

It is a Small World: Building a Physics-Based Simulator For Magnetic Microrobot Manipulation

Tom GOALARD
ENS Rennes
Rennes, France

Leon Raphalen
Inria Rennes
Rennes, France

Marco Ferro
Inria Rennes
Rennes, France

Claudio Pacchierotti
Inria Rennes
Rennes, France

I. INTRODUCTION

This project is a part of the European project REGO. The goal of this project is to create small robots to do micromanipulation tasks. The goal of the RAINBOW team is to create a simulation of a small environment where the robots can be manipulated to test human-robot interaction.



Fig. 1: Logo of the REGO project

The project is supervised by Claudio Pacchierotti in collaboration with Leon Raphalen and Marco Ferro.

A. Motivation

There already exists an electronic system that is able to actuate a pair of micro-robots (Fig. 2). This system is ment to experiment with shared control. To facilitate the expérimentations, a simulation has been developed.

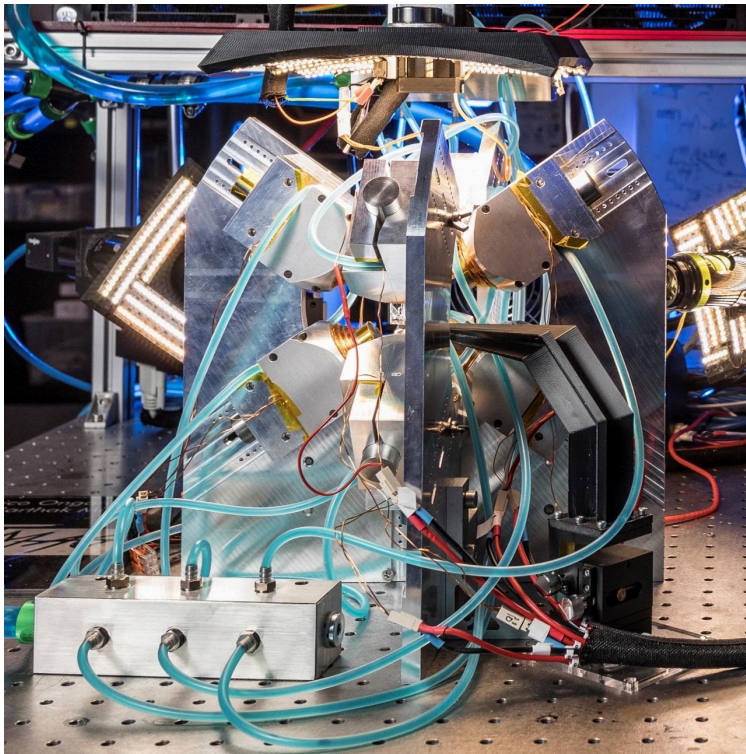


Fig. 2: Close-up view of the BatMag magnetic system [1]

The current simulation uses Coppeliassim to simulate the environment. This solution comes with a lot of limitations. One of the main ones is the precision of the physics engine: Due to the way Coppeliassim 2.7 does the simulation, the current solution is “cheating” to create a somewhat realistic simulation. For instance, the cubes that are used for manipulation are not grabbed by the robots. Instead, when the simulation thinks that the robot has grabbed the cube, it just moves the cube to the position of the robot. This leads to sometimes not-so-realistic behavior.

Another goal of this internship was to bring Virtual Reality to the simulation. This is not possible with Coppeliassim as it is not maint for this kind of purpose. The VR simulation is needed to experiment with new ways of interacting with the robots. The real world system uses two orthographic cameras to track the robots. Based on that we can reconstruct a 3D representation of the environment to better interact with the system, and VR seems to be a good fit for this purpose.

Thus, the goal was to create a new simulation. On one hand, this simulation had to make use of a physics engine capable enough to simulate the environment more realistically. On the other hand, it had to be able to run in Virtual Reality to experiment with new ways of interacting with the robots.

B. Unreal Engine

The choice for the new simulation engine settled on Unreal Engine. The reasons for this choice are multiple. To begin with, it makes use of C++ which is a language we are familiar with. Another good point is the nodes system that allows rapid prototyping of new ideas; this, however, is also a downside as it doesn't tinker well with git. The last good point is the fact that it doesn't require a special license to be used by anyone on the team.

