Sensor Fusion for Proximity Estimation

Afonso Alemão
MEEC
Instituto Superior Técnico
Student Number: 96135
Lisbon, Portugal
afonso.alemao@tecnico.ulisboa.pt

Rui Daniel
MEEC
Instituto Superior Técnico
Student Number: 96317
Lisbon, Portugal
ruipcdaniel@tecnico.ulisboa.pt

Tomás Fonseca
MEEC
Instituto Superior Técnico
Student Number: 66325
Lisbon, Portugal
tomasfonseca@tecnico.ulisboa.pt

***Abstract*—This work proposes a ultrasonic and infrared sensor fusion for proximity estimation in order to create a unified sensor that produces the highest precision for the full range of tested distances (0-100 cm). It takes advantage of interrupts to produce a new pulse as soon as the echo signal is received. The ultrasonic measurements are used to compensate infrared response to different refraction rates of objects with different colors. The accelerometer is used to compensate for sensor tilting. The unified sensor achieved a relative error less than 25% for all tests. For most of the measurements this value was below 10%.**

I. Introduction

This project includes the assembly, testing and calibration of a proximity sensing system. The base of the design is an Arduino [1] electronic platform, equipped with an nRF52840 Microcontroler bearing an Arm Cortex-M4 32-bit Processor [2], a 12-bit ADC, and an internal inertial measurement unit (IMU), among other things. The specific Arduino platform used in this project was Nano 33 BLE [3]. We used two different external proximity sensors, based on distinct physical principles. One is based on ultrasonic time of flight and the other in infrared refraction intensity. These sensors have different precisions for different distances. Our goal is to create a unified sensor with high precision for the full range of tested distances (0-100 cm).

The Arduino's IMU can be used to measure acceleration, and on the earth's surface it can determine the devices angle regarding the earths gravitational force. So, it can be used in order to compensate for the systems tilt that induces errors in the proximity measurements: angle compensation.

Contrary to ultrasonic sensors, infrared sensors are highly dependent on the surface color of the reflecting obstacle. We also perform color compensation to avoid this phenomenon.

II. Theory

A. CPU

The Arm Cortex-M4 processor [2] with floating-point unit (FPU) has a 32-bit instruction set (Thumb-2 technology) that implements a superset of 16- and 32-bit instructions to maximize code density and performance. This processor implements several features that enable energy-efficient arithmetic and high-performance signal processing, including Digital Signal Processing (DSP) instructions, single-cycle multiply and accumulate (MAC) instructions, Hardware divide, 8- and 16-bit single instruction multiple data (SIMD) instructions and single-precision floating-point unit (FPU). In addition, the Arm Cortex Microcontroller Software Interface Standard (CMSIS) hardware abstraction layer for the Arm Cortex processor series is implemented and available for the M4 CPU. Additionally, real-time execution is highly deterministic in thread mode, to and from sleep modes, and when handling events at configurable priority levels via the nested vectored interrupt controller (NVIC). Executing code from flash will have a wait state penalty on the nRF52 series. An instruction cache can be enabled to minimize flash wait states when fetching instructions.

B. Ultrasonic Sensor

Based on ultrasonic time of flight, ultrasonic sensors are mostly independent from environmental factors like light intensity, dust, smoke, etc.

In fact, the HC-SR04 [4] employs ultrasonic echo-based distance estimation. The proximity measurement starts when a pulse of 10 μs is applied to the trigger pin of the module. The sensor will emit a burst of sonic pulses with a known signature that allows the system to differentiate from interfering signals. The pulses travel through the air (Fig. 1 of [5]). During this time the echo pin goes HIGH.

If there is no reflection the echo pin will timeout after $38\,\mathrm{ms}$. If the emitted pulses are reflected, the echo pin goes LOW as soon as the signal is received. This produces a pulse duration related to the distance of the reflecting obstacle, which can be used to compute that distance.

As a matter of fact, the width of the received pulse is then used to calculate the distance to the object from which it reflected. This can be worked out using the equation Distance $=$ Time $\cdot$ Speed, where the speed is equal to the speed of sound ($v_{sound} = 340\,\mathrm{m/s}$).

Our circuit for the interconnection of this sensor and the Arduino is presented in the attachments in Fig. 4.

