**a) What are the parameters that should be considered while calculating maximum angle of inclination in Two wheeled self-balancing robots (TWSBR) ?**

When calculating the maximum angle of inclination for a two-wheeled self-balancing robot (TWSBR), several key parameters need to be considered to ensure stability and proper functioning. Here are the main parameters:

1. Center of Gravity (CoG):
   * The height of the CoG relative to the wheelbase.
   * The lateral position of the CoG, which affects the balance in the sideways direction.
2. Wheelbase:
   * The distance between the two wheels, affecting the stability and the ability to balance.
3. Weight Distribution:
   * The distribution of the robot's weight, ensuring it is balanced around the CoG.
4. Inertia:
   * The moment of inertia of the robot, particularly around the pivot point between the wheels.
5. Motor Specifications:
   * Torque and speed of the motors, as they determine the robot’s ability to correct itself when it starts to tip.
6. Wheel Radius:
   * The size of the wheels, which impacts the robot's ability to climb inclines and navigate uneven surfaces.
7. Sensor Accuracy:
   * The precision and responsiveness of the accelerometers and gyroscopes used for detecting inclination and balance.
8. Control System:
   * The algorithm used (e.g., PID controller) and its tuning parameters, which dictate how quickly and effectively the robot responds to changes in inclination.
9. Surface Conditions:
   * The type of surface the robot is operating on, including factors like friction and slipperiness.
10. Battery and Power Supply:
    * The capacity and voltage of the battery, as well as the power consumption of the motors and sensors, which affect how long the robot can operate and its ability to handle inclines.
11. Environmental Factors:
    * External conditions such as wind, which can affect the robot’s stability.

### Detailed Explanation:

1. Center of Gravity (CoG):
   * Importance: A lower CoG increases stability and makes it easier to balance. If the CoG is too high, the robot will be more prone to tipping over.
   * Calculation: The vertical and horizontal positions of the CoG are critical. They can be determined through physical measurements or calculations based on the distribution of the robot's mass.
2. Wheelbase:
   * Importance: A wider wheelbase provides more stability and allows for greater angles of inclination without tipping over.
   * Calculation: Measured as the distance between the contact points of the two wheels on the ground.
3. Weight Distribution:
   * Importance: Even weight distribution ensures the robot does not lean to one side, which could affect its ability to balance.
   * Calculation: The weight of each component and its position relative to the CoG should be taken into account.
4. Inertia:
   * Importance: The moment of inertia affects how the robot responds to angular accelerations. A higher moment of inertia means the robot is more resistant to changes in its rotational motion.
   * Calculation: Depends on the mass distribution and geometry of the robot.
5. Motor Specifications:
   * Importance: Motors need to provide sufficient torque to counteract the force of gravity when the robot inclines.
   * Calculation: Torque requirements can be calculated based on the weight of the robot, the radius of the wheels, and the desired maximum angle of inclination.
6. Wheel Radius:
   * Importance: Larger wheels can help in navigating over obstacles and inclines but may affect the balance due to increased height of the CoG.
   * Calculation: Directly measured and factored into torque calculations.
7. Sensor Accuracy:
   * Importance: Accurate sensors are essential for detecting the angle of inclination and making precise adjustments to maintain balance.
   * Calculation: Sensor specifications, such as sensitivity and response time, need to be considered.
8. Control System:
   * Importance: The control algorithm determines how the robot responds to deviations from its balanced state.
   * Calculation: Tuning of control parameters (e.g., PID gains) is done through testing and adjustment.
9. Surface Conditions:
   * Importance: Different surfaces offer different levels of grip, affecting the robot's ability to balance and climb inclines.
   * Calculation: Friction coefficients and surface texture can be considered in the design phase.
10. Battery and Power Supply:
    * Importance: Adequate power supply ensures the motors and sensors function correctly, especially when climbing inclines.
    * Calculation: Power requirements based on motor specifications and expected operational duration.
11. Environmental Factors:
    * Importance: Conditions like wind or uneven ground can affect stability.
    * Calculation: Environmental factors are typically considered in robustness testing and design adjustments.

By carefully considering and calculating these parameters, a two-wheeled self-balancing robot can be designed to handle the desired maximum angle of inclination while maintaining stability and proper functionality.