Originally, the Unreal Engine was also chosen for the better physics engine. However, it turns out that Chaos, the physics engine used by Unreal Engine, is not deterministic. This means that the simulation will not be the same every time it is run. This is a problem for us as we need to be able to reproduce the simulation based on a recording of user inputs.

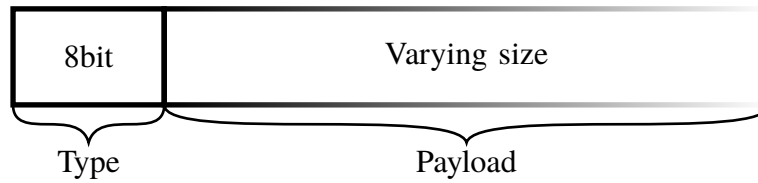
To solve this issue, we used the Bullet physics engine. This engine is deterministic and super precise so it suits perfectly our needs. Furthermore, there is an open-sourced plugin available on Git Hub that allows the use of Bullet with Unreal Engine. This plugin is called BulletPhysicsPlugin [2] and is made by Yadhu-S. A modified version is used in the project but all the heavy lifting is done by the original plugin. The modifications are only to make some Bullet-specific functions available in Unreal Engine and to change the gravity to be more precise.

II. IMPLEMENTATION

A. Communication Protocol Between The Controller and Unreal Engine

The main problem with starting a new simulation was the communication between the simulation and the controller. There is no built-in way to communicate between an Unreal Instance and a C++ program. The built-in TCP/IP communication is meant for games and can't be exploited for our needs. The solution chosen was to use the socket library provided by the system. Contrary to other solutions like shared memory or named pipes, sockets offer an easy interface for synchronous communication. Moreover, on a local machine, the implementation uses shared memory so the communication doesn't suffer from network latency.

The communication in itself is a common client/server architecture: The simulation acts as a server that accepts only one client at a time and the controller acts as a client. The communication between the two is then done with a simple protocol based on binary messages (Fig. 3).



This is sufficient to communicate between the two programs. It currently allows to send integers, floating points, and strings or arrays of these types. In the end, this part of the implementation works the same as Coppeliasim. This way, a common interface has been created to allow the use of the same controller for both simulations. The switching between Unreal and Coppeliasim is done by changing a Cmake variable to switch between the Unreal interface and the Coppeliasim interface without changing the code. This separation allows to use any simulation environment that follows the same communication protocol without changing the controller code.

B. Physics Simulation

The bullet physics engine allows us to simulate a way more precise physics than the version Coppeliasim uses. Hence, the cubes use a more precise collision detection.

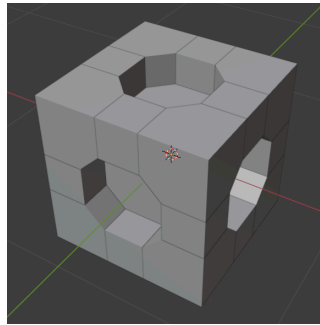


Fig. 4: The collision mesh used for the cubes

This allows us to do things that were not possible before. For instance, the cubes can be grabbed two at a time if they are close enough.

However, it is still an approximation as the center holes on each side are not round; This is due to the fact that the collision mesh has to be a convex shape. This means that we had to cut the collision shape into smaller convex pieces. This makes nevertheless the simulation accurate enough for our needs. It is moreover way better than the previous implementation in Coppeliasim. There is no longer any need for hacks to make the simulation work.

C. Global Project Architecture

The project in itself follows a quite simple architecture where all the components used are sorted in folders with appropriate names (Blueprints, Textures,...). However, there are some important points to highlight:

- The files used for each environment are part of the project and not independent files. Contrary to Coppeliasim, the environment is packaged with the simulation. This can be changed but then, we would have to implement a way to load the environment from a file (or use an already existing one if available).
- The server code is located under the `Source` folder. This folder is not accessible from the Unreal Engine editor. To edit it, we have to use Visual Studio.

