Faculty of Computers & Informatics

Benha University

Obstacle-Avoiding Robot

Developing an obstacle-avoiding robot using Arduino Uno and AVR ATmega328P for efficient navigation and collision avoidance.

Project Team Members

1- Ahmed Mohamed Kassem Mohamed
2- Ahmed Mohamed Saeed Mohamed
3- Ibrahim Mohamed Meshref
4- Ahmed Sayed Ahmed Abd elAati
5- Ahmed Adel Hassan Fahim
6- Ahmed Mohamed Abd eltawab
7- Ahmed Taha Ahmed Abd elkreem
8- Ahmed Mohamed Mahmoud Mohamed

Under Supervision of

Dr. Ahmed Shalaby
Eng. Ghada Refaay
Eng. Abdullah El Ghamry

Benha, December 2023

ABSTRACT

The obstacle-avoiding robot project involves creating a smart robotic car that can navigate autonomously while avoiding obstacles. Equipped with ultrasonic sensors to detect obstacles and determine their proximity, the robot's control system processes sensor data in real-time. This allows the robot to make quick decisions, adjusting its direction or speed to steer clear of obstacles. The project includes essential hardware components such as motors, wheels, a chassis, and a **microcontroller** serving as the robot's brain. The distance sensors, usually ultrasonic, emit signals, measuring the time it takes for signals to bounce back from obstacles. The microcontroller interprets this data, directing the motors to change direction accordingly. Effective programming is key to implementing an efficient obstacle avoidance algorithm, allowing the robot to interpret sensor data, make real-time decisions, and execute precise control commands for motor movements.

Table of Contents

a) Defining Project goals and identifying its objectives

The goal of an obstacle-avoiding robot using Arduino is to create a mobile robot that can navigate its environment while avoiding obstacles in its path. The primary objective of this project is to design and build a robot that can autonomously detect obstacles and take appropriate actions to avoid them. The specific objectives of the project can include:

1. **Object detection**: Implementing sensors such as ultrasonic sensors to detect obstacles in the robot's surroundings.
2. **Distance measurement**: Measuring the distance between the robot and obstacles to determine their proximity and potential collision risk.
3. **Collision avoidance**: Developing algorithms or control mechanisms to steer the robot away from obstacles or stop its movement to prevent collisions.
4. **Path planning**: Implementing logic to plan the robot's path, considering the detected obstacles and finding alternative routes to reach its destination.
5. **Robot mobility**: Integrating motors and wheels to provide the robot with mobility and the ability to move in different directions.
6. **Real-time responsiveness**: Ensuring the robot can quickly respond to changes in the environment and adjust its path or speed accordingly.
7. **Power management**: Developing an effective power supply system to sustain the robot's ongoing functionality, emphasizing the use of rechargeable batteries for prolonged operation without the necessity for frequent replacements.
8. **Robustness and reliability**: Building a sturdy robot structure and refining the algorithms to enhance the overall reliability and performance of the obstacle-avoiding capabilities.
9. **Documentation and presentation**: Documenting the project, including circuit diagrams, code explanations, and project reports, and presenting the project findings and outcomes.

These objectives collectively contribute to achieving the project's goal of creating an obstacle-avoiding robot using Arduino that can effectively navigate its environment while avoiding collisions

b) Identifying the input and output and briefly describing their meaning

1. Inputs

Name	Description
Ultrasonic sensor	Measuring the distance between the robot and obstacles.
Motor Control Signals	Receive commands to control the movement and direction of the robot's motors.

2. Outputs

Name	Description
Motor Control	Signals sent to the motors to control the robot's movement and direction.

These inputs and outputs collectively enable the obstacle-avoiding robot to perceive its environment, make decisions, control its motion, and communicate with the user. The inputs, such as ultrasonic and infrared sensors, provide information about the presence and distance of obstacles. The motor control signals, and user commands allow for controlling the robot's movements and behavior.

c) Hardware Components Used

Hardware	Description
Microcontroller	The microcontroller serves as the brain of the robot, controlling its behavior and coordinating different components. Popular microcontrollers used in robotics projects include Arduino boards (such as Arduino Uno, Arduino Mega)
DC Motors	DC motors are commonly used to provide motion to the robot. They convert electrical energy into mechanical energy, allowing the robot to move in different directions.
Motor Driver	A motor driver is an electronic circuit or module that controls the speed and direction of the DC motors. It receives signals from the microcontroller and provides power to the motors
Ultrasonic Sensors	Ultrasonic sensors emit sound waves and measure the time taken for the waves to bounce back after hitting an obstacle. They are commonly used for distance measurement and obstacle detection
Wheels	Wheels provide mobility to the robot, allowing it to move smoothly on the ground. They are typically attached to the DC motors and can vary in size and type depending on the robot's design
Power Supply	The power supply provides electrical energy to the robot's components. It can be a battery pack or an external power source, supplying the required voltage and current for the microcontroller, motors, and other electronic components.
Chassis	The chassis is the physical structure or frame of the robot that holds all the components together. It provides support, protection, and stability to the robot
Cables and Connectors	Various cables and connectors are used to interconnect the components, allowing electrical signals and power to flow between them.

