Waterwell monitor - web interface - Instructions

Thank you for purchasing the Hladinomer project, which supports the use of open hardware from the Arduino series (AVR), Espressif Systems (ESP8266, ESP32). The project consists of a web application that is used to collect data on the level of water from a microcontroller. It can represent the data to the user who uses the web application. The web application can represent the last measured data in real time with automatic update in the main report, but also historically in a tabular report, where it is possible to find the last 100 records, or historically all available records.

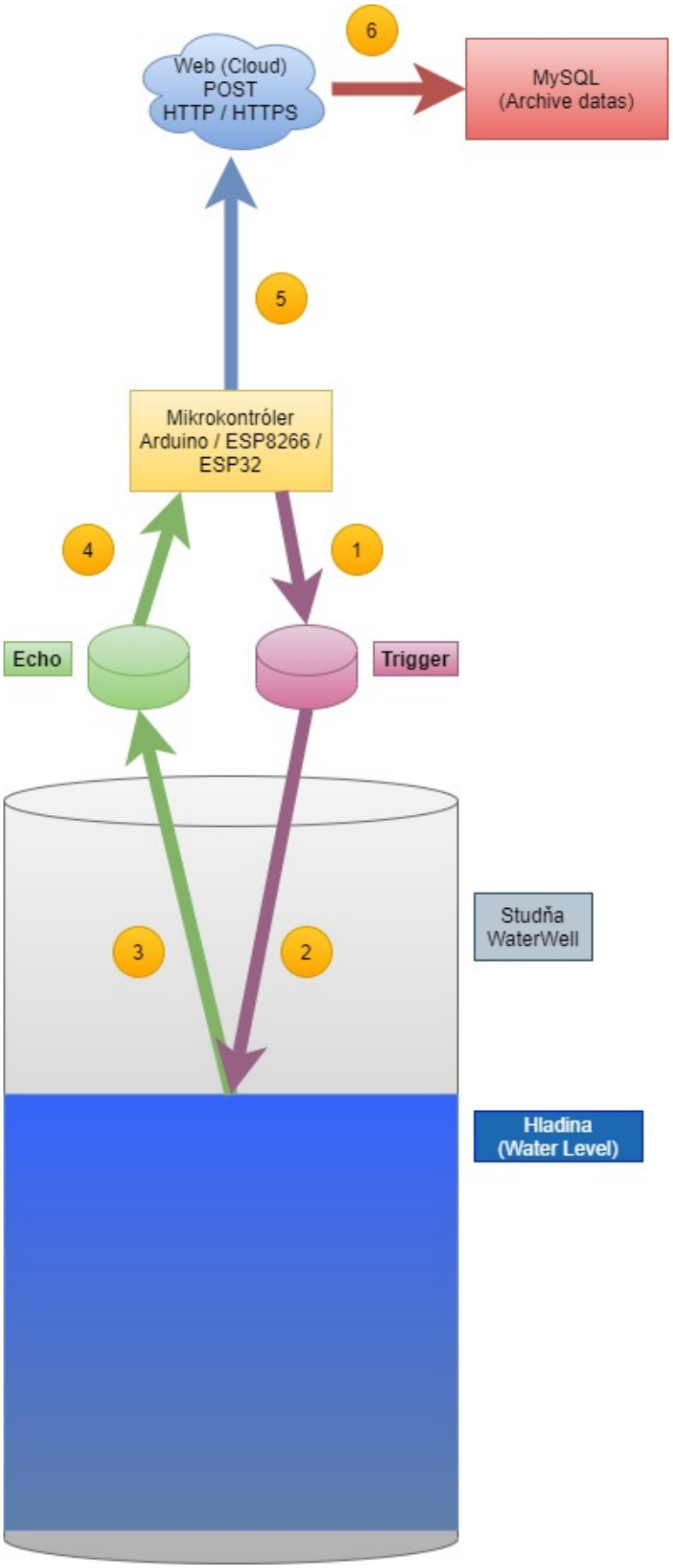
For the development of the water level over time, it is possible to use a graphical overview with a bar graph, which allows an overview of measured data for the last 2 hours, 24 hours, 7 days, 30 days and 365 days. Through alarm representations, the user can see the measured record of the minimum and maximum in 24 hours, 7 days, 30 days. The microcontroller collects water level data and operates an ultrasonic distance sensor. During the measurement, the ultrasonic sensor Trigger sends a signal with a length of 10 microseconds, which bounces off the surface and returns to the receiver - the Echo signal. Based on the speed of sound propagation, it is possible to calculate the distance between the sensor and the level in centimeters.

