**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.