Autonomous Vehicle Project

1st Luca Brodo
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
luca.brodo@stud.hshl.de

2st Oguzkaan Tuna
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
oguz-kaan.tuna@stud.hshl.de

3st Giuseppe Scalora
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
giuseppe.scalora@stud.hshl.de

4st Gordan Konevski
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
gordan.konevski@stud.hshl.de

***Abstract*—The world is moving rapidly towards automation. Machines capable of solving tasks autonomously would make people's life less complicated and effortless. Moreover, they can also help people with disabilities, because machines can handle everyday tasks for them such as going to work or to the shop. A big topic in automation is in fact transportation. The direction in which the industry is moving is towards an always more automated driving experience. The prototype described in this paper is a first attempt of developing a vehicle capable of solving the most basic tasks of driving which are moving in different directions, following a path and avoiding obstacles**

I. Introduction

As a project for the 3rd semester, this group has been asked to develop a prototype of an autonomous RC vehicle

II. Ultra Sound Sensors

In parallel with the infrared sensors at the bottom, the prototype described in this paper collects data from the environment using three ultrasound sensors mounted at the front. The main purpose of these three ultrasound sensors is to detect the presence of the object and send information to the Arduino which will then elaborate the informations and react to it. The sensors used in these project are the ultrasonic rage finders HC-SR04. (Fig. 1)

A. Principle of Operation

The basic principle of these ultrasound sensors is to use sound waves and calculate the distance from an object on function of the duration for which the signal is emitted. The entire process is based on the speed of sound, which is approximately 343 m / s at 20 degrees Celsius at dry air. This specification is necessary since the speed of sound, or a wave in general, is dependent on the material which it goes through. Since sound in this context propagates through air, humidity and temperature are important factors to take into consideration. Air and humidity change the way particles are disposed in space and those are the elements that vibrate and make it possible for the sound to propagate. Although all the previous discussion is true, using the known speed of sound would be enough since the variation in humidity and temperature will not influence it so drastically.

These sensors are essentially composed of 2 parts, a transmitter (T) and a receiver (R). The names themselves are pretty self-explanatory. The transmitter is responsible for emitting the sound waves at 40kHz. Those waves will eventually bounce back from a surface and the receiver is responsible for receiving them. More specifically, HC-SR04 sensors have a pin for the transmitter (Trig) and one for the receiver (Echo) .The other 2 will be ground and 5v.(Fig. 1) At the beginning, those pins need to be both LOW (or 0). In order to start the process, the Trig Pin needs to be set as 1 (or HIGH) from the Arduino using the function digitalWrite. This will make the Transmitter send 8 short pulses to simulate a sound wave. After a delay of 10 milliseconds (achievable with the function delayMilliseconds) the pin has to be set back to 0 again. Once those pulses are sent, the Echo Pin needs to be set as 1 and it will stay in this state until the pulses come back to the receiver, which will then set it back to 0. This can be simply done by the function PulseIn in Arduino. The duration corresponds to the amount of time during which the Echo was set as 1 (Fig. 2). Obviously, the key in this process is timing. The more precise it is, the more accurate the value of the duration will be. Having this in mind, predefined functions such as digitalWrite in the Arduino standard library might not seem perfectly suitable. They in fact hide some lines of code which make the thread a little bit slower and can be substituted by manipulating ports. In this way, the thread will run a bit faster. However, readability would be affected by that and so would be troubleshooting. This bargain might be beneficial in some other context, but not here.

The equation used to calculate the distance once the duration is calculated is the following:

$$S = v \times T$$

Where S is the distance, v is the speed of sound and T the time of the duration. However, this equation must be adjusted in this way:

$$S = v \times Tim/2 \times 10^{-6}$$

Where Tim is in microseconds. The first equation does not take in consideration that the duration outputted by the sensor is in microseconds and that the wave travels half the time for reaching the object and the other half for going back to the sensor. Finally, if we want the distance in centimeters, the final equation will result as follows:

$$Scm = 0.0017 \times Tim$$

Where Scm and Tim are respectively in centimeters and milliseconds

For the main purpose of the project, these ultrasound sensors work perfectly fine. In fact, even though they present limitations such as the angle of reflection and the material of the object, during the tests they behaved most of the times as expected, reacting quite accurate to most of the objects. A problematical situation faced more frequently than the other was with small objects which can not be detected by the sensors. It is also important to note that the range of this sensor is limited both in distance and width. In fact, they can only detect objects in a range of 2 meters and whose position is in their 30 degrees spectrum. This spectrum limitation is the reason why there are 3 of them at the front, situation that create an approximate wide 90 degrees aperture.

