***Raspberry Pi 3 and Camera Module V2 - Live Streaming***

One of the requirements of the project is streaming a live video from the vehicle prototype to any given computer, and for that, a Raspberry Pi 3 with one of its accessories (Camera Module V2) was used. Below, the workflow steps that led to the fulfillment of the requirement are explained.

## I. RASPBERRY PI 3

Raspberry Pi 3 is a small single-board computer for which a standard keyboard and mouse, a monitor and a power supply is required in order to use it. This device operates in different programing languages, like Scratch or Python, and can execute every functionality just like a desktop computer.

## A. Installing the operating system on the microSD card

## Just like any other computer, Raspberry Pi needs an operating system to function. The official operating system ‘Raspbian’ was used on the Raspberry Pi. After downloading the image, the next step was installing it on the card, and for that, ‘balenaEtcher’ was used, as a graphical SD card writing tool.[1] Therefore, the microSD card was formatted and made ready to operate on the Raspberry Pi.

## B. Configuration

On the first booting into Raspbian, after running on the

Terminal the command line ‘*sudo raspi-config’,* a new tab with a few options available appears (*Figure 1*). The main functionality of this command is providing the most common configurations. Moving down in this menu bar, it is possible to change options related to network, booting, localization or interface.[2]

II*.* *CAMERA MODULE V2 CONFIGURATION*

The V2 Camera Module has a Sony IMX219 8-megapixel sensor. It can be used for taking pictures, live streaming as well as recording videos.

The configuration starts by inserting the ribbon cable in the slot situated between the Ethernet and HDMI ports, while making sure the silver pins are facing the HDMI port. Then, the ‘*sudo raspi-config’* command line can be run, and after navigating down the menu, under the ‘*Interfacing Options*’, the camera can be enabled. Hence, a reboot of the Raspberry Pi is needed.[3]

Going back in the Terminal, it is now accessible to run a few command lines like: ‘*raspistill*’ (for capturing pictures) or ‘*raspivid*’ (for recording videos).

III. *TCP/IP NETWORKING*

TCP/IP is a group of network protocols. To identify themselves, computers participating in this network use IP addresses. In order to configure TCP/IP across all of its network interfaces, Raspberry Pi uses ‘*dhcpcd*’ which means assigning each interface an IP address as well setting netmasks and configuring DNS resolutions via the Name Service Switch facility. The ‘*ip link*’command line can be used to find the names of the interfaces that are present on the system.[4]

IV. *LIVE VIDEO STREAMING*

The Raspberry Pi camera should be able to stream live videos into a web page that is accessible in any device that has a browser and is also connected to the same network as the Raspberry Pi.

After making sure the camera is enabled in the system preferences, the camera should be connected to the specific port. As a next step the ‘*if config*’ command is run, which displays several information, including the IP addresses of the Raspberry Pi, that will be needed for the streaming. Thereafter, the command line ‘python3 rpi\_camera.py’ is used to run a prewritten script on Python.[5]

Finally, once the script is running, the ‘http://your-pi-address:8000/’ webpage can be used to view the video streaming.

A close up of text on a white background

Description automatically generated

*Figure 1*

(Workflow Diagram)

V. *I2C COMMUNICATION BETWEEN THE RASPBERRY PI AND ARDUINO*

The Inter-Integrated Circuit (I2C) is a hardware protocol which offers flexibility to interface slave integrated circuits with one or more masters. For the communication it uses two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors. Usually, the voltages used are +3.3V or 5V.[6]

It was decided to operate with this protocol because, for example, compared to other alternatives, like the SPI protocol, it has several advantages.

Firstly, I2C requires only two cables to connect to any number of slaves, whereas the SPI needs four of them for the same communication.

Secondly, I2C supports a multi-master hardware connection, which was useful considering it was worked with multiple sensors on the vehicle and therefore, they can be accessed and controlled by different master devices.

Using this protocol, the Raspberry Pi will behave as a single master, which means it will take the data from the user on the terminal, and then send it to the Arduino Uno, which will serve as a slave, thus, it will receive data at the I2C interface and display it on the Serial Monitor.

Below, is presented a picture (*Figure 2*) that shows how the connection was realized, by using an LED as an indicator to demonstrate the communication, by sending commands from the Raspberry Pi, to the Arduino, in order to turn the LED On and Off.

A circuit board on a table

Description automatically generated

*Figure 2*

***UML Diagrams***

*VI. USE CASE DIAGRAM*

The use case diagram is a simplified representation of the interactions the user can execute with the system and the relationship it can build with different use cases.

Four main use cases that the target user can interact with, were identified. (Figure 3)

1. *Interaction with the Qt Interface*

A simple graphical user interface was created for the vehicle, using the Qt framework. The development started by doing multiple paper prototyping and testing them on different people, outside of the working team, to evaluate the user experience. After deciding on the prototype that would be used as a base for the interface, the following step was the graphical design.

The Qt interface includes: a small display for the video streaming; a joystick which is used to accelerate the vehicle, as well as choosing the direction; a button for turning the lights ON/OFF; a real time feedback box which informs the user about activities happening at a specific time; a menu bar for different settings; a speedometer and a status bar (for e.g. the wi-fi signal, current battery charge, etc.).

*2*. *Manual driving*

*The* Qt interface provides the option of choosing the driving mode, i.e. whether you want to drive it manually or not - in which case, alternatively, the car will drive autonomously. When in manual mode, the user would, ideally, be able to drive the car manually by using the joystick, but this has yet to be implemented in the future improvements of the car.

3. *Autonomous driving*

The user can choose the autonomous driving option, hence, two main functionalities of the car are enabled: the line tracking assisting (realized by the infrared sensors) and the collision detection (realized by the ultrasound sensors). Therefore, the vehicle is capable of not only following the path and keeping track of the black lines, but also detecting obstacles and preventing crashing into them.

4. *Camera usage*

As presented on the Raspberry Pi and the Camera Module V2 section, there are several interactions accessible by the user, like capturing pictures recording videos or streaming a live video from the vehicle to the Qt interface.

A close up of text on a white background

Description automatically generated

*Figure 3*

(Use Case Diagram)

*B. State Machine Diagram*

The purpose of the state machine diagram is to present the behavior of the system through different state transitions. While developing, it is very important to know the current state of the system and what kind of activity is exactly happening there.

The diagram (*Figure 4*), starts with the first state *‘Motor OFF’,* and after switching it on, it proceeds to the ‘*Motor On*’ state, where the car is ready to execute its functionalities.

Initially, the vehicle starts checking the surroundings (‘*Check 1*’ state), using the ultrasonic sensors, in order to prevent collisions to any obstacle. In this state, three decisions are available:

1. If an obstacle is detected, the car will go to the ‘*Car Stopped*’ state, wherefrom, it continuously jumps to the previous state, while checking for a change that will initiate another decision;

2. If there is no obstacle detected, the car will proceed to the ‘*Check 2*’ state (explained below);

3. At any time, the user can choose to switch the car off, thus, the car would go back to ‘*Motor OFF*’ state.

On the ‘Check 2’ state, the infrared sensors play their role for assisting the vehicle to ‘read’ the black line, by constantly checking from left to right. Hence, the car stays on track and follows the path. Just like on the ‘*Check 1*’ state, in this state, it is also available to switch the car off.

There is a constant loop that jumps from ‘*Check 1*’ to ‘*Check 2*’ state very fast, which makes them run almost in parallel, by making sure the car, not only stays on the line, but also prevents crashing into obstacles on the way and responds in a real-time manner.

A close up of a map

Description automatically generated