PROJECT BATTLEBOT DOCUMENTATION

By Stefani Margaritova

NHL Stenden
Information Technology

1. Requirement Analysis

1.1 The BattleBot must:

- be autonomous.
- be able to follow a black line and stay on it.
- be able to detect an object in front of it and be able to grasp it.
- be able to stop moving on a black box and if it's holding an object drop it.

1.2 Game Specification

Line Tracking

Within this sub-part of the BattleBot game, the robot must track a black line on a white surface and successfully navigate to the end of the prepared route while staying on the line the whole time.

Object Gripping

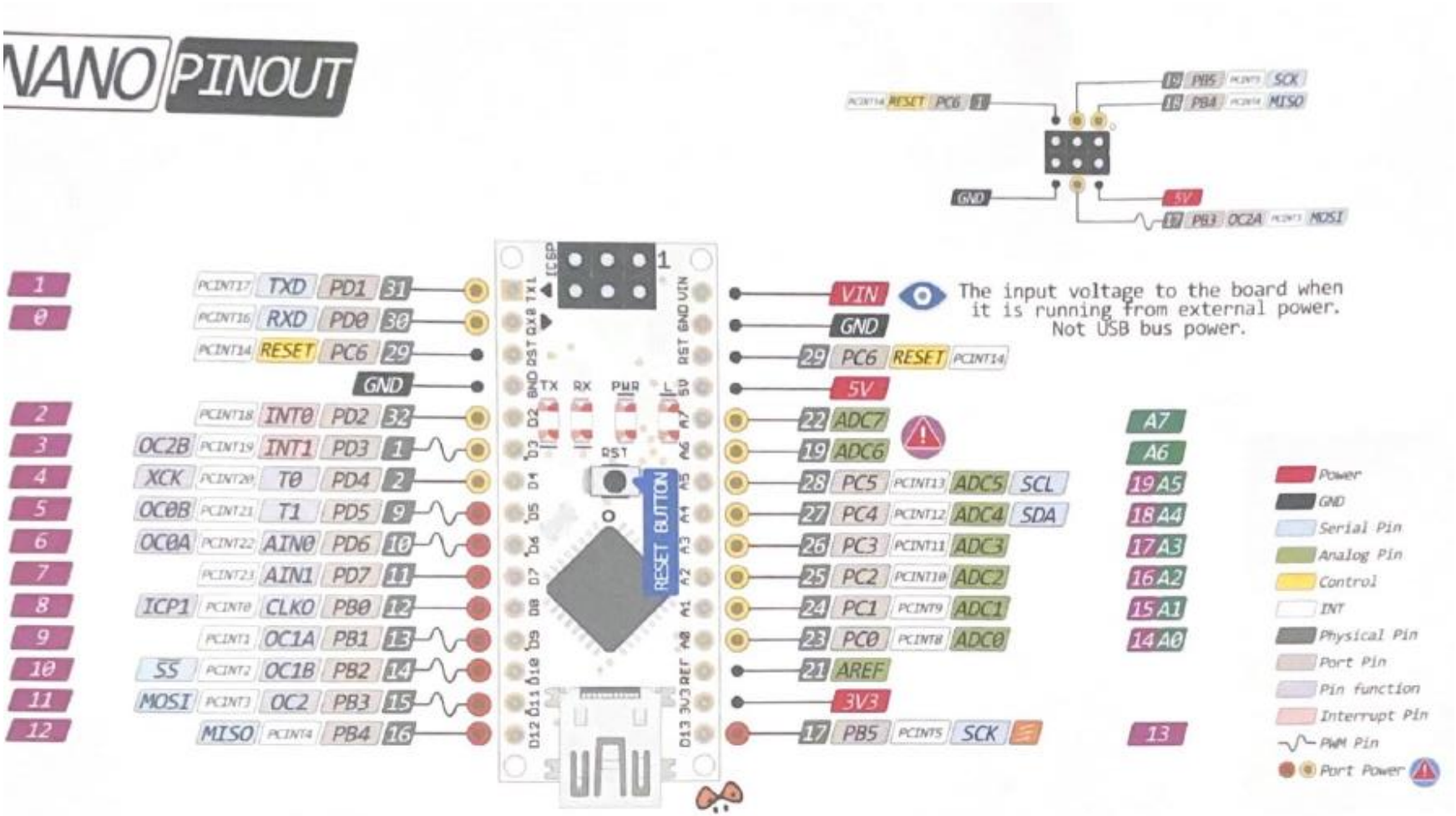
Within this sub-part the BattbleBot must detect if there's an object in front of it and be able to grasp it using the servo gripper. The goal is to bring it successfully to the end of the track designed for Line Tracking and drop the gripped object.

2. Hardware Design

Hardware components used:

- Arduino NANO microcontroller
- PCB with an Arduino UNO footprint
- Metal chassis
- Powerbank (10.000 mAh)
- 2 On/Off switches, for the logic circuit and the motor circuit
- A gripper with a servo motor
- A motor driver for two electromotors
- 2 Electromotors
- Ultrasonic distance sensors
- 8 Analog line follower sensors

Microcontroller Pinout



Code elaboration

Setup():

Initializes the system, configures the pin modes, attaches the servo motor to its control pin, as well as initializes its position.

