, the cost of this kind of equipment is very high and is only feasible for very advanced research in the field of aeronautics and autonomous control systems. In this research, the project tackles this issue by using a low-cost, high sensitivity IMU (attached to a UAV test-bed) to measure orientation, with the earth acting as the inertial frame of reference. The IMU can also be connected to the computer in the absence of the test-bed to navigate through a 3D generated map.

Software containing 3D environment, collision detection and complex physics is very difficult to build from the ground up and can be overwhelming and time consuming for a small research group. Therefore, to improve productivity and to facilitate rapid prototyping, the VE was built using the Unity game engine [11]. Although the idea of using commercial and open source game engines to develop 3D simulators is not a new and there are instances where researchers have used game engines such as Unreal [12] [13] [14], however, those are very few and the literature offer limited examples. This is simply because (in the past) game engines did not possess the necessary capabilities to create fully featured 3D simulation software, and the ones that did were sold at very high prices. It is only in the past decade that the price of game engines dropped significantly, due to the rapid increase in processing power, graphical detail and affordability of computers, which has led to an increasing number of open source and free versions of commercial game engines available for use.

3 Related Work

For the purpose of the study, a suitable programing language or an Integrated Development Environment (IDE) was required to build a VE with specific capabilities. What follows is a complete list of capabilities the IDE and the VE had to possess in order to be viable for use as part of the project. The Virtual Environment:

- creates 3D multi-level maps
- saves and loads maps from a popular text file format
- reads and parses the output data from an IMU
- uses the parsed data to interact with a virtual flying object
- implements collision detection for the virtual objects
- has an interactive Graphical User Interface (GUI)
- runs on multiple platforms (Windows, Mac OS X, Linux)
- utilises preferably open source IDE, or free proprietary software.

Upon thorough study of the literature, it was observed that a few game engines and several different types of robot simulators have been used by researchers in the past to simulate robotic movement and create VEs. All of them have pros and cons, depending upon the type of VE being created.

There are many robot simulators available that have integrated physics and are designed for rapid prototyping such as:

- USARSim is a high-fidelity, open source, robot simulation software based on the open source version of Unreal Engine 3. It is widely used in various robotics competitions such as RoboCup rescue virtual robot competition (RoboCup) and the IEEE Virtual Manufacturing Automation Competition (VMAC).
- Gazebo [15] is another such 3D robotics simulator that uses Open Dynamics Engine (ODE) as the physics engine along with Ogre rendering engine, which is an open source engine that provides high quality visual rendering.
- Webots [16] also uses the ODE as the physics engine with a C/C++ and Java Application Programmer's Interface (API). It is commercial simulation software that is normally used for algorithm testing and provides the option to transport its controller programs to commercially available real robots.

Besides open source and commercial robot simulators, a few game engines like Unreal engine (mentioned above) and Panda3D have previously been used to create VEs. In this study, Panda3D and Unity were tested to decide on which one is to be used for future developments.

Panda3D developed by Disney is an open source game engine and a framework for 3D object rendering and game development. Although the engine itself is developed in C++, the intended game-development language is Python. Since it is a 3D game engine, some of the facilities that Panda3D provides include collision detection, fully integrated physics system, support for I/O devices and 3D audio support. However, Panda3D has a steep learning curve and a small developer community, preventing rapid development and prototyping. The documentation related to its API is also not complete, making it difficult for beginners to fully understand and utilize the tools that Panda3D provides for the development of the VE.

