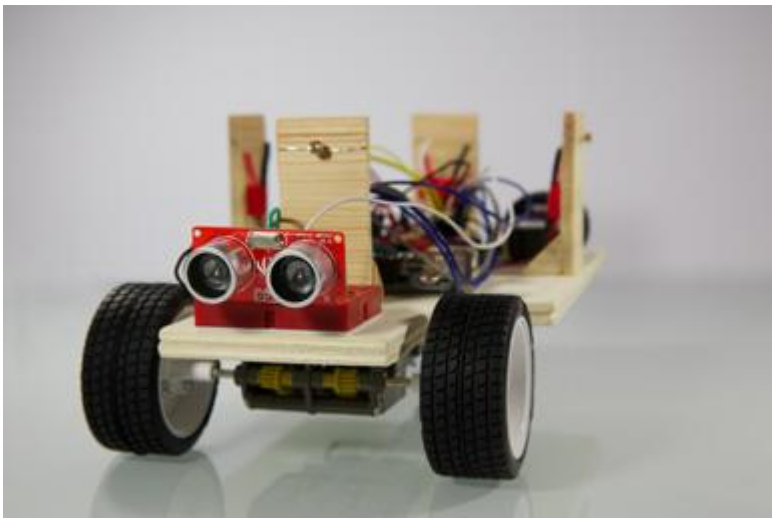
VP Industrial & Scientific Robotics, Winter 2013

Light-Bot

Final Report



Team: Thomas Herzog, Nicole Kowatsch

Betreuung: Dr. techn. Rainer Trummer

Inhalt

Einführung

Light-Bot ist ein mobiler Roboter, der in seiner Umgebung die hellste Lichtquelle erkennt und darauf zufährt. Dabei versucht er, gegebenenfalls auftretende Hindernisse zu umfahren.

Komponenten

- Arduino Uno R3[1]
  - Integrierter Mikrocontroller
  - Programmierung in einer auf C++ basierenden Sprache
  - Verbindung zu PC über USB
  - Stromversorgung über USB oder direkt an Pins
  - 14 digitale I/O Pins (6 davon als PWM Output nutzbar)
  - 6 analoge I/O Pins
- Platform
  - Aus Holz gebaut
  - Grundplatte und vier Erhöhungen für Licht-Widerstände
- 2x Rad
- Stützrad (ball caster)
  - Als dritter Kontaktpunkt, für stabilen Stand
- Dual Motor Getriebebox[2]
- 2x DC Motor
  - Zum Antrieb der Räder
- Dual Motor Driver[3]
  - Zur Ansteuerung der Motoren
- Ultraschall Entfernungssensor[4]
  - Zum Erkennen von frontalen Hindernissen
- 4x Fotowiderstand
  - Zum Erkennen von Lichtquellen (Stärke und Richtung)
- 2x Batterie
  - Getrennte Stromversorgung von Arduino und Motoren

[1] http://arduino.cc/de/Main/ArduinoBoardUno
[2] http://www.pololu.com/product/61
[3] https://www.sparkfun.com/products/9457
[4] http://www.seeedstudio.com/depot/Ultra-Sonic-range-measurement-module-p-626.html

Planung

Wie in Abbildung 1 zu sehen, war ursprünglich geplant, in etwa gleich viel Zeit für den Hardware-Teil (Beschaffung der Bauteile, Zusammenbau) wie für den Software-Teil (Programmierung, Testung) aufzuwenden. Im Endeffekt war der Hardware-Teil schneller abgeschlossen als geplant, der Software-Teil stellte sich dagegen als aufwändiger heraus.

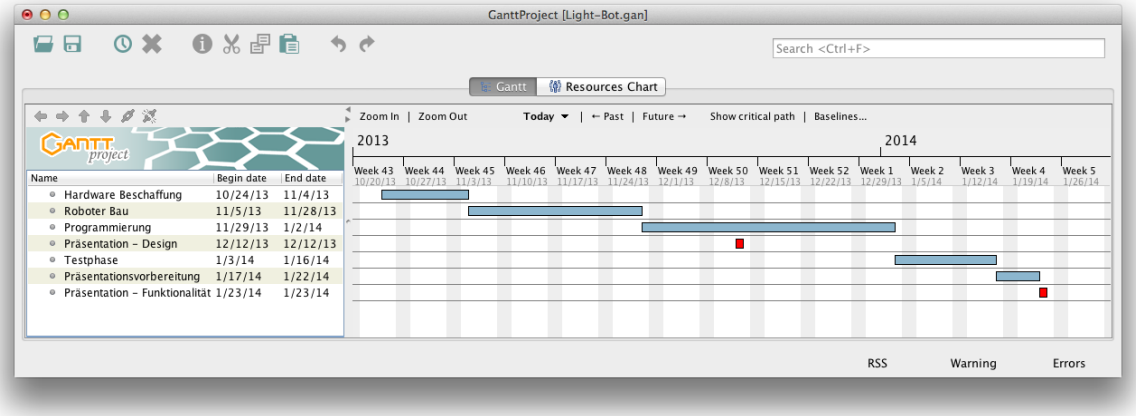


Abbildung 1

Abbildung 2 zeigt einen frühen Entwurf des Roboters. Die Grundform (zwei Räder frontal, Arduino und sonstige Elektronik in der Mitte, umgeben von erhöhten Lichtwiderständen, vorne der Ultraschall-Sensor) wurde bis zur Fertigstellung eingehalten.

