Using an ultrasonic Sensor to build a mobile Sonar

Daniel Bencic, Simon Mennig
HTWG Konstanz
Faculty of Computer Science
Brauneggerstr. 55, 78462 Konstanz (Germany)
simon.mennig, daniel.bencic@htwg-konstanz.de

Ralf Seepold
HTWG Konstanz
Faculty of Computer Science
Brauneggerstr. 55, 78462 Konstanz (Germany)
ralf.seepold@htwg-konstanz.de

***Abstract*—The ability to detect objects using non-optical methods has many applications. Radar technology is widely used in aviation, shipping, weather forecast and many other fields including autonomous driving. Sonar (Sound Navigation And Ranging) technology is mainly used for underwater navigation and detection by the military and commercial or scientific ships.**

The aim of this project was to explore the possibilities of sonar technology in a near-field environment above the surface. This was realized as a low-cost sonar device that can be placed on a desk and detect near-distance objects, and then visualizes their shape on a smartphone. In order to achieve that, an Arduino, a servomotor and a sonic distance sensor was used. The smartphone can communicate directly with the sonar via Bluetooth Low Energy (BLE). The visualization was achieved using low-level graphics APIs, to ensure enough performance on constrained smartphones.

Keywords—sonar; Arduino; app; radar; android, OpenGL

Motivation and Introduction

Autonomous robots are becoming increasingly popular. The demand for household robots alone is projected to rise from 3.6 million units worldwide in 2015 to 30 million units in 2019 [1]. These robots need to navigate in a partially known environment, as rooms and the general floor map does not change frequently, but movable objects like chairs or pets and humans do. Therefore, any autonomous home robot needs a sensor to constantly sense its environment with enough precision and tolerance for environmental influences like bad or no light, dust or steam.

Possible problems that can be solved using the input data can range from real-time mapping of the environment, over path-finding and collision avoidance to pose detection and shape recognition. For this task, cameras can be used in combination with image recognition software. However, distance tracking is not very accurate with standard cameras and their performance is limited by the lighting conditions. Also, they are more expensive than other solutions like infrared and ultrasonic sensors. Infrared sensors can be used to accurately measure distances indoors and they are relatively low-cost, but they can be easily disturbed by bright sunlight or dust in the air. Ultrasonic distance sensors use sound waves instead of electromagnetic waves, are low-cost and can operate under any lighting conditions. Also, they do not react to small particles like dust or steam, making them work very reliable under most circumstances. Disadvantages can include possible interference from other ultrasonic devices and the distraction of pets that can hear in the ultrasonic spectrum.

To gather a deeper understanding of the possibilities and challenges using an Arduino combined with an ultrasonic distance sensor, we chose to constrain our project to a sonar that scans its surrounding in an angle of 180 degrees and then visualizes the results on a smartphone.

State of the Art

In this chapter, we mention applications of sonar sensors in indoor environments. One paper describes, how sonar sensors can be used for path planning of a mobile robot in a home environment. Therefore, a hierarchical sonar grid map with a resolution of 5cm * 5cm is created. The sonar sensor scans the environment at a rate of 4 Hz [2]. Another publication uses sonar for 3-D indoor in-air localization of robots. They use a random array of receivers and a wide field of view sonar. Using this approach, the sensor can operate in an environment with multiple overlapping echoes, without mechanical scanning [3]. The last reference we want to mention uses the fusion of depth data provided by Microsoft Kinect and sonar for obstacle avoidance in indoor an environment. The combination of these two inputs allows for real-time path-planning around obstacles, even in the dark and can operate in unknown environments [4].

Proposed approach

A. *Architecture and Used Technology*

The project's hardware consists of three main components. The Arduino board as a control unit and communication relay, a servomotor and a sonic distance sensor.