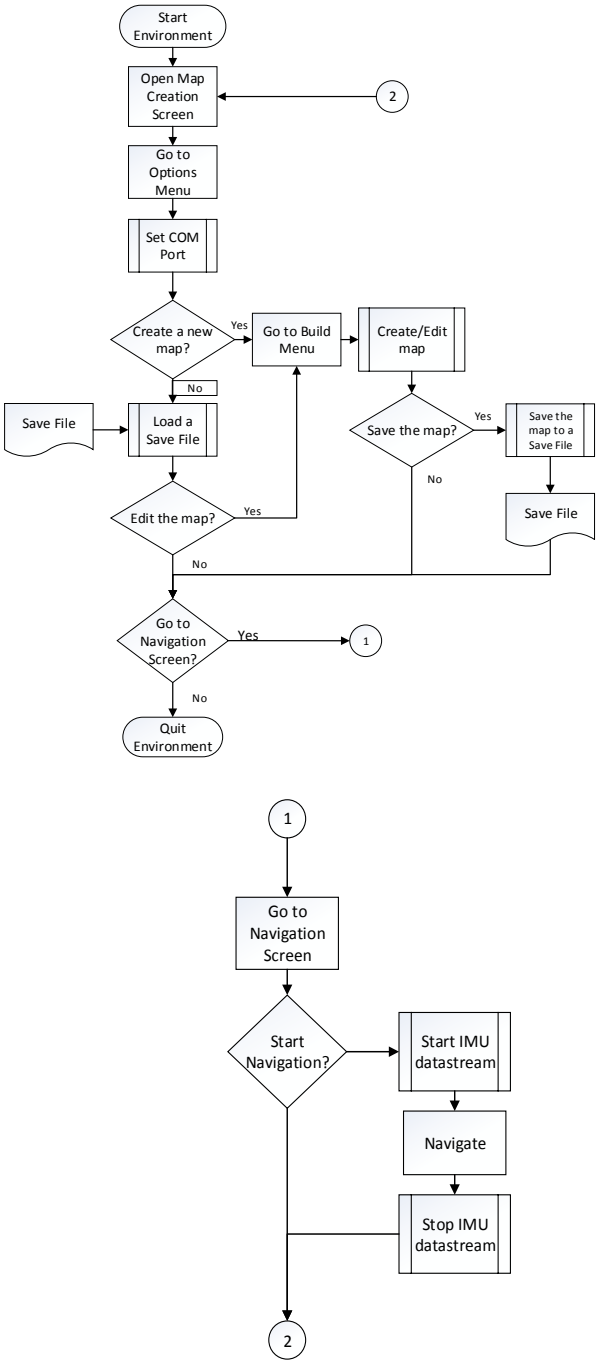


Figure 1. System architecture of the virtual environment

Thorough testing of both game engines reveals that Panda3D surpassed Unity in terms of the cost benefit, as it is an open source game engine. In all other aspects related to documentation, developer community, and user-friendliness, Unity was found to be far more flexible and easy to use. Besides, the free version of Unity provided all the necessary tools required to make quality software without any added cost. The existence of a huge community on the internet also helped in reducing the learning curve and resulted in rapid progress in software development. Therefore, Unity was chosen as the game engine to be used for the project.

4 System Architecture

Figure 1 shows the complete system architecture. The novelty of the research required the development of a system architecture that enabled VE development with hardware integration. The VE design was divided into the following parts:

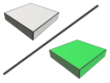
- Graphical User Interface (GUI)
- Creating and Destroying Objects
- Save/Load SystemGraphical User Interface (GUI): A minimalistic approach was used while designing the GUI, with primary and secondary menus. The primary menu has six primary controls namely Build, Options, Save, Load, Navigate, and Quit. Placing mouse over the Build or Options button opens up their respective secondary menus that deal with map building and software settings respectively.

Creating and Destroying Objects: At a very basic level, the main objective of the VE is to create and destroy objects (henceforth termed as "cubes") when the user clicks the appropriate mouse button. A single cube acts as a brick, and thus, multiple bricks can be stacked on top of each other to build a column. Similarly, multiple columns can be placed side-by-side to create a wall. Thus, using this simple strategy, an environment can be built to represent almost any map or lower resolution approximations. These maps are known as "Voxel" maps, and the software created to build these maps is termed as a "Voxel Engine".

The term "Voxel" is a combination of the word "volume" and "pixel". A pixel is the smallest unit in a 2D screen and every image displayed on the screen is a combination of different pixels. Similarly, the cube is the smallest unit in a Voxel Engine and the entire map is created using a combination of these cubes. So, the cubes can be thought of as the pixels of the Voxel Engine, and since these cubes have volume, they are termed as Voxels.