D. Visualization

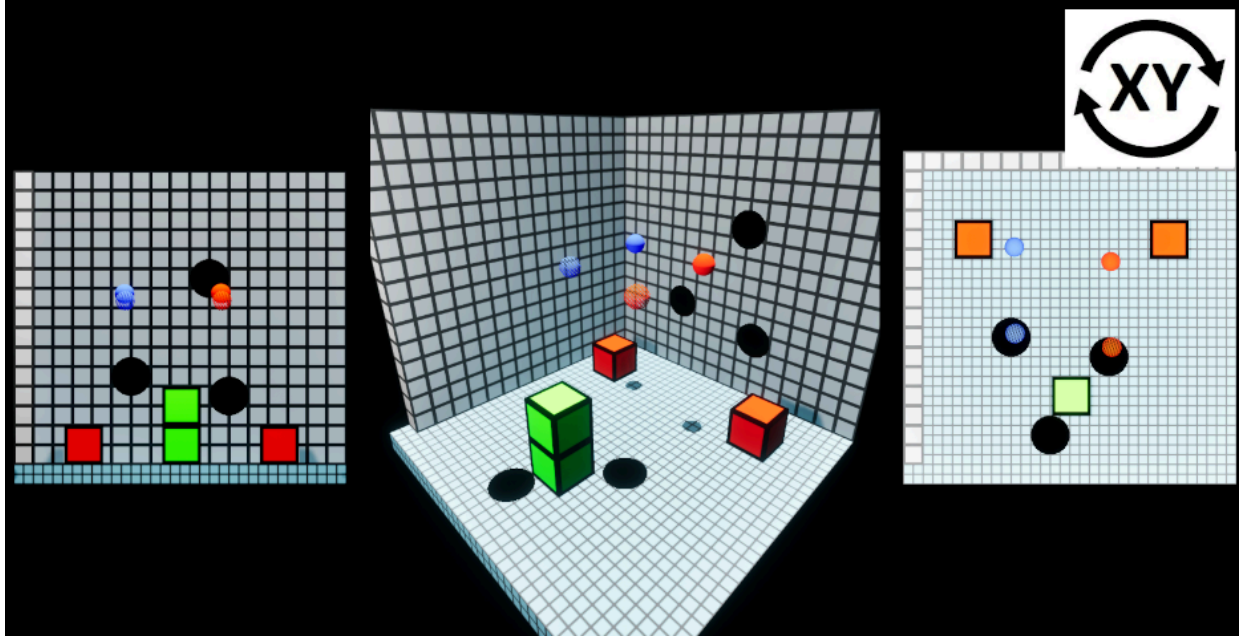


Fig. 5: Render of the tutorial environment

The visuals were made to look as similar to the Coppeliasim simulation as possible. More details are given in the Experiment description (Section IV). But on Fig. 5, we can see three parts:

- On the left and right panels, we see the two orthogonal views of the environment. Those are what the real-life system sees in the real environment. The view mimic the orthogonal cameras of the real systems. Those views are meant to be used as a reference for the user.

This way, the user can understand where are the occlusions and why the robots can't go behind them. Indeed, the system can only track the robots if it sees them.

- On the center pane, we see the 3D render of the simulation. This is the view that the user will use when not in VR mode. In this view, the camera can be fixed or can move in the environment to follow the robot's movements.

E. Virtual Reality

The main goal of this project was to be able to use VR to control the camera and see the environment. This part is pretty much straightforward as Unreal Engine has built-in support for VR.

One of the main problems was to be able to enable and disable VR at will. This was however addressed by running some commands using a special node. This way, it works as expected and we can switch between VR and non-VR modes during the experiment without restarting the simulation.

III. USER EXPERIMENT

Now that we have a working simulation, we can start experimenting with it. The goal here is to test if VR is a better way or not to interact with the robots. Paired with that there is also a test to see the influence of hints on occlusion avoidance. Those hints are either visual or haptic. The visual hints are arrows displayed at the robot's position. The haptic is a force feedback given by the manipulation device.

