FHV
Vorarlberg University
of Applied Sciences

Fledermaus
Flying Classroom Workshop

Du hast noch keinerlei Erfahrung mit Mikrocontrollern und Programmierung? Keine Angst! Wir starten mit einer Einführung in die Thematik und beginnen mit kleinen, leichten Übungen auf dem Weg zur Fledermaus. In diesem Workshop wollen wir das Ortungssystem einer Fledermaus mit Elektronik und etwas Software nachbauen.

Hierzu verwenden wir

- einen Ultraschallsensor (USS)
- einen Mikrocontroller Arduino Nano
- einen Lautsprecher

Das Programm, das wir schreiben werden, wird aus einer Zeitmessung eine Distanz berechnen.

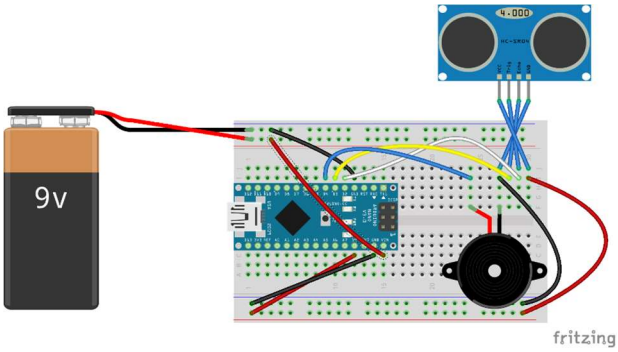
Mit dem Distanzwert wird dann ein Ton erzeugt, dessen Frequenz von eben diesem Distanzwert abhängt.

Den USS befestigen wir durch Einstecken in das Breadboard. So verbinden wir den USS mit dem Breadboard elektrisch und mechanisch, brauchen also kein Klebeband, um ihn irgendwo zu befestigen.

Glossar

Es werden uns immer wieder Begriffe und Abkürzungen begegnen, die wir in diesem Kapitel erläutern bzw. definieren wollen.

Breadboard: Steckbrett, auf dem wir unsere Schaltungen aufbauen.
USS: Ultraschallsensor
Steckplan:



Achtung: Es kann sein, dass bei eurem Breadboard die rote Schiene oben ist und die blaue unten. Dann müsst ihr entsprechend umdenken und umstecken. Denn es gilt immer:

- Rot ist 5V
- Blau ist GND!

Berechnung des Objektstandes aus der Zeitmessung eines Ultraschallsensors

Wie man den Abstand zu einem möglichen Hindernis berechnet, findest du im Abschnitt Peripherieelemente.

Software

Ultraschallsensoren wie der hier eingesetzte HC-SR04 können auf einem Arduino sehr einfach in Software eingebunden werden.

Wir wissen, dass das Messprinzip auf einer Zeitmessung beruht. Suchen wir in der Language Reference nach pulse, finden wir pulseIn() und die Dokumentation zu pulseIn.

