Smart Firetruck

Reiner Katharina
University of Applied Sciences Hof
Hof, Germany
katharina.reiner@hof-university.de

Wilfert Luca
University of Applied Sciences Hof
Hof, Germany
luca.wilfert@hof-university.de

Raithel Tim
University of Applied Sciences Hof
Hof, Germany
tim.raithel@hof-university.de

Schlitter Moritz
University of Applied Sciences Hof
Hof, Germany
moritz.schlitter@hof-university.de

Manig Ralf
University of Applied Sciences Hof
Hof, Germany
ralf.manig@hof-university.de

***Abstract*—As part of the lecture "Applied Robotics" at the university of applied sciences Hof, an intelligent firefighting robot is built. The robot is supposed to drive autonomously to the destination, where it should recognize a picture of a flame, then put out that flame with water and lastly return to the fire station. This functionality is implemented based on the chassis of an RC car with a single board computer that uses the input of a camera to navigate. The car contains a water tank with a pump to spray the water on the flame picture. Additionally, the vehicle has working blue light and can play a siren sound.**

Keywords— intelligent, robot, extinguish, firefighters

I. STATE OF THE ART

There were already multiple projects done at the university of applied science Hof, in which the same type of electric vehicle was used. Reference [1], the practical work „Aufbau und Implementierung eines Elektrofahrzeugs mit REST API zur Steuerung und Kamerabildübertragung", written by Daniel Hanik, on which this project is based on, describes steps to build up the car. Amongst other things, the paper includes the structure and configuration of all the hardware components, the installation of the operating system and the software components, as well as the Python code that controls the servomotors. The steps of these paragraphs can be adapted to fit this project.

II. APPROACH

A. Goal

The goal of this project is to build an intelligent firefighting robot, based on the vehicle, camera, and Nvidia Jetson Xavier NX, provided by the university. The robot is supposed to look like a firefighter truck with working flashing blue light and siren sound. When the user manually activates an alarm, the vehicle autonomously follows a yellow line on the floor, by evaluating the camera input and controlling the servomotors accordingly. During that time, two blue LEDs are flashing alternately, and the siren sound file is played via a small Bluetooth speaker. Concurrently, an AI application for image recognition monitors the camera input, looking for a previously determined symbol of a flame or a fire station. When the AI application recognizes symbol of the flame, the vehicle stops and begins to put out the flame. This is accomplished by activating a small water pump inside a water tank within the body of the car for a few seconds. The water will spray out of a jet pipe mounted on the roof of the truck. After the vehicle finished this job, it will switch off the blue light and the siren. It then autonomously follows the yellow line back, until the AI application recognizes the fire station symbol. Here the vehicle will come to a hold and the process is finished.

In case the implementation of the goals stated above can be finished early in the time space of the project, there are some additional goals, which can be implemented afterwards. For example, the signal to start the whole process could be send autonomously by another image recognition software that sends an alarm when recognizing a flame symbol. The user would not have to start the alarm process manually. Furthermore, the robot's ability to navigate could be enhanced. One option is installing multiple possible destinations, that means the robot would have to decide where to go when encountering a crossing in the yellow line. Autonomous parking in a defined area or avoiding obstacles on the yellow line are additional extensions. On top of that, aiming with the water jet while putting out the flame can be enhanced. The vehicle would position itself autonomously so that the water lands in the desired container. Additionally, the image recognition software can be expanded to recognize different types of fires and put out a message containing the correct method to extinguish this kind of flame.

B. Steps

1) Preparing the hardware

a) Installing the Operating system

To install the operating system, a computer with a SD-card slot and a SD-card is needed. The Jetson operating system image is downloaded from NVIDIA and the SD-card is formatted with the "SD Memory Card Formatter" program's "quick format" option. The image is flashed onto the SD-card with the "Etcher" program. After the program has flashed the image and verified its integrity, it can be inserted into the SD-card port on the Jetson.

Now the initial setup can be conducted. A monitor via HDMI, a keyboard and a power cable are connected. The Jetson now powers on and boots. The NVIDIA Jetson software EULA must be reviewed and accepted. The system language, keyboard layout and time zone are configured. A user is created, and the hostname is setup. The partition size is set to the recommended value.

The official guide from NVIDIA was used [2].

b) Move Operating system to SSD

For Performance reasons, right after flashing the SD-card and initially setting the Jetson up, the boot process is changed so that the Jetson only uses the SD-card to boot the image and runs on the SSD afterwards.

