ASSIGNMENT NO: 2

NAME: Akshaj Chainani PRN: 22070127008 BATCH: RA1

AIM:

Create a world with objects such as two chairs, one stool and one ball that is supported by four pillars, without using Proto (Attached).

APPARATUS: Webots R2023b

THEORY:

Definitions

- Compound Solids: Compound solids are 3D objects that consist of multiple simple shapes (e.g., boxes, spheres) combined into a single structure. They are used to create detailed and complex models like robots efficiently.

- Physics Attribute: The "physics" attribute describes the physical properties of a 3D object, such as mass, friction, and density, which affect how it behaves in simulations, especially regarding movement and collisions.

- Solid Node: The term "solid nodes" refers to the Solid node and all its derived nodes together.

- Robot Node: In a robot model, the structure is organized as a tree of Solid nodes. The root of this tree must be a Robot node.

- Joint Node: Joint nodes connect Solid nodes. A Device node must be a direct child of a Robot node, a Solid node, or a Joint node. A Joint node provides one or two degrees of freedom (DOF) between its parent and child, which are both Solid nodes.

- Hinge Joint: The initial position of a wheel is set by the translation and rotation fields of the Solid node. The rotation origin (anchor) and axis are defined by the optional HingeJointParameters child of the HingeJoint node.

Concepts

- Compound Solids: Compound solids can be organized with a hierarchical structure where each part (or sub-component) can be positioned, rotated, and scaled independently. Each transformation is relative to the parent object, allowing for flexible designs.

- Application: Compound solids are suitable for building robots or other complex models requiring multiple components to function as a single unit, such as a robot with multiple limbs or a vehicle with several parts.

Physics Attributes

- Mass: Determines the weight of an object and impacts how it responds to forces, gravity, and collisions. Accurate mass is crucial for realistic simulations.

- Friction: Determines the resistance an object experiences when sliding across surfaces. Higher friction means more resistance, influencing how the object moves and stops.

- Bounciness (Restitution): Specifies how much an object bounces back after a collision. A higher value results in more bounce, while a lower value reduces bounce.

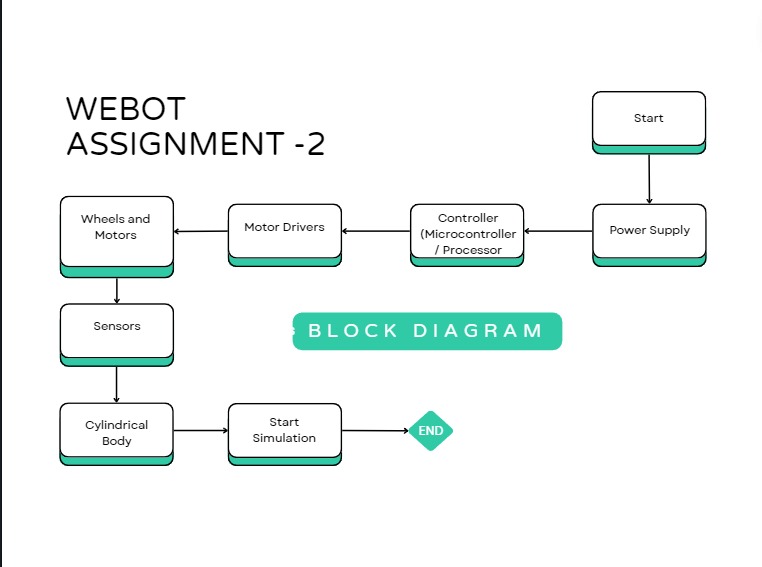
What Is a World?

In Webots, a "world" is a 3D environment that contains the properties of robots and their surroundings. It includes the position, orientation, geometry, appearance (e.g., color, brightness), and physical attributes of every object. Worlds are structured hierarchically so that objects can contain other objects. A world file specifies the controller required for each robot but doesn’t include the actual controller code. World files use the ".wbt" format and are stored in the "worlds" subdirectory of each Webots project.

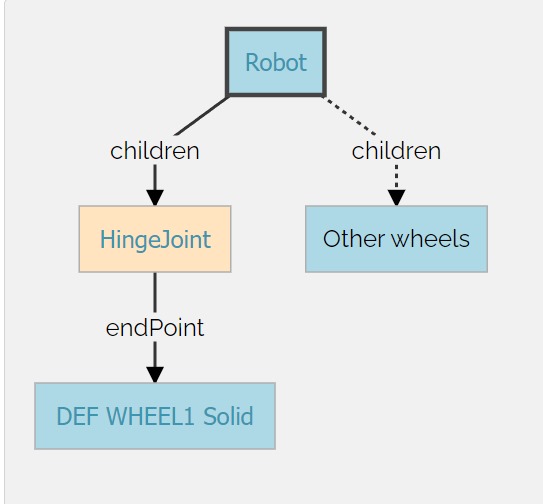
What Is a Controller?

