DTE-3605-1

Asal Asgari

UiT the Arctic University of Norway, Narvik, Norway.

Abstract

The Sphero BOLT is an advanced, programmable robot that includes various sensors, such as a compass, gyroscope, accelerometer, and light sensor on board. We created a virtual Sphero in this task that maps rooms based on inputs from a file containing movement and collision information. Starting with a simple plane, we add geometry depending on collision input, Through a custom C++ module.

1 Introduction

The Sphero BOLT is an advanced, programmable robot that includes various sensors. We began by testing the sensors and how they function by using the Sphero Edu app and coding in javascript. Then we planned on how to read and use the data from them. After creating a custom C++ module for this purpose we started working on our simulation with Godot. Developing a way to detect collisions based on sensor data, and eventually spawning objects in real time, was the next step in making the sphere move closer to reality. Moreover, we discuss ways in which this algorithm can be developed further in the future.

2 Methods

2.1 Step 1: Sphero edu app

Using JavaScript, we wrote some simple code in the Sphero Edu app to move the sphero around the room, change the LEDs' color when it collided with an obstacle, and change the direction of the sphero after a collision. Firstly, the purpose of this section was to determine the changes that take place in sensor data as a result of collisions, and second, to create a module based on the comparative data. Detecting collisions was accomplished by testing an

argument that whenever a collision happens, the sphero's velocity decreases as it stops for a moment after every collision, and the sphero's acceleration also increases. A series of tests were run on the sphero to see whether there is a threshold for these changes, or whether these two variables correlate when collisions occur.

Newtonian law of motion proves this theory, meaning acceleration is dependent on the amount of velocity change over time $a = \Delta v/\Delta t$. Whenever a collision happens the velocity of the moving object decreases drastically. This causes an increase in Δv and since Δv and a have a direct relationship as can be seen from the formula, hence there would be an increase in acceleration.

Using the Sphero Edu app, we generated all of these sensor data sets and began to work on a costume module through C++ that would enable us to read and process data for our Godot module.

2.2 Step 2: Creating C++ Module

For the C++ module, I began by creating a structure that holds all the sensor data for a single time step. The structure contains information such as time, velocity in each direction, acceleration, gyroscope data, sphero orientation, location, and distance traveled. A function called "readFiles" reads all the sensor data files by calling a function called "read" for every file, which reads the CSV file line by line and creates a 2-dimensional vector called content that contains all the data from that particular file. After that, in "readFiles", I iterate through these vectors simultaneously since they all have the same size and time details, creating a single structure for every time step, which is added to a vector of structures called sphero. A series of getters were then created in order to allow Godot to retrieve specific sensor data from a time frame by getting an index and returning the specific data..

2.3 Step 3: Creating shapes in Godot and attaching scripts

For the Godot module, I first created a static body called ground with a collision shape and mesh instance in the form of a cube, then created the sphero as a kinematic body with a collision shape and mesh instance in the form of a sphere. After that, I attached a script to the sphero(named as player in GDScript) to begin reading from the C++ module and making the simulation. For enabling sphero movement, I used the velocity data from the sensors, and so I created an object within my module and called the appropriate functions to get the sphero's speed for each time step. It should be noted that the sphero also has gravity applied to it toward the ground by setting a certain speed in its Y direction.

It had, however, caught my attention that the sphero's movement in the simulation is robotic and not a smooth one, since it jumps between time windows that the sensor data has provided, which are approximately 10 times larger than Godot's frame rate in the "_physics_process".

2.4 Step 4: Calculating speed for every frame in C++ module

Given the result from the last step, I decided to calculate the speed for every frame based on the frame rate in the custom module. This was so that the change in speed would be gradual and that it would help smooth out the sphero movement in the simulation.

I added two functions in the custom module for calculating speed. These functions get the index of the time step we are in and output the amount of change needed to apply to the velocity to make the movement smoother. To begin, it divides the difference between these two times by the frame rate. This is done to check how many times the "_physics_process" in the Godot is called between the given two time steps. After that, we divide the difference in velocity between these two steps by the amount of time we expect the function to run between time steps. The GD Script uses the value from the custom module to adjust the speed until we reach a valid time step.

At the end of this step I had a clear simulation of the sphero bolt moving on the ground based on the values from the sensor. The next step would be to detect the collisions.

2.5 Step 5: Collision Detection

The next step was to detect collisions in the custom module and eventually see it in Godot. For detecting collisions based on a series of test cases we have noticed that we can use the change in acceleration and velocity for this purpose. So in the custom module, we have a function that compares the acceleration of the point we are at to the time frame before. This function checks if there is an increase in the acceleration and if the acceleration has a value greater than one. Observing from the data sheets that whenever a collision occurs the acceleration exceeds one, we chose this threshold. As well, we check if a decrease in velocity occurs. If all these conditions are true we find our collision and we return True to Godot.

