Major Project

ROBOTICS & AI

Priyanshu Bist | 8-December-2023

Introduction

In recent years, the use of mobile robots has been increasing rapidly and has found an important and inevitable position in automating an industry, or society, and so on. These mobile robots are broadly divided into two classes – Wheeled robots and legged robots. Wheeled robots are primarily used in places where the surface is flat and smooth. Wheeled robots are not suited to operate in uneven terrain which is harsh, rough, and bumpy. It has been proved experimentally that legged robots play an effective role. As compared to wheeled robots, legged robots perform the task with high accuracy and improve the efficiency of the overall process, as they can overcome the disadvantages faced by wheeled robots operating on even as well as uneven surfaces legged robots experience a huge magnitude of external forces are capable of regaining their original state or original position. There are certain areas where the wheeled robot cannot provide more value and is not handy in uneven, bumpy terrains. On the other hand, legged robots which often mimic animal actions, are well structured and can be more stable in every environment than just a wheeled robot. Legged robots can be used for different kinds of applications like border surveillance in military and defense, used for navigation in shopping malls, in industries for performing operations like pick and place with a robotic arm fitted on the body of the robot, and so on, a quadruped robot consists of four legs with a greater number of degrees of freedom.

Methodology

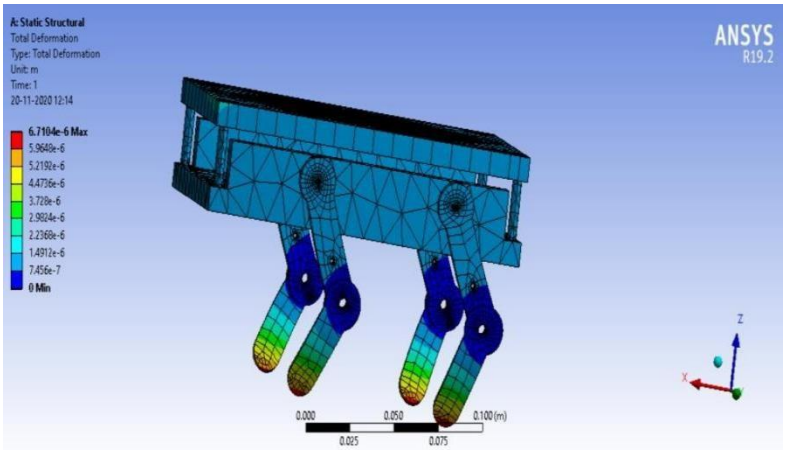
The design and development of the proposed quadruped robot focus on optimization of CAD design, Static & Transient Structural Analysis.

DESIGN CONSIDERATION

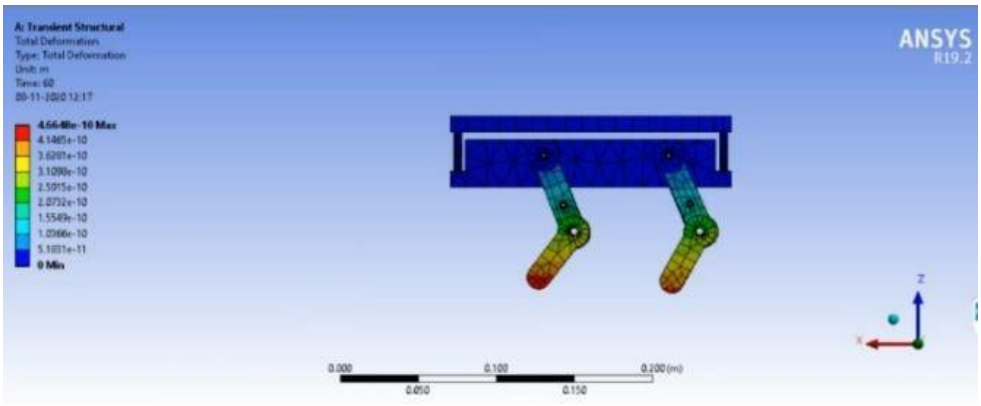
As this robot is specifically concentrated for military surveillance applications, it is intended to keep the robot light and small. So, every design has similar dimensions. The robot's basic payload is 1.5 kg, which includes the weight acting on the robot due to chassis, miscellaneous weight due to electronic components, etc. It is assumed that the robot will not carry any external load except the basic load. A robot is said to be stable if the robot can regain its original state when external forces or motion start acting on the robot. In quadruped robots, the robot is stable if it can gain its original position or the

home position after external forces act on it. This stability in a quadruped robot is of two types, static and dynamic stability.