C. Infrared Sensor

Infrared (IR) sensors operate on the principle of reflected light intensity. An IR emitter produces IR light that is consequently reflected by objects in the surrounding area. The IR receiver absorbs the IR light and produces energy. There is a correlation between the absorbed energy and the object proximity that can be used to measure the object distance with respect to the sensor. IR sensors are widely used for measuring close distances, as they are cheaper and faster in response than the ultrasonic (US) sensors. However, they are very susceptible to the properties of the reflecting object (different absorption rates) and to external light sources that emit infrared radiation. They also have a non-linear characteristic and depend on the reflective properties of the object surfaces. As such, to obtain consistent output results, knowledge of ambiance and surface properties must be known and controlled. Therefore, the breadboard was protected by a cardboard cage and the IR receiver limited by black plastic in order to receive only radiation in front of it (directly coming from the detected object). The setup can be seen in Fig. 3 in the attachments.

A circuit was designed to implement the IR sensor. For convenience, the circuit is divided in two: one responsible for the IR emission and another for the IR reception. The IR emitter consists in a LED [6] in series with a $33\,\Omega$ resistor, fed by $3.3\,\mathrm{V}$ supplied by the Arduino. The purpose of the resistor is to both control the

intensity of the light emitted by the LED and to guard the circuit of external noise.

The IR receiver circuit consists in a photodiode [7], a $470\,\mathrm{k}\Omega$ resistor and a operational amplifier (op-amp) [8]. Photodiodes generate current proportional to the absorbed light. To convert current in voltage, a simple transimpedance amplifier was designed [9]. The photodiode is connected to the inverting input, v_-, and the non-inverting input, v_+, is grounded. Then, the resistor, R_f, connects v_- to the output of the op-amp, V_{out}, that is connected to the Arduino analogic input pin (A0). A Passive Low Pass RC Filter should have been put between V_{out} and A0, in order to reject all unwanted high frequencies.

As expected, the received energy increases as we bring an object (e.g., white sheet) closer. To obtain an accurate distance measurement, it is required to perform a calibration. The calibration process, using different materials and colors, is explained further ahead.

A schematic of the IR sensor circuit is included in the attachments in Fig. 5.

D. Accelerometer (IMU)

The LSM9DS1 module in the Arduino platform is a 9-axis inertial sensor [10]. It can sense 3 acceleration channels, 3 angular rate channels and 3 magnetic channels. In fact, LSM9DS1 embeds 32 slots of 16-bit data FIFO for each of the gyroscope's three output channels, yaw, pitch and roll, and 16-bit data FIFO for each of the accelerometer's three output channels, X, Y and Z.

This way, the accelerometer (IMU) is used to compensate for sensor tilting and for calculating the real distance to an obstacle relative to the systems geometric center. Indeed, sensor tilting between the movement plane can introduce significant errors in the eventual traveling distance measure to the actual obstacle. Thus, angle compensation should be introduced, which can be performed by estimating the inclination or tilting angle using the Arduino's accelerometer.

For this case, we get the rate of change in velocity of our Arduino board, from all channels. However, we only make use of channel Z to obtain the angle relative to z-axis, θ_Z. Consequently, the distance to an object considering tilt, $d_{\text{CONSIDERING TILT}}$, as a function of the distance measured by the sensors, d, can be expressed by (1).

$$d_{\text{CONSIDERING TILT}} = d \cdot \cos\left(\frac{\pi}{2} - \theta_Z\right); \tag{1}$$

III. Implementation

A. Interrupts and Timers

1) Timers: `micros()` [11] returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On our Arduino board, this function has a resolution of 4 μs.

In order to understand the internal mechanism of this function, we must understand how counts are converted to time and what is a prescaler value. The timer can operate in two modes: timer mode and counter mode.

In counter mode, the timer's internal counter register is incremented by one each time the count task is triggered, that is, the timer frequency and the prescaler are not utilized. Counter frequency is always the same ($f_{\text{COUNTER}} = 16\,\mathrm{MHz}$).

The timer runs on the high-frequency clock source (HFCLK) and includes a four-bit (1/2X) prescaler that can divide the timer input clock from the HFCLK controller.

A $16\,\mathrm{MHz}$ oscillator results in the microcontroller having a clock cycle once every 16-millionth of a second. A clock cycle is roughly the time it takes for one instruction cycle (there are exceptions). Therefore, on the Arduino, each clock cycle is $1/f_{\text{COUNTER}} = 0.0625\,\mu\mathrm{s}$.