A controller is a program that runs a robot described in a world file. Controllers can be written in languages like C, C++, Java, Python, or MATLAB. When the simulation starts, Webots launches each controller as a separate process and associates it with the corresponding robot. Controllers in C and C++ are compiled into platform-specific executables (e.g., ".exe" in Windows), while Python and MATLAB controllers are interpreted by their runtime systems. Controller files are stored in a controller directory within the "controllers" subdirectory of each Webots project.

Block Diagram & Flow Chart:



Flowchart of a 2-wheel Robot



High level Representation of a 2-wheel robot

Program Code (Following a square path):

include <webots/robot.h>

include <webots/motor.h>

define TIME\_STEP 64

define MAX\_SPEED 6.28

int main(int argc, char argv) {

// Initialize Webots API

wb\_robot\_init();

// Get motor devices

WbDeviceTag left\_motor = wb\_robot\_get\_device("motor\_1");

WbDeviceTag right\_motor = wb\_robot\_get\_device("motor\_2");

// Set motors to velocity control mode

wb\_motor\_set\_position(left\_motor, INFINITY);

wb\_motor\_set\_velocity(left\_motor, 0.0);

wb\_motor\_set\_position(right\_motor, INFINITY);

wb\_motor\_set\_velocity(right\_motor, 0.0);

int num\_side = 4;

double length\_side = 0.25;

double wheel\_radius = 0.025;

double linear\_velocity = wheel\_radius MAX\_SPEED;

double duration\_side = length\_side / linear\_velocity;

double start\_time = wb\_robot\_get\_time();

double angle\_of\_rotation = 6.28 / num\_side;

double distance\_between\_wheels = 0.090;

double rate\_of\_rotation = (2 linear\_velocity) / distance\_between\_wheels;

double duration\_turn = angle\_of\_rotation / rate\_of\_rotation;

double rot\_start\_time = start\_time + duration\_side;

double rot\_end\_time = rot\_start\_time + duration\_turn;

// Main loop

while (wb\_robot\_step(TIME\_STEP) != -1) {

double current\_time = wb\_robot\_get\_time();

double left\_speed = MAX\_SPEED;

double right\_speed = MAX\_SPEED;

// Check if the robot needs to turn

if (current\_time > rot\_start\_time && current\_time < rot\_end\_time) {

left\_speed = -MAX\_SPEED;

right\_speed = MAX\_SPEED;

}

// Update the rotation timings

else if (current\_time > rot\_end\_time) {

rot\_start\_time = current\_time + duration\_side;

rot\_end\_time = rot\_start\_time + duration\_turn;

}

// Set motor velocities

wb\_motor\_set\_velocity(left\_motor, left\_speed);

wb\_motor\_set\_velocity(right\_motor, right\_speed);

}

// Cleanup

wb\_robot\_cleanup();

  return 0;

SNAPSHOTS:

EXPANDED SCENE TREE:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

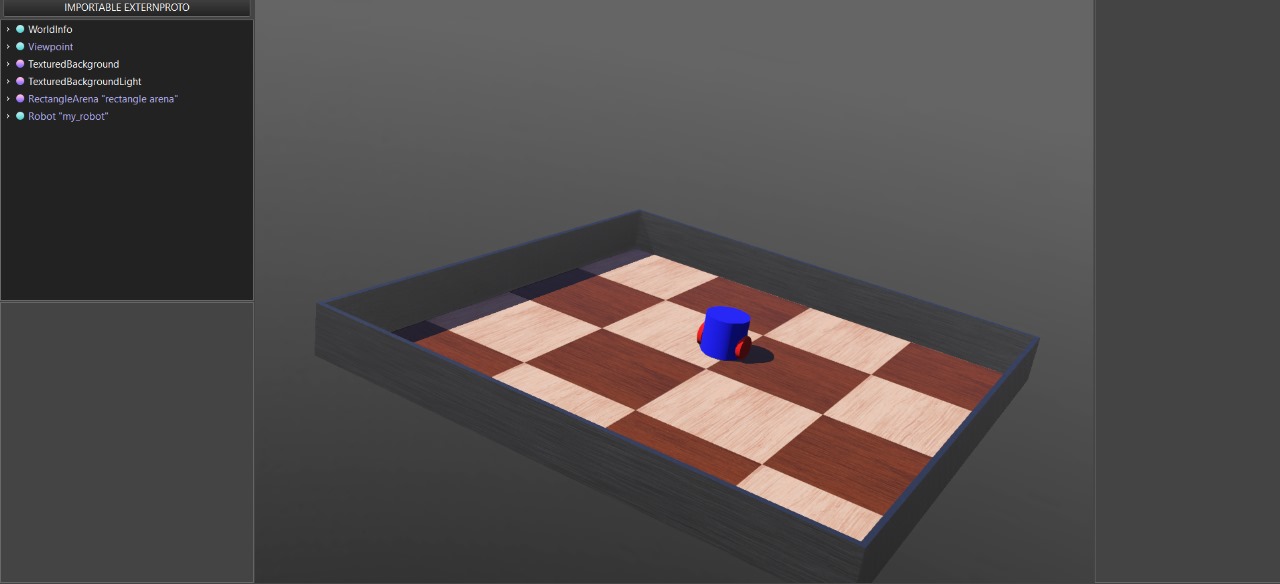
Description automatically generated

**CODE IN C**

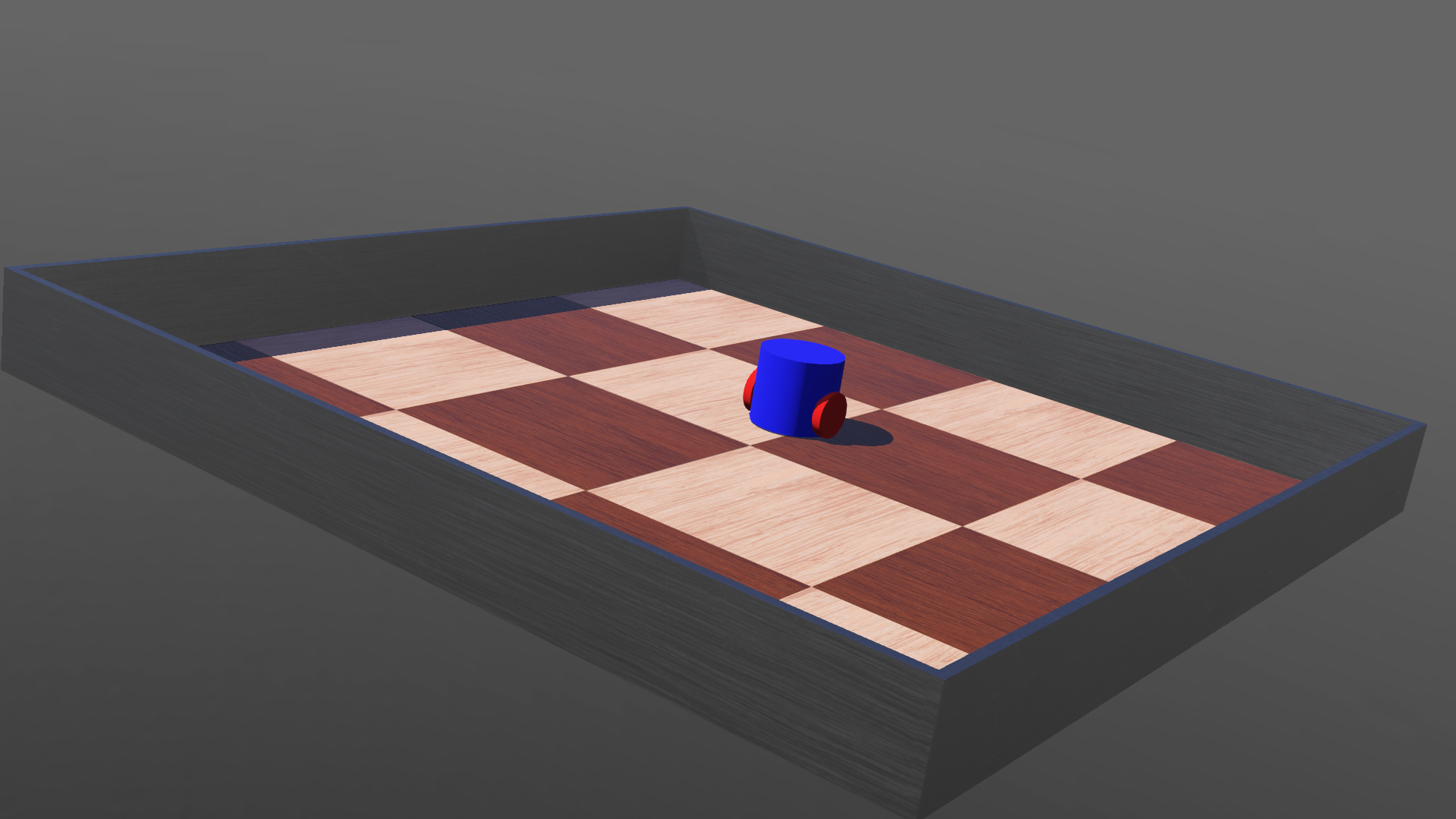
A screen shot of a computer

Description automatically generated

FULL WINDOW WITH DIRECTORY:



OUTPUT:



WBT TEXT FORMAT:

VRML\_SIM R2023b utf8

EXTERNPROTO "https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/backgrounds/protos/TexturedBackground.proto"

EXTERNPROTO "https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/backgrounds/protos/TexturedBackgroundLight.proto"

EXTERNPROTO "https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/floors/protos/RectangleArena.proto"

WorldInfo {

}

Viewpoint {

orientation -0.08592590714502174 -0.31363361336332407 0.9456482935267057 5.719127272383435

position -1.7334857217842412 1.103309355546052 0.584479087007047

}

TexturedBackground {

}

TexturedBackgroundLight {

}

RectangleArena {

translation -0.00442659 0.00551414 0

wallHeight 0.03

}

Robot {

translation -0.012008888137962881 0.048958187629633665 -0.0002045183218206681

rotation 0.03949686463254615 0.014480608754160278 -0.999114763004885 -0.706214610476791

children [

DEF Body Pose {

translation 0 0 0.0415

children [

Shape {

appearance PBRAppearance {

baseColor 0 0 1

roughness 1

metalness 0

}

geometry Cylinder {

height 0.08

radius 0.045

}

}

]

}

HingeJoint {

jointParameters HingeJointParameters {

position 3788.108101743404

anchor 0.045 0 0.025

}

device [

RotationalMotor {

name "motor\_1"

}

]

endPoint Solid {

translation 0.045 0 0.025

rotation 0.3054787890505593 -0.9019500666472192 0.305235624911773 4.610167676512081

children [

DEF wheel Shape {

appearance PBRAppearance {

