# PROJECT BATTLEBOT DOCUMENTATION

By Abu Hasib Shanewaz

# Contents

# Chapter 1: BattleBot Hardware

This section provides detailed information about the hardware components used for the BattleBot. Each component plays a crucial role in ensuring the BattleBot operates efficiently and meets its designed objectives.

## 1.1 Arduino NANO Microcontroller

The Arduino NANO serves as the brain of the BattleBot. It is a compact, yet powerful microcontroller based on the ATmega328P. It manages the robot's operations, including reading sensor inputs, processing data, and controlling the motors and gripper. The small size of the Arduino NANO makes it ideal for compact robot designs.

## 1.2 PCB with Arduino UNO Footprint for Additional Shields

A custom PCB with an Arduino UNO footprint is used to accommodate additional shields and expand the functionality of the BattleBot. This setup allows for easy integration of various modules and sensors, facilitating a modular and versatile design.

## 1.3 Metal Chassis

The metal chassis provides a sturdy and durable framework for the BattleBot. It houses all the electronic components and ensures the robot can withstand the mechanical stresses encountered during operation. The metal construction also adds weight, improving traction and stability.

## 1.4 Powerbank (10,000 mAh)

A 10,000 mAh powerbank supplies the necessary power to the BattleBot. This high-capacity battery ensures the robot can operate for extended periods without frequent recharging. The powerbank powers both the logic circuits and the motors, providing consistent and reliable energy.

## 1.5 On/Off Switches

Two On/Off switches are used: one for the logic circuit and one for the motor circuit. These switches allow for easy and independent control of the robot's power systems, enabling safe and convenient operation.

## 1.6 Gripper with Servo Motor

The BattleBot is equipped with a gripper controlled by a servo motor. This gripper allows the robot to grasp and manipulate objects. The servo motor provides precise control over the gripper's movements, enabling secure gripping and accurate placement of objects.

## 1.7 Electromotors

The BattleBot is driven by two electromotors, providing the necessary propulsion. These motors are powerful enough to move the robot and carry objects while maintaining control and stability.

## 1.8 Ultrasonic Distance Sensor

An ultrasonic distance sensor is used for object detection. This sensor measures the distance to objects in front of the robot, allowing it to detect obstacles and interact with them using the gripper.

# Chapter:2 Requirement Analysis

## 2.1. Essential Functions

### Autonomy

- After it is begun, the robot must function independently without assistance from a human.

### Line Tracking

- On a white surface, the robot needs to be able to recognize and follow a black line.
- A responsive and accurate line-following system is necessary to guarantee that the robot remains on the line at all times.

### Identifying Objects

- The robot needs to be able to use an ultrasonic sensor to identify objects that are in its path.
- To ensure rapid reaction, objects should be detected within an 8 cm range.

### Gripper Function

- A servo-operated gripper that can identify and grasp objects is a must for the robot.
- When needed, the gripper should be able to open to release things after closing securely on them.

### Mechanism of Stopping

- When the robot gets to a specified spot (such as a black box), it has to stop.
- When stopping, the robot has to release whatever object it is holding.

## 2.2. Non-Operational Conditions

### Achievement

- The robot should react quickly to obstacles and follow the line.
- To grab and release things, the gripper needs to work fast and effectively.

### Dependability

- Over several runs, the robot must reliably follow the line and recognize items.

- Robust hardware and software are necessary to provide uninterrupted operation without errors.

**Utilization**

- With little assistance from the operator, the robot should be simple to start and operate.
- A serial monitor should be used by the system to offer feedback for debugging and status updates.

**Security**

- If any of the sensors identify a black surface, the robot will not be able to continue along its intended path and must stop.
- Safe handling without overheating or overloading should be ensured by the power supply and motor operations.

## 2.3. Hardware Specifications

**Microcontroller**

- Robotics is controlled via an Arduino Nano.

**Sensors**

- Eight line follower sensors are used to find the black line.
- Ultrasonic object detection sensor (HC-SR04).

**Motors**

- The robot is propelled by two DC motors.
- A motor driver is used to manage the motors.

**Gripper**

- Servo motor for driving the gripper mechanism (SG90).

**Power Source**

- A power bank that can power the robot and its parts with a minimum capacity of 10,000 mAh.

**Framework**

- A sturdy metal chassis to safely contain every component.

**Flips**

- The motor circuit control and logic circuit are separated by two On/Off switches.

## 2.4. Necessary Software

**Functions of Sensor Reading**

- Features that allow line sensor analog values to be read and, using preset thresholds, converted to digital values.

**Functions of Motor Control**

- Functions to regulate the motors' direction and speed in response to sensor input.

**Functions of Ultrasonic Sensors**

- Calculates the distance to detected objects and initiates the ultrasonic sensor.

**Gripper Control Features**

- Operates the servo motor gripper in order to grasp and release items.

**Primary Control Loop**

- A loop function that runs the gripper, checks for things, reads sensor values continuously, and controls the motors.