For the Arduino, there were two main requirements, Bluetooth communication and a compact size, so it can be fitted in a small case. With these aspects in mind, the *Beetle BLE* by DFRobot was chosen. It is based on the Arduino Uno board and has Bluetooth Low Energy (BLE) capabilities. BLE is not compatible with the standard Bluetooth stack however, it offers significantly lower power consumption. This can be achieved because BLE devices are in sleep mode when no data is transmitted. They only wake up for a short time, compared to classic Bluetooth, which stays active. In addition to lower power consumption latencies with BLE can go as low as 3ms, whereas classic Bluetooth only offers a minimum of 100ms [5]. This approach ensures that the power can be provided by a battery if needed.

The servomotor used in the project is the *Microservo SG9Rr*. It can turn by 1 degree in 1 step, so the orientation of the sensor can be controlled very accurately. The ultrasonic sensor is the *HC-SR04*. It can be used for distances from 2 cm to 300 cm. Measurements can be done at a frequency of 50 Hz. The sonic sensor is mounted on top of the servo so that it can rotate freely in an angle of 180 degrees. A wider angle is not used because of the more complex wiring. The servo, therefore, oscillates from one side to the other in steps of 1 degree. The measured distance and angle are then sent to a connected smartphone via BLE. Fig. 2 shows an overview of the whole system.

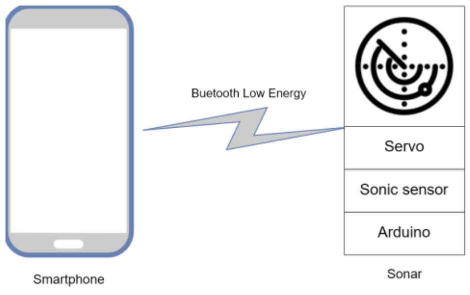


Fig. 2 System overview.

B. Hardware

Fig. 2 shows how the Arduino board and the other electrical components are wired together. The sonic sensor is connected to pin 2 and 4. Pin 2 is the echo pin. The distance measurement result signal is transmitted through this pin. Pin 4 is used to trigger a new measurement. The servo is connected to the Arduino through pin 3.

C. Software

The Arduino sends the measured distance and angle via BLE to the smartphone app. The data is sent for every step the servo performs as a byte-array.

The app utilizes the Bluetooth library provided by DFRobot [6]. The library provides basic connectivity with the Beetle BLE and is customized to fit the project.

The visualization of the sonar is done via an Android NDK App using the graphics API OpenGL ES 2.0.

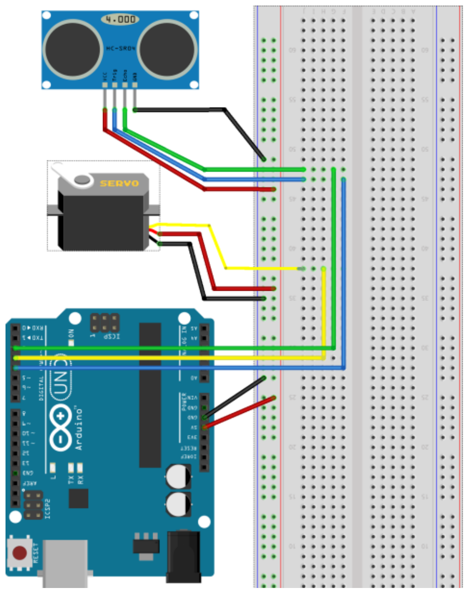


Fig. 1 Circuit schematics.

Methods and Algorithms

This section describes the methods and algorithms we used to realize our approach. First, we are going to give detailed information about the actual sonar device and how the distance is measured. After that, we describe our solution of displaying the measured distances in the Android App.

D. Distance Measurement

For the distance measurements we are using the low-cost ultrasonic sensor *HC-SR04*. The module sends out ultrasonic waves and collects them if they get reflected by an object. Given the time between sending out and receiving these ultrasonic waves and the speed of sound, we can calculate the distance to this object [7].

To start the measurement, we have to send a low signal for at least 10 µs on the trigger pin of the *HC-SR04*. The module sends out a 40 kHz signal and the output of the modules echo pin instantly jumps from low to high. The echo pin output stays high until the echo of the 40 kHz signal is received [7].