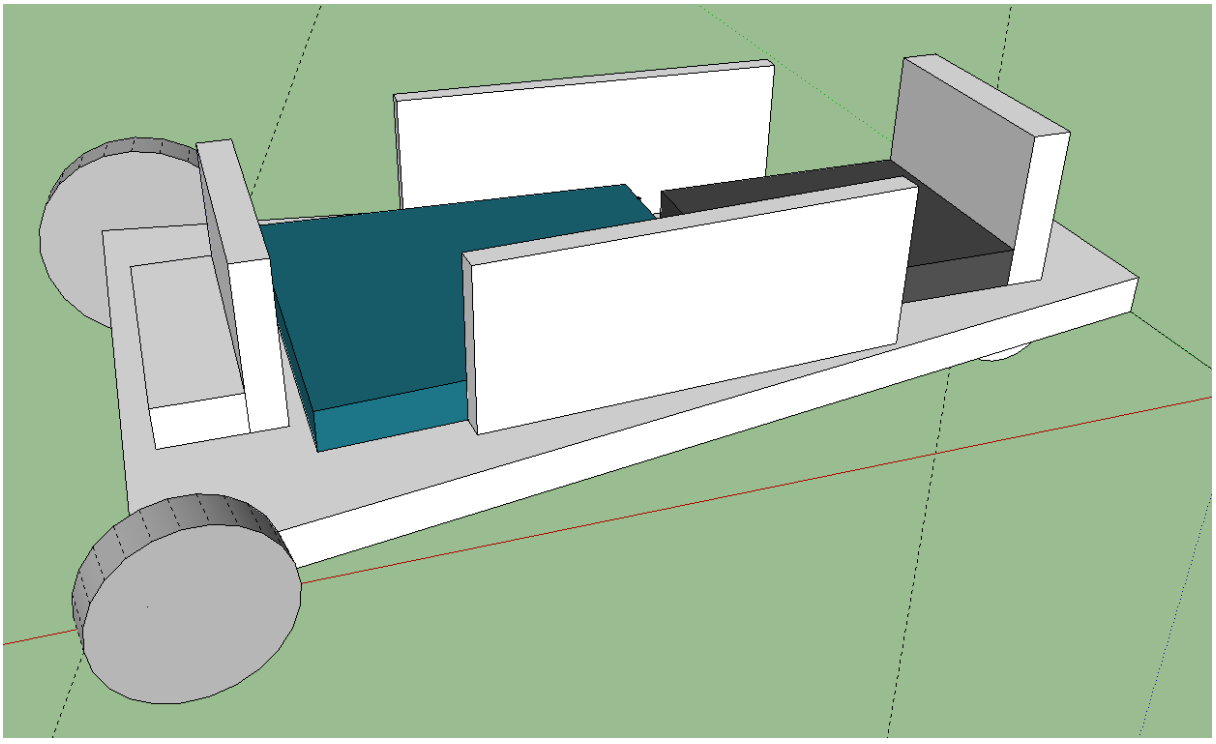


Abbildung 2

Schaltplan

Der Schaltplan (Abbildung 3) wurde zuerst am PC entworfen, bevor er physisch umgesetzt wurde. Dazu wurde die Software Fritzing[5] verwendet. Zusätzlich zum Arduino Uno wurden noch zwei kleine Steckplatinen verwendet, um den Motor Driver und die benötigten Widerstände unterzubringen.