These **hardware components** work together to create an obstacleavoiding robot that can sense its environment, make decisions, and move accordingly. The specific components used may vary based on the project's requirements and the desired functionality of the robot

d) Used Algorithm

1. **Stop the car:** Initially, the function stops the car to prepare for obstacle avoidance.

```
stopCar();
delay(100);
```

2. **Check if obstacles are close on all sides:** If the distances to obstacles on the left, in the middle, and on the right are all below predefined threshold values (DISTANCE_THRESHOLD_FORWARD and DISTANCE_THRESHOLD_SIDE), the robot takes evasive action. It stops, moves backward for a short duration, and then turns to avoid the obstacle.

```
if (middleDist <= DISTANCE_THRESHOLD_FORWARD && rightDist <=
DISTANCE_THRESHOLD_SIDE && leftDist <= DISTANCE_THRESHOLD_SIDE) {
  stopCar();
  delay(150);
  moveBackward();
  delay(70);
}
```

   2.1. **Handle obstacles in the middle:** If an obstacle is detected in the middle (front) of the robot, it stops, moves backward, and then decides whether to turn left or right based on the distance measurement on the right side.

```
else {
  if (middleDist <= DISTANCE_THRESHOLD_FORWARD) {
    stopCar();
    delay(100);
    moveBackward();
    delay(50);
    if (rightDist <= DISTANCE_THRESHOLD_SIDE ) {
      moveBackward();
      delay(20);
      turnLeft();
      delay(750);
    } else {
      turnRight();
      delay(700);
    }
  }
```

3. **No obstacles in the middle:** If there are no obstacles in the middle, the robot resumes moving forward.

```
else {
   stopCar();
   delay(150);
   moveForward();
   delay(100);
}
```

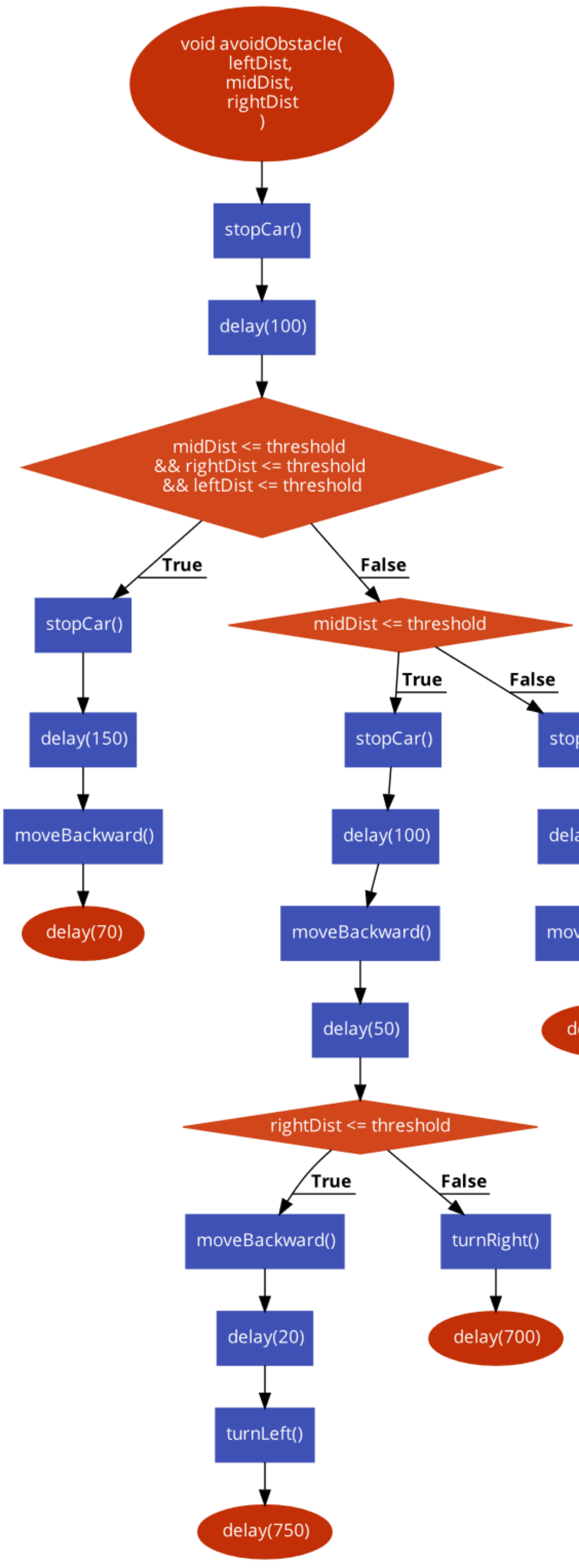
4. **Full Algorithm.**

```
void avoidObstacle(int leftDist, int middleDist, int rightDist) {
  stopCar();
  delay(100);
  if (middleDist <= DISTANCE_THRESHOLD_FORWARD && rightDist <=
DISTANCE_THRESHOLD_SIDE && leftDist <= DISTANCE_THRESHOLD_SIDE) {
    stopCar();
    delay(150);
    moveBackward();
    delay(70);
  }
  else {
    if (middleDist <= DISTANCE_THRESHOLD_FORWARD) {
      stopCar();
      delay(100);
      moveBackward();
      delay(50);
      if (rightDist <= DISTANCE_THRESHOLD_SIDE ) {
        moveBackward();
        delay(20);
        turnLeft();
        delay(750);
      }
       else {
        turnRight();
        delay(700);
      }

    }
    else {
      stopCar();
      delay(150);
      moveForward();
      delay(100);

    }
  }
}
```