baseColor 1 0 0

roughness 1

metalness 0

}

geometry Cylinder {

height 0.01

radius 0.025

}

}

]

boundingObject USE wheel

physics DEF wheel\_physics Physics {

}

linearVelocity 0.100524999825089 -0.11946089740009769 -1.0112598691463742e-05

angularVelocity 4.775652757433535 4.080113661484263 -0.002920179472802596

}

}

HingeJoint {

jointParameters HingeJointParameters {

position 6308.0180235522275

anchor -0.045 0 0.025

}

device [

RotationalMotor {

name "motor\_2"

}

]

endPoint Solid {

translation -0.045 0 0.025

rotation -0.1479240254153559 0.9778915008085394 -0.14780627642742197 1.592354884725012

children [

USE wheel

]

name "solid(1)"

boundingObject USE wheel

physics USE wheel\_physics

linearVelocity 0.10166698612622033 -0.11967596179130115 -1.4327532919974888e-05

angularVelocity 4.787655629745545 4.066019722583425 -0.0031786184379903916

}

}

Solid {

translation 0.015 -0.045 0.07

children [

DEF eye Shape {

appearance PBRAppearance {

baseColor 1 0 0

roughness 1

metalness 0

}

geometry Box {

size 0.005 0.005 0.005

}

}

]

name "solid(2)"

}

Solid {

translation -0.015 -0.045 0.07

children [

USE eye

]

name "solid(3)"

}

]

boundingObject USE Body

physics Physics {

}

controller "drive\_my\_robot"

linearVelocity 0.10110619608665458 -0.11959094898754581 -1.3000784793183e-05

angularVelocity 0.0010162726837399823 0.0007336437846886255 0.006165931046016371

}

LEARNING OUTCOME:

* We have gained the ability to design basic robot models, implement them, and develop their controllers.
* Specifically, we learned about the various nodes involved in robot model design, techniques for translating and rotating one solid relative to another, and how to control a rotational motor through the controller.