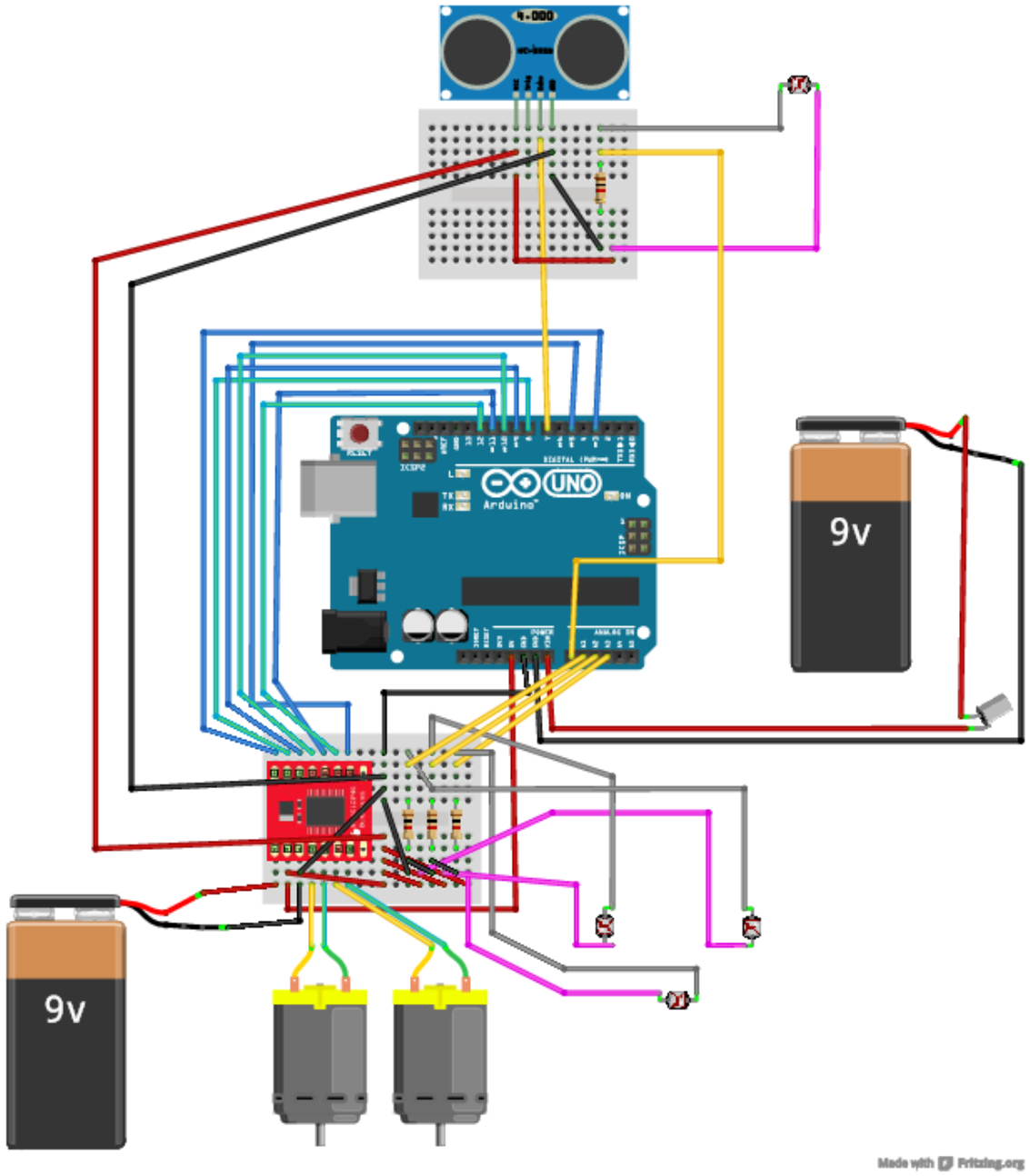


Abbildung 3

[5] http://fritzing.org/download/

Konstruktion

Die Konstruktion des Roboters erfolgte nach dem geplanten Design (Abbildung 2), mit kleinen Modifikationen (schmalere Frontseite für die Räder, frontale Abschirmung der seitlichen Lichtwiderstände). Der Schaltplan (Abbildung 3) wurde 1:1 umgesetzt. Der fertige Roboter ist in Abbildung 4 zu sehen.

Unerwartete Probleme bereitete die Getriebebox, die nicht vorhersah, die beiden Räder unabhängig ansteuern zu können. Dieses Problem konnte gelöst werden, indem die verbindende Metallachse durchgeschnitten wurde.

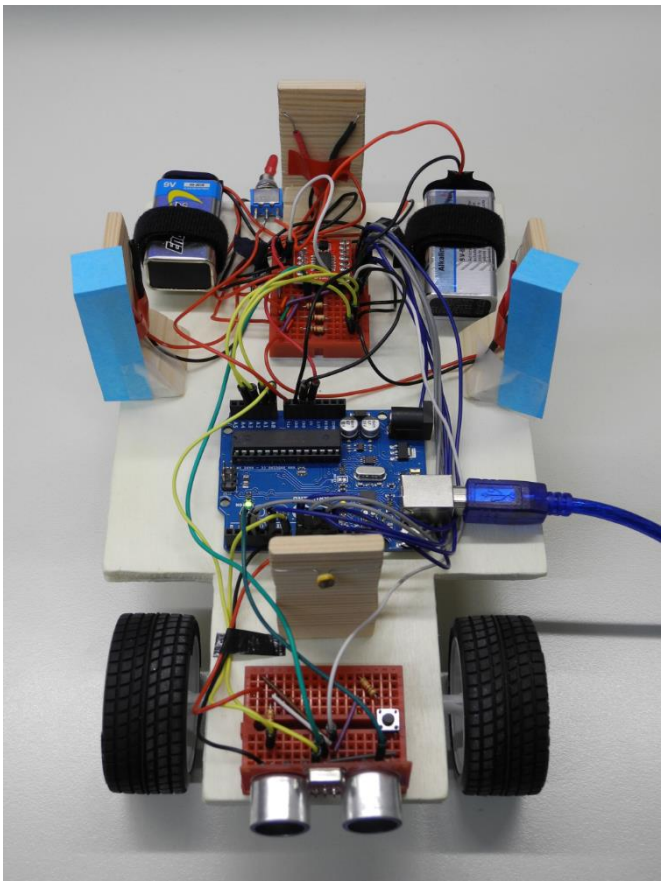


Abbildung 4

Logik

Vor Beginn der Implementierung wurde ein Zustandsdiagramm (Abbildung 5) entworfen, um alle möglichen Zustände des Roboters abzudecken.

Der Ausgangszustand ist „Pause“. In diesen Zustand kann man aus jedem beliebig anderen gelangen, indem man den am vorderen Steckbrett befindlichen Tastschalter betätigt. Dieser muss ebenfalls betätigt werden, um aus dem „Pause“-Zustand in den „Light Amplitude“-Zustand zu gelangen. Darin wird überprüft, ob ein Lichtvektor (in irgendeiner Richtung) von ausreichender Stärke vorhanden ist. Wenn ja, wird in „Light Direction“ die Richtung dieses Vektors bestimmt, und in „Light Turn“ sich solange in Richtung Licht gedreht, bis es sich vor dem Roboter befindet. In „Drive Forward: Start“ werden die Motoren für eine Vorwärtsfahrt gestartet. In „Drive Forward“ wird ständig überprüft:

- Ist das Licht noch erkennbar (stark genug)? Sonst: „Light Amplitude“
- Ist das Licht noch vor uns (± einige Grad Toleranz)? Sonst: „Light Direction“
- Ist vor uns (Abstand <= Toleranz) ein Hindernis? Dann: „Avoid Obstacle“

In „Avoid Obstacle“ wird geprüft, ob sich das Licht links oder rechts befindet. Dahin dreht sich der Roboter dann in „Avoid Obstacle: Turning“ für eine vordefinierte Zeitdauer. Wenn danach kein Hindernis mehr zu erkennen ist, wird in „Avoid Obstacle: Drive Ahead“ eine vordefinierte Zeitdauer lang geradeaus gefahren, um hoffentlich am Hindernis vorbei zu fahren. Danach wird wieder in „Light Amplitude“ übergegangen. Falls nach der ersten Ausweich-Drehung immer noch ein Hindernis zu erkennen ist, dreht sich der Roboter (2x so lange als bei der ersten Drehung) zurück. Falls der Weg diesmal frei ist, „Avoid Obstacle: Drive Ahead“ und „Light Amplitude“. Ansonsten geht der Roboter davon aus, in einer Sackgasse zu stecken, und warten im Zustand „Wait for Light from Behind“ darauf, dass die Lichtquelle hinter ihm erscheint, und geht dann wieder in „Light Amplitude“ über.