Therefore, a guide was used [3].

c) Installung the software prerequisites

To use the camera, the corresponding software 'Pyrealsense2' is needed. Because there are no prebuilt

978-0-7381-4484-9/21/$31.00 ©2021 IEEE

binaries for ARM processors like the one the Jetson has, it must be built from source. [1]

Additional required python modules were installed with the package manager pip.

d) Chassis

As in the practical work referenced above, the given chassis with the pre-mounted servomotors is used. Additionally, a wooden platform was placed on four stud bolts on top of the chassis. This platform serves as a foundation for the 3D-printed body parts, which give the vehicle the look of a firefighter truck. The body is composed of three parts that are fixated on the platform with multiple small blocks of wood that are glued on the platform. The blocks prevent the body parts from slipping horizontally, but the parts can still be removed by lifting them.

The front part is used as a mount for the camera so that the camera points downwards at a 45° angle.

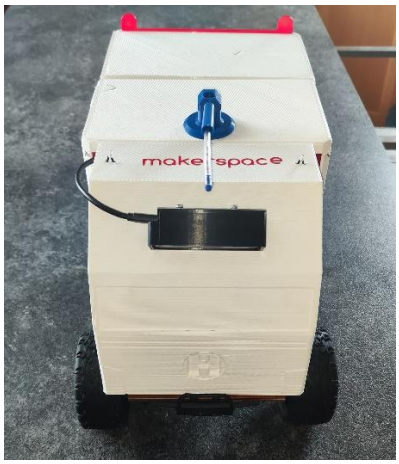

Fig. 1. Chassis front view.

The middle part contains the water tank and has lids on both sides, to grant easy access to the tank. The water tank itself is a lunch box, with the water pump inside. There are two holes in the tank to allow the cable and hose from the pump to run to the outside. From there the hose runs through the roof of the middle part, inside a 3D-printed jet pipe. In the front of the hose, there is a small 3D-printed outlet, which reduces the diameter of the hose. This allows greater range of the water jet.

The back part contains electronic devices like the Jetson, the voltage converter and the Bluetooth speaker. It is separated from the middle part to prevent water from leaking near the electronics. A trunk lid allows easy access to the electronic parts.

Fig. 2. Chassis side view.

e) Connecting electronics

Following the previously mentioned practical work [1], the Nvidia Jetson Xavier NX as well as the servomotors are powered by the given 4200mAh battery pack. Since the Jetson requires an input voltage of 18-19 volts, a voltage converter was used as described in the paper. The ground (GND), power supply (VCC) and signal cables of the servomotor used for steering are connected to the corresponding pins of row 2 on a servo driver, while the cables of the electronic speed controller are connected to the first row similarly. The servo driver itself is connected to the Jetson as described in [1]. The ground pin is connected to any ground pin of the Jetson, the SCL and SDA pins are connected to the corresponding pins of the I2C1 port (pin 5 and 3) on the Jetson and VCC is connected to one of the two 3.3V power supply pins.

The camera and the water pump are both connected via USB. It is important that the pump is connected to one of the two outputs of the USB hub closer to the power supply (port 1 or 2), while the camera is connected to either USB port 3 or 4. This is because, in order to switch off the pump, the whole USB hub is cut from power and this would affect the camera as well if connected to this hub.

The Jetson can establish Bluetooth connections, which is used to communicate with the small speaker, which plays the siren sound.

The two blue LEDs are each connected to a ground pin with their black cable. The red cable of one LED is connected to pin 11 while the other LED is connected to pin 12 with its red cable.

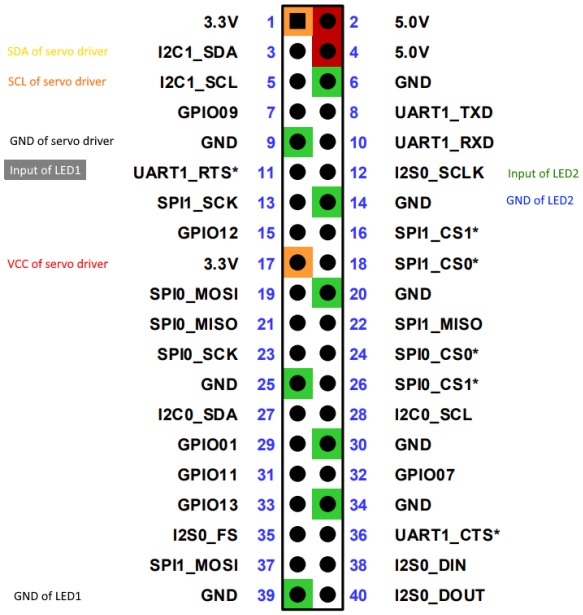

Fig. 3. Jetson Xavier pin layput [1].