In timer mode, the timer's internal counter register is incremented by one for every tick of the timer frequency f_{TIMER}. The timer frequency is derived from (2), using the values specified in the prescaler register [2].

$$f_{\text{TIMER}} = \frac{f_{\text{COUNTER}}}{2^{\text{PRESCALER}}} = \frac{16\,\mathrm{MHz}}{2^{\text{PRESCALER}}} \tag{2}$$

Prescaler must only be updated when the timer is stopped. If these registers are updated while the timer is started, then this may result in unpredictable behavior. Timer has a 24-bit counter register which is incremented by 1 every $1/f_{\text{TIMER}}$. The maximum 24-bit value the timer can hold is 16 777 215 (0xFFFFFF16). Therefore, when the timer attempts to count to this value, it "rolls over" to 0. This "roll over" is called an overflow in microcontroller parlance. The counter register overflow can trigger an interrupt. When an interrupt occurs, the Arduino halts execution of the running program and then calls the specific interrupt subroutine. This subroutine is typically called an "interrupt handler." Therefore a timer's counter "overflow interrupt" occurs each time the timer's counter rolls over [12] [13].

Taking the above math into account, this occurs every $1/f_{\text{COUNTER}} \cdot 2^{\text{PRESCALER}} \cdot 16\ 777\ 215$. Consequently, there is also an overflow counter. This overflow handler increments its value and clears the counter register, that is, its internal value is set to zero explicitly, by triggering the CLEAR task.

2) Interrupts: An interrupt's job is to make sure that the processor responds quickly to important events. When a certain signal is detected, an interrupt interrupts whatever the processor is doing, and executes some code designed to react to whatever external stimulus is being fed to the Arduino. It structures the system to react quickly and efficiently to important events that aren't easy to anticipate in software [14].

Whenever we want to do something else in `loop()` besides just reading a pin, instead of just watching that pin all the time, we can farm the work of monitoring that pin to an interrupt, and free up `loop()` to do what we need in the meantime.

`attachInterrupt()` [15] in the `setup()` routine is what sets up the interrupt for the whole system. This function takes three arguments: the interrupt vector, the function name of the interrupt service routine and the interrupt mode.

The interrupt vector determines what pin can generate an interrupt. It's actually a reference to where in memory the Arduino processor has to look to see if an interrupt occurs. A given space in that vector corresponds to a specific external pin, and not all pins can generate an interrupt.

The function name of the interrupt service routine determines the code that gets run when the interrupt condition is met.

The interrupt mode determines what pin action triggers an interrupt. The Arduino Nano BLE 33 supports four interrupt modes: RISING, FALLING, CHANGE and LOW.

3) Ultrasonic sensor implementation: First, we start by attaching pin D2 (one of the available interrupt pins) and D4 of the Arduino Nano BLE 33 to, respectively, the echo and the trigger pin from HC-SR04.

Then, in order to produce a new pulse as soon as the echo signal is received, we take advantage of interrupts. Actually, interrupts are useful for get things done automatically in microcontroller programs and can help solve timing problems.

We begin each distance measurement by clearing the trigger pin condition (from last measure), setting it to value 0 (LOW) for $2\,\mu$s. Afterwards, we set the trigger pin to HIGH (ACTIVE) for $10\,\mu$s, with the purpose of emitting a burst of sonic pulses. Then we put its value back to LOW.

Whenever there is a change in the echo pin value, the interrupt is triggered (`mode` parameter from `attachInterrupt()` function [16] is CHANGE).

Each time the interrupt occurs, ISR (Interrupt Service Routine) `doInterrupt()` is called. This function takes no parameters and

returns nothing. Its only purpose is to change the state of volatile short variable `flag`.

In the beginning of each measurement the echo pin is LOW, and `flag` is initialized with a `WAITING` state (value 0). After the echo pin rises to HIGH, inside the IRS, if the `flag`'s value is equal to 0, then is changed to 1: `RECEIVING_ECHO` state.

When `flag` value is set to one, we measure (for only one time) inside the `loop()` the number of microseconds since the Arduino board began running the current program until this point, using `micros()` function (`time1 = micros()`). During this period and still inside the `loop()`, we measure as much times as possible the voltage of the analogic pin 0, in order to get the average voltage (V_{out}) received from the photodiode presented in the infrared sensor circuit.

When the echo pin's state falls form HIGH to LOW, the interrupt is triggered and, inside IRS, `flag` is set to `SENDING_SIGNAL` state (value 2), assuming its previous value was 1. Right after this, in the `loop()`, we measure the number of microseconds since the Arduino board began running the current program until this point with `time2 = micros()`. This way, the time interval in which the pulses travel through the air can be measured by `Time = time2 - time1`. The distance between the ultrasonic sensor and some object is given by (3).

$$d_{\text{ultrasonic}} = \frac{\texttt{Time} \cdot v_{sound}}{2} \tag{3}$$

When `flag` is equal to 2, we also measure for 20 times the voltage that is being received by the analogic pin 0 for infrared sensor distance measurement. This is done, because in the case the time the echo pin is HIGH is too small, the amount of samples measured from analogical pin 0 is not enough to ensure a reasonable and robust V_{out} measure.