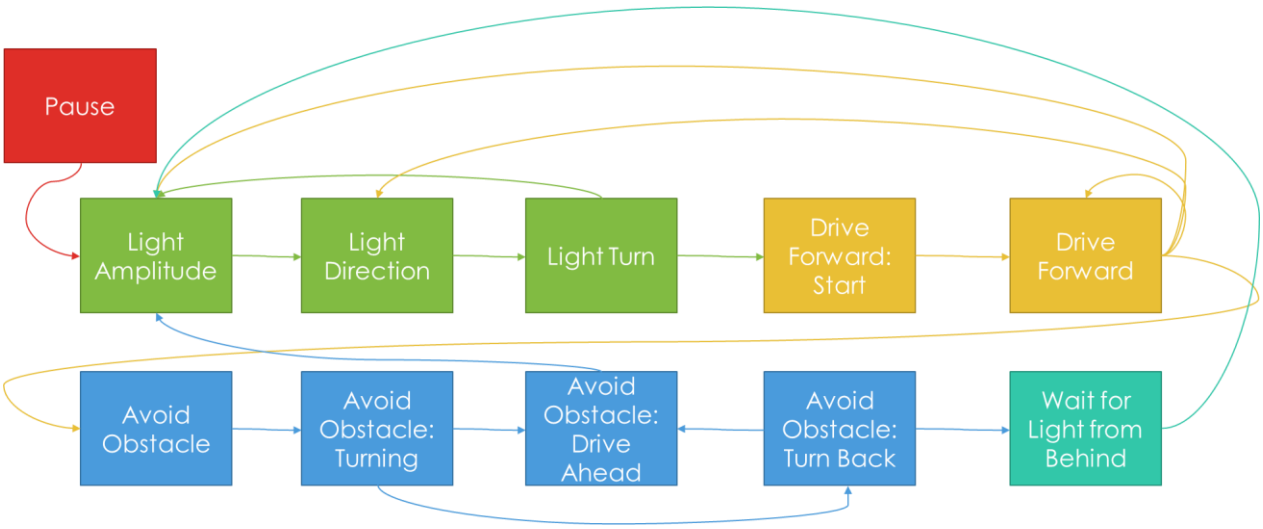


Abbildung 5

Software

Visualisierung

Um zu testen, ob der Aufbau der Elektronik fehlerfrei ist, und welche Daten die Sensoren liefern, wurde zuerst (mit Processing[6]) ein kleines Programm zur Visualisierung des Sensor-Inputs erstellt (lights.pde).

Wie in Abbildung 6 zu sehen, werden per USB-Verbindung zum Arduino Uno die Sensor-Werte ausgelesen (4x Lichtwiderstand, 1x Entfernung) und ausgegeben. Zusätzlich wird aus den Rohdaten ein Richtungsvektor zur hellsten Lichtquelle berechnet, und visualisiert.

Dieses Setup ermöglichte es, relativ schnell die Grundidee der Lichtlokalisation und den korrekten Aufbau zu verifizieren.

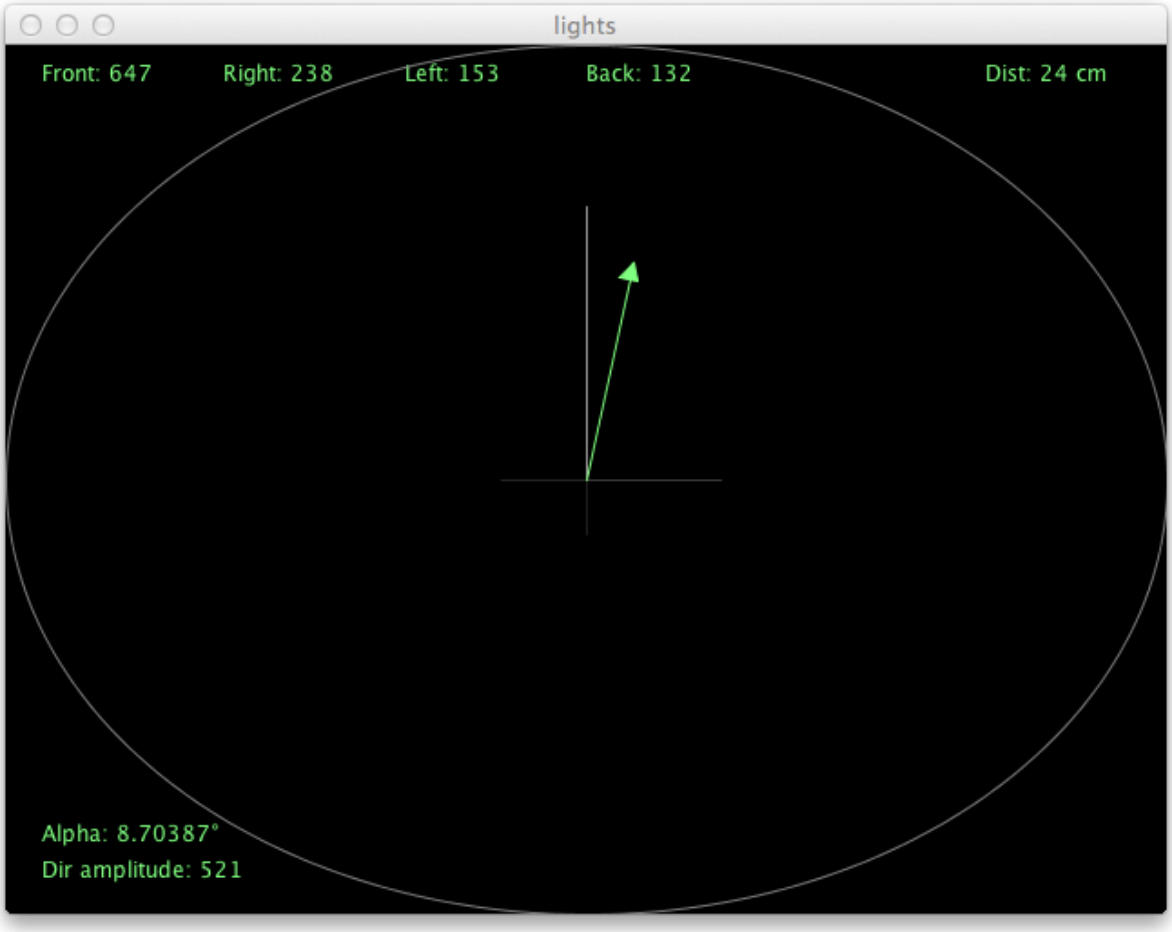


Abbildung 6

[6] http://processing.org/

Ultraschall-Modul

Zur erhöhten Übersichtlichkeit und Wartbarkeit wurde die Ultraschall-Entfernungssensor-Logik in ein eigenständiges Modul gekapselt (Ultrasonic.h und Ultrasonic.cpp). Dieses Modul definiert eine Klasse `Ultrasonic`, der im Konstruktor die Pin-Nummer des Steuer-Pins des Sensors übergeben wird.
Bei Aufruf der Methode `measure_distance()` wird mit dem Sensor kommuniziert, und das Ergebnis von Mikrosekunden in Zentimeter umgerechnet, bevor dieser Wert zurückgegeben wird.