Our first approach was, to measure the time of this high pulse with the pulseIn() function of the Arduino core library. This function counts the clock cycles between start and end of the pulse, therefore to get the most accurate measurement results interrupts have to be turned off before the invocation of the pulseIn() function [8]. Using this approach however caused our servomotor to stutter. Due to this problem we make use of a third-party library called "NewPing", which uses a different method to measure the time of a pulse. The mentioned problem does not occur with this library and the library also offers a useful feature to specify a maximum distance measured [9].

After measuring the distance, a byte array of length two is constructed. The first element of this array is the measured distance and the second element is the servomotor’s angle while the distance was measured. This array is then sent to the

Android App. Sending data with the Beetle BLE is very straightforward. The data is just written to the serial [10].

E. *Android App*

The Android App consists mainly out of two components. The first component is the main activity, which is responsible for establishing the BLE connection to the Beetle BLE and receiving the data sent by the sonar. It also sets up the view.

To achieve the best performance, the rendering is done via OpenGL ES 2.0 and implemented in C++. Here we are using the Java Native Interface to enable the communication (handing over angle and distance data) between the Java and the C++ part of the App.

F. *Sonar Rendering with OpenGL ES 2.0*

The OpenGL ES 2.0 rendering in Android is done on a *GLSurfaceView*. This type of view holds a reference to an object implementing the *GLSurfaceView.Renderer* interface. This interface defines the three methods *onSurfaceCreated*, *onSurfaceChanged* and *onDrawFrame*. The implementations of this methods are calling the respective native implementations.

In computer graphics small building blocks are used to form more complex shapes. This building blocks are for example points (vertices), lines or triangles. They are called geometric primitives [11]. Our sonar has a circular shape (semicircle) so, we have to combine the primitives in a certain way to get the desired result.

We construct our semicircle by splitting it into multiple triangles. For every degree of the semicircle we introduce a single triangle. Every triangle T consists of three vertices. All triangles have one vertex v_0 in common, which is the center of the circle. Given the parametric equation of a circle we can compute the other two vertices v_1 and v_2 of each triangle with these formulas:

$$x = x_c + r * cos\left(\frac{angle * \pi}{180}\right)$$

$$y = y_c + r * sin\left(\frac{angle * \pi}{180}\right).$$

We start at an angle of 359,5 ° and end with the angle 180,5 ° to center the triangles on integer angles. It holds that,

$$T_i.v_0 = T_{i+1}.v_0 \wedge T_i.v_2 = T_{i+1}.v_1 \text{ with } 0 \leq i \leq 180.$$

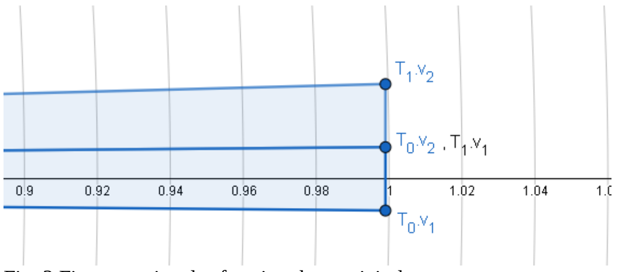
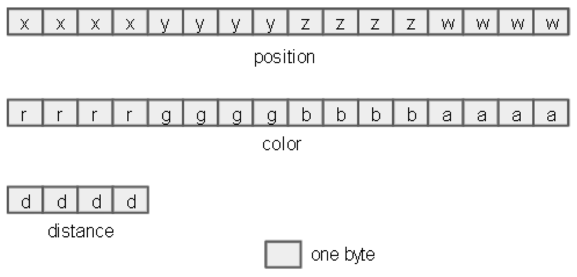


Fig. 3 First two triangles forming the semicircle.