Moreover, when `Time` is higher than 38 ms, the echo pin will timeout, and the program output is "Distance: Out of bounds".

B. I2C

In order to access the accelerometer we use a library called `Wire.h` that assists the handling of serial communications [17]. Actually, the I2C communication with the accelerometer must use this library, by implementing custom read and write functions [18].

We start by implementing a class called `LSM9DS1Class_new` with a set of methods (public and private) that enable communication with the accelerometer [19]. Then, we initialize an object of this class and name it `IMU`.

After that, we do `IMU.begin()` that allows the peripheral device we are using (our personal computer, in this case) to join the I2C bus, with `TwoWire.begin()`. Still in this method, we write to register CTRL_REG8 the hexadecimal value 0x05, i.e, we write IF_ADD_INC bit equal to 1 that means register address will be automatically incremented during a multiple byte access with a serial interface (I2C or SPI) and we write the last bit (SW_RESET) equal to 1 that means device will be reseted (this bit is cleared by hardware after next flash boot). Then we read from WHO_AM_I accelerometer register in order to confirm its value is equal to 0x68. Afterwards, we write to register CTRL_REG6_XL the value 0x70, in order to define that the accelerometer operate in normal mode, the gyroscope is powered down (operating mode), ODR_XL $= 011$ that correspondes to output data rate (ODR) selection of 119 Hz and accelerometer full scale selection of $\pm 4g$ (FS_XL = 10).

After that, we call the `IMU.accelerationSampleRate()` method, that returns the acceleration sample rate, in our case equal to 119 Hz. In order to verify if the accelerometer has new data avaliable we use `IMU.accelerationAvailable()` that reads STATUS_REG register which tells us if there is new data available: if XLDA bit (last bit) is 1 a new set of data is available, otherwise a new set of data is not yet available. We are doing a bitwise AND operation between the contents of this register and 0x01, to get the XLDA bit.

Knowing that there is new data available in the accelerometer, we perform multiple reads starting by register OUT_X_XL (linear acceleration sensor x-axis output register, whose value is expressed as a 16-bit word in two's complement) and ending in OUT_Z_XL (linear acceleration sensor z-axis output register, whose value is also expressed as a 16-bit word in two's complement) to get the values from the 3 output channel from the accelerometer. The raw acceleration data is represented as a 16-bit signed integer (between -32768 to 32767), which is then normalized (divided by 32768) and multiplied by 4 in order to be in the correct range of $[-4; 4]$ g.

In class `LSM9DS1Class_new` we implement methods for reading and writing from/to registers, which are `readRegister()`, `readRegisters()` and `writeRegister()`.

In `readRegister()`, we start by beginning the transmission with `TwoWire.beginTransmission(slaveAddress)`. Then we perform `TwoWire.write(address)` that writes the `address` in the I2C bus. When an address is sent, each device in the system compares the first seven bits after a start condition with its address. If they match, the device considers itself addressed by the master. Then, the address register data will be sent to master register from a slave device in response to a request from a master with `TwoWire.requestFrom(slaveAddress, 1)` (in this case a request of only one register). Finally, it reads a byte that was transmitted from a slave device to a master (`TwoWire.read()`).

In `readRegisters()`, it performs multiple reads using the same logic that we used for `readRegister()`, although this time we request a number of registers to read with `TwoWire.requestFrom(slaveAddress, length)` (since the size of the output data from accelerometer is 3, then we will set `length` as 3). Note that, as we mentioned earlier, register address will be automatically incremented during a multiple byte access. Then, the received data is stored in variable `data` by doing three `TwoWire.read()`.

Finally, for `writeRegister()`, we also begin the transmission using `TwoWire.beginTransmission(slaveAddress)`. Then, in order to write `value` in `address`, we perform `TwoWire.write(address)` and `TwoWire.write(value)`.

C. Infrared calibration including Color Compensation

It is possible to use ultrasonic based measures to compensate infrared response to different refraction rates of objects with different colors.

Contrary to ultrasonic sensors, infrared sensors are highly dependent on the surface color of the reflecting obstacle. Thus, it is expected that for different colors the sensor will produce different output results.

For calibrating the IR sensor, we used different objects with different color and surface smoothness. They were white sheet, black notebook cover, cardboard, white plastic, green notebook cover and blue notebook cover. We obtain the calibration curve for the IR sensor in Fig. 1, using the results in Table II in the attachments.

For the ADC we use the default resolution value of 10-bits that converts 0 to 3.3 V on its input into a digital number from 0 to 1023, that we called $u.m.$ units.