Hauptprogramm

Die Dauerschleife `loop()` des Programms ultrasonic.ino besteht aus zwei Hauptkomponenten:

- Überprüfung des Tastschalters
  - Zum Wechseln in den bzw. aus dem „Pause"-Zustand
- Zustands-Maschine
  - Je nach aktuellem Zustand wird die zugehörige Funktion aufgerufen

Jede der Zustands-Funktionen ist selber dafür verantwortlich, ein sinnvolles Delay zu setzen. So wird z.B. im Zustand „Light Amplitude" ein weit höheres Delay verwendet als in „Light Turn".

Zur Kontrolle des Motors wurden zwei Hilfsfunktionen geschrieben, um einerseits den Roboter einfach anhalten zu können (`motor_stop()`) bzw. mit `motor_move()` komfortabel über drei Parameter (welcher Motor, welche Richtung, welche Geschwindigkeit) die Motoren steuern zu können. Damit wird die Ansteuerungs-Logik des Motor Drivers gekapselt.

Programmcode

lights.pde

```
import processing.serial.*;

Serial port;
String val;
int lf = 10;      // ASCII linefeed

int front = 0;
int right = 0;
int left = 0;
int back = 0;
int dist = 0;

int WIDTH = 640;
int HEIGHT = 480;

void setup() {
  size(WIDTH, HEIGHT);
  println(Serial.list());
  String portName = Serial.list()[4];
  port = new Serial(this, portName, 9600);
  port.bufferUntil(lf);
}

void draw()
{
  background(0);

  text("Front: " + front, 20, 20);
  text("Right: " + right, 120, 20);
  text("Left: " + left, 220, 20);
  text("Back: " + back, 320, 20);
  text("Dist: " + dist + " cm", WIDTH - 100, 20);

  int MID_X = WIDTH / 2;
  int MID_Y = HEIGHT / 2;

  stroke(128);
  noFill();
  ellipse(MID_X, MID_Y, 2 * MID_X, 2 * MID_Y);

  stroke(256 * front / 1024);
  line(MID_X, MID_Y, MID_X, MID_Y - (MID_Y * front / 1024));
  stroke(256 * right / 1024);
  line(MID_X, MID_Y, MID_X + (MID_X * right / 1024), MID_Y);
  stroke(256 * left / 1024);
  line(MID_X, MID_Y, MID_X - (MID_X * left / 1024), MID_Y);
  stroke(256 * back / 1024);
  line(MID_X, MID_Y, MID_X, MID_Y + (MID_Y * back / 1024));

  int dir_x = right - left;
  int dir_y = back - front;

  int dir_amp = (int) sqrt(dir_x * dir_x + dir_y * dir_y);
  text("Dir amplitude: " + dir_amp, 20, HEIGHT - 20);

  float alpha = acos((float)-dir_y / dir_amp) * 180.0 / PI;
  text("Alpha: " + alpha + "°", 20, HEIGHT - 40);
```

```
  int x = MID_X + (MID_X * dir_x / 1024);
  int y = MID_Y + (MID_Y * dir_y / 1024);
  stroke(127, 255, 127);
  fill(127, 255, 127);
  line(MID_X, MID_Y, x, y);
  pushMatrix();
    translate(x, y);
    rotate(atan2(y-MID_Y, x-MID_X));
    triangle(0, 0, -10, 5, -10, -5);
  popMatrix();
}

void serialEvent(Serial p) {
  val = p.readString();
  if (val != null) {
    String[] m = match(val, "FRONT: (.*?);");
    if (m != null && m.length > 1) {
      front = Integer.parseInt(m[1]);
    }

    m = match(val, "RIGHT: (.*?);");
    if (m != null && m.length > 1) {
      right = Integer.parseInt(m[1]);
    }

    m = match(val, "LEFT: (.*?);");
    if (m != null && m.length > 1) {
      left = Integer.parseInt(m[1]);
    }

    m = match(val, "BACK: (.*?);");
    if (m != null && m.length > 1) {
      back = Integer.parseInt(m[1]);
    }

    m = match(val, "DIST: (.*?);");
    if (m != null && m.length > 1) {
      dist = Integer.parseInt(m[1]);
    }
  }
}
```

Ultrasonic.h

```
#ifndef __ULTRASONIC_H__
#define __ULTRASONIC_H__

#include <inttypes.h>

class Ultrasonic {
public:
  Ultrasonic(const uint8_t pin);
  const unsigned long measure_distance();
  void output_distance(const unsigned long duration);
private:
  const unsigned long microseconds_to_cm(const unsigned long microseconds);

  uint8_t _pin;
};

#endif
```