Static structural analysis determines the displacements, stresses, strains, and forces in structures or components caused by loads that do not induce significant inertia and damping effects.



To find the dynamic response of the structure with the presence of time, displacement, and velocity transient structural analysis is applied.



CAD MODEL PROVIDED IN ZIPPED FOLDER

Reason for cuboidal type design:

Stability and Balance: The cuboidal body design offers inherent stability due to its low center of gravity. This stability is crucial for load-carrying applications and ensures safe transportation of payloads.

Structural Integrity: Cuboidal bodies are known for their structural strength and rigidity allowing the robot to withstand the stresses and strains associated with carrying substantial loads.
Payload Distribution: The cuboidal shape provides an even distribution of weight across the robot's base, which is advantageous for load-carrying robots. This design minimizes the risk of overloading specific areas and maintains overall balance.
Interior Space: The ample interior space within the cuboidal body can house components, electronics, batteries, and additional equipment. This interior space can be efficiently utilized for housing the necessary circuits and control systems, contributing to an organized and compact design.
Versatility: Cuboidal designs offer versatility and adaptability. They provide a versatile platform for integrating and customizing additional features, such as sensors, cameras, and communication equipment, making the robot suitable for a wide range of tasks.

Electronic work

Introduction: Simulating Obstacle Detection and Robot Movement in TinkerCAD

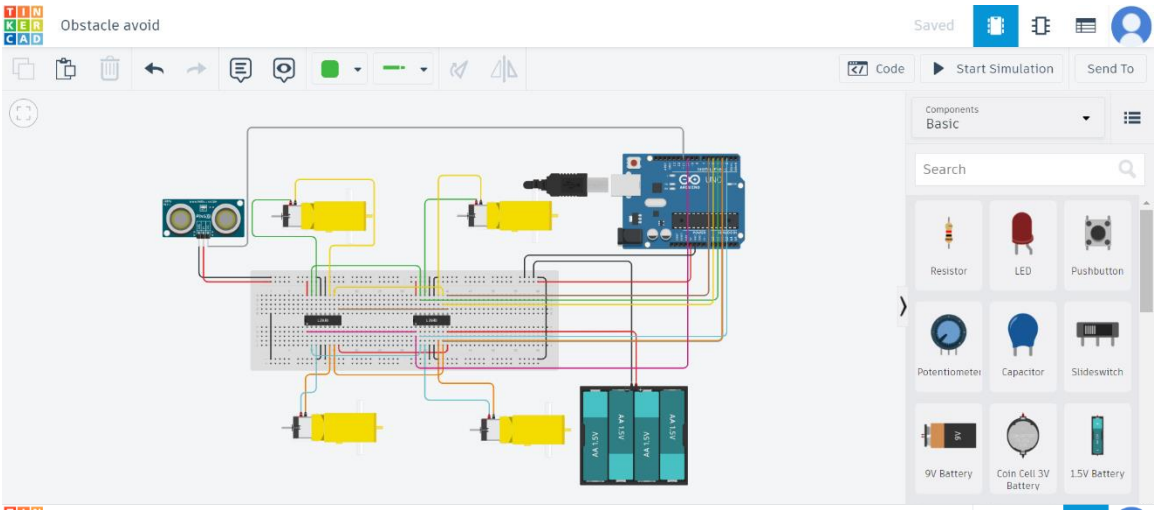
Robotics represents a dynamic field at the intersection of hardware and software, pushing the boundaries of innovation to address real-world challenges. This project focuses on the crucial aspects of obstacle detection and controlling robot movement, leveraging the versatile simulation platform, TinkerCAD.

Project Objectives: The primary objective of this project is to develop and simulate a robotic system capable of autonomously detecting obstacles in its environment and adjusting its movement accordingly.

Overview of TinkerCAD: TinkerCAD provides a virtual workspace for designing, testing, and simulating electronic circuits and systems.

Scope and Limitations: *Focus on Obstacle Detection:* The scope of this project centers on the integration of ultrasonic sensor for obstacle detection and the implementation of a responsive motor control system.