This value is sent by the microcontroller to the HTTP web interface using the POST method. Data can be sent via HTTP, but also HTTPS protocol on ESP8266 and ESP32 platforms, which support encryption. For data transmission it is possible to use transmission technology Ethernet, WiFi, IoT network Sigfox. The ultrasonic distance sensor is placed perpendicular to the surface and ideally in the middle of the well for optimal measurement without reflections and other disturbing phenomena that would inaccurate the measurements, as the emitted beam has a circular (conical) characteristic. For consistent measurement results, the measurement is used 10 times in a row, which is averaged to achieve the final value.

The web interface performs a correction after receiving data from the microcontroller, as the measured level does not represent the actual level from the bottom, but only the distance from the lid of the well where the sensor is located. For this reason, it is possible to enter the depth of the well into the web interface, which serves as a correction for the correct conversion of the water level to the actual (from the bottom) well. The second configurable parameter in the web interface is the diameter of the well, which is used for possible conversion of the water level in the well per liter. The well diameter also uses a trigonometric - theoretical estimate of the measurable level without reflections. The estimate expresses the maximum depth of the well at which it is possible to measure without reflections on the basis of known detection characteristics (known beam width) of ultrasonic distance sensors.

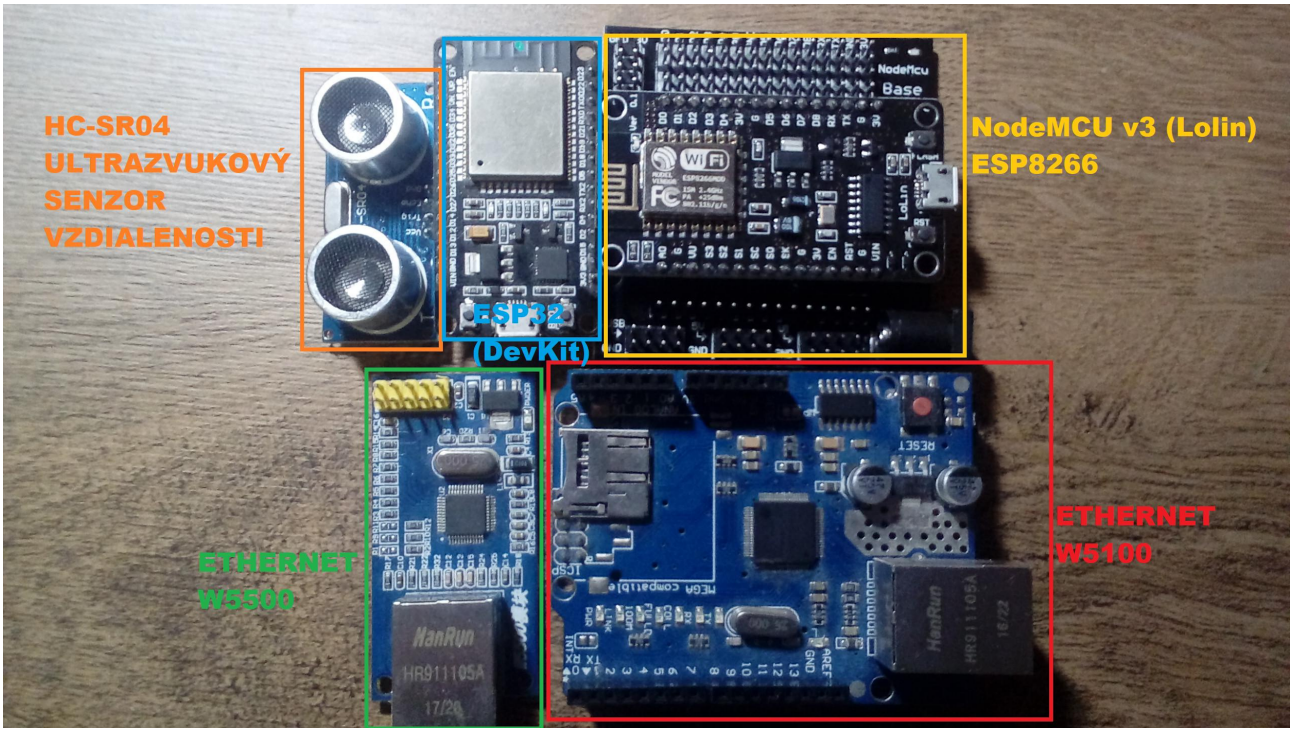
Program implementation for microcontrollers using ultrasonic sensors HC-SR04 (fully compatible and HY-SRF05), or JSN-SR04T (waterproof sensor) in a 4-pin version (with a small modification of the program, it is possible to use a 3-pin version with one pin for Trigger and Echo). The sensors are able to measure up to a maximum of approx. 450 cm. HC-SR04 has a detection characteristic of 15 ° and is suitable for most wells, but it is not waterproof and there is a high risk of oxidation due to moisture in the well, it is suitable to place it above the well. The waterproof sensor JSN-SR04T has a 60 ° detection characteristic, which results in a wide beam, so a well with a sufficient diameter is required (min. 6 meters at a depth of 4 and a half meters). Ultrasonic sensors cannot be used in drilled wells, the characteristics are too great! Both sensors must be powered to 5V, control signals are sufficient for 3.3V.