e) Flowchart And Required Functions



Constants

ENA, IN1, IN2, IN3, IN4, ENB, Speed: Constants defining motor control pins and speed.

DISTANCE_THRESHOLD_FORWARD, DISTANCE_THRESHOLD_SIDE: Threshold distances for obstacle detection.

trig_forward, echo_forward, trig_right, echo_right, trig_left, echo_left: Pins for ultrasonic sensors.

Required Functions

Motor Control Functions:

1. moveForward()

Description: Moves the robot forward.

Motor Control:

Motor A: Forward

Motor B: Forward

Speed: Defined by Speed constant.

Serial Output: "ROBOT_MOVING_FORWARD"

2. moveBackward()

Description: Moves the robot backward**.**

Motor Control:

Motor A: Backward

Motor B: Backward

Speed: Defined by Speed constant.

Serial Output: "ROBOT_MOVING_BACKWARD"

3. turnRight()

Description: Turns the robot to the right.

Motor Control:

Motor A: Forward

Motor B: Stops

Speed: Defined by Speed constant**.**

Serial Output: "ROBOT_MOVING_RIGHT"

4. **turnLeft()**

Description: Turns the robot to the left.

Motor Control:

Motor A: Stops

Motor B: Forward

Speed: Defined by Speed constant.

Serial Output: "ROBOT_MOVING_LEFT"

5. **stopCar()**

Description: Stops the robot.

Motor Control:

Motor A: Stops

Motor B: Stops

Speed: 0

Serial Output: "ROBOT_STOP"

Ultrasonic Distance Measurement:

measureDistance(int trigPin, int echoPin)

Description: Measures the distance using an ultrasonic sensor.

Parameters:

trigPin: Ultrasonic sensor trigger pin.

echoPin: Ultrasonic sensor echo pin.

Returns: Calculated distance in centimeters.

Obstacle Avoidance Algorithm:

avoidObstacle(int leftDist, int middleDist, int rightDist)

Description: Implements an obstacle avoidance algorithm based on ultrasonic sensor readings.

Parameters:

leftDist: Distance from the left ultrasonic sensor.

middleDist: Distance from the front (middle) ultrasonic sensor.

rightDist: Distance from the right ultrasonic sensor.

Setup and Loop:

1. **setup()**

Description: Arduino setup function.
Configures pins, initializes serial communication.

2. **loop()**

Description: Arduino main loop function.
Repeatedly measures distances and performs obstacle avoidance.
These functions collectively control the behavior of the obstacle-avoiding robot, responding to ultrasonic sensor readings to navigate and avoid obstacles in real-time.

f) Ultrasonic code in C language for the AVR microcontroller

```c
/*
 * ultrasonic.c
 *
 * Created: 12/25/2023 6:55:06 AM
 * Author : Ahmed Sanad
 */
#define F_CPU 16000000 // 16 MHz Clock
#define trig_pin 4 // PB4 (Digital pin 12)
#define echo_pin 5 // PB5 (Digital pin 13)

#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>

float Distance;

float MeasureDistance();

int main(void)
{
    DDRB |= (1<<trig_pin); // setting trig as a output
    DDRB &= ~(1<<echo_pin); //setting echo as a input
    while (1)
    {
              Distance = MeasureDistance();
    }
}

float MeasureDistance(){
       PORTB &= ~(1<<trig_pin); // trigger = low
       _delay_us(2); // ensure that trig is low for 2 microseconds
       PORTB |= (1<<trig_pin); // trigger = high
       _delay_us(10); // sending signals to objects
       PORTB &= ~(1<<trig_pin); // trigger = low

       while(!(PINB & (1<<echo_pin))) // while waiting echo to go high
       TCNT1 = 0; // starts timer (TCCR1B 16-bit Counter/Timer)
       // we can use CS10 without dividing by 8
       TCCR1B |= (1<<CS11) /8; // generating 8 pulses from trigger in 10us and set prescaler
to 1

       while(PINB & (1<<echo_pin)) // while wating echo to go low
       TCCR1B = 0; // stop timer

       //calculate distance
       float dist = TCNT1*(1/29.1) /2;

       return dist;
}
```