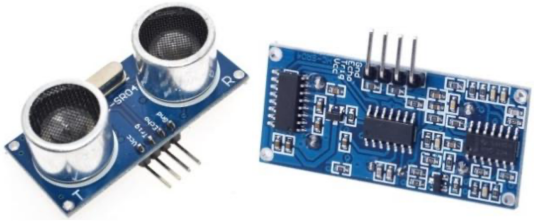

Fig. 1. Picture showing an example of HC-SR04

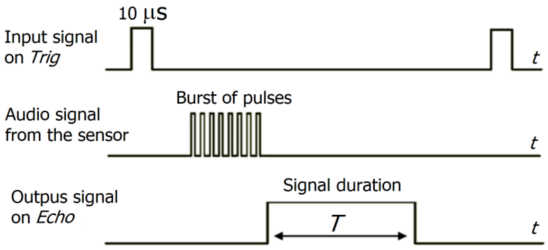

Fig. 2. Representation of how the ultrasound sensors work

B. Implementation of the sensors according to the context

The whole object detection process takes place in a protothread, different from the main one. Arduino is a mono core microprocessor, so the creation of real parallel threads is impossible. A solution to this problem is to use protothreads using the library pt.h [3]. This library provides a conditional blocking wait operation that can be used to call threads which will run until blocked or the execution is finished. This wait conditional blocks make possible to call the threads after a specific time interval. In this scenario, the protothread is called every 50 ms. In addition, the object detection functionality has been implemented having the concept of a state machine in mind. As shown in the diagram in Fig. n (8), there are 4 different states in which the prototype can stay. The central one, namely the default state, represents the state in which the car stay when no objects are detected and it is free to go forward. The other states are related to each sensor. If the distance calculated in one of the sensor is less than 20 cm, then the prototype goes into one of these states depending on which sensors calculates the 20cm distance first. In the diagram, the number ”1” symbolizes that the distance from the object is less than 20 cm, while ”0” refers to a distance inferior to 20cm. If the car finds itself in one of these states, its purposes are to turn 180 degrees in order to avoid the obstacle in order to come back on track and send to the interface which sensor detected the obstacle. In order to do so, a ’if else’ statement has been implemented where the distance is calculated for every sensor. Even though this is not as fast as a switch case (a switch case implementation will have a O(1) asymptotic growth, while this has O(n) in the worst case), it seemed a suitable option since there are not many conditions and both finding errors and trouble shooting result easier. In addition, this implementation suits the diagram thoroughly because the prototype returns to the default state once all the operations end.

III. User Interface

One of the most important part of the project is represented by the user interface. The main purpose of having it is to allow the user to make important decisions and impart commands to the prototype. In addition, the user interface is responsible for notifying important messages to the user. The entirety of the interface was designed having in mind one of the Dieter Rams’ ten principle of design which goes as following : “Good design is as little design as possible”. [1]

The interface is in fact devoid of unnecessary elements, or to quote Rams it is ”not burdened with non-essentials”. What the user sees is a clean and minimalistic interface where elements are disposed in an easy-to-understand way. Both the layout and the design of the elements put at ease also the less experienced user who also does not feel overwhelmed by the amount of objects. The two predominant colors are dark blue and white. The dark blue is used as background because darker color are known to tire less the eye of the viewer. On the other hand, white is used for the elements allowing them to stand out meanwhile creating a nice and elegant contrast with the background.