Block diagram of level meter operation - principle:



Supported hardware and transfer technology:

- **Arduino Uno / Nano / Mega + Ethernet** (Wiznet W5100 / W5500 (i v2 USR-ES1), ENC28J60) - HTTP support
- **ESP8266** (NodeMCU, Wemos D1 Mini) - built-in WiFi - HTTP + HTTPS support
- **ESP32** (Devkit, ESP32-WROOM-32) - built-in WiFi - HTTP + HTTPS support
- Any MCU + Sigfox WISOL SFM10R1 UART Modem - HTTP + HTTPS support



The wiring diagram for the Level Meter is available directly in the web interface. Since the ESP8266 and ESP32 platforms have 3.3V operating logic, it is necessary to connect the Echo signal via a voltage divider, which will reduce the logic level from 5V to 3.3V. ESP microcontroller terminals are not 5V tolerant! The connection of the terminals from the diagram is also used in the software implementation for microcontrollers, so no modification of the source codes is necessary. These are also available directly in the web application, which generates their code based on the domain where the level meter is running - it finds the path to the .php file, which is used to write data to the web interface. All you have to do is copy the code to the Arduino IDE environment and upload it to the microcontroller.

If a Sigfox module is used, it is necessary to communicate with this sensor via the UART bus. It is possible to use a hardware UART line (eg UART1, UART2 - if the microcontroller supports it), or emulate software via the SoftwareSerial library - it only supports low data rates. The level meter message has size 4B (INT - integer), the remaining 8B, which can be used in the message can be used for system data transmission (GPS coordinates, RSSI, message number), callback must be set for automated data entry from Sigfox backend to web interface.

For ESP8266, ESP32 platforms, data transfer is also available via HTTPS protocol. To implement a secure connection, the ESP32 microcontroller uses a Root CA certificate in .pem format, ESP8266 SHA1 fingerprint of the web server certificate. It is necessary to implement a certificate in the program implementations, resp. fingerprint manually into source code (program.php). It is possible to obtain a fingerprint and a Root CA certificate in .pem format via the OpenSSL cryptographic tool.