g) Test strategy to verify the operation of project

Test Strategy Overview: The test strategy for the obstacle-avoiding robot project aims to verify the correct functioning of individual components and the overall system. This includes testing ultrasonic sensors, motor movements, obstacle avoidance algorithms, hardware integration, power management, and real-time decision-making.

Test Cases:

1. **Ultrasonic Sensor Testing:**
   - Verify accurate distance measurements.
   - Confirm sensor responsiveness to obstacles.
2. Motor Movement Testing:
   - Ensure correct response to movement commands.
   - Verify the intended direction and speed of wheel movement.
3. Obstacle Avoidance Algorithm:
   - Test the robot in environments with various obstacles.
   - Confirm successful obstacle avoidance.
4. Hardware Integration Testing:
   - Check proper connection of motors, wheels, chassis, and sensors.
   - Verify absence of loose connections or faulty wiring.
5. Power Management Testing:
   - Verify efficiency of the power supply system.
   - Test continuous operation without frequent battery replacements.
6. Boundary Testing:
   - Test robot performance at different distances from obstacles.
   - Evaluate behavior in extreme cases.
7. Real-Time Decision Testing:
   - Simulate real-time scenarios with obstacles at different intervals.
   - Confirm appropriate and timely decision-making.
8. Programming Logic Testing:
   - Inspect code logic for sensor data interpretation.
   - Test decision-making in various sensor input scenarios.
9. Environment Testing:
   - Test robot in different environmental conditions.
   - Evaluate performance under varying lighting and surface textures.
10. Durability Testing:
    - Conduct endurance tests over an extended period.
    - Evaluate wear and tear on mechanical components.
11. Documentation Review:
    - Cross-verify project documentation with the implemented functions.
    - Ensure alignment between documented and observed behavior.
12. Edge Case Testing:
    - Test robot in challenging scenarios (tight spaces, sharp turns).
    - Evaluate performance in complex obstacle configurations.

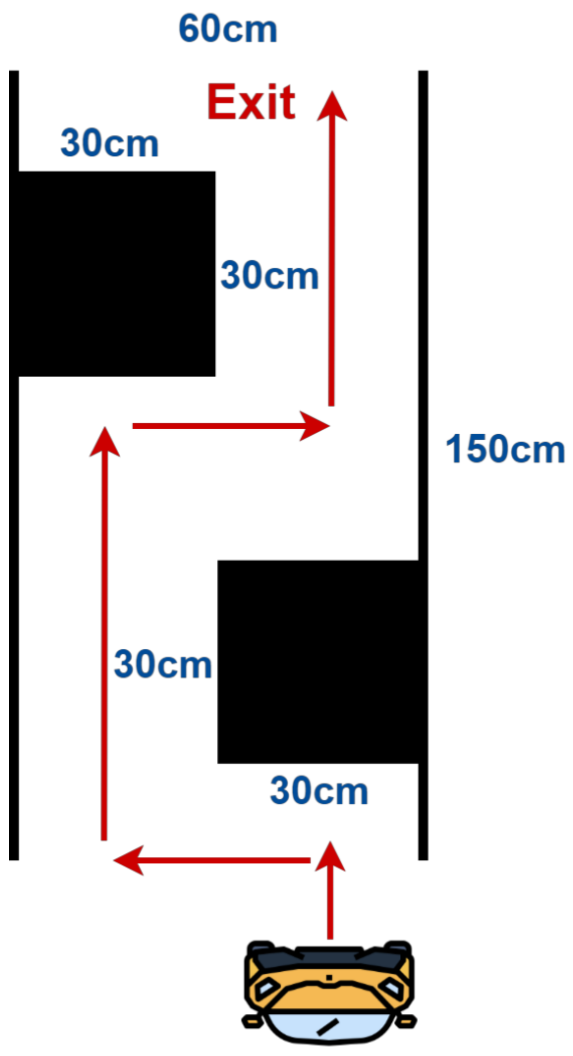
13. Emergency Stop Testing:

- Verify functionality of emergency stop mechanism.

14. Communication (if applicable):

- Test reliability and accuracy of communication protocol (if applicable).

Conclusion:
This overall test strategy aims to ensure the obstacle-avoiding robot functions reliably and safely across various scenarios and conditions.


60cm
Exit
30cm
30cm
150cm
30cm
30cm