SCREENSHOTS OF MULTISIM ADDED BELOW



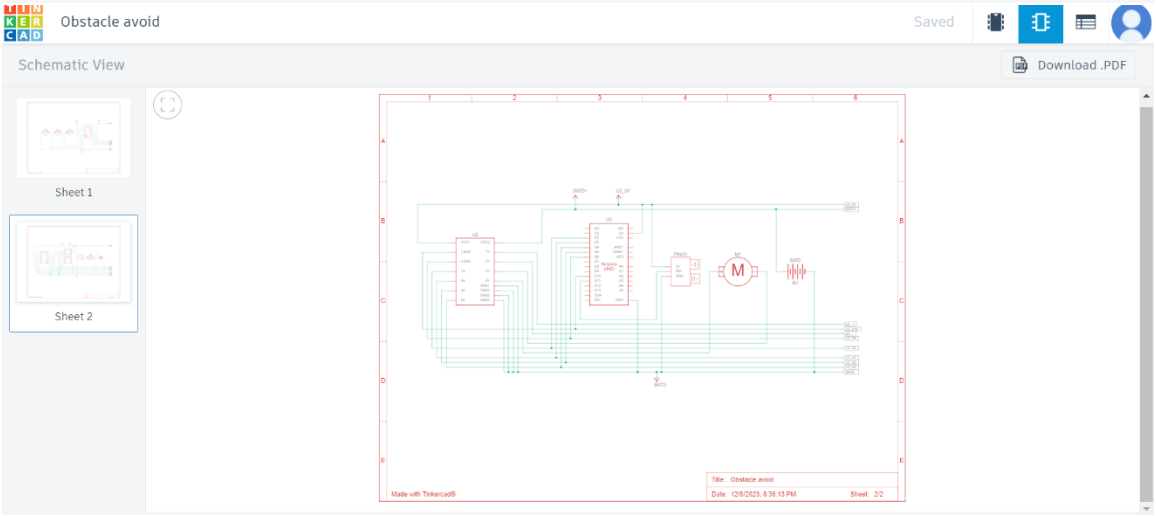
Obstacle avoid

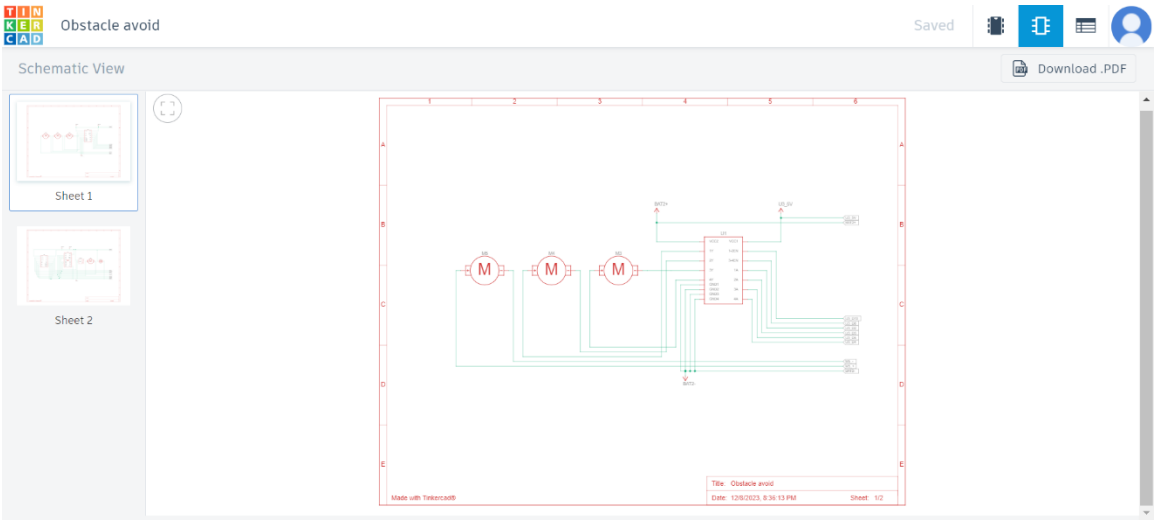
Saved

Component List

Download CSV

Name	Quantity	Component
Bat2	1	4 batteries, AA, no 1.5V Battery
M1 M3 M4 M5	4	Hobby Gearmotor
U1 U2	2	H-bridge Motor Driver
U3	1	Arduino Uno R3
PING1	1	Ultrasonic Distance Sensor





