**Unique Project Idea: Autonomous Two-Wheeled Self-Balancing Delivery Robot with Dynamic Obstacle Avoidance and Solar Charging**

**Project Overview**

The project focuses on designing and developing an autonomous two-wheeled self-balancing robot capable of delivering parcels while navigating complex environments. The robot will be equipped with dynamic obstacle avoidance capabilities and a solar charging system to extend its operational time and reduce reliance on conventional charging methods. This project will integrate advanced sensor technologies, control algorithms, and sustainable energy solutions.

**Repository Structure: BotBrains\_Battle\_Round-2**

**1. Zipped Folder Contents**

* **Solution Files**:
  + solution\_a.md: Solution for part a (parameters for maximum angle of inclination).
  + solution\_b.md: Core concept explanation.
  + solution\_c.md: Additional component and proof of concept.
  + solution\_d.md: Programming and circuitry details.
* **Documentation Files**:
  + ideation.md: Detailed ideation process.
  + diagrams.md: Diagrams and flowcharts.
  + theory.md: Theoretical background.
  + calculations.md: Calculations for parameters.
  + references.md: References and sources.
* **Code Files**:
  + robot\_code.py: Python code for robot control.
* **Readme File**:
  + README.md: Overview and instructions.

**Project Details**

**a) Parameters for Calculating Maximum Angle of Inclination in TWSBR**

1. **Center of Gravity (CoG)**
   * Determines stability and maximum tilt angle.
2. **Wheelbase and Wheel Diameter**
   * Affects balance and maneuverability.
3. **Motor Torque and Power**
   * Impacts the ability to correct tilt.
4. **Sensor Accuracy (Gyroscopes and Accelerometers)**
   * Crucial for detecting tilt and balance.
5. **Weight Distribution**
   * Affects overall stability.
6. **Surface Friction**
   * Influences grip and stability.

**b) Core Concept of TWSBR**

* **Balancing Mechanism**: Utilizes gyroscopes and accelerometers for tilt detection and motor adjustments.
* **Dynamic Control Systems**: Implements PID control for maintaining balance.
* **Autonomous Navigation**: Uses sensors (LIDAR, ultrasonic, cameras) for obstacle detection and path planning.
* **Communication Systems**: Employs wireless communication for remote control and monitoring.

**c) Additional Component: Solar Charging System**

**Proof of Concept**:

* **Description**: Integrate a lightweight, flexible solar panel on top of the robot to charge the battery.
* **Advantages**:
  + **Extended Operation Time**: Reduces downtime by charging during operation or idle periods.
  + **Environmental Benefit**: Utilizes renewable energy, making the robot eco-friendly.
  + **Cost-Effective**: Lowers operational costs by harnessing solar energy.
* **Implementation**:
  + Use a solar charge controller to manage power input to the battery.
  + Position the solar panel for maximum exposure to sunlight.

**d) Programming the Robot for Parcel Delivery**

**Approach**:

1. **Path Planning**:
   * Implement A\* or Dijkstra’s algorithm for optimal route finding.
2. **Obstacle Detection and Avoidance**:
   * Use LIDAR and ultrasonic sensors to detect and navigate around obstacles.
   * Implement a reactive control algorithm for dynamic environments.
3. **Balance and Navigation Control**:
   * Use a PID controller for maintaining balance.
   * Employ encoders and an IMU for precise movement tracking.
4. **Code Implementation**:

python

import RPi.GPIO as GPIO

from time import sleep

import math

# PID control constants

Kp = 2.0

Ki = 0.0

Kd = 1.0

# Initialize sensors and motors

def init\_sensors():

# Code to initialize gyroscope, accelerometer, and other sensors

pass

def get\_sensor\_data():

# Code to read sensor data

pass

def set\_motor\_speed(left\_speed, right\_speed):

# Code to set motor speed

pass

# PID control loop

def balance\_control():

error\_sum = 0

last\_error = 0

while True:

angle = get\_sensor\_data()

error = 0 - angle

error\_sum += error

d\_error = error - last\_error

control = Kp \* error + Ki \* error\_sum + Kd \* d\_error

set\_motor\_speed(control, -control)

last\_error = error

sleep(0.01)

# Main function to run the robot

def main():

init\_sensors()

while True:

balance\_control()

# Add path planning and obstacle avoidance logic here

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. **Circuit Design**:
   * Use a Raspberry Pi or Arduino as the main controller.
   * Connect motor drivers to control the motors.
   * Integrate sensors (gyroscope, accelerometer, ultrasonic sensors) with the controller.
2. **Testing and Calibration**:
   * Test in a controlled environment to fine-tune PID parameters.
   * Calibrate sensors for accurate readings.

**Documentation**

* **Ideation**: Explanation of design choices and development process.
* **Diagrams**: Block diagrams and flowcharts.
* **Theory**: Detailed explanation of balancing mechanisms, control algorithms, and navigation systems.
* **Calculations**: Math behind inclination angle, motor torque, and PID control.
* **References**: List of sources and tools used.