Operating modes of microcontrollers:

Software implementations for microcontrollers were created for various needs related to the operation of the project and the possibilities of its management. StandBy mode is available for all platforms - i. Arduino + Ethernet / ESP8266 / ESP32. In this mode, the microcontroller is in active operating mode, executing requests, handling the WiFi / Ethernet stack, "feeding" the watchdog, timing the sending of data via millis (). The second mode of operation is StandBy + OTA, which is only available for the ESP8266 and ESP32 platforms.

When loading the OTA program, the microcontrollers operate as in StandBy mode, but in addition they send a network OTA port, through which it is possible to load a new program - firmware remotely - into the microcontroller via LAN directly from the Arduino IDE environment. The port is displayed in the Tools → COM ports section and is in the form ESPXX-MAC address at IP_ADDRESS.

In order for OTA functionality to be available even after uploading a new program, it must include a Basic OTA implementation, otherwise the OTA network port will stop broadcasting and it will not be possible to update it remotely. You cannot display a UART monitor with a network OTA port. For Windows devices, it is necessary to turn off the Firewall, as it blocks the microcontroller's response to the computer and the program cannot be loaded with the “ESP timed out” error.

The device that performs the OTA update can be connected to the LAN network via Ethernet / WiFi. In addition to the Arduino IDE, you must have Python at least version 3 installed. Python automatically manages updates and data transfers. The ESP platform must have an OTA-enabled partition table for compatibility with the upgrade method.

For the mentioned platforms from the Espressif Systems series, it is possible to use another operating mode, namely ULP - Ultra Low Power. This operating mode uses the deep sleep of the main processor, which dramatically reduces power consumption to the order of 10 microamperes at 3.3V power supply. This mode is thus suitable for battery level operation. It is also possible to add to the program implementation the possibility of controlling the transistor, which can be used to switch the power supply of the ultrasonic sensor, as it has a current consumption of 1mA with a 5V power supply in the StandBy state.

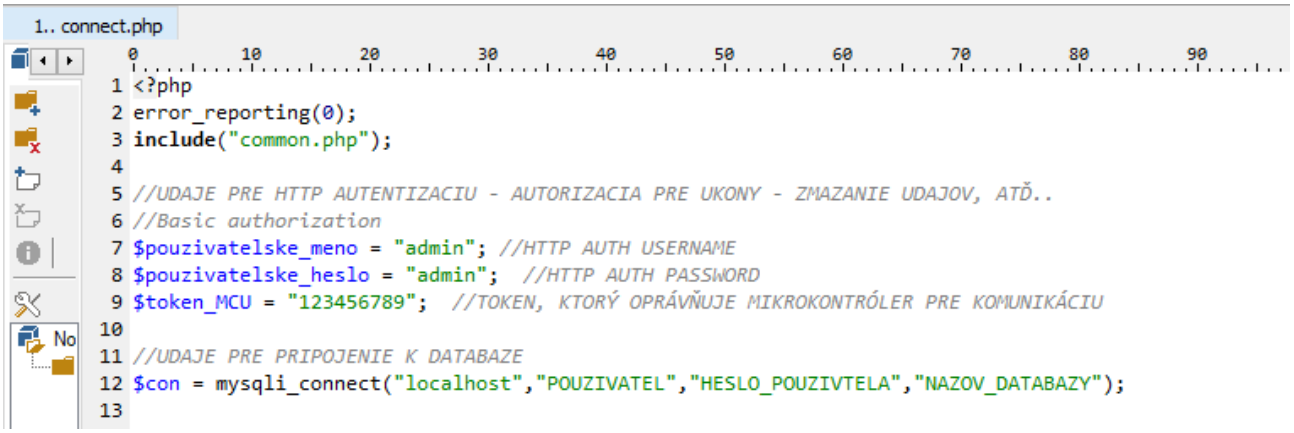
When the main processor falls asleep, the method of waking it up after 300 seconds (5 minutes) is set, when we want to send data to the web server and perform measurements from the sensor. The ESP8266 platform uses the Timer wake source. The timer sends a signal (GND) connected to the RST from the GPIO16 (D0) terminal of the WAKE. This signal restarts the microcontroller and starts the main chip - this connection is included in the wiring diagram in the web application. If D0 is connected to RST, it is not possible to upload a new program! When recording a program, it is necessary to disconnect this jumper and reconnect it after recording the program, otherwise the microcontroller will not wake up from sleep mode.