A. Set-up

TABLE I: THE DIFFERENT SETUPS USED FOR THE EXPERIMENT

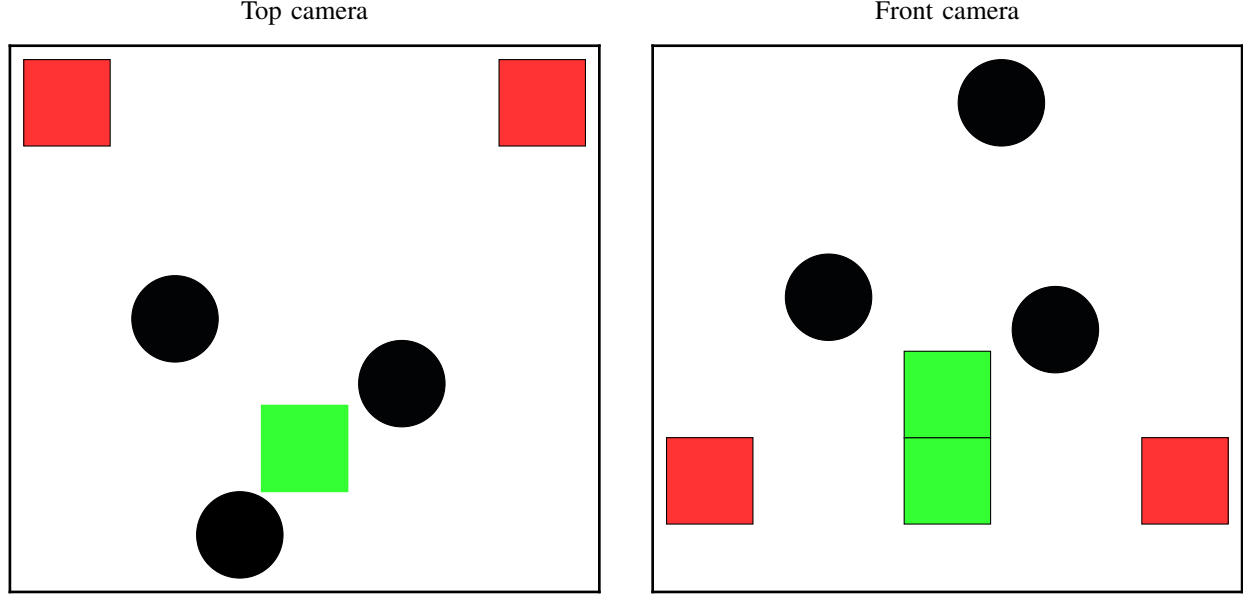
VR	Off	Off	Off	Off	Off	Off	On	On	On
Hints	None	Visual	Haptic	None	Visual	Haptic	None	Visual	Haptic
Camera	Static	Static	Static	Dynamic	Dynamic	Dynamic	N/A	N/A	N/A

The user experiment consists of a tutorial phase followed by 9 different setups Table I. Each setup is a combination of the three parameters: VR, Hints, and Camera.

The static camera is a camera that is fixed in the environment. The dynamic camera is a camera that follows the robot's movements by orbiting around the scene. When the VR is enabled, the camera is not controlled by the simulation.

During the experiment, each task is given randomly to the user. Each task consists of moving the cubes to build a tower. The thing is that there are occlusions in the environment. The user has to avoid those occlusion spots to build the tower.

TABLE II: OCCLUSION SPOTS (BLACK CIRCLES) POSSIBLE PLACEMENT FOR EACH CAMERA



For each task, four occlusion spots are in the environment, two for each real-world camera. The spots are randomly chosen among the three possible placements (Table II). For the tutorial phase, however, the occlusion spots are all enabled for each step. This is however too hard for someone who never used the system before whence the choice to only use two occlusion spots for the experiment phase.

IV. EXPERIMENT DESCRIPTION

The following is a description of the experiment that can be used as an introduction for the user. It is a bit long but part of it can be cut off. This is mainly meant to be read to the user.

A. Introduction

First, thank you for participating in this experiment. This experiment will last for about an hour. It is divided into two parts:

- A tutorial phase where you will learn how to use the system
- An experiment phase where you will have to do some tasks

The goal of this experiment is to test the influence of the camera and the hints on occlusion avoidance. The task that you will have to complete is a simple tower-building task. Each task lasts at most 5 minutes.

You can ask any question at any time. If you feel tired, uncomfortable or simply want to pause, please tell us. We can take a short break at any time.

B. What you will see

You will see displayed on the screen a simulation of a real environment. This environment is a 2 by 2 by 2cm cube. It contains 4 small cubes that you can manipulate using 2 small spheres.