Maps created in a Voxel Engine are an easy and effective way of testing a simulation if very high environmental detail is not a priority. In this way, a basic map can be efficiently constructed in a very short time, which makes it ideal for rapid prototyping. There are three types of cubes used in the VE, namely, Wall Cube, Floor Cube and Gap. Although these cubes serve as Voxels, they are named "cubes", and in reality they are cuboid in shape of a "cube.

Table 1. GUI description

Icon	Description
	Open Build menu
	Select Grid Size
	Place a Wall cube on left click
	Place a Floor and Gap cube on left and right mouse click respectively
	Open Options menu
	Set COM Port number
	Save the created map
	Load previously created map
	Go to Navigate screen
	Connect hardware with VE
	Go Back to map creation screen
	Exit VE

The Wall Cube is a single, opaque element that can be used to construct walls inside the VE and has the dimensions 1x1x1 unit3. It can be placed on top of or beside another object and can be deleted if need arises. For convenience, clicking on the floor creates a column that consists of 3 Wall Cubes instead of instantiating a single cube. In this way, time can be saved by reducing the number of clicks when building a map. However, clicking on a Wall Cube instead of the floor will result in the instantiation of a single Wall Cube at that point. This gives the

user more freedom in customizing the map as the user can vary the height of the wall at different locations or create tall structures with wide openings inside the walls etc.

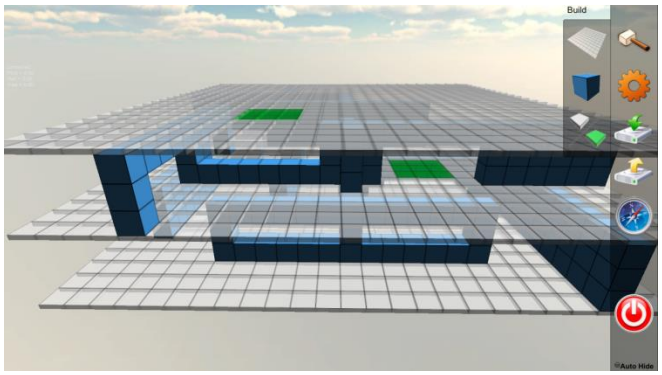


Figure 2. A 3D map created inside the VE

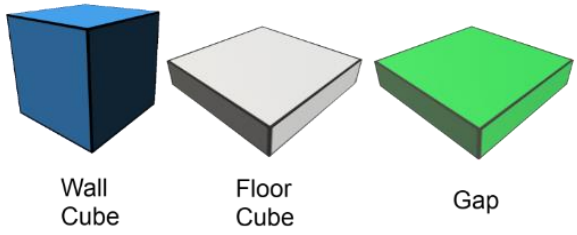


Figure 3. The basic building blocks of the VE

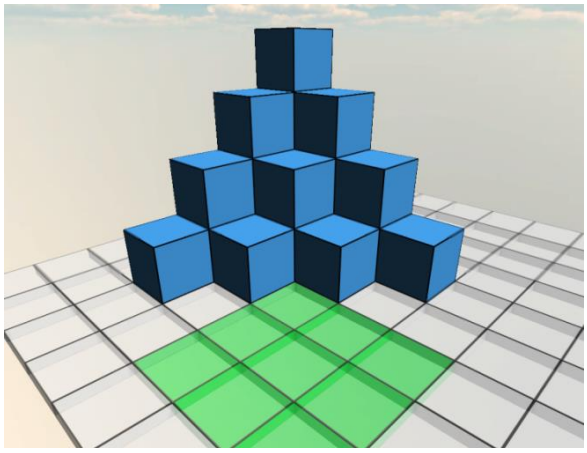


Figure 4. Interaction of the cubes with each other

The Floor Cube and Gap Cube are, basically, different sides of the same coin. The Floor Cube is a single semi-transparent element that can be used to construct floors and has the dimensions 1x0.2x1 unit3. A Floor can only be constructed as an entire unit instead of placing individual cubes side by side which means that an entire floor will be instantiated on a single mouse click.