**b) What is the core concept of the Two wheeled self-balancing robots (TWSBR).**

### **Core Concept of Two-Wheeled Self-Balancing Robots (TWSBR)**

The core concept of two-wheeled self-balancing robots (TWSBR) revolves around maintaining balance and stability through continuous adjustments based on sensor feedback and control algorithms. This involves several key principles:

1. Inverted Pendulum Model:
   * Concept: TWSBRs operate similarly to an inverted pendulum, where the robot’s body is the pendulum that needs to be balanced upright on its wheels. The goal is to keep the center of gravity directly above the pivot point (wheels).
2. Feedback Control System:
   * Sensors: TWSBRs use sensors like accelerometers and gyroscopes to measure the tilt angle and rate of rotation of the robot.
   * Control Algorithms: A control algorithm, often a PID (Proportional-Integral-Derivative) controller, processes the sensor data and determines the necessary wheel movements to maintain balance. The algorithm continuously adjusts the wheel speeds to correct any deviations from the upright position.
3. Actuators (Motors):
   * Wheel Control: The wheels are driven by motors that can move forward and backward. By adjusting the speed and direction of the wheels, the robot can counteract the tilting motion and stay balanced.
   * Torque Application: The motors apply torque to the wheels to generate the necessary corrective forces to keep the robot upright.
4. Real-time Processing:
   * Fast Response: The control system must operate in real-time, with rapid sensor data acquisition and processing to provide immediate corrective actions.
   * Stability: Ensuring stability requires precise and timely adjustments, as even small delays or inaccuracies can lead to a loss of balance.
5. Sensor Fusion:
   * Combining Data: Combining data from multiple sensors (accelerometers, gyroscopes, sometimes magnetometers) helps improve the accuracy of the robot's state estimation, leading to better balance control.
   * Kalman Filter: Often, a Kalman filter is used for sensor fusion to provide a more accurate estimate of the robot's tilt angle and angular velocity.

### Detailed Explanation:

1. Inverted Pendulum Model:
   * The robot is modeled as an inverted pendulum, where the wheels are the pivot point and the body is the pendulum that needs to be balanced.
2. Feedback Control System:
   * Sensors: Measure the robot’s tilt angle and angular velocity.
   * PID Controller: Adjusts the motor commands to maintain the balance. The PID controller continuously calculates the error between the desired upright position and the current tilt angle, and then it applies corrective actions based on proportional, integral, and derivative terms.
3. Actuators (Motors):
   * Motors: Drive the wheels forward or backward to correct the robot’s tilt. By moving the wheels in the direction of the tilt, the robot can bring itself back to an upright position.
   * Wheel Encoders: Provide feedback on the wheel positions and speeds, which helps in precise control.
4. Real-time Processing:
   * The control system must process sensor data and compute the motor commands within milliseconds to ensure timely corrections and maintain stability.
5. Sensor Fusion:
   * Combining Sensor Data: Improves the accuracy of tilt and velocity estimates.
   * Kalman Filter: A commonly used method for sensor fusion, providing an optimal estimate by considering the noise and uncertainties in sensor measurements.

By integrating these concepts, a two-wheeled self-balancing robot can effectively maintain its balance, navigate its environment, and respond to disturbances. The combination of accurate sensor measurements, robust control algorithms, and responsive actuators enables the robot to perform tasks that require dynamic stability.

**c) What is the additional thing/component that you can add to make it unique and explain the same by giving its proof of concept.**

### Additional Component for Two-Wheeled Self-Balancing Robot: Autonomous Navigation with LIDAR

Component to Add:

* LIDAR (Light Detection and Ranging) Sensor for autonomous navigation.

Purpose:

* Enhance the robot's functionality by enabling it to navigate autonomously in its environment, avoid obstacles, and map its surroundings.

### Proof of Concept

1. Integration of LIDAR Sensor:

* LIDAR Sensor Selection: Choose a suitable LIDAR sensor like the RPLIDAR A1, which is compact and provides 360-degree scanning capabilities.
* Mounting: Secure the LIDAR sensor on the top of the robot to get an unobstructed 360-degree view of the environment.

2. Data Processing:

* LIDAR Data Acquisition: Connect the LIDAR sensor to the robot’s onboard computer (e.g., a Raspberry Pi or an Arduino with a processing shield) and acquire real-time distance measurements.
* SLAM Algorithm: Implement Simultaneous Localization and Mapping (SLAM) algorithm to create a map of the environment and localize the robot within that map. Libraries such as Gmapping or Hector SLAM can be used for this purpose.