2.6 Step 6: Real time obstacles

In the Godot script, we use the output from the custom module regarding the collision. In every time step, we check if there was a collision in the data. Whenever a collision happens we generate an obstacle using the location of the sphere in the simulation. The collision detection phase of GD Script has been implemented in the main scene script. The obstacle that we generate is simply a kinematic body with a mesh instance of a cube.

After generating obstacles I noticed that the position of the obstacles was not correct relative to the position of the sphero in terms of orientation. As a result of my search for patterns in the sensor data, I realized that there is a pattern in the roll value of the sphero and the orientation of the obstacle with which it is colliding, and used the patterns as a set of conditional statements in

the obstacle (also called the wall in Godot script) to move the obstacles in the correct orientation, hence the degree of the roll of the sphero in that time step.

2.7 Step 7: Adding Textures

Using Godot's Spatial Material menu, I created and applied textures to all three objects after I finished the simulation logic.

3 Result

Through the above-mentioned method, we have created a model of a sphero bolt robot that simulates the movement of the sphero in a room using data obtained from sphero bolt sensors. Furthermore, it detects collisions between the sphero and objects in the room and spawns obstacles representing the objects that were met in the room inside Godot.

3.1 Features

1. Reading data from a test subject
2. Simulating Sphero bolt with a smooth movement through frames
3. Following rules of gravity in the simulation
4. Detecting collision
5. Spawning realtime objects
6. Objects with textures and lighting
7. Ability to map any room