The ESP32 platform also allows timer wake-up, but uses the RTC Timer, a real-time clock timer. This timer is more accurate compared to the ESP8266 and does not require a jumper as the timer restarts the ESP32 system call. For ESP8266, a timer variance of 10 to 15% is possible.

Examples of possible placement of sensors in a well

Installation and commissioning of the Level meter system:

1. Unzip the .zip archive of the Hladinomer project
2. Log in to your MySQL database management interface (PHPMyAdmin)
3. Select the database into which you want to import the project table with the structure
4. Click Import in the top bar and select the file contained in the / sql / folder
5. Open the connect.php file and change the data to your MySQL database (host, username, password and database name where you imported the well2 table from the .sql file in question)
6. Also modify the token that will authorize your microcontroller to send data to the server, it can be formed by characters, numbers (for example: dsf4ds98f4s9df9). Change the username and password for HTTP Basic Authentication, which will authorize you to make changes to the data (ATTENTION, THIS LOGIN INFORMATION IS NOT ENCRYPTED! THEIR BASE64 VALUE HAS BEEN SENT! VALUE IS TRANSMITTED TO THE ENCRYPTED CHANNEL OF SERVER

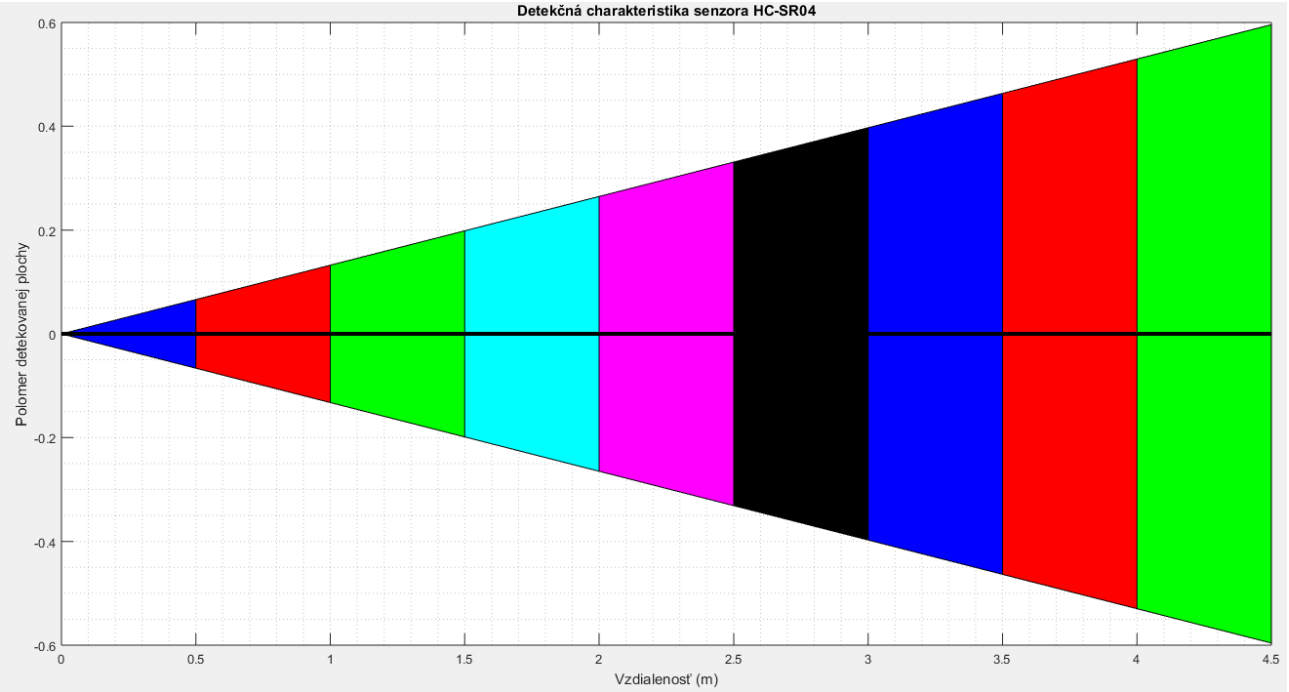


7. Copy all files and folders (sql folder not needed) to your webserver to the www folder (it can be copied to a separate folder under the main www as a separate project, for example / level meter / files ...)