3. Autonomous Navigation:

* Path Planning: Use path planning algorithms like A\* or Dijkstra to navigate the mapped environment. The algorithm will plan the optimal path to a specified destination while avoiding obstacles.
* Obstacle Avoidance: Implement real-time obstacle avoidance using LIDAR data to dynamically adjust the robot’s path if an obstacle is detected.

4. Control System Integration:

* Sensor Fusion: Combine LIDAR data with the existing IMU (accelerometer and gyroscope) data for more accurate localization and navigation.
* Control Loop: Integrate the autonomous navigation system with the robot’s control loop to send motor commands for movement and balance adjustments.

5. Software Implementation:

* Robot Operating System (ROS): Use ROS to handle the communication between different components, process sensor data, and control the robot. ROS has built-in packages for LIDAR integration, SLAM, and navigation.
* Custom Scripts: Write custom scripts to handle the specific requirements of the robot, such as initializing the LIDAR, processing the SLAM output, and executing the planned path.

### Example Workflow:

1. Initialization:
   * Initialize the LIDAR sensor and start acquiring data.
   * Initialize the IMU sensors for tilt and orientation data.
   * Start the SLAM algorithm to build a map and localize the robot.
2. Mapping:
   * As the robot moves, the LIDAR sensor collects distance measurements.
   * The SLAM algorithm processes these measurements to create a map of the environment.
   * The robot continuously updates its position on the map using sensor data.
3. Path Planning and Navigation:
   * Set a destination point on the map.
   * The path planning algorithm calculates the optimal path to the destination, avoiding obstacles.
   * The control system sends motor commands to follow the planned path while maintaining balance.
4. Obstacle Avoidance:
   * Continuously monitor LIDAR data for any new obstacles.
   * If an obstacle is detected, adjust the path dynamically to avoid it while maintaining the target destination.

### Proof of Concept - Practical Example:

Scenario:

* The robot needs to navigate from one end of a cluttered room to the other.

Steps:

1. Mapping:
   * The robot starts moving and the LIDAR sensor begins scanning the environment.
   * The SLAM algorithm builds a real-time map as the robot moves around.
2. Path Planning:
   * Once the map is sufficiently detailed, set a target location on the opposite side of the room.
   * The path planning algorithm computes a path avoiding detected obstacles.
3. Navigation:
   * The robot follows the computed path, using the LIDAR and IMU data to adjust its movements.
   * If the LIDAR detects an unexpected obstacle, the robot recalculates its path in real-time.
4. Reaching the Destination:
   * The robot successfully navigates to the target location, demonstrating autonomous navigation and obstacle avoidance.

### Advantages:

* Autonomous Operation: Enables the robot to operate without human intervention.
* Enhanced Functionality: Adds advanced capabilities such as environment mapping and dynamic path planning.
* Real-World Applications: Suitable for applications in warehouse automation, search and rescue, and home robotics.

By integrating a LIDAR sensor and implementing autonomous navigation, the two-wheeled self-balancing robot can become a versatile and intelligent system capable of performing complex tasks in dynamic environments.

**d) Suppose you have been given the computer aided design model of the robot satisfying the necessary clearances and dimensions to hold the parcel. Using the necessary components, you have been assigned the task of programming the robot such that it can carry out the assigned task of delivering the parcel to the consumer doorstep navigating the traffic, turns and other obstacles. Explain how you would achieve this with the code and circuitry to substantiate your claim. It is allowed to take the necessary assumptions if you are certain that it is outside the scope of robotics to obtain the necessary data and then derive conclusions pertaining to that specific aspect.**

**Project Overview**

The project involves designing a two-wheeled self-balancing robot that can autonomously deliver parcels to consumer doorsteps. The robot must navigate through traffic, make turns, and avoid obstacles while maintaining balance and stability. The core components include Lidar sensors for mapping and obstacle detection, an IMU for balance control, ultrasonic sensors for close-range detection, and a single-board computer for high-level processing and navigation.