As for the Gap Cube, as the name suggests, it is used to create a gap in the floor which serves as a gateway between two adjacent floors. It has the same dimension as the Floor Cube and is also semi-transparent. The only difference is that the Gap Cube has a different colour as compared to the Floor Cube and does not have a collider attached to it. This means that objects can pass through it as if the Gap Cube never existed. The reason for creating a gap cube instead of leaving the opening empty was to make it convenient for the user to identify an opening in the ceiling or on the floor and thus, make navigation through the map much easier. Every cube instantiated is also made as a child object of another object called “Cube Container”.

Save/Load System: There are many different ways in which information can be saved to and loaded from a file and all of them are highly dependent on the nature of the software being developed. Here, the aim was to create a file saving and loading system that is lightweight and can be modified in any text editor. Moreover, the file format should also be both human and machine readable, to make it convenient for other students working on the project to modify it and work with it.

For this reason, the XML (Extensible Markup Language) was chosen as a default Save/Load file format. The format defines a set of rules for encoding documents in a format that is both human and machine-readable. The Unity Engine has its own built-in XML serializer that can be used to serialize information to and from an XML file. The Save/Load functionality saves information related to the present state of the environment by encoding it in XML format before saving it to a text file. This can later be retrieved and decoded back into the original data. An example of the XML encoded information is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<BuildingInfo
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <cubeList>
    <CubeInfo>
      <prefabName>Floor</prefabName>
      <position>
        <x>0</x>
        <y>0</y>
        <z>0</z>
      </position>
    </CubeInfo>
    <CubeInfo>
      <prefabName>Wall</prefabName>
      <position>
        <x>1</x>
        <y>0</y>
        <z>0</z>
      </position>
    </CubeInfo>
```

Save/Load works by iterating through all of the objects inside the "Cube Container" and stores them inside a List. The name and position (x,y,z) in 3D space of each object stored in the List are, then, retrieved and the data converted into XML format which is eventually saved into a ".XML" text file. To load the cubes back again, the .XML file is decoded and the cubes are instantiated, based on the coordinates written in the text file along with their names.

5 Hardware Integration

The hardware that is integrated with the VE consists of an Inertial Measurement Unit (IMU) to sense the orientation of an object it is attached to with respect to the earth as an inertial frame of reference; a microcontroller to retrieve the orientation data from the IMU and process it; and a pair of XBee modules to wirelessly transmit orientation data from the microcontroller to the Laptop, where the data is parsed by the VE. Figure 5 is a picture of the hardware design, with the Arduino in this case is only used to supply power to the IMU and the XBee modules [17].

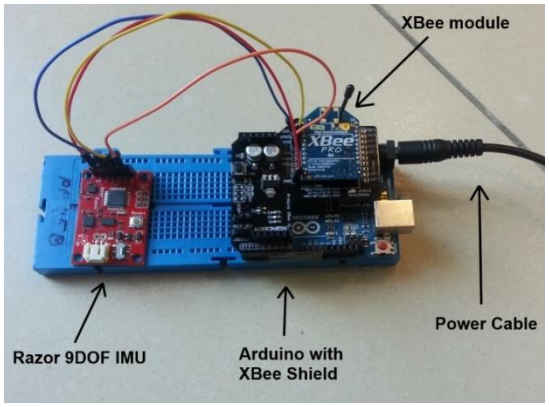


Figure 5. The final hardware design

6 Multiplatform Testing

The VE was exported as a standalone multiplatform application that is functional in Windows, Mac OS X and Linux environments. The integrated VE and hardware was tested on all operating systems, where hardware manoeuvres were reflected in the VE, as shown in Figure 6.