2) Individual hardware control

a) Flashing light

To implement the flashing light of the firetruck, there are two blue LEDs on top of the car body. The LEDs have integrated resistors and are addressed by powering the I/O pins of the Jetson on and off. They are wired to two different I/O pins and a ground pin each.

The default state of both LEDs must be off. When powered on, they flash alternately. For this, there exists a python script

that uses the “RPi.GPIO” library, which wraps addressing the single pins.

b) Water pump

The fire extinguishing process is implemented by using a small USB-powered water pump that pumps some water from a small tank via a hose to a nozzle. For addressing the water pump, one of the USB-ports of the Jetson is used.

By default, the pump must be powered off. This is achieved by running a bash script using “uhubctl” that powers off the used USB-port before plugging in the pump.

If the Robot is commanded to release some water, there is another script that turns the respective USB-Port on. After enough water has been pumped out, the same script as in the beginning can be used again to turn the pump back off.

c) Output siren via loudspeaker

The implementation of the siren is achieved by using a small Bluetooth speaker which plays a mp3 file of a siren.

First, the Jetson must be initially paired with the speaker via Bluetooth. This is a one-time configuration task in the beginning which must be performed manually.

After the devices have been paired once, the connection-process can be automated via the command “bluetoothctl connect”. This process is supported by another script, checking regularly if the device is still connected.

For playing the mp3-file, the “playsound”-library is used in a python script.

3) Camera

a) Path guidance

The firetruck will be guided to its destination via a yellow line on the ground. To steer it accordingly, the camera mounted on the front of the firetruck will be used to detect the yellow line. This is achieved by finding the largest area of yellow pixels in the frame.

To do so, some preprocessing of the frame is necessary. First, the RGB camera stream is converted into the HSV (Hue-Saturation-Value) color space. This is done to isolate and find the yellow pixels more easily. Then, binary masks of the regions with yellow pixels are created. By applying morphological operations, noise and minor gaps are cleaned up to get smooth and solid edges of the detected yellow area for improved further processing and visual representation.

After the preprocessing, the largest contour is selected, and the center calculated. Depending on its position relative to the center of the camera frame, notifications with instructions of how the vehicle needs to steer are sent. A stop message is sent if no yellow line is detected for 1,5s.

b) Detecting start- and endpoints

For detecting the stopping points of the firetruck, an AI-object-detection model was trained using Roboflow.

The model was trained on two classes each consisting of a symbol, the first symbol being a fire, which is the destination of where the firetruck needs to go to put out the fire, and the second symbol being a fire station, where the firetruck will return to after all fires are extinguished. These symbols were printed out and various pictures were taken of them from different angles, with different lighting and from different distances.

Roboflow was utilized for the entire process of preprocessing the images and training the model. 358 pictures were labeled and normalized. Additional images were created using data augmentation and the model was trained on 500 images in total. With a precision of 97.1%, the model was deployed via the Roboflow API, which the firetruck calls. Depending on the result of the model, appropriate notifications including the information of which destination was reached are sent.

4) Integration

a) Wrapping the components with shared base class

The functionalities for controlling the individual components are summarized below. To standardize their use, all controlling elements inherit from a common base class Component. The requirement to be able to be started and stopped or interrupted, applies to all controls. For example, if the car drives off to a fire destination, the blue lights and siren must be switched on. However, as soon as the destination is reached, both must be stopped.

The base class “Component” encapsulates the usage of a thread including its start and stop process. This enables the GlobalController to manage the standardized interface according to the application.

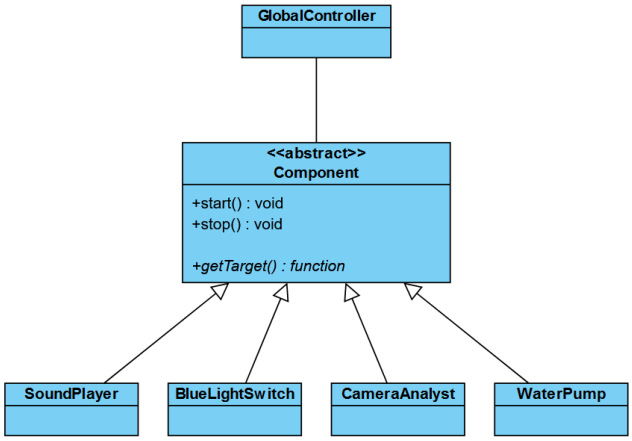

Fig. 4. UML diagram of the Component class und its subclasses.