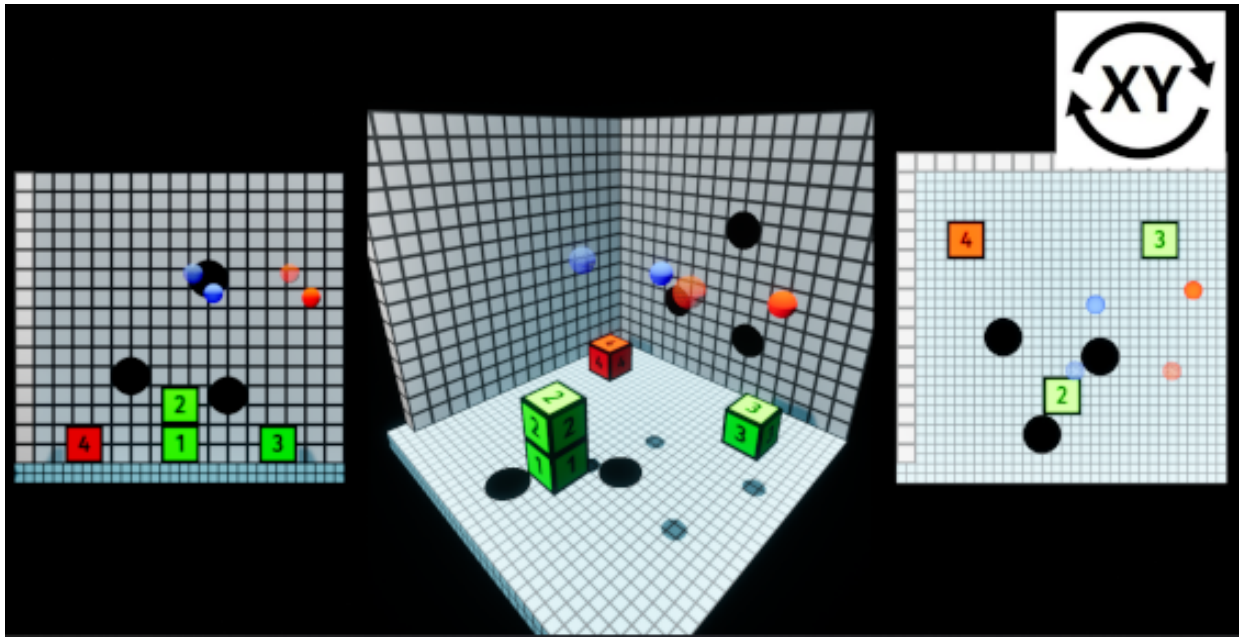


Fig. 6: Initial setup of the experiment

Once the experiment starts, a number will appear on each cube. Those represent the order in which the tower has to be built. The cube number 1 has to be placed on the cube number 2, and so on.

In the environment, cubes 1 and 2 are already placed. You have to place Cube 3 on top of Cube 2 and Cube 4 on top of Cube 3.

In the tutorial phase, the screen is divided into three parts. The left and right panels are the views of the real-world cameras: The real system being really limited in terms of vision, the cameras are only two orthographic cameras. The center view is a 3D reconstruction of the environment from the two cameras. Once the tutorial phase is over, the left and right panels will be removed. The center view will be used in itself to show the simulation.

You will also see black circles on the floor and back walls. Those are occlusion spots: Due to system limitations, the robots can't be moved behind the cubes or in the occlusion spots. Your goal is to avoid those occlusion spots while building the tower.

C. Occlusion avoidance

You will quickly notice that the system won't allow you to move the robots anywhere they are occluded for one of the cameras.

To help you avoid those occlusions, you will be given hints. Those hints are either visual or haptic. The visual hints are arrows displayed at the robot position Fig. 7. The haptic is a force feedback given by the manipulation device.