Functions:

- readSensors
    - This function reads the output of the line tracking sensors. It iterates over the array of sensor pins (SENSOR_PINS), reads the analog value from each pin, and determines if it exceeds a predefined threshold (SENSOR_THRESHOLD). Each result is stored in the sensorValues array as true or false, indicating whether each sensor detects the line (black) or not.
- calculateLinePosition
    - This function calculates the position of the line relative to the line tracking sensors. It computes a weighted average (position) based on which sensors detect the line. Sensors detecting the line contribute their indices, multiplied by 1000, to a sum (position), which is then averaged over the number of sensors detecting the line (numActiveSensors). The function then adjusts this average to a scale centered around zero. It does this by subtracting (NUM_SENSORS - 1) * 500 from the result. This subtraction centers the position such that:
        - If all sensors on one end detect the line, the position skews to either positive or negative.
        - If the line is exactly in the center of the sensor array, the position is zero.
- 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.
- stopMotors
    - This function stops all motor activity and opens 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.
- checkForObject
    - This function checks if there is an object within a close distance and operates the gripper based on this. If the distance is 5 cm or less it closes the gripper.

Loop():

Continuously does the following:

Reads sensor values to detect line position. Checks if all sensors detect a black surface, and handles stopping the motors after a delay if this condition persists (to make sure that the robot is on the black "box" and not on the line still). If not, all sensors detect black, it calculates the line position, and calls the function controlMotors based on this position, then it checks for nearby objects to operate the gripper.

Code:

```
#include <Arduino.h>
#include <Servo.h>

// Motor pins
const int MOTOR_RIGHT_BACKWARD = 4;
const int MOTOR_LEFT_BACKWARD = 7;
const int MOTOR_RIGHT_FORWARD = 5;
const int MOTOR_LEFT_FORWARD = 6;

// Sensors configuration
const int NUM_SENSORS = 8;
const int SENSOR_PINS[NUM_SENSORS] = {A6, A7, A0, A2, A1, A3, A4, A5};
const int SENSOR_THRESHOLD = 576;
const int ECHO_PIN = 12;
const int TRIGGER_PIN = 8;
const int GRIPPER_PIN = 13;

// Miscellaneous variables
Servo servoGripper;
int isInputAllBlackDetectedStart = 0;
const int blackWaitTime = 100;
bool isInputAllBlackEventActive = false;

// Functions
void readSensors(bool sensorValues[]) {
  for (int i = 0; i < NUM_SENSORS; i++) {
    sensorValues[i] = analogRead(SENSOR_PINS[i]) > SENSOR_THRESHOLD;
  }
}

int calculateLinePosition(const bool sensorValues[]) {
  int position = 0;
  int numActiveSensors = 0;

  for (int i = 0; i < NUM_SENSORS; i++) {
    if (sensorValues[i]) {
      position += (i * 1000);
      numActiveSensors++;
    }
  }

  return numActiveSensors > 0 ? position / numActiveSensors - (NUM_SENSORS - 1) * 500 : 0;
}
```

```
void controlMotors(int position) {
  int baseSpeed = 255;

  if (abs(position) > 300) { // For sharp turns
    if (position < 0) {
      analogWrite(MOTOR_RIGHT_FORWARD, baseSpeed);
      analogWrite(MOTOR_LEFT_FORWARD, LOW);
    } else {
      analogWrite(MOTOR_RIGHT_FORWARD, LOW);
      analogWrite(MOTOR_LEFT_FORWARD, baseSpeed);
    }
  } else if (abs(position) > 150) {  // For gentler turns
    analogWrite(MOTOR_RIGHT_FORWARD, baseSpeed - position / 2);
    analogWrite(MOTOR_LEFT_FORWARD, baseSpeed + position / 2);
  } else {  // For a straight path
    analogWrite(MOTOR_RIGHT_FORWARD, baseSpeed);
    analogWrite(MOTOR_LEFT_FORWARD, baseSpeed);
  }
}

void stopMotors() {
  analogWrite(MOTOR_RIGHT_FORWARD, LOW);
  analogWrite(MOTOR_LEFT_FORWARD, LOW);
  analogWrite(MOTOR_LEFT_BACKWARD, LOW);
  analogWrite(MOTOR_RIGHT_BACKWARD, LOW);
  servoGripper.write(180);
}

double calculateDistanceFromObject() {
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  long timeTaken = pulseIn(ECHO_PIN, HIGH);
  return timeTaken * 0.034 / 2;
}

void checkForObject()
{
  if (calculateDistanceFromObject() <= 5)
  {
     servoGripper.write(0);
  }
}
```

```
void setup() {
  Serial.begin(9600);
  pinMode(MOTOR_RIGHT_BACKWARD, OUTPUT);
  pinMode(MOTOR_LEFT_BACKWARD, OUTPUT);
  pinMode(MOTOR_RIGHT_FORWARD, OUTPUT);
  pinMode(MOTOR_LEFT_FORWARD, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(TRIGGER_PIN, OUTPUT);
  servoGripper.attach(GRIPPER_PIN);
  servoGripper.write(180);
}

void loop() {
  bool sensorValues[NUM_SENSORS];
  readSensors(sensorValues);

  // Check if all sensors detect black
  bool isInputAllBlack = true;
  for (int i = 0; i < NUM_SENSORS; i++) {
    if (!sensorValues[i]) {
      isInputAllBlack = false;
      break;
    }
  }

  // Handle an all black response from line sensors
  if (isInputAllBlack) {
    if (!isInputAllBlackEventActive) {
      isInputAllBlackEventActive = true;
      isInputAllBlackDetectedStart = millis();
    } else if (millis() - isInputAllBlackDetectedStart > blackWaitTime) {
      stopMotors();
    }
  } else {
    isInputAllBlackEventActive = false;
    int position = calculateLinePosition(sensorValues);
    controlMotors(position);
    checkForObject();
  }
}
```