Ultrasonic.cpp

```
#include <ctype.h>

#if defined(ARDUINO) && ARDUINO >= 100
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

#include "Ultrasonic.h"

const float MICROSECONDS_PER_CM = 29.155;

Ultrasonic::Ultrasonic(const uint8_t pin) {
  _pin = pin;
}

const unsigned long Ultrasonic::measure_distance() {
  pinMode(_pin, OUTPUT);
  digitalWrite(_pin, LOW);
  delayMicroseconds(2);

  digitalWrite(_pin, HIGH);
  delayMicroseconds(5);
  digitalWrite(_pin, LOW);

  pinMode(_pin, INPUT);
  return microseconds_to_cm(pulseIn(_pin, HIGH));
}

const unsigned long Ultrasonic::microseconds_to_cm(const unsigned long
microseconds) {
  return microseconds / MICROSECONDS_PER_CM / 2;
}

void Ultrasonic::output_distance(const unsigned long duration) {
  Serial.print("Distance to nearest object: ");
  Serial.print(duration);
  Serial.println(" cm");
}
```

ultrasonic.ino

```
#include "Ultrasonic.h"
#include <Bounce.h>

// Light sensors
const unsigned int LIGHT_FRONT_PIN = A0;
const unsigned int LIGHT_RIGHT_PIN = A1;
const unsigned int LIGHT_LEFT_PIN = A2;
const unsigned int LIGHT_BACK_PIN = A3;

// Ultrasonic sensor
const unsigned int PING_SENSOR_IO_PIN = 7;
Ultrasonic ultrasonic(PING_SENSOR_IO_PIN);

// Motor Pins
const unsigned int MOTOR_STBY_PIN = 10;

const unsigned int MOTOR_PWMA_PIN = 3;
const unsigned int MOTOR_AIN1_PIN = 9;
const unsigned int MOTOR_AIN2_PIN = 8;

const unsigned int MOTOR_PWMB_PIN = 5;
const unsigned int MOTOR_BIN1_PIN = 11;
const unsigned int MOTOR_BIN2_PIN = 12;

// Motor constants
const unsigned int MOTOR_LEFT = 0;
const unsigned int MOTOR_RIGHT = 1;
const unsigned int MOTOR_FWD = 2;
const unsigned int MOTOR_BACK = 3;

const unsigned int MOTOR_SPEED = 31;
const unsigned int MOTOR_TURN_SPEED = 31;

const unsigned int MOTOR_90DEGREE_TURN_TIME_MS = 3000;
const unsigned int MOTOR_AVOID_OBSTACLE_DRIVE_TIME_MS = 2000;

// Button
const unsigned int BUTTON_PIN = 2;
Bounce button = Bounce(BUTTON_PIN, 5);

// Serial baud rate
const unsigned int BAUD_RATE = 9600;

// Tolerances etc,
const unsigned int MIN_LIGHT_AMP = 50;
const unsigned int MAX_LIGHT_ANGLE_DEVIATION = 30;
const unsigned int MAX_LIGHT_ANGLE_BACK_DEVIATION = 25;
const unsigned int MIN_OBSTACLE_DISTANCE = 25;

// Turn direction (for obstacle avoiding)
bool turnedLeft = false;

// Start turn time (for obstacle avoiding)
unsigned long startTurnTime = 0;

// Start drive ahead time (for obstacle avoiding)
unsigned long startDriveTime = 0;

// States
```

```
const unsigned int STATE_PAUSE = 0;
const unsigned int STATE_LIGHT_AMPLITUDE = 1;
const unsigned int STATE_LIGHT_DIRECTION = 2;
const unsigned int STATE_LIGHT_TURN = 3;
const unsigned int STATE_DRIVE_FWD_START = 4;
const unsigned int STATE_DRIVE_FWD = 5;
const unsigned int STATE_AVOID_OBSTACLE = 6;
const unsigned int STATE_AVOID_OBSTACLE_TURNING = 7;
const unsigned int STATE_AVOID_OBSTACLE_DRIVE_AHEAD = 8;
const unsigned int STATE_AVOID_OBSTACLE_TURNING_BACK = 9;
const unsigned int STATE_WAIT_FOR_LIGHT_FROM_BEHIND = 10;

unsigned int state = STATE_PAUSE;

// Standby states
boolean turn_left = false;
boolean turn_right = false;
boolean move_fwd = false;
boolean move_back = false;

void setup() {
  Serial.begin(BAUD_RATE);
  pinMode(BUTTON_PIN, INPUT);
}

void loop() {
  // Test for button press
  if (button.update()) {
    if (button.read() == HIGH) {
      if (state == STATE_PAUSE) {
        state = STATE_LIGHT_AMPLITUDE;
      } else {
        motor_stop();
        state = STATE_PAUSE;
      }
    }
  }

  // State machine
  Serial.print("State: ");
  Serial.println(state);
  switch (state) {
    case STATE_PAUSE:
      standby();
      break;
    case STATE_LIGHT_AMPLITUDE:
      lightAmplitude();
      break;
    case STATE_LIGHT_DIRECTION:
      lightDirection();
      break;
    case STATE_LIGHT_TURN:
      lightTurn();
      break;
    case STATE_DRIVE_FWD_START:
      driveFwdStart();
    case STATE_DRIVE_FWD:
      driveFwd();
      break;
    case STATE_AVOID_OBSTACLE:
```

```
      avoidObstacle();
      break;
    case STATE_AVOID_OBSTACLE_TURNING:
      avoidObstacleTurning();
      break;
    case STATE_AVOID_OBSTACLE_DRIVE_AHEAD:
      avoidObstacleDriveAhead();
      break;
    case STATE_AVOID_OBSTACLE_TURNING_BACK:
      avoidObstacleTurningBack();
      break;
    case STATE_WAIT_FOR_LIGHT_FROM_BEHIND:
      waitForLightFromBehind();
      break;
    default:
      Serial.print("Unhandled state: ");
      Serial.println(state);
      delay(50);
  }
}

void serialEvent() {
  while (Serial.available()) {
    char c = (char)Serial.read();
    if (c == 'a') {
      turn_left = true;
    } else if (c == 'd') {
      turn_right = true;
    } else if (c == 'w') {
      move_fwd = true;
    } else if (c == 's') {
      move_back = true;
    }
  }
}

// Standby functionality
void standby() {
  const unsigned int light_front = measure_light(LIGHT_FRONT_PIN);
  const unsigned int light_right = measure_light(LIGHT_RIGHT_PIN);
  const unsigned int light_left = measure_light(LIGHT_LEFT_PIN);
  const unsigned int light_back = measure_light(LIGHT_BACK_PIN);

  const unsigned long dist = ultrasonic.measure_distance();