Except for a very absorbent material color (black), the shape of the curve is similar and there is only an offset that should also be compensated. The V_{out} amplitude from the IR sensor is dependent on the reflectivity of the obstacle as expected.

Now we are going to explain our procedure for color compensation, from which we obtained the data for calibrating the distance of the infrared sensor.

For each material, except black notebook cover, we create an individual calibration. It is evident that the IR sensor has non-linear characteristic. So we use the multi-segment curves approach: each segment is a linear regression and store the parameters in a lookup

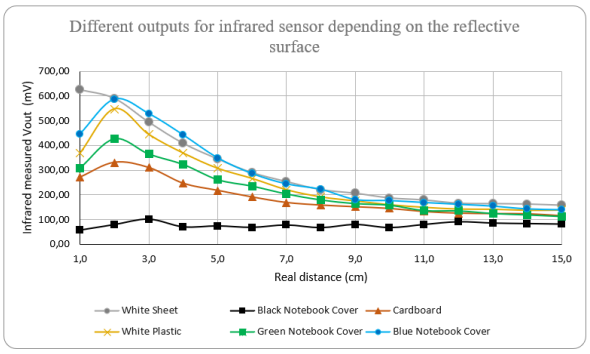

Fig. 1. Calibration curves of different materials for a single IR sensor.

table. For V_{out} that leads to real distances greater than 15 cm we use a power trend-line using all the points and for V_{out} that leads to real distances less than or equal to 3 cm, we use the linear regression of the segment used in range $[2; 3]$ cm. To this end we eliminate some data outliers and bounded the real distance to be non-negative.

Initially, we check if it is a very absorbent material (if $V_{out} < 32u.m. \approx 103.2\,\text{mV}$). In that case, the measurement of the infrared sensor is neglected turning infrared calibrated distance equal to ultrasonic measured distance.

Otherwise, through each of the calibrations for all tested materials, we calculate a candidate for the distance measured by the infrared sensor. Next, for each of the candidates we calculate the absolute value of its error against the ultrasonic sensor measurement. The candidate that minimizes this value is then chosen for the calibrated measurement of the infrared sensor. In this way, it becomes possible to measure distances with the infrared sensor independently of the object characteristics (for the tested materials).

Furthermore, the V_{out} amplitude from the IR sensor depends on the environmental conditions such as sunlight or artificial lights, unless the external source is directly pointed towards the sensor.

D. Sensor Fusion

After testing both sensing systems we come to some conclusions about the optimal sensor for the estimated distance. Ultrasonic sensors are not ideal for edge detection or close distance measurements. On the other hand, infrared sensors are usually more accurate for close distance measurements.

To achieve better precision in the full distance range $[0; 100]$ cm we determine the optimal range for each sensor. For the ultrasonic sensor the optimal range is $[10; 100]$ cm. On the other hand, for the infrared sensor the optimal range is $[0; 10]$ cm, and for distances greater than 20 cm the measurements are completely dominated by noise, and therefore they are negligible.

So, using as criterion the distance given by ultrasonic, we use weighted average for the distance estimation (Table I) where the ultrasonic output has more weight for longer distances and the infrared has more weight for shorter distances.

TABLE I
WEIGHTS FOR SENSOR FUSION IN EACH RANGE

Range [a; b[		Weights (%)	
a (cm)	b (cm)	Ultrasonic	Infrared
0	5	10	90
5	10	25	75
10	15	90	10
15	20	95	5
20	$+\infty$	100	0

E. Additional Functionality - Proximity Detection Alert

As an additional feature, the inbuilt RGB LED in Arduino was used to output a specific color based on the object distance with respect to the sensor [20]. If the object distance is less than or equal to 7 cm, the LED blinks red; for distances between 7 cm and 15 cm it blinks blue; if the distance is greater than 15 cm it blinks green and if the nearest object is out of bounds, the LED blinks a mix of red and blue.

F. Results

Lastly, we present the experimental results obtained (reported in Table II in the attachments) in order to evaluate the performance of the developed unified sensor. In Fig. 2 is the plot of the relative error (%) of the distance measured by the unified sensor as a function of the real distance. We define relative error in (4).

We also tested the unified sensor for distances between 15 cm and 100 cm and we obtained a relative error below 10% for almost all of the measurements.

$$\text{Relative error } (d) = \frac{|d_{\text{measured}} - d_{\text{real}}|}{d_{\text{measured}}} \tag{4}$$

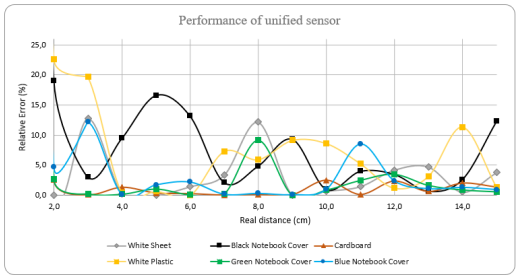

Fig. 2. Relative error (%) of the distance given by unified sensor as a function of the real distance.