8. The web interface is ready for operation and data recording
9. Download the Github repository: https://github.com/martinius96/hladinomer-studna-scripty, click on the green button on the right Code → Download ZIP
10. Unzip the archive and copy the contents of the / src / folder (Entire folders Ethernet2, Ethernet3, NewPing, NewPingESP8266, UIPEthernet) to:
11. C:/Users/[Current_User]/Documents/Arduino/libraries - Windows 10 - predefined location (may vary depending on where you have the Arduino IDE installed on your system!) - individual folders are libraries that are required for supported hardware in system and for successful program compilation.
12. Open the web interface, go to the Program subpage, log in to the system via HTTP Auth (you entered the name and password in connect.php), after granting access you can see the source code for your microcontroller.
13. Copy the program to the Arduino IDE, select the target board, COM port where the board is connected, if necessary, enter the name of the home WiFi network and password in case of WiFi platform, uncomment the library and comment out the original (different libraries for different Ethernet shields and modules), click on Upload - parts of the program implementation are marked in red if they need your attention / change based on the connected hardware for easy orientation
14. After uploading the program, the microcontroller will start sending data at 5-minute intervals, or in the case of using a Sigfox modem, it is 11 minutes.

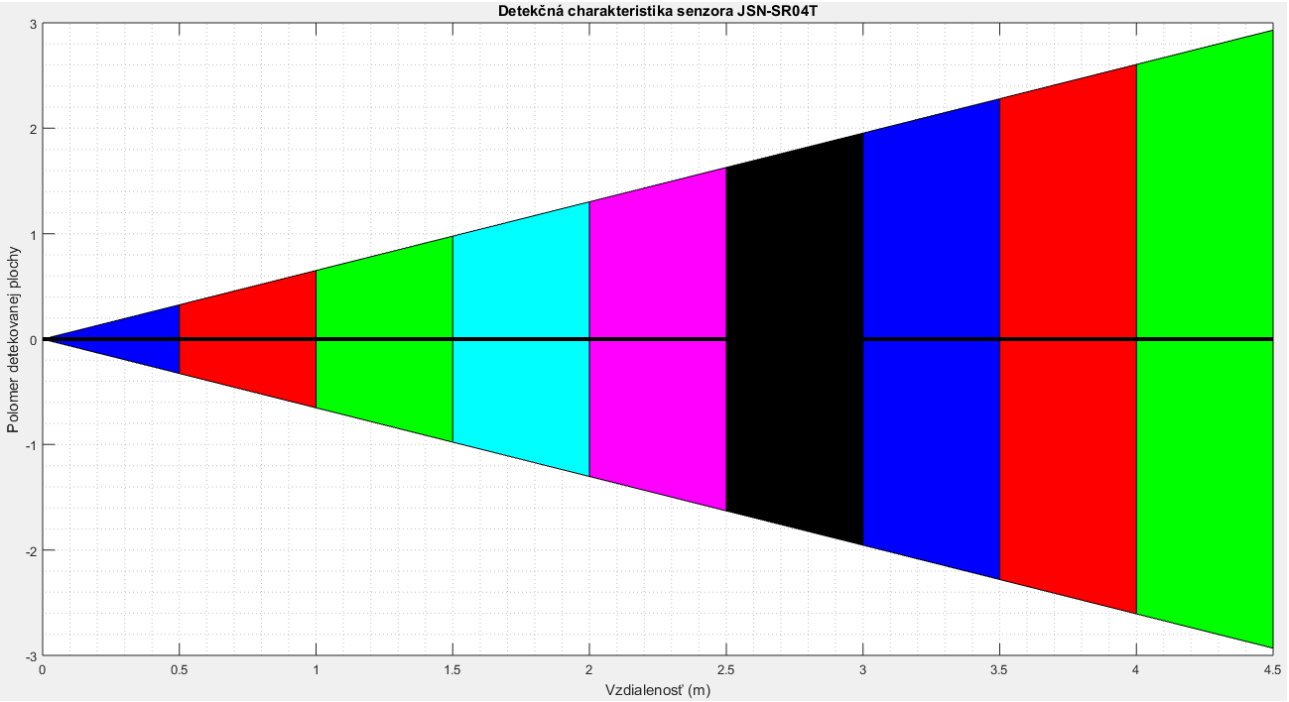
Problems, hardware support:

- The ESP8266 and ESP32 development platforms are not supported by the Arduino IDE environment by default, it is necessary to install their support via the Board Manager in the Arduino IDE environment according to the instructions:
- ESP8266 - https://github.com/esp8266/Arduino#installing-with-boards-manager - select 2.7.4. (the latest stable version for which the level implementations of Hladinomer are compatible)
- ESP32 -
- https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md - compatible any version from 1.0+
- If the COM port is not shown in the Arduino IDE after connecting the board (the option is gray and does not allow selection), or is still set to COM1, it is necessary to install the driver of the USB-UART converter, or check that you use a USB cable with data wires, some cables they only have power supply without data wires!
- The ESP8266 development platform (NodeMCU, Wemos D1 Mini) is most often equipped with a USB-UART converter CH340, which must be manually installed: https://sparks.gogo.co.nz/ch340.html
- ESP32 (Devkit) contains the most common USB-UART converter CP2102 from Silicon Labs. The converter must be installed manually from the official Silicon Labs website:
- https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers - this type of converter also occurs with some ESP8266 boards, such as NodeMCU v2 (Amica) due to its compact size
- MacOS users - If the compiler reports an error with PROGMEM functions, they must be manually removed from the program, but the strings will be assigned to the RAM memory of the microcontroller
- ESP32 Devkits have a problem design with two identical capacitors for the EN and BOOT signal (GPIO 0) and the board does not boot. When the system is started (power-on cycle, power supply connection), due to the same GND potentials on both terminals, the microcontroller switches to download mode and waits for a manual restart with the EN (RESET) button. The repair is possible by connecting GPIO 0 via a 10K pullup resistor to Vcc (3.3V), or it is possible to desolder the capacitor or replace it with another with a different capacitance - a known Devkit error.
- If the web server does not display the page, but only the texts of the program code, it is necessary to install Apache / Nginx, or HTTPD service, which allows you to run backend - server side scripts in PHP - It is possible to use PHP version 5.6+ up to the latest PHP versions
- If you see the information on the project website - “Failed to connect to DataBase”, you have not entered the correct information about your database in the connect.php file
- To obtain a Root CA certificate using OpenSSL, we will use the command openssl s_client -showcerts -verify 5 -connect myserver.com:443 </ dev / null - this solution will last in the implementation of the order of 5 to 20 years, depending on the validity of the CA certificate the certificate for the web server is issued after a time by another authority, it needs to be changed, the connection will not be performed otherwise)
- To get the server certificate fingerprint in SHA1 format with OpenSSL, use the openssl s_client -connect myserver.com:443 -showcerts </ dev / null 2> / dev / null | openssl x509 -in / dev / stdin -sha1 -noout -fingerprint - this solution lasts in the implementation of the order of 1 to 2 years, which is the period of a valid webserver certificate, each time it is renewed, the fingerprint changes, which needs to be implemented again in source code for the ESP8266 microcontroller

Detection characteristic of HC-SR04:



Detection characteristic JSN-SR04T:



Author: Bc. Martin Chlebovec **4. Mar. 2021**