#### **Implementatio**n Steps:

1. Hardware Assembly:
   * Chassis Design: Ensure the chassis is robust and can accommodate all components with the necessary clearances and dimensions to hold the parcel.
   * Mounting Sensors and Controllers: Securely mount the Lidar, IMU, ultrasonic sensors, and motor controllers on the chassis.
   * Wiring and Connections: Properly connect all sensors and controllers to the single-board computer and microcontroller, ensuring reliable power supply and data communication.
2. Software Development:
   * ROS Setup: Install and configure ROS on the Raspberry Pi, creating a workspace and setting up necessary packages for SLAM, navigation, and sensor integration.
   * Sensor Integration: Write ROS nodes to handle data acquisition from the Lidar, IMU, and ultrasonic sensors, and process this data for real-time navigation and obstacle avoidance.
   * SLAM and Mapping: Implement a SLAM algorithm to create a map of the environment and keep track of the robot’s position within this map.
   * Path Planning: Develop path planning algorithms to determine the optimal route to the consumer’s doorstep, taking into account the current map and dynamic obstacles.
   * Motor Control: Implement a PID controller on the Arduino for real-time balance control, ensuring the robot remains upright while moving.
3. Testing and Iteration:
   * Component Testing: Individually test each component (sensors, motors, controllers) to ensure proper functionality.
   * Integrated System Testing: Conduct integrated tests in a controlled environment to validate the overall system performance, making adjustments as necessary.
   * Real-world Testing: Test the robot in a real-world scenario, navigating through an environment with traffic, turns, and obstacles to ensure reliable delivery performance.

### **Design and Hardware Components**

#### Components List:

* Lidar Sensor: For mapping and obstacle detection.
* Ultrasonic Sensors: For close-range obstacle detection.
* IMU: For tilt and orientation detection.
* Motors: DC motors with encoders.
* Motor Controllers: H-Bridge motor drivers.
* Single-board Computer: Raspberry Pi for high-level processing.
* Microcontroller: Arduino for real-time control.
* Power Supply: Batteries to power all components.
* Chassis: A sturdy frame to hold all components.

#### **Circuit Connections:**

1. Raspberry Pi Connections:
   * Lidar Sensor:
     + Connect Lidar TX to Raspberry Pi RX.
     + Connect Lidar RX to Raspberry Pi TX.
     + Power Lidar with 5V and GND from Raspberry Pi.
   * Arduino:
     + Connect Raspberry Pi GPIO (e.g., pin 14 for TX, pin 15 for RX) to Arduino RX and TX respectively for serial communication.
   * Ultrasonic Sensors:
     + Connect trigger pins to Raspberry Pi GPIO pins.
     + Connect echo pins to Raspberry Pi GPIO pins.
     + Power ultrasonic sensors with 5V and GND from Raspberry Pi.
2. Arduino Connections:
   * IMU (MPU6050):
     + Connect VCC to 3.3V on Arduino.
     + Connect GND to GND on Arduino.
     + Connect SCL to A5 (SCL) on Arduino.
     + Connect SDA to A4 (SDA) on Arduino.
   * Motor Drivers (L298N):
     + Connect ENA to Arduino PWM pin (e.g., pin 9).
     + Connect IN1 and IN2 to Arduino digital pins (e.g., pins 10 and 11).
     + Connect ENB to Arduino PWM pin (e.g., pin 6).
     + Connect IN3 and IN4 to Arduino digital pins (e.g., pins 5 and 4).
   * Motors:
     + Connect motor A to OUT1 and OUT2 on L298N.
     + Connect motor B to OUT3 and OUT4 on L298N.
3. Power Connections:
   * Connect a 12V battery to the power inputs of the L298N motor drivers.
   * Use a DC-DC converter to step down the voltage to 5V for the Raspberry Pi and Arduino.
   * Ensure all components share a common ground.

**Code**

Path Planning and Navigation Node:

import rospy

from move\_base\_msgs.msg import MoveBaseAction, MoveBaseGoal

import actionlib

def movebase\_client():

client = actionlib.SimpleActionClient('move\_base', MoveBaseAction)

client.wait\_for\_server()

goal = MoveBaseGoal()

goal.target\_pose.header.frame\_id = "map"

goal.target\_pose.header.stamp = rospy.Time.now()

# Set the goal position and orientation

goal.target\_pose.pose.position.x = 2.0

goal.target\_pose.pose.position.y = 2.0

goal.target\_pose.pose.orientation.w = 1.0

client.send\_goal(goal)

wait = client.wait\_for\_result()

if not wait:

rospy.logerr("Action server not available!")

else:

return client.get\_result()

if \_\_name\_\_ == '\_\_main\_\_':

try:

rospy.init\_node('movebase\_client\_py')

result = movebase\_client()

if result:

rospy.loginfo("Goal execution done!")

except rospy.ROSInterruptException:

rospy.loginfo("Navigation test finished.")