  Serial.print("FRONT: ");
  Serial.print(light_front);
  Serial.print("; RIGHT: ");
  Serial.print(light_right);
  Serial.print("; LEFT: ");
  Serial.print(light_left);
  Serial.print("; BACK: ");
  Serial.print(light_back);
  Serial.print("; DIST: ");
  Serial.print(dist);
  Serial.println(";");

  delay(200);

  if (turn_left) {
```

```
    turn_left = false;
    Serial.println("Turning left");
    motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_TURN_SPEED);
    motor_move(MOTOR_LEFT, MOTOR_BACK, MOTOR_TURN_SPEED);
  }
  if (turn_right) {
    turn_right = false;
    Serial.println("Turning right");
    motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_TURN_SPEED);
    motor_move(MOTOR_RIGHT, MOTOR_BACK, MOTOR_TURN_SPEED);
  }
  if (move_fwd) {
    move_fwd = false;
    Serial.println("Moving forward");
    motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_SPEED);
    motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_SPEED);
  }
  if (move_back) {
    move_back = false;
    Serial.println("Moving back");
    motor_move(MOTOR_LEFT, MOTOR_BACK, MOTOR_SPEED);
    motor_move(MOTOR_RIGHT, MOTOR_BACK, MOTOR_SPEED);
  }

  delay(800);
  motor_stop();
}

// STATE: Light Amplitude
void lightAmplitude() {
  const unsigned int light_amp = calculateLightAmplitude();

  Serial.print("Light amplitude: ");
  Serial.println(light_amp);

  if (light_amp >= MIN_LIGHT_AMP) {
    state = STATE_LIGHT_DIRECTION;
  } else {
    delay(200);
  }
}

const unsigned int calculateLightAmplitude() {
  const unsigned int light_front = measure_light(LIGHT_FRONT_PIN);
  const unsigned int light_right = measure_light(LIGHT_RIGHT_PIN);
  const unsigned int light_left = measure_light(LIGHT_LEFT_PIN);
  const unsigned int light_back = measure_light(LIGHT_BACK_PIN);

  const unsigned int light_x = light_right - light_left;
  const unsigned int light_y = light_back - light_front;
  const unsigned int light_amp = (int) sqrt(light_x * light_x + light_y *
light_y);

  return light_amp;
}

// STATE: Light Direction
void lightDirection() {
  const int angle = calculateLightAngle();
  const int side = calculateLightSide();
```

```
  Serial.print("Light angle: ");
  Serial.println(angle);

  if (side > 0) {
    Serial.println("Right!");
    motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_TURN_SPEED);
    motor_move(MOTOR_RIGHT, MOTOR_BACK, MOTOR_TURN_SPEED);
    state = STATE_LIGHT_TURN;
  } else {
    Serial.println("Left!");
    motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_TURN_SPEED);
    motor_move(MOTOR_LEFT, MOTOR_BACK, MOTOR_TURN_SPEED);
    state = STATE_LIGHT_TURN;
  }

  delay(1000);
}

const int calculateLightAngle() {
  const unsigned int light_front = measure_light(LIGHT_FRONT_PIN);
  const unsigned int light_right = measure_light(LIGHT_RIGHT_PIN);
  const unsigned int light_left = measure_light(LIGHT_LEFT_PIN);
  const unsigned int light_back = measure_light(LIGHT_BACK_PIN);

  const int light_x = light_right - light_left;
  const int light_y = light_back - light_front;
  const unsigned int light_amp = (int) sqrt(pow(light_x, 2) + pow(light_y, 2));

  int alpha = (int)(acos(((float)-light_y) / light_amp) * 180.0 / PI);

  return alpha;
}

const int calculateLightSide() {
  const unsigned int light_right = measure_light(LIGHT_RIGHT_PIN);
  const unsigned int light_left = measure_light(LIGHT_LEFT_PIN);

  return light_right - light_left;
}

// STATE: Light Turn
int previousAngle = 999;
void lightTurn() {
  // check for light intensity
  const unsigned int light_amp = calculateLightAmplitude();
  if (light_amp < MIN_LIGHT_AMP) {
    motor_stop();
    Serial.print("Light intensity too low: ");
    Serial.println(light_amp);
    state = STATE_LIGHT_AMPLITUDE;
    return;
  }

  const int angle = calculateLightAngle();
  if (angle > previousAngle) {
    motor_stop();
    previousAngle = 999;
    state = STATE_DRIVE_FWD_START;
  } else {
```

```
    previousAngle = angle;
    delay(50);
  }
}

// STATE: Drive Fwd Start
void driveFwdStart() {
  motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_SPEED);
  motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_SPEED);
  state = STATE_DRIVE_FWD;
}

// STATE: Drive Fwd
int side_adjustment = 0;
void driveFwd() {
  // check for obstacle
  const unsigned long dist = ultrasonic.measure_distance();
  if (dist < MIN_OBSTACLE_DISTANCE) {
    motor_stop();
    Serial.print("Avoiding obstacle: ");
    Serial.print(dist);
    Serial.println(" cm");
    side_adjustment = 0;
    state = STATE_AVOID_OBSTACLE;
    return;
  }

  // check for light intensity
  const unsigned int light_amp = calculateLightAmplitude();
  if (light_amp < MIN_LIGHT_AMP) {
    motor_stop();
    Serial.print("Light intensity too low: ");
    Serial.println(light_amp);
    side_adjustment = 0;
    state = STATE_LIGHT_AMPLITUDE;
    return;
  }

  // check for angle deviation
  const int angle = calculateLightAngle();
  if (angle > MAX_LIGHT_ANGLE_DEVIATION) {
    motor_stop();
    Serial.print("Light angle deviation too much: ");
    Serial.println(angle);
