BotBrains\_Battle\_Round-2

Two Wheeled Self-Balancing Robots (TWSBR)

-Mokshyada Mishra

**Ideation for Two Wheeled Self-Balancing Robot (TWSBR) Code**

In this project, the main challenge was to make the robot balance on two wheels while moving and carrying parcels. We also wanted our robot to stand out, so we decided to use LIDAR for advanced navigation.

**Problem Breakdown**

1. **Balancing Mechanism**:
   * **Sensors**: A gyroscope and accelerometer (MPU6050) to get real-time orientation data.
   * **Control System**: PID controller to adjust the motor speeds based on how much the robot tilts.
   * **Code**: The main loop would constantly read sensor data and adjust the motors.
2. **Navigation**:
   * **LIDAR Integration**: Adding LIDAR helps the robot map its surroundings and plan paths.
   * **Path Planning**: writing a basic path planning algorithm using the LIDAR data, and also implementing real-time obstacle avoidance is a key step.
   * **Code**: Read LIDAR data, plan paths, and adjust motor speeds as needed.
3. **Payload Management**:
   * **Sensors**: We thought about using load sensors to check if the parcel is properly placed and to detect if it falls off.
4. **Communication**:
   * **Wireless Module**: We used WiFi or Bluetooth for communication.

**IDEATION FOR CODE:**

1. Include Necessary Libraries:

The code starts by including libraries to interact with the sensors and components:

* + 1. Wire.h: For I2C communication with the MPU6050.

2.MPU6050.h: A library for interfacing with the MPU6050 sensor.

3.TinyGPS++.h: For parsing GPS data.

4.SoftwareSerial.h: To create a serial interface for the GPS module.

2. Instantiate Sensor Objects:

instances for the MPU6050 and GPS sensors:

1.MPU6050 mpu;

2.TinyGPSPlus gps;

3.SoftwareSerial gpsSerial(4, 3); // RX, TX pins for GPS communication.

C). Defined PID Constants and Motor Pins:

Defining constants for the PID control will help maintain balance:

1.Kp, Ki, Kd: Proportional, Integral, and Derivative constants.

2.setPoint: The desired angle (ideally 0 degrees for upright position).

3.Defining the motor control pins:

motorLeftPin1, motorLeftPin2, motorRightPin1, motorRightPin2: Pins connected to the motor driver to control the left and right motors.

D). Setup Function:

Initializing the sensors and pins:

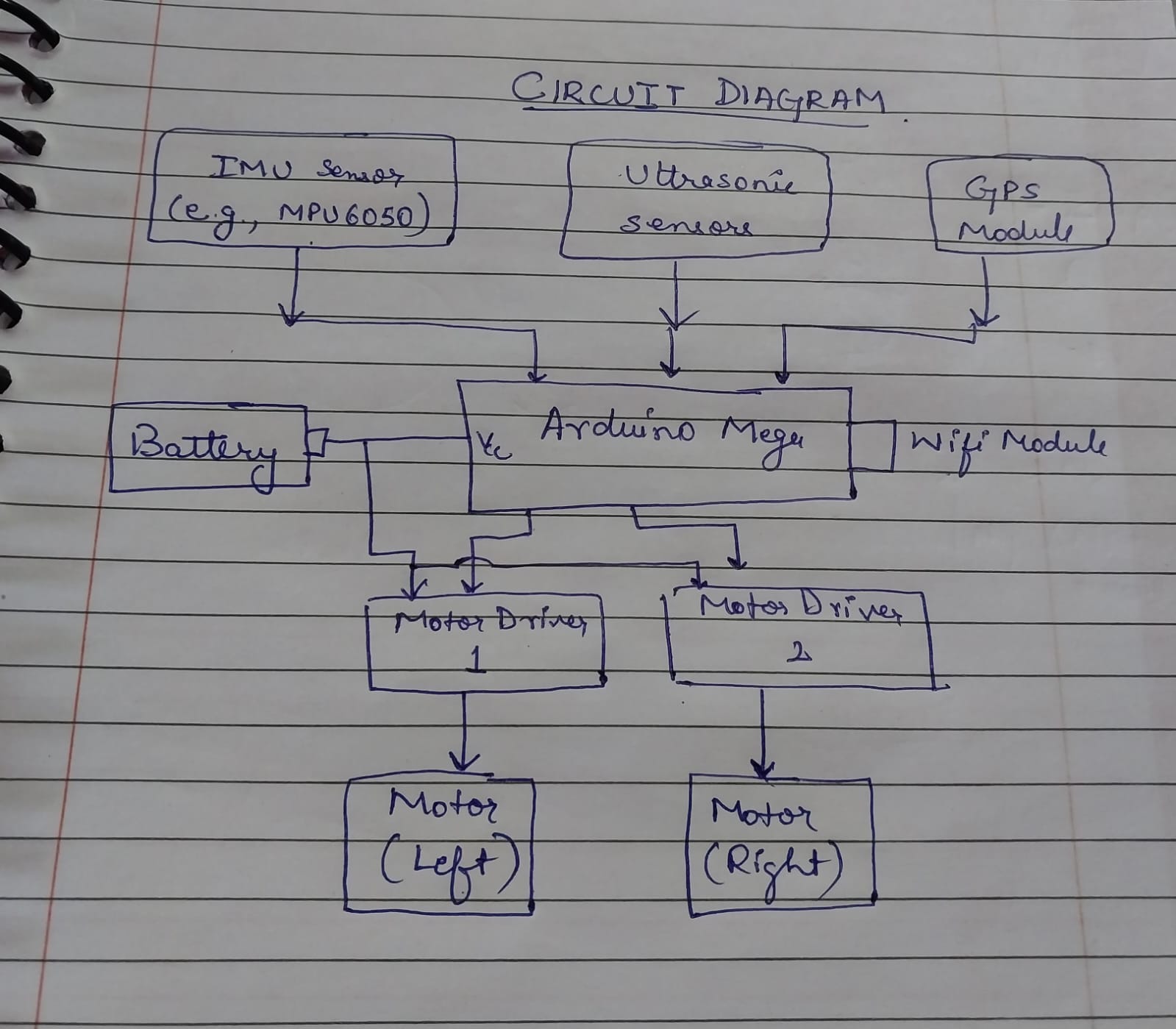
1.Serial.begin(9600);: Initialize the serial communication for debugging and GPS data.