#### **Obstacle Avoidance:**

* Ultrasonic Sensor Integration:

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

TRIG = 23

ECHO = 24

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

def distance():

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

start\_time = time.time()

stop\_time = time.time()

while GPIO.input(ECHO) == 0:

start\_time = time.time()

while GPIO.input(ECHO) == 1:

stop\_time = time.time()

time\_elapsed = stop\_time - start\_time

distance = (time\_elapsed \* 34300) / 2

return distance

try:

while True:

dist = distance()

print("Measured Distance = %.1f cm" % dist)

time.sleep(1)

except KeyboardInterrupt:

print("Measurement stopped by User")

GPIO.cleanup()

#### **Motor Control and Balance Algorithm:**

* Arduino Balance Control:

#include <Wire.h>

#include <MPU6050.h>

MPU6050 mpu;

const int motorPinA1 = 9;

const int motorPinA2 = 10;

const int motorPinB1 = 6;

const int motorPinB2 = 5;

double Kp = 30;

double Ki = 0;

double Kd = 0;

double setPoint = 0;

double input, output;

double lastInput;

double integral;

void setup() {

Wire.begin();

Serial.begin(9600);

mpu.initialize();

pinMode(motorPinA1, OUTPUT);

pinMode(motorPinA2, OUTPUT);

pinMode(motorPinB1, OUTPUT);

pinMode(motorPinB2, OUTPUT);

}

void loop() {

input = mpu.getAngleX();

double error = setPoint - input;

integral += error;

double derivative = input - lastInput;

output = Kp \* error + Ki \* integral + Kd \* derivative;

if (output > 0) {

analogWrite(motorPinA1, output);

analogWrite(motorPinA2, 0);

analogWrite(motorPinB1, output);

analogWrite(motorPinB2, 0);

} else {

analogWrite(motorPinA1, 0);

analogWrite(motorPinA2, -output);

analogWrite(motorPinB1, 0);

analogWrite(motorPinB2, -output);

}

lastInput = input;

delay(10);

}

### **Calculations and Theory**

#### Center of Gravity and Balance Control:

* Center of Gravity Calculation:

CoG=∑mi​∑(mi​×xi​)​

Where mi​ is the mass of each component, and xi​ is the position of each component relative to a reference point. Ensuring the CoG is directly above the wheelbase is critical for maintaining balance.

* Moment of Inertia:

I=∑mi​⋅ri2​

Where ri​ is the distance of each mass element mi​ from the axis of rotation. The moment of inertia affects the torque required to maintain balance and perform maneuvers.

#### Torque and Motor Specifications:

* Torque Calculation:

τ=Iα

Where τ is the torque, I is the moment of inertia, and α is the angular acceleration. This helps determine the motor specifications needed to achieve desired acceleration.

* Motor Power:

P=τ⋅ω

Where P is power, τ is torque, and ω is angular velocity. This equation helps in selecting motors that can provide the necessary power for movement and balance.

#### Sensor Fusion:

* Kalman Filter for Sensor Fusion:

x^k∣k​=x^k∣k−1​+Kk​(zk​−Hx^k∣k−1​)

Where x^k∣k​ is the estimated state, Kk​ is the Kalman gain, zk​ is the measurement, and H is the measurement model. This filter combines data from the IMU and Lidar to provide accurate position and orientation estimates.

* PID Controller Tuning:

u(t)=Kp​e(t)+Ki​∫e(t)dt+Kd​dtde(t)​

Where u(t) is the control input, Kp​, Ki​, and Kd​ are the proportional, integral, and derivative gains respectively, and e(t) is the error signal. Proper tuning of these parameters is essential for responsive and stable balance control.

#### Battery Life Calculation:

* Energy Consumption:

E=P×t

Where E is energy consumed, P is power, and t is time. Calculating energy consumption helps in selecting an appropriate battery to ensure sufficient operation time.

* Battery Capacity:

C=VE​

Where C is the battery capacity in ampere-hours (Ah), E is energy, and V is the battery voltage. Ensuring the battery can supply the required energy for the duration of the delivery task is crucial.

#### Stability Analysis:

* Natural Frequency:

ωn​=mk​​

Where ωn​ is the natural frequency, k is the stiffness of the system, and m is the mass. Understanding the natural frequency helps in designing a control system that avoids resonant frequencies.

* Damping Ratio:

ζ=2km​c​

Where ζ is the damping ratio, c is the damping coefficient, k is the stiffness, and m is the mass. Proper damping ensures the system quickly returns to stability after a disturbance.