7 Conclusion

This paper presents the development of real-time UAV navigation inside a VE. The paper mainly focused on the VE system architecture and the different associated developed mechanisms that makes it a powerful tool for UAV movement tracking.

The GUI was minimalistic and was specifically designed to provide more flexibility for the user. The save/load functionality was integrated into the VE to save time, keep record and improve efficiency. The voxel approach to the VE ensured a light smooth run on low-end computers.

The VE has proven to be simple, yet very robust and can be deployed on multiple platforms, thus benefiting users and eliminating compatibility issues for various types of UAVs.

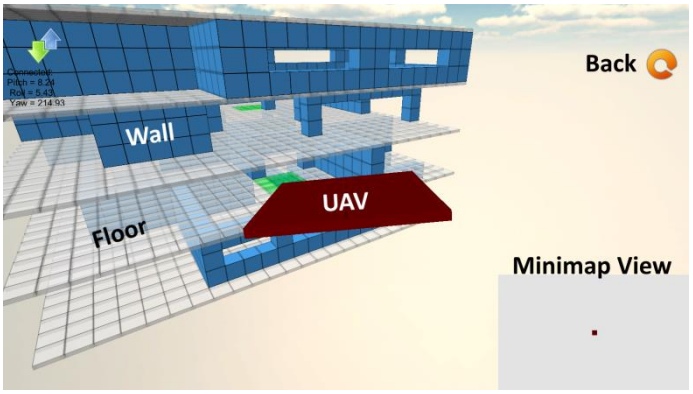


Figure 6. Navigation inside the VE

References

[1] F. Y. Annaz, "A Mobile Robot Solving a Virtual Maze Environment", *International Journal of Electronics, Computer and Communications Technologies, IJECCT,* **Vol**. 2, No. 2, pp. 1-7, Jan 2012.

[2] F. Y. Annaz, "Real Time Robot Navigation in Virtually Created Environments", *International Journal of Electronics, Computer and Communications Technologies, IJECCT,* **Vol**. 3, No. 4, pp. 7-13, Jul 2013.

[3] F. Y. Annaz, "Path-Whispering in a Virtual Environment", *International Review of Mechanical Engineering (IREME),* **Vol**. 5, No. 5, 2011.

[4] J. Faust, C. Simon and W. D. Smart, "A Video Game-Based Mobile Robot Simulation", *Proceedings of the IEEE/RSJ International Conference on Robots and Systems (IROS 2006)*, 2006.

[5] J. Craighead, J. Burke and R. Murphy, "Using the Unity Game Engine to Develop SARGE: A Case Study", *Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008)*, 2008.

[6] U. H. Hernandez-Belmonte, V. Ayala-Ramirez and R. E. Sanchez-Yanez, "A Mobile Robot Simulator Using a Game Development Engine", *Proceedings of the ROboticS SUmmer Meeting ROSSUM 2011*, Xalapa,

Veracruz, Mexico, 2011.

[7] R. Cory and R. Tedrake, "Experiments in fixed-wing UAV perching", *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2008.

[8] D. Mellinger, N. Michael and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors", *The International Journal of Robotics Research 2012,* **Vol**. 31, No. 5, pp. 664-674, 2012.

[9] "VICON Motion Capture System", VICON Motion Systems Ltd. http://www.vicon.com/Application/Engineering.

[10] N. Michael, D. Mellinger, Q. Lindsey and V. Kumar, "The GRASP Multiple Micro-UAV Testbed", *Robotics & Automation Magazine, IEEE ,* **Vol**. 17*,* No.*3,* pp. pp.56-65, September 2010.

[11] "Unity Game Engine", Unity Technologies, [Online]. https://unity3d.com/. [Accessed 07 May 2014].

[12] "Unreal Engine Technology", Epic Games, [Online]. Available: https://www.unrealengine.com/. [Accessed 07 May 2014].

[13] S. Balakirsky, C. Scrapper, S. Carpin and M. Lewis, "USARSim: providing a framework for multi-robot performance evaluation", *Proceedings of the 6th Performance Metrics for Intelligent Systems (PerMIS)*, 2006.

[14] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, "Usarsim: a robot simulator for research and education", *Proceedings of the IEEE 2007 International Conference on Robotics and Automation*, 2007.

[15] "Gazebo", Open Source Robotics Foundation (OSRF), [Online]. Available: http://gazebosim.org/. [Accessed 31 May 2014].

[16] "Webots", Cyberbotics, [Online]. http://www.cyberbotics.com/. [Accessed 31 May 2014].

[17] F. Y. Annaz and H. K. Wazir, "Hardware-Virtual Environment Integration", *BICET 2014*, Brunei Darussalam, 2014.