2.Wire.begin();: Start I2C communication.

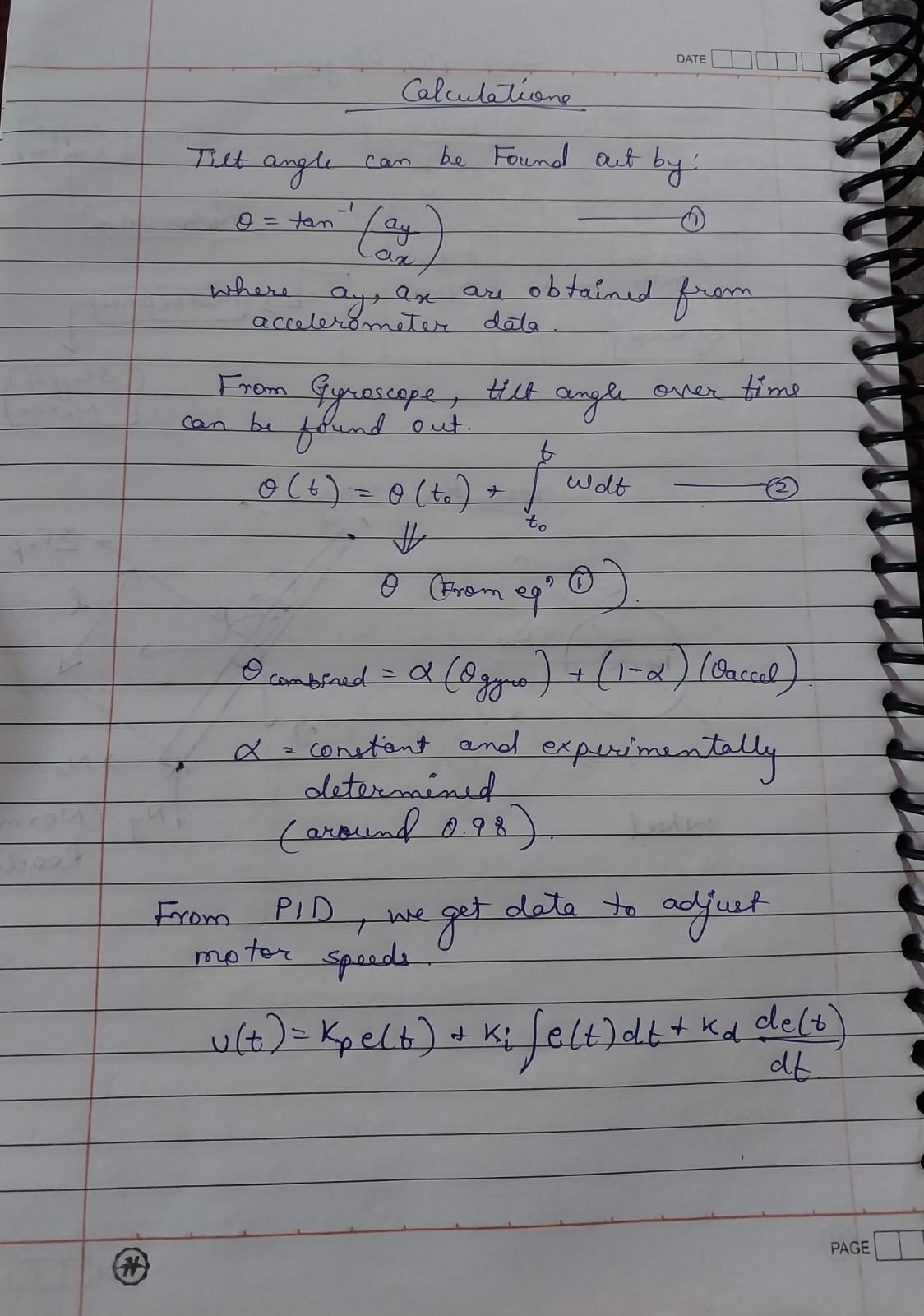
3.mpu.initialize();: Initialize the MPU6050 sensor.