IV. Conclusion

In this work, we demonstrate the use of a ultrasonic sensor and a infrared sensor in order to create a unified sensor that produces the highest precision for the full range of tested distances (0-100 cm). We proposed an eletronic circuit aiming to couple the proper functioning of these sensors to their transmission of information to the Arduino platform Nano 33 BLE. We also propose a calibration model for the infrared measurement with the objective of obtaining an estimate for the distance based on the voltage measurement, independently of the object color and surface smoothness. Finally, we test our model in real world conditions (Fig. 3 in the attachments) using a white sheet, a black notebook cover, a cardboard, a white plastic, a green notebook cover and a blue notebook cover. The unified sensor achieved a relative error less than 25% for all tests. For most of the measurements this value was below 10%. In conclusion, the infrared sensor performs better for shorter distances (except for millimeter distances due to the infrared emitter almost touching the object) and the ultrasonic sensor performance is indeed better in longer distances.

Acknowledgment

This work was done in the Technologies of Computing Systems course from Instituto Superior Técnico, Lisbon, teached by Professor Leonel Augusto Pires Seabra de Sousa and Professor Diogo Miguel Bárbara Coroas Prista Caetano.

References

[1] Arduino, "Arduino General Website," Arduino, [Online]. Available: https://www.arduino.cc/.

[2] Nordic, "nRF52840 Product Specifications," Nordic, [Online]. Available: https://content.arduino.cc/assets/Nano_BLE_MCU-nRF52840_PS_v1.1.pdf.

[3] Arduino, "Arduino Nano 33 BLE product page," [Online]. Available: https://docs.arduino.cc/hardware/nano-33-ble.

[4] A. Electronics, "Description of the HC-SR04 Ultrasonic Sensor," [Online]. Available: https://ampere-electronics.com/p/hc-sr04-ultrasonic-sensor-module3/.

[5] 1st Semester 2022/2023, Technologies of Computing Systems, 1st Project, Sensor Fusion for Proximity Estimation, by Prof. Leonel Sousa, Prof. Diogo Caetano, Prof. Ruben Afonso.

[6] Vishay, "High Speed Infrared Emitting Diode, 890 nm," [Online]. Available: https://www.vishay.com/docs/81040/tssf4500.pdf.

[7] Vishay, "Silicon PIN Photodiode TEFD4300," [Online]. Available: https://www.vishay.com/docs/83471/tefd4300.pdf.

[8] T. Instruments, "OPAx340 Single-Supply, Rail-to-Rail Operational Amplifiers," Texas Instruments, [Online]. Available: https://www.ti.com/lit/ds/symlink/opa2340.pdf.

[9] L. Orozco, "Optimizing Precision Photodiode Sensor Circuit Design," Analog Devices, [Online]. Available: https://www.analog.com/en/technicalarticles/optimizing-precision-photodiode-sensor-circuit-design.html.

[10] ST, "LSM9DS1 9-axis iNEMO inertial module (IMU)," [Online]. Available: https://www.st.com/en/mems-and-sensors/lsm9ds1.html.

[11] Reference, Language, Functions, Time, Micros: micros(). Available: https://www.arduino.cc/reference/en/language/functions/time/micros/.

[12] µC eXperiment: "Examination of the Arduino micros() Function". Available: https://ucexperiment.wordpress.com/2012/03/17/examination-of-the-arduino-micros-function/.

[13] µC eXperiment: "Examination of the Arduino millis() Function". Available: https://ucexperiment.wordpress.com/2012/03/16/examination-of-the-arduino-millis-function/.

[14] All About Circuits: "Using Interrupts on Arduino". Available: https://www.allaboutcircuits.com/technical-articles/using-interrupts-on-arduino/.

[15] Arduino Reference, Language, Functions, External interrupts, Attachinterrupt: attachInterrupt(). Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/.

[16] Arduino Forum: micros() on interrupt. Available: https://forum.arduino.cc/t/micros-on-interrupt/538302/7.

[17] Arduino Docs: "A Guide to Arduino & the I2C Protocol (Two Wire)". Available: https://docs.arduino.cc/learn/communication/wire.

[18] Arduino Docs: "Connecting Two Nano 33 BLE Sense Boards Through I2C". Available: https://docs.arduino.cc/tutorials/nano-33-ble-sense/i2c.