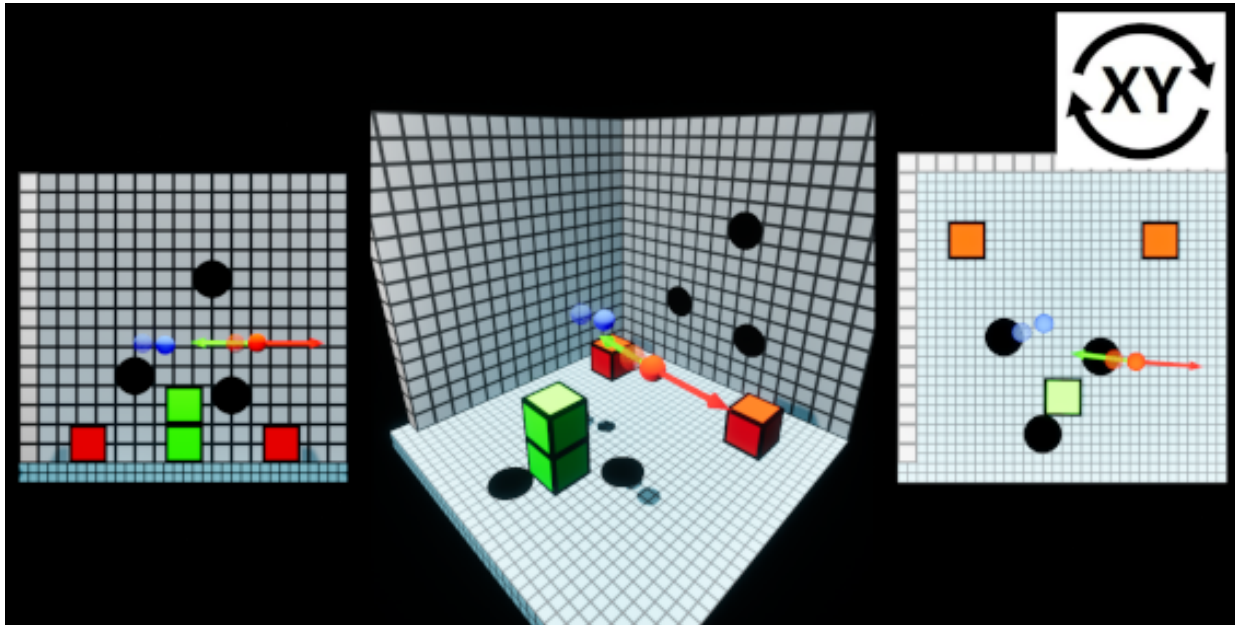


Fig. 7: Visual hints. The green arrow indicates the force you apply to the robots. The red arrow indicates the force the system is applying to the robots for them to avoid the occlusion.

Finally, on the top right corner of the screen, you will see a small panel. This panel contains the current rotation plane of the robots. If it displays “XY” then the robots can rotate horizontally. If it displays “XZ” then the robots can rotate vertically.

D. How to control the robots

The control of the robots uses two components: A robotic arm and a little box with three buttons. The robotic arm is used to control the position of the robots. The little box is used to control the rotation of the robots and the experiment.



Fig. 8: The robotic arm used to control the robots

When the experiment starts, a text prompt will ask you to press the “force feedback” button. This is the green button at the base of the robotic arm. Press it once. You will hear a little beep. This means that the force feedback is enabled. You can now use the robotic arm to control the robots.

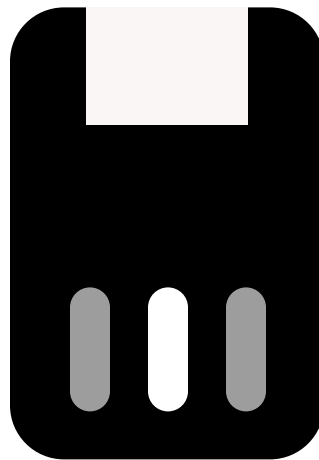


Fig. 9: The little box used to control the experiment. The left button is used to launch the experiment and to skip phases (more on that later). The middle button is unused. The right button is used to change the rotation plane of the robots.

E. Ahh I'm stuck!

Sometimes, a cube falls or is pushed in a way that it is in an occlusion space. In this case, the system will not allow you to move the robots to grab the cube again. It happens and it is not a problem. You can, in this case, press **once** the left button of the little box. This will skip the current cube and will allow you to continue the experiment.

V. CONCLUSION

This summary concludes the internship. The goal to create a new simulation for the REGO has been reached. Although there wasn't enough time to do the user experiment before the end of the internship, the simulation is ready to be used. Leon Raphalen and Marco Ferro will continue the work to conduct the user experiment. This should lead to a publication hence showing the work done during this internship is useful.

REFERENCES

- [1] F. Ongaro, S. Pane, S. Scheggi, and S. Misra, "Design of an electromagnetic setup for independent three-dimensional control of pairs of identical and nonidentical microrobots," *IEEE Trans. robotics*, vol. 35, no. 1, pp. 174–183, 2018.
- [2] Yadhu-S, "Yadhu-S/BulletPhysicsPlugin." Accessed: May 05, 2025. [Online]. Available: <https://github.com/Yadhu-S/BulletPhysicsPlugin>