Das Programm kann dann z.B. so aussehen:

```
/*****************************************************************************
*
*   FILE:    /atlantis/p/projects/arduino/sketchbooks/fledermaus/fledermaus.ino
*
*   SYNOPSIS:
*       Ansteuerung eines Beepers mit Frequenzen, die von der Abstandsmessung
*       eines Utraschallsensors abhängig sind.
*
*
*   REVISION HISTORY:
*       2024-02-23/FGE:
*           -   Zeitgesteuertes Heartbeat.
*           -   Kommentare
*
*       2017-11-08:
*           Änderungen im Mapping d -> f für KUni 2017.
*
*       2017-03-23:
```

```
*           Erstellt anlässlich des JUni-Workshops, der in Zusammenarbeit mit
*           der inatura am 12. April 2017 stattfindet.
*
******************************************************************************/

const int Frequency_base = 440/4;
                               // Grundfrequenz
const int Frequency_hi = 440 * 8;
const int Frequency_lo = Frequency_base;

unsigned long Time_of_last_heartbeat = 0;
int           Togglevalue_for_heartbeat = 0;

const int Distance_min_cm = 3;
const int Distance_max_cm = 300;

const int Duration_tone_ms = 50;
//const int Duration_sleep_ms = Duration_tone_ms/2;
const int Duration_sleep_ms = Duration_tone_ms * 3;

#define PINNBR_USS_TRIGGER  2
#define PINNBR_USS_ECHO     3
#define PINNBR_SPEAKER      4
#define PINNBR_HEARTBEAT    13


/*****************************************************************************
*/
float delaytime_ms( unsigned int frequency)
/*
* Berechnet aus einer Frequenz eine Wartezeit für die Funktion `delay` in ms.
*
* Parameters:
*      frequency (int):
*          Frequenz in Hz.
* Returns:
*      float:
*          Wartezeit für `delay` in der Funktion `loop`, damit ein Ereignis
*          genau `frequency`-mal ausgeführt wird.
*
******************************************************************************/
{
  return 1000.0/(2.0*frequency);
}


/*****************************************************************************
*/
void setup()
{
    // put your setup code here, to run once:
    pinMode( PINNBR_SPEAKER, OUTPUT);
    pinMode( PINNBR_USS_TRIGGER, OUTPUT);
    pinMode( PINNBR_USS_ECHO, INPUT);
    pinMode( PINNBR_HEARTBEAT, OUTPUT);

    digitalWrite( PINNBR_USS_TRIGGER, LOW);
}

/*****************************************************************************
*/
```

```
void loop()
{
    /*
    * Heartbeat
    */
    if (millis() - Time_of_last_heartbeat > delaytime_ms( 5))
    {
        Togglevalue_for_heartbeat = 1 - Togglevalue_for_heartbeat;
        digitalWrite( PINNBR_HEARTBEAT, Togglevalue_for_heartbeat);
        Time_of_last_heartbeat = millis();
    }

    /*
     * Ping senden
     */
    digitalWrite( PINNBR_USS_TRIGGER, LOW);
    delayMicroseconds( 2);
    digitalWrite( PINNBR_USS_TRIGGER, HIGH);
    delayMicroseconds( 10);
    digitalWrite( PINNBR_USS_TRIGGER, LOW);

    /*
     * Echo empfangen
     */
    int duration_us = pulseIn( PINNBR_USS_ECHO, HIGH);
                                    //  pulseIn(pin, value, timeout=1000000);
                                    //  Reads a pulse (either HIGH or LOW) on a pin. For
example, if value is HIGH,
                                    //  pulseIn() waits for the pin to go HIGH, starts
timing, then waits for the
                                    //  pin to go LOW and stops timing. Returns the length of
the pulse in microseconds
                                    //  or 0 if no complete pulse was received within the
timeout.
    /*
     * Zeit, die der Ping braucht, um wieder am Sensor als Echo
     * gemessen werden zu können, umrechnen in einen Hindernisabstand
     */
    int distance_cm = (int)((float)duration_us/29.0/2.0);
                                    //  v = 0.034 cm/us = 1/29 cm/us. Die Distanz ist die
halbe "Reisedistanz" des "Pings".
    /*
     * Aus Abstand die Frequenz des Wartons ableiten
     */
    int frequency = map( distance_cm, Distance_min_cm, Distance_max_cm, Frequency_hi,
Frequency_lo);
                                    //  map(value, fromLow, fromHigh, toLow, toHigh);
                                    //  Wenn also der Abstand zu einem Hindernis kleiner
wird, soll die Frequenz größer,
                                    //  der Ton also höher werden.
    tone( PINNBR_SPEAKER, frequency, Duration_tone_ms);
                                    //  tone(pin, frequency); oder tone(pin, frequency,
duration) ;
    /*
     * Zuykluszeit
     */
    delay( Duration_sleep_ms);


}
```