[19] Arduino Libraries: "Arduino_LSM9DS1". Available: https://github.com/arduino-libraries/Arduino_LSM9DS1.

[20] Arduino Docs: "Proximity Detection with the Nano 33 BLE Sense". Available: https://docs.arduino.cc/tutorials/nano-33-ble-sense/proximity-sensor.

V. Authors Bio

Afonso Alemão received a Bachelor of Science degree in Electrical and Computer Engineering from Instituto Superior Técnico, Lisbon, Portugal in 2022. He is currently pursuing a Master of Electrical and Computer Engineering at the Instituto Superior Técnico, Lisbon, Portugal. He is also participating at "JEEC - Jornadas da Engenharia Electrotécnica e de Computadores" as a member of the Web Development Team.

Rui Daniel received a Bachelor of Science degree in Electrical and Computer Engineering from Instituto Superior Técnico, Lisbon, Portugal in 2022. He is currently pursuing a Master of Electrical and Computer Engineering at the Instituto Superior Técnico, Lisbon, Portugal. He is also participating at "JEEC - Jornadas da Engenharia Electrotécnica e de Computadores" as a member of the Web Development Team.

Tomás Fonseca received a Bachelor of Science degree in Electrical and Computer Engineering from Instituto Superior Técnico, Lisbon, Portugal in 2022. He is currently pursuing a Master of Electrical and Computer Engineering at the Instituto Superior Técnico, Lisbon, Portugal.

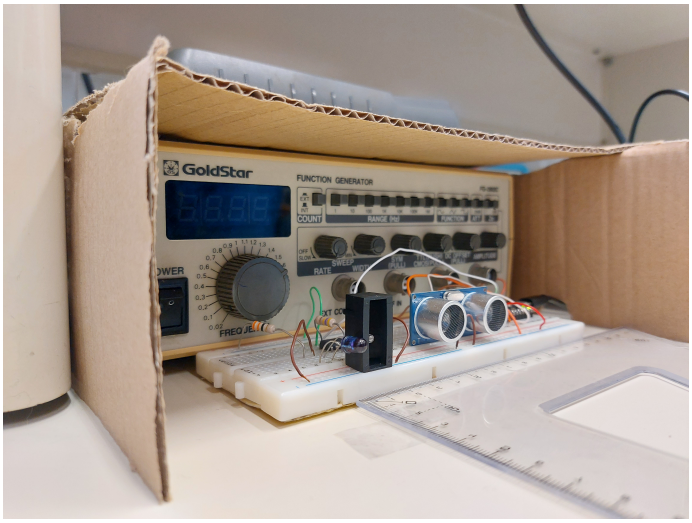

Fig. 3. Testing environment.

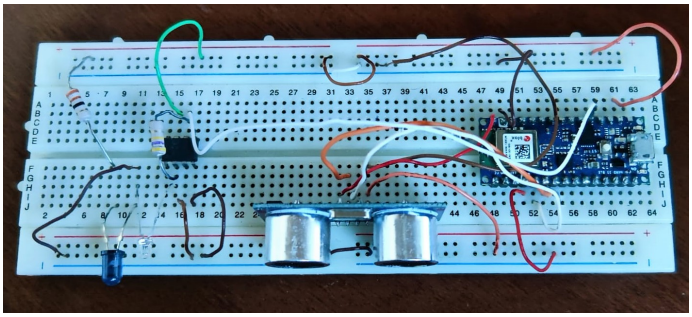

Fig. 4. Eletronic circuit implemented on a breadboard.

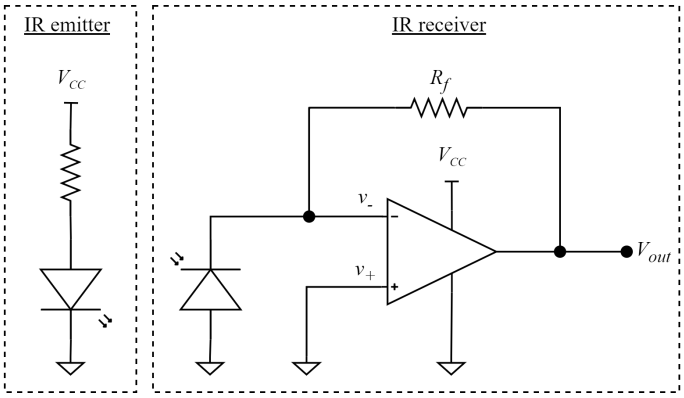

Fig. 5. Schematic of the IR sensor circuit.