Code:

```
int distance = 0;
int i = 0;
long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  return pulseIn(echoPin, HIGH);
}
void setup()
{
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(3, OUTPUT);
```

```
  pinMode(2, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(10, OUTPUT);
}
void loop()
{
  distance = 0.01723 * readUltrasonicDistance(11, 11);
  distance = (distance / 2.54);
  if (distance > 15) {
    // MoveForward
    digitalWrite(5, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(2, HIGH);
    analogWrite(6, 255);
    analogWrite(10, 255);
  } else {
    // TurnRight
    digitalWrite(5, LOW);
    digitalWrite(4, LOW);
    digitalWrite(3, LOW);
    digitalWrite(2, LOW);
    analogWrite(6, 80);
    analogWrite(10, 80);
  }
  delay(10); // Delay a little bit to improve simulation performance
}
```

Explanation:

1. **Ultrasonic Sensor Function (readUltrasonicDistance): This function is responsible for reading the distance from an ultrasonic sensor. It triggers the sensor, measures the time it takes for the ultrasonic waves to bounce back, and calculates the distance in centimeters.**
2. **Setup Function (setup): Configures the necessary pins for motor control (5, 4, 3, 2) and motor speed control (6, 10) as OUTPUT.**
3. **Loop Function (loop):**
    - **In the loop, the distance is continuously measured using the ultrasonic sensor.**
    - **If the distance is greater than 15 cm, the robot is instructed to move forward by activating specific motor control pins and setting the motor speeds to maximum.**
    - **If the distance is 15 cm or less, the robot is instructed to turn right by deactivating specific motor control pins and setting the motor speeds to a lower value.**

SIMULATION and Testing: VIDEO included in Folder

Code taken from a Github Repo for Coppeliasim simulation-

```
function sysCall_init()
  -- do some initialization here
  RobotBase=sim.getObjectAssociatedWithScript(sim.handle_self)
  leftMotor=sim.getObjectHandle("motor_kiri")
  rightMotor=sim.getObjectHandle("motor_kanan")
  noseSensor=sim.getObjectHandle("front_prox")
  minMaxSpeed={50*math.pi/180,300*math.pi/180}
  backUntilTime=-1
  xml = '<ui title="'..sim.getObjectName(RobotBase)..'speed" closeable="false" resizable="false" activate="false">'..[[
    <hslider minimum="0" maximum="100" on-change="speedChange_callback" id="1"/>
    <label text="" style="* {margin-left: 300px;}"/>
```

```
    </ui>
  ]]
  ui=simUI.create(xml)
  speed=(minMaxSpeed[1]+minMaxSpeed[2])*0.3
  simUI.setSliderValue(ui,1,100*(speed-minMaxSpeed[1])/(minMaxSpeed[2]-
minMaxSpeed[1]))
end

function speedChange_callback(ui,id,newVal)
  speed=minMaxSpeed[1]+(minMaxSpeed[2]-minMaxSpeed[1])*newVal/100
end

function sysCall_actuation()
  -- put your actuation code here
  result=sim.readProximitySensor(noseSensor)
  if(result>0) then backUntilTime=sim.getSimulationTime()+4 end

  if(backUntilTime<sim.getSimulationTime()) then
    sim.setJointTargetVelocity(leftMotor,-speed)
    sim.setJointTargetVelocity(rightMotor,-speed)
  else
    sim.setJointTargetVelocity(leftMotor,speed/5)
    sim.setJointTargetVelocity(rightMotor,speed/7)
  end
end

function sysCall_sensing()
  -- put your sensing code here
end

function sysCall_cleanup()
```

```
    -- do some clean-up here
    simUI.destroy(ui)
end
-- See the user manual or the available code snippets for additional callback functions and details
```

CONCLUSION:

Although the goal of seamless ROS2 integration remained beyond immediate reach, significant strides were made in other pivotal aspects.

1. Circuit Design with Multisim:

2. 3D CAD Model:

3. Simulation with CoppeliaSim:

4. Learnings and Adaptability: Drew from examples and tutorials available online, showcasing a commitment to adaptability. The project's journey underscores the fusion of theoretical knowledge and practical application, marking it as a valuable exploration of robotics concepts.

In summary, this project, though diverted from its initial trajectory, has yielded tangible outputs and instilled an appreciation for the iterative nature of innovation. The circuit design, 3D CAD model, and CoppeliaSim simulation collectively form a robust foundation for future endeavors.