## 2.5. Environmental Conditions

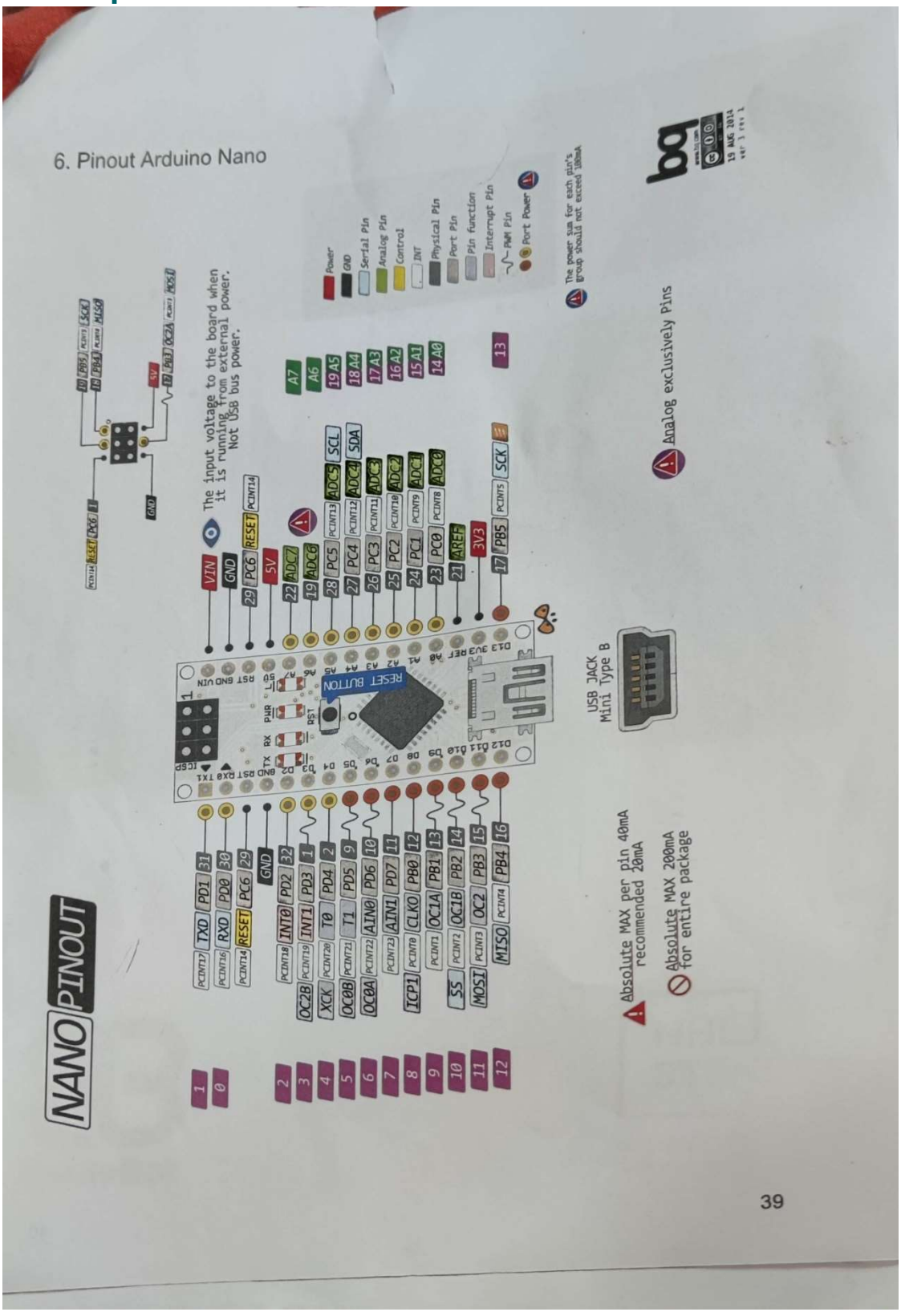
**Surface of Operations**

- A clean, white surface with a distinct black line should be used for the robot's operations.

**Lighting Specifications**

- Sufficient and uniform illumination to guarantee precise sensor measurements.

# Chapter 3: Arduino Pinout



# Chapter 4: Code Elaboration

## Setup ()

The setup() function initializes the system, configures the pin modes, attaches the servo motor to its control pin, and sets the initial position of the servo motor.

```
void setup() {
  Serial.begin(9600);

  // motor pins as output
  pinMode(MOTOR_RIGHT_FORWARD, OUTPUT);
  pinMode(MOTOR_LEFT_FORWARD, OUTPUT);
  pinMode(MOTOR_LEFT_BACKWARD, OUTPUT);
  pinMode(MOTOR_RIGHT_BACKWARD, OUTPUT);

  // Ultrasonic sensor pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  myServo.attach(servoPin);
  myServo.write(130); // Open the gripper initially to 130 degrees
}
```

## Loop()

The loop() function runs continuously, performing the following tasks:

1. Reads sensor values to detect the line position.
2. Checks if all sensors detect a black surface, stopping the motors if this condition persists.
3. If not all sensors detect black, it calculates the line position and calls controlMotors() based on this position.
4. Checks for nearby objects using the ultrasonic sensor to operate the gripper.

```
void loop() {
  int sensorValues[8];
  int digitalValues[8];

  // Read the IR sensors values from A0 to A7
  for (int i = 0; i < 8; i++) {
```

```
  sensorValues[i] = analogRead(A0 + i);
  digitalValues[i] = sensorValues[i] > thresholds[i] ? 1 : 0;
 }

 // Debugging tool: Print the sensor values and digital states to the serial monitor
 for (int i = 0; i < 8; i++) {
  Serial.print("Sensor A");
  Serial.print(i);
  Serial.print(": ");
  Serial.print(sensorValues[i]);
  Serial.print(" (");
  Serial.print(digitalValues[i]);
  Serial.print(") ");
 }
 Serial.println();

 // Determine the line position and control the motors
 controlMotors(digitalValues);

 // Check for objects with ultrasonic sensor
 checkForObject(digitalValues);

 delay(100);
}
```

### readSensors

This function reads the output of the line tracking sensors. It iterates over the array of sensor pins, reads the analog value from each pin, and determines if it exceeds a predefined threshold. Each result is stored in the digitalValues array as true or false, indicating whether each sensor detects the line (black) or not.

```
void readSensors(int* sensorValues, int* digitalValues) {
 for (int i = 0; i < 8; i++) {
  sensorValues[i] = analogRead(A0 + i);
  digitalValues[i] = sensorValues[i] > thresholds[i] ? 1 : 0;
 }
}
```

### controlMotors

This function controls the robot's motors based on the line's position relative to the sensor array. It adjusts the speeds of the motors to turn the robot towards the line. Sharp turns are made when the line is far from the center, and gentler adjustments or straight movements are made when the line is closer to the center.

```
void controlMotors(int* digitalValues) {
  int position = 0;
  int count = 0;
  bool allBlack = true;

  // Calculate the position by summing the weighted sensor values
  for (int i = 0; i < 8; i++) {
    if (digitalValues[i] == 1) {
      position += (i - 3.5) * 2; // Weighted position: -7, -5, -3, -1, 1, 3, 5, 7
      count++;
    } else {
      allBlack = false; // If any sensor does not detect black, set allBlack to false
    }
  }

  if (allBlack || count == 0) {
    // If all sensors detect black or no line is detected, stop the robot and open the gripper
    stop();
    myServo.write(130); // Open the gripper when all sensors detect black
  } else {
    // Calculate the average position
    position = position / count;

    // Proportional control
    int adjustment = position * (MAX_ADJUSTMENT / 7);

    int leftSpeed = BASE_SPEED + adjustment;
    int rightSpeed = BASE_SPEED - adjustment;

    // the speed values to the range [0, 255]
    leftSpeed = constrain(leftSpeed, 0, 255);
    rightSpeed = constrain(rightSpeed, 0, 255);

    moveForward(leftSpeed, rightSpeed);
  }
}
```

### stopMotors

This function stops all motor activity and opens the gripper.

```
void stopMotors() {
  analogWrite(MOTOR_LEFT_FORWARD, 0);
  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_RIGHT_FORWARD, 0);
  analogWrite(MOTOR_RIGHT_BACKWARD, 0);
  myServo.write(130); // Open the gripper
}
```

### calculateDistanceFromObject

This function calculates the distance to an object using an ultrasonic sensor. It sends short pulses to the ultrasonic sensor's trigger pin and measures the duration until the returned echo pulse. This duration is then converted into a distance using the speed of sound and returned.

```
long calculateDistanceFromObject() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in microseconds
  long duration = pulseIn(echoPin, HIGH);

  // Calculating distance
  long distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)

  return distance;
}
```

## checkForObject

This function checks if there is an object within a close distance and operates the gripper based on this. If the distance is 8 cm or less, it closes the gripper.

```
void checkForObject(int* digitalValues) {
  long distance = calculateDistanceFromObject();

  // Debugger: Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Distance 8 cm or less
  if (distance <= 8) {
    Serial.println("Object detected within 8 cm. Closing gripper.");
    closeGripperFaster(); // Close the gripper faster
  }
}
```

## closeGripperFaster

This function closes the gripper at a faster rate by adjusting the servo motor position in small increments.

```
void closeGripperFaster() {
  int currentAngle = myServo.read();
  while (currentAngle > 0) {
    currentAngle -= 4;
    myServo.write(currentAngle);
    delay(5); // Faster closing
  }
}
```

## moveForward

This function sets the speed and direction for the left and right motors to move the BattleBot forward.

```
void moveForward(int leftSpeed, int rightSpeed) {
  analogWrite(MOTOR_LEFT_FORWARD, leftSpeed);
  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_RIGHT_FORWARD, rightSpeed);
  analogWrite(MOTOR_RIGHT_BACKWARD, 0);
}
```