The first recognizable element is the picture which represents a well-known real car in the middle of the interface .(Fig. 6) The angle of view tricks the eye simulating a real 3D scenario of the car in a road. This ploy is used to display whether or not the car is crossing the street line using the two lines as signifiers. The two figures at the sides of the car which simulates the road turn red if the line is crossed either left or right. In order to avoid confusion, they do not turn red at the same time but they change color corresponding to the side in which the line has been crossed. This whole scenario makes possible even for not-well trained eye to extrapolate information from it and react to them as fast as possible. As a first approach, a 2D top view of the car was taken in consideration. This model had the same characteristics as the first one mentioned. However, it did not reflect the reality as close as the other and it resulted unpleasant to the eyes. A 2D

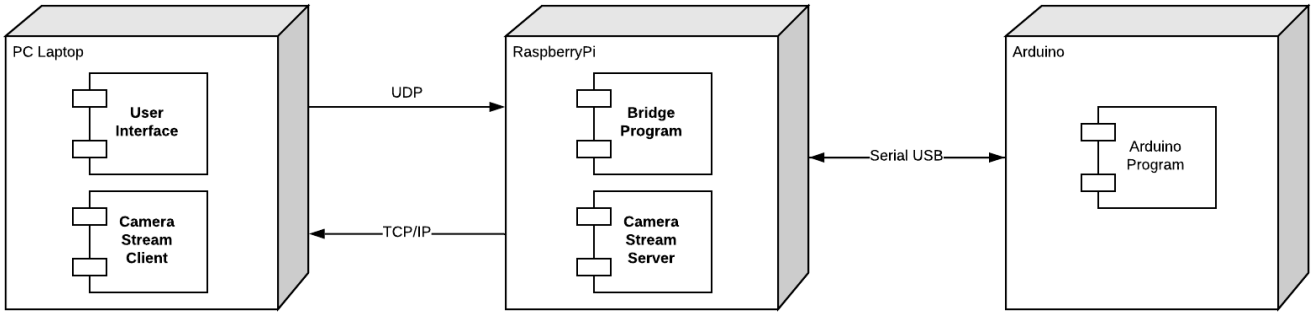

Fig. 3. Diagram displaying the architecture of the system.

top view has been instead chosen to display the location of a near object. An example of it can be seen in Fig. n (Fig. 4). This mechanism is triggered when one of the ultrasound sensors in the front detects an object which is closer than 20cm. The immediate change in perspective is clear and the user immediately recognize it which makes it faster for him to react. In addition, the bright red color of the icon suggests danger and the eye is captured by it. Moreover, this new 2D view of the car is a zoom of the front part of the car and this suggests the object is very close. On the left side of the interface a virtual joystick is placed and it represents the core of the interface (Fig. 5).The joystick is made of two parts. The outer circle, which is a bit darker than the other, represents the boundary while the inner one is the fundament part to give commands to the car. The inner circle is in fact a moving element and the user can directly interact with it by moving it around. The direction in which the inner circle is moved will then be the direction in which the car will go. As long as the inner circle is moved or pressed, the back and front lights of the picture of the car will light up which signifies that the car is turned on and moving. (Fig. 7) When the circle is released, it goes back to its normal position, in the center of the other, the car will immediately stop and the lights will turn off. (Fig. 6) Finally, the other elements that compose the user interface are two switch buttons, a divider, which is a design element and not a usable one, and a progress bar. The two switch buttons are positioned at the sides of the divider. The left one, when switched on, shows the camera view and if switched off, it interrupts it. On the other hand, the right one switches on and off the automatic mode, namely when the car goes around following a line and avoiding obstacles.

IV. Connection between Devices

The purpose of having a form of communication between the three devices is to allow the user to control the movement of the prototype directly via the interface. The basic idea behind it is to use the RaspberryPi as a bridge between the Arduino and the user interface taking advantage of its integrated network card. As shown in the diagram (Fig. 3), the RaspberryPi and the laptop are connected with a Wifi connection using both User Datagram Protocol (UDP) and Internet protocol suite (TCP/IP) protocol. Those protocols have been implemented for two different functionalities.