    side_adjustment = 0;
    state = STATE_LIGHT_DIRECTION;
    return;
  }

  const int side = calculateLightSide();
  if (side > 0) {
    // right
    side_adjustment = min(side_adjustment + 1, 10);
  } else {
    // left
    side_adjustment = max(side_adjustment - 1, -10);
  }
  motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_SPEED + side_adjustment);
  motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_SPEED - side_adjustment);
```

```
  delay(100);
}

// STATE: Avoid Obstacle
void avoidObstacle() {
  // Start turning towards the direction of light
  const int side = calculateLightSide();

  if (side > 0) {
    Serial.println("Light source is right");
    motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_TURN_SPEED);
    motor_move(MOTOR_RIGHT, MOTOR_BACK, MOTOR_TURN_SPEED);
    turnedLeft = false;
    startTurnTime = millis();
    state = STATE_AVOID_OBSTACLE_TURNING;
  } else {
    Serial.println("Light source is left");
    motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_TURN_SPEED);
    motor_move(MOTOR_LEFT, MOTOR_BACK, MOTOR_TURN_SPEED);
    turnedLeft = true;
    startTurnTime = millis();
    state = STATE_AVOID_OBSTACLE_TURNING;
  }
}

// STATE: Avoid Obstacle (turning)
void avoidObstacleTurning() {
  unsigned long elapsedTurnTime = millis() - startTurnTime;
  if (elapsedTurnTime >= MOTOR_90DEGREE_TURN_TIME_MS) {
    motor_stop();
    delay(100);

    // check for obstacle
    const unsigned long dist = ultrasonic.measure_distance();
    if (dist >= MIN_OBSTACLE_DISTANCE) {
      Serial.print("No obstacle found: ");
      Serial.print(dist);
      Serial.println(" cm");
      motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_SPEED);
      motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_SPEED);
      startDriveTime = millis();
      state = STATE_AVOID_OBSTACLE_DRIVE_AHEAD;
    } else {
      Serial.print("Avoiding obstacle, turning back again: ");
      Serial.print(dist);
      Serial.println(" cm");
      if (turnedLeft) {
        // Turn right
        motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_TURN_SPEED);
        motor_move(MOTOR_RIGHT, MOTOR_BACK, MOTOR_TURN_SPEED);
      } else {
        // Turn left
        motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_TURN_SPEED);
        motor_move(MOTOR_LEFT, MOTOR_BACK, MOTOR_TURN_SPEED);
      }
      startTurnTime = millis();
      state = STATE_AVOID_OBSTACLE_TURNING_BACK;
    }
  }
```

```
  delay(50);
}

// STATE: Avoid obstacle (turning back)
void avoidObstacleTurningBack() {
  unsigned long elapsedTurnTime = millis() - startTurnTime;
  if (elapsedTurnTime >= 2 * MOTOR_90DEGREE_TURN_TIME_MS) {
    motor_stop();
    delay(100);

    // check for obstacle
    const unsigned long dist = ultrasonic.measure_distance();
    if (dist >= MIN_OBSTACLE_DISTANCE) {
      Serial.print("No obstacle found: ");
      Serial.print(dist);
      Serial.println(" cm");

      motor_move(MOTOR_LEFT, MOTOR_FWD, MOTOR_SPEED);
      motor_move(MOTOR_RIGHT, MOTOR_FWD, MOTOR_SPEED);
      startDriveTime = millis();
      state = STATE_AVOID_OBSTACLE_DRIVE_AHEAD;
    } else {
      Serial.print("Found obstacle, waiting for light from behind: ");
      Serial.print(dist);
      Serial.println(" cm");

      state = STATE_WAIT_FOR_LIGHT_FROM_BEHIND;
    }
  }

  delay(50);
}

// STATE: Avoid obstacle (drive ahead)
void avoidObstacleDriveAhead() {
  unsigned long elapsedDriveTime = millis() - startDriveTime;
  if (elapsedDriveTime >= MOTOR_AVOID_OBSTACLE_DRIVE_TIME_MS) {
    motor_stop();
    state = STATE_LIGHT_AMPLITUDE;
  }

  delay(50);
}

// STATE: Wait for light from behind
void waitForLightFromBehind() {
  const int angle = calculateLightAngle();

  if (180 - angle <= MAX_LIGHT_ANGLE_BACK_DEVIATION) {
    state = STATE_LIGHT_AMPLITUDE;
  }

  delay(50);
}

// Light sensors
const unsigned int measure_light(const unsigned int pin) {
  unsigned int reading = analogRead(pin);

  return reading;
```

```
}

void output_light(const unsigned int light, const char *where) {
  Serial.print("Light(");
  Serial.print(where);
  Serial.print("): ");
  Serial.println(light);
}

// Motor
void motor_stop() {
  // put controller in standby mode
  digitalWrite(MOTOR_STBY_PIN, LOW);
  analogWrite(MOTOR_PWMA_PIN, 0);
  digitalWrite(MOTOR_AIN1_PIN, LOW);
  digitalWrite(MOTOR_AIN2_PIN, LOW);
  analogWrite(MOTOR_PWMB_PIN, 0);
  digitalWrite(MOTOR_BIN1_PIN, LOW);
  digitalWrite(MOTOR_BIN2_PIN, LOW);
}

void motor_move(const unsigned int side, const unsigned int direction, const
unsigned int speed) {
  // disable standby
  digitalWrite(MOTOR_STBY_PIN, HIGH);

  boolean in1 = LOW;
  boolean in2 = LOW;

  if (direction == MOTOR_FWD) {
    in1 = HIGH;
    in2 = LOW;
  } else if (direction == MOTOR_BACK) {
    in1 = LOW;
    in2 = HIGH;
  }

  if (side == MOTOR_LEFT) {
    digitalWrite(MOTOR_AIN1_PIN, in1);
    digitalWrite(MOTOR_AIN2_PIN, in2);
    analogWrite(MOTOR_PWMA_PIN, speed);
  } else if (side == MOTOR_RIGHT) {
    digitalWrite(MOTOR_BIN1_PIN, in2);
    digitalWrite(MOTOR_BIN2_PIN, in1);
    analogWrite(MOTOR_PWMB_PIN, speed);
  }
}
```