TABLE II
RESULTS FOR UNIFIED SENSOR TEST AND CALIBRATION WITH DIFFERENT MATERIALS

White Sheet				Black Notebook Cover			
Real distance (cm)	Ultrasonic measured distance (cm)	Infrared measured Vout (mV)	Distance given by unified sensor (cm)	Real distance (cm)	Ultrasonic measured distance (cm)	Infrared measured Vout (mV)	Distance given by unified sensor (cm)
1.0	2.98	625.81	1.49	1.0	2.76	56.45	2.76
2.0	2.01	590.61	2.00	2.0	2.38	79.03	2.38
3.0	3.29	494.32	3.38	3.0	3.09	101.29	3.09
4.0	3.88	408.55	3.99	4.0	3.62	69.90	3.62
5.0	5.01	345.16	5.00	5.0	4.17	73.74	4.17
6.0	5.69	289.68	5.91	6.0	5.21	67.74	5.21
7.0	6.72	253.42	6.77	7.0	7.15	77.42	7.15
8.0	7.25	220.71	7.03	8.0	7.61	66.52	7.61
9.0	9.07	206.10	9.01	9.0	8.16	79.48	8.16
10.0	10.08	187.19	10.07	10.0	9.90	66.81	9.90
11.0	11.18	178.84	11.16	11.0	11.44	79.45	11.44
12.0	12.45	165.61	12.51	12.0	12.42	90.65	12.42
13.0	13.65	164.00	13.61	13.0	13.09	85.19	13.09
14.0	14.13	162.32	14.08	14.0	14.37	83.23	14.37
15.0	15.69	157.65	15.57	15.0	16.84	81.10	16.84
Cardboard				White Plastic			
Real distance (cm)	Ultrasonic measured distance (cm)	Infrared measured Vout (mV)	Distance given by unified sensor (cm)	Real distance (cm)	Ultrasonic measured distance (cm)	Infrared measured Vout (mV)	Distance given by unified sensor (cm)
1.0	2.52	270.97	3.50	1.0	2.92	367.74	2.95
2.0	2.52	330.65	2.05	2.0	2.50	547.58	2.45
3.0	3.04	309.68	3.00	3.0	3.62	444.10	3.59
4.0	3.46	246.23	3.95	4.0	4.17	368.68	4.02
5.0	4.80	217.74	4.98	5.0	4.66	307.26	4.97
6.0	6.18	191.13	6.02	6.0	6.00	266.45	6.00
7.0	6.96	168.81	7.00	7.0	6.64	220.81	6.49
8.0	8.11	158.42	8.01	8.0	7.61	191.55	7.52
9.0	8.89	151.61	8.99	9.0	9.88	175.71	9.82
10.0	9.72	144.77	9.75	10.0	10.96	159.94	10.86
11.0	10.98	132.26	10.98	11.0	11.68	149.68	11.58
12.0	12.25	125.81	12.28	12.0	12.07	142.55	12.15
13.0	13.09	124.03	13.08	13.0	13.45	141.81	13.41
14.0	13.75	123.42	13.71	14.0	15.66	137.52	15.59
15.0	15.21	115.29	15.21	15.0	15.25	138.32	15.21
Green Notebook Cover				Blue Notebook Cover			
Real distance (cm)	Ultrasonic measured distance (cm)	Infrared measured Vout (mV)	Distance given by unified sensor (cm)	Real distance (cm)	Ultrasonic measured distance (cm)	Infrared measured Vout (mV)	Distance given by unified sensor (cm)
1.0	2.98	308.39	3.02	1.0	2.98	445.81	2.98
2.0	2.54	426.61	2.05	2.0	2.54	586.29	2.09
3.0	3.09	365.16	3.01	3.0	2.52	528.23	2.64
4.0	3.91	323.13	3.99	4.0	4.06	441.94	4.01
5.0	4.46	261.74	4.95	5.0	4.70	349.19	4.91
6.0	6.07	235.81	6.01	6.0	6.33	285.65	6.13
7.0	7.17	204.03	7.02	7.0	6.86	243.81	6.99
8.0	8.66	179.77	8.73	8.0	8.27	221.90	8.03
9.0	8.95	165.13	9.00	9.0	9.11	180.65	9.01
10.0	9.90	158.65	9.91	10.0	9.90	176.29	9.90
11.0	11.27	137.00	11.28	11.0	11.88	168.55	11.94
12.0	11.58	135.65	11.58	12.0	11.68	161.13	11.71
13.0	12.78	125.06	12.78	13.0	12.85	153.97	12.85
14.0	14.13	119.48	14.13	14.0	14.17	142.48	14.19
15.0	14.90	113.16	14.90	15.0	15.19	139.32	15.15