Every vertex is a block of memory containing 36 bytes. The first 16 bytes consist of 4 floating-point numbers representing the coordinates (x, y, z and w). The following 16 bytes consist of 4 floating-point numbers defining the color of this vertex in RGBA. The last 4 bytes represent a floating-number storing the measured distance at this angle. The last 20 bytes of a vertex are identical for every vertex of a triangle.


x x x x y y y y z z z z w w w w
position
r r r r g g g g b b b b a a a a
color
d d d d
distance
one byte
Fig. 4 Memory layout of a vertex.

This is necessary because OpenGL ES 2.0 does not allow disabling interpolation [12]. To get the correct value in our fragment shader we therefore have to save per triangle attributes in every vertex of this triangle, because the interpolated value of three times the identical value is that value.

To see the whole shape of an object detected by the sonar we added an afterglow effect. This effect is implemented as a linear function which calculates the alpha value based on how long the triangle is already visible. The equation for this function is

$$alpha = \frac{-duration}{duration_{max}} + 1.$$

We are lowering the alpha value ($1 \geq alpha \geq 0$) to make the triangle more transparent and let it fade away over time. Fig. 5 shows the graph of this function.

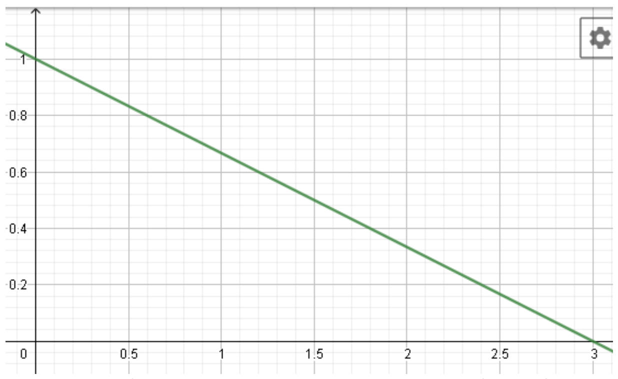
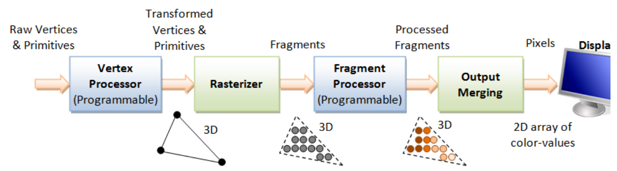


Fig. 5 Graph of the equation to calculate the alpha value for an afterglow effect of 3 seconds (y = alpha value, x = duration).

The last algorithm we want to discuss is the algorithm to determine the color of a triangle. This algorithm is run in the fragment shader, hence it is executed on the GPU. The color of a triangle depends on the distance measured at this angle. A measured distance of 0 cm means no object was detected by the sonar, so we color this triangle green. If the measured distance is however greater than 0 cm, we have to color a part of the triangle red, representing the detected object. We applied a range cap of 50 cm to the sonar so, given the distance measured, we can calculate a radius

$$r = r_{max} * \frac{distance}{50}.$$

After processing the vertices and producing grid-aligned fragments for each primitive through the rasterizer, the next step in the OpenGL rendering pipeline is processing each fragment through a program called fragment shader (Fig. 6). The fragment shader is run for every pixel on the rendering surface and its purpose is to output a color for the given fragment [13].


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
3D
Fragments
Fragment Processor (Programmable)
3D
Processed Fragments
Output Merging
3D
Pixels
Displa
2D array of color-values

3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attribute such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up o one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, b interpolating the vertices.

Fig. 6 OpenGL rendering pipeline [14].

It takes in several built-in parameters , we can use to calculate the color of this pixel [12]. The parameter we use is called gl_FragCoord and represents to window coordinates of the pixel. We use the built-in function distance() to calculate the euclidean distance between the center of the circle and the pixel. If the distance is greater than our calculated radius, we color this pixel red, otherwise it is getting the color green.