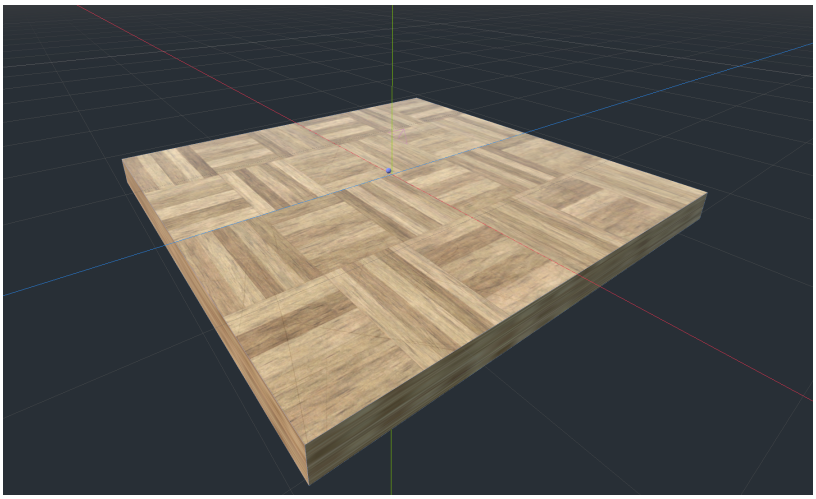

Fig. 1 Ground/Room Area

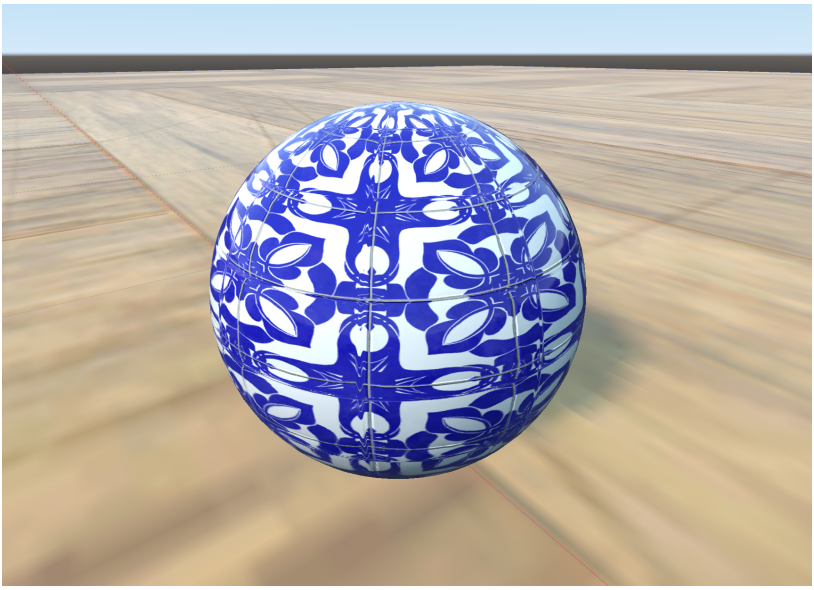

Fig. 2 Sphero Texture

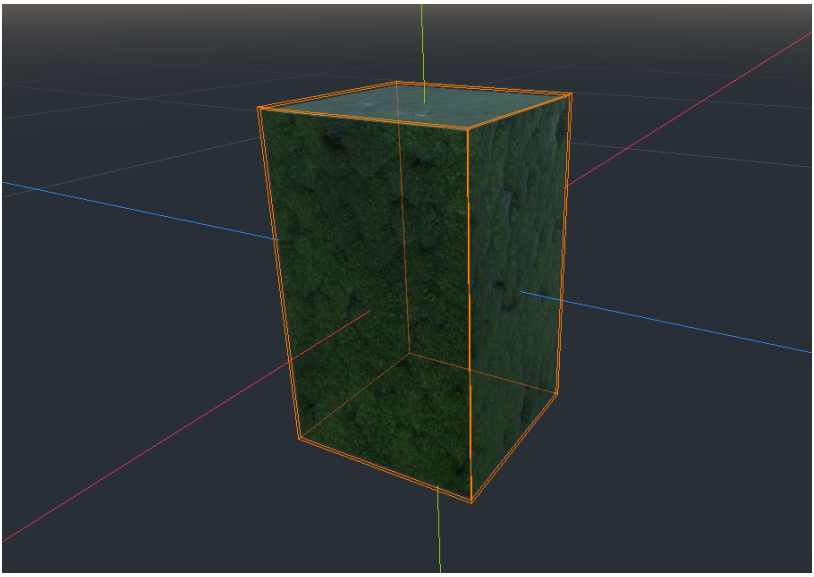

Fig. 3 a moss wall representing obstacles in the room

4 Discussion

I noticed that there were instances when there was a collision happening. However, the sensors were not picking up on it or in some cases my method was not accurate enough to detect that specific collision. I have also tested

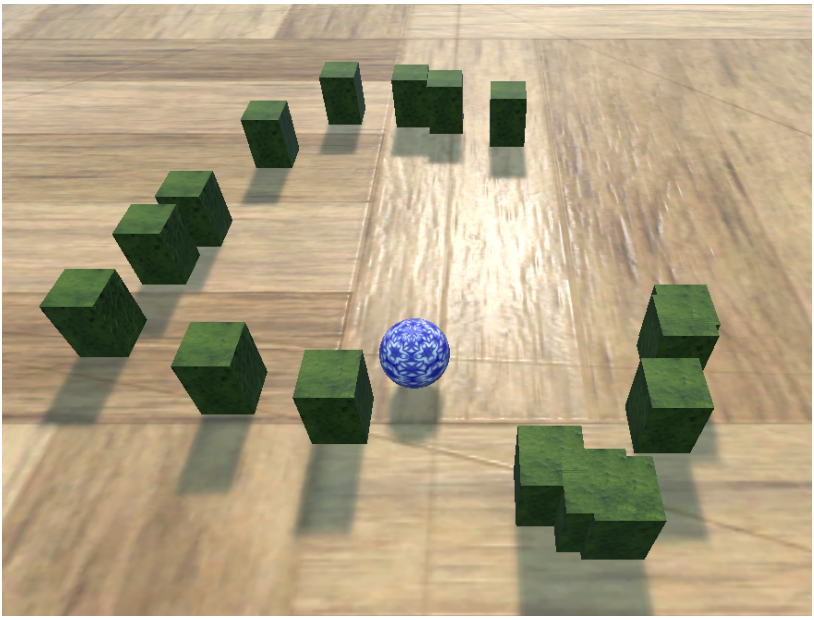

Fig. 4 Room Mapping

another idea to use for collision detection, which would be the roll value of the orientation of the sphero. In this method we would check as if there was a certain amount of change in the roll value between two timesteps. This means that the sphero has collided with an object and now it's changing its heading direction based on the JavaScript code that we have written for the sphero bolt movement. However, this method was not consistent across all scenarios. For example, in a case where the sphero bolt was given a higher speed, this pattern was not visible in the data set.

In the case of further development, improving the accuracy of collision detection for the sphero bolt would be a worthwhile subject to work on.

References

[1] Sphero Edu JavaScript Wiki. https://sites.google.com/sphero.com/sphero-edu-javascript-wiki/getting-started?authuser=0

[2] Godot Docs. https://docs.godotengine.org/en/stable/