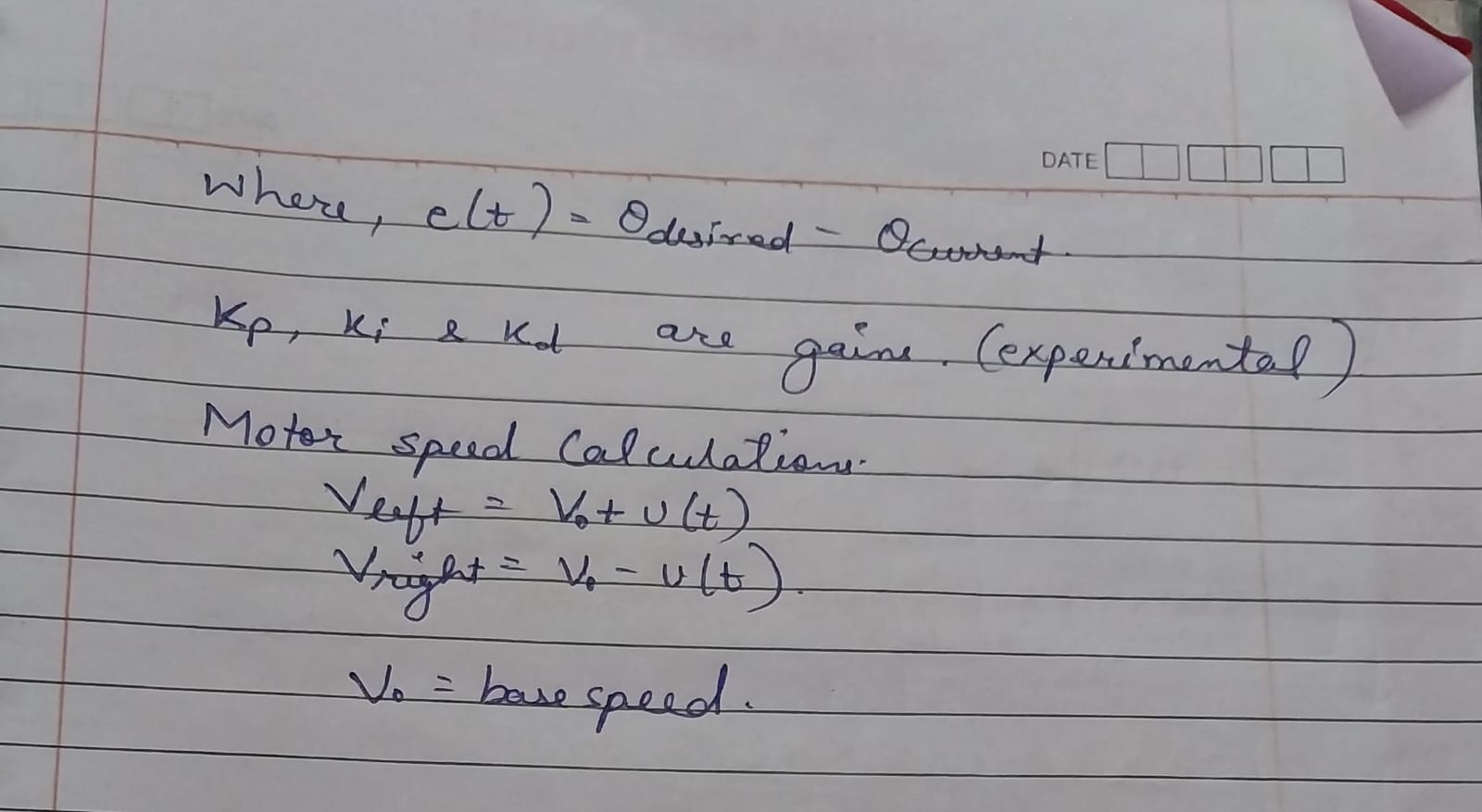
4.gpsSerial.begin(9600);: Start the serial communication for the GPS module.

5.pinMode(motorLeftPin1, OUTPUT); and similar lines: Set motor control pins as outputs.



Calculations:





References:

* ResearchGate
* SpringerLink: <https://link.springer.com/article/10.1007/s40435-021-00832-1>
* Medium: <https://medium.com/geekculture/dynamics-modelling-and-simulation-of-self-balancing-robot-in-c-d32a3b835bbf>
* Slideshare: <https://www.slideshare.net/slideshow/selfbalancing-robot-modeling-and-comparative-analysis-between-pid-and-linear-quadratic-regulator/259882380>
* ArduinoIDE:for compiling