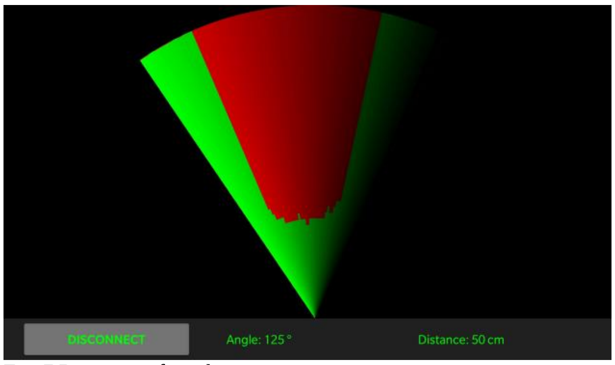


Fig. 7 Detection of an object.

Results and Conclusion

Our work showed, that it is possible to do very accurate distance measurements with the low-cost ultrasonic sensor *HC-SR04*. Combined with a servomotor a robot can easily detect objects around itself and measure the distance to these objects. The *HC-SR04* can therefore be a good sensor to get measurements for techniques like SLAM or a Kalman Filter, to make the localization and navigation of a robot more accurate.

Furthermore we showed a high-performance approach to visualize the resulting distance measurements on an Android phone with the use of the JNI, OpenGL ES 2.0 and basic methods of computer graphics. We chose OpenGL ES 2.0 to also support older versions of Android, however it turned out, that some parts of the implementation got more complex – compared to the implementation with OpenGL ES 3.0 - due to missing features like flat interpolation in OpenGL ES 2.0

References

[1] International Federation of Robotics, "31 million robots helping in households worldwide by 2019", https://ifr.org/news/31-million-robots-helping-in-households-worldwide-by-2019. Accessed 28.12.2018

[2] B. Park, J. Choi and W. K. Chung, "An efficient mobile robot path planning using hierarchical roadmap representation in indoor environment", in 2012 IEEE International Conference on Robotics and Automation, 2012

[3] J. Steckel, A. Boen and H. Peremans, "Broadband 3-D Sonar System Using a Sparse Array for Indoor Navigation," IEEE Transactions on Robotics, vol. 29, pp. 161-171, 2013.

[4] J.-D. Lee and Z.-Y. Dang, "Dual-sensor fusion for obstacle avoidance in indoor environment," in 2015 International Conference on Advanced Robotics and Intelligent Systems (ARIS), 2015.

[5] Bluetooth Special Interest Group, "About Bluetooth® Low Energy Technology", https://web.archive.org/web/20140214120404/http://www.bluetooth.com/Pages/low-energy-tech-info.aspx, Accessed 10.01.2019

[6] DFRobot,"BlunoBasicDemo, https://github.com/DFRobot/BlunoBasicDemo, Accessed 10.01.2019

[7] KT-Electronic, "Ultraschall Messmodul HC-SR04", January 2012, https://www.mikrocontroller.net/attachment/218122/HC-SR04_ultraschallmodul_beschreibung_3.pdf, Accessed 14.01.2019.

[8] Arduino, "pulseIn()", https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/, Accessed 14.01.2019.

[9] T. Eckel, "NewPing Library for Arduino", February 2017, http://playground.arduino.cc/Code/NewPing, Accessed 14.01.2019.

[10] DFRobot, "Bluno Beetle Simple Tutorial (With Explanatory Images)", https://www.dfrobot.com/blog-283.html, Accessed 14.01.2019.

[11] D. J. Eck, "Introduction to Computer Graphics", January 2018, http://math.hws.edu/eck/cs424/downloads/graphicsbook-linked.pdf, Accessed 14.01.2019.

[12] Khronos Group, "OpenGL ES 2.0 API Quick Reference Card", 2010, https://www.khronos.org/opengles/sdk/docs/reference_cards/OpenGL-ES-2_0-Reference-card.pdf, Accessed 14.01.2019.

[13] G. Sellers, R. S. Jr. Wright, N. Haemel, "OpenGL Superbible: Comprehensive Tutorial and Reference", Addison-Wesley, 2016.

[14] C. Hock-Chuan, "3D Graphics with OpenGL: Basic Theory", July 2012, https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html, Accessed 14.01.2019.