A. Implementation of the UDP protocol

A type of communication using a UDP protocol is achieved by transmitting information from client to server without verifying the state of the receiver. Packages of data are sent in datagrams which are small in size and fast since the protocol is not designed to check the order of arrival nor the state of arrival. Even though in this way congestions are prevented, the communication may result in an unreliable state since the order can not be predicted. Given these characteristics, the UDP protocol is suitable for sending directions and commands to the vehicle. Using the available class in the qt library called “QUdpServer”, the laptop is able to open a socket-type communication with the Raspberry Pi. The user interface presents to the user a virtual joystick which would be used to give directions to the car. This area is divided in 8 different equally large zones (right, forward-right, forward, forward-left, left, backwards-left , backwards and backwards-right) and moving the inner circle of the joystick all the way to a specific area results in the program sending the corresponding characters to the RaspberryPi. This program will then receive the character and transmit it to the Arduino via serial communication. The Arduino will then map the characters to the specific position using a switch case. This is very easy to implement and very fast to execute because a switch case will be compiled in a jump table. However, the way this function is implemented is an oversimplification of what it really should be. As a matter of fact, the best implementation would be to map the position of the inner circle in the joystick with a function that gives as output direction and speed. Another functionalities which uses the UDP protocol is sending data from the Arduino to the Qt program. When the Arduino detects an object with one of three ultrasound sensors in the front, it sends to the RaspberryPi the position using the three characters ‘r’, as in right, ‘c’, as in center, and ‘l’ as in left. The RaspberryPi will then send them to the computer, which is constantly checking those in a designed thread, and then display the position of the object in the interface.(Fig. 4)

Fig. 4. Example of the interface showing where the object is.

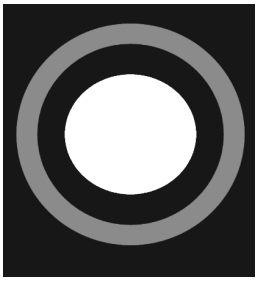

Fig. 5. Picture showing the joystick situated in the left part of the interface.

Fig. 6. Representation of the car in the interface

Fig. 7. Detail of the car when the controller is moved

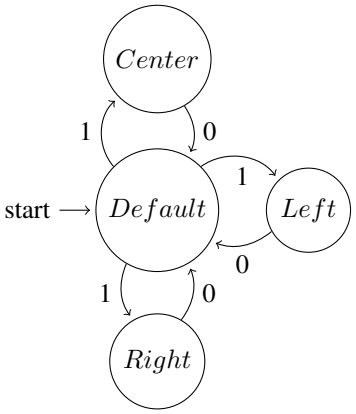

Fig. 8. Figure displaying the states in which the car can stay. "1" symbolize that the distance from the object is less than 20 cm, while "0" refers to a distance inferior to 20cm. "Left","Right" and "Center" are the states triggered when a object is detected in the corresponding position

B. Implementation of the TCP protocol

When it comes to streaming the camera, however, the UDP protocol is hard to implement given the the fact that the image needs to be fragmented into packages and those might get lost. A better approach would be to use the TCP protocol. This protocol allows to stream heavier packages whose order is checked every time. This makes the connection more reliable but a bit slower than the one with UDP protocol, hence more suited for a camera stream. The program utilizes the OpenCV library which requires to be installed in both the raspberryPi and the laptop. Using OpenCV, the raspberry Pi is able to capture data from the camera frame by frame, store it in a "cv::Mat" type and send the data to the TCP socket. Once the data has been entirely collected by the laptop, the picture needs to be reconstructed pixel by pixel. It's a very intricate process, which however allows a wide degree of freedom. In fact, once the picture has been reconstructed, it can be processed. For example, the program so far is able to find contours and draw them. More importantly, this implementation can be embedded with the interface since both utilize C++.

REFERENCES

[1] Norman, Donald A. The Design of Everyday Things. New York: Doubleday, 1990. Print.
[2] Zhmud, Vadim and Kondratiev, N and Kuznetsov, K and Trubin, V and Dimitrov, Lubomir,2018, Application of ultrasonic sensor for measuring distances in robotics, 1015,Journal of Physics: Conference Series,
[3] Dunkels, Adam and Schmidt, Oliver and Voigt, Thiemo and Ali, Muneeb, Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems,2006