The previously implemented processes for controlling the hardware are each integrated into an encapsulating class. The logic is transferred to a function which is used as the Component’s thread’s target. Additional cleanup code before stopping the hardware control can be linked into the stop process. For example, the flashing of the LED first ends with a LOW output before the thread is terminated.

b) Identifying the destinations

Ids are used to define the destination to be reached. To differentiate between the journey to the fire destinations and the return journey, the home ID is defined as 0. The controller can now use this information to determine whether privileges are required for a journey or not. This attribute is defined as follows:

If the destination ID is that of a fire destination, the blue lights and siren must be switched on during the journey. As soon as the destination is reached, these components stop. In return, the water pump is activated. The recognition of a successful extinguishing is simulated by a three-second wait. The pump is then no longer activated. Finally, the car reverses and the home ID is activated.

If the destination ID is the known home ID, neither privileges need to be switched on during the journey nor an extinguishing process started when the destination is reached.

c) Wrapping the REST-API

The existing REST interface [1] is controlled by an API adapter. This encapsulates the sending and configuration of the requests and only forwards the corresponding responses. In particular, the logically correct use of right, left and central steering is ensured.

d) Communication with Control-API

The car is still steered by the camera analysis. To be able to pass on current observations to the GlobalController, it offers a communication interface. A dedicated method is available for each possible event.

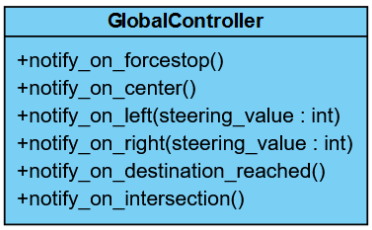

Fig. 5. UML diagram of the GlobalController methods.

These process the signal in each case. If a direct stop or steering signal is detected, corresponding POST requests are sent with the API adapter. The camera analyst continuously sends observations to the controller. To avoid overloading the REST interface, the latest messages received are cached and only new signals are forwarded to the API.

An interface for processing the detection of an intersection is also available for further development in the future. The signal is forwarded to an IntersectionGuide which would internally determine the direction to take and return a corresponding instruction to the controller.

If the destination is detected, the necessary follow-up actions are carried out as described in B.

e) Usage

In order to use the built environment, first of all the REST-Api to control the servo-motors must be started. Next, the script controlling all components must be run with sudo rights in order to have the needed access to control hardware components. This script instantiates the global controller and manually triggers the alarm. Due to python package interoperability version 3.11 must be utilized for running both scripts with the currently installed packages on the Jetson.

When the car's IP address changes, this must be applied in the 'config'-file, which is used by the ApiAdapter to communicate with the REST-Api.

III. Experiments

During the development process, multiple experiments to test the robot were conducted to make sure that every hardware and software component work both individually and together.

This included individual tests for controlling the flashing lights, enabling and disabling the USB-port to which the water pump is connected, which was visualized by using a USB-lamp, outputting the siren sound via loudspeaker, detecting the yellow line and detecting the two symbols the AI was trained on for the start- and endpoints.

After testing the components individually, everything was put together to be managed by the GlobalController and the first driving tests with the chassis were conducted. The setup for these tests were as follows: The printed-out symbols for the fire station and the fire were laid on the ground, with a curved yellow tape line connecting the two. The REST-API of the car and the GlobalController were run to start the car.

With these tests, the steering was modified for the firetruck to follow the line more accurately, and any communication errors between the individual software components and the GlobalController were ironed out.

However, the 3D-printed body of the firetruck and the water tank made the firetruck too heavy, which resulted in worse steering and the battery failing to move the vehicle continuously, therefore a ball roller was added to relief a bit of weight.

The final acceptance test is conducted in a more real environment outside. Unfortunately, the combination of the rougher ground and the weak battery resulted in the firetruck having to be supportively pushed to drive. Everything else worked well, as indicated by the following table.

Test	**Components**				
	Blue lights	***Water Pump***	***Siren***	***Path guidance***	***Start- and endpoints***
successful	✓	✓	✓	✓	✓

The final acceptance test reached all the initial goals of the project.

IV. Conclusion

This project built and implemented an intelligent firefighting robot based on a vehicle with a camera and a Nvidia Jetson Xavier NX. After being activated, the truck follows a yellow line with active blue lights and a siren, until a fire symbol is detected and puts out the fire via a water pump. Having finished, it drives back with no flashing blue lights and siren, until a fire station symbol is detected, where the process is finished.

Unfortunately, due to the many hardware complications and errors, none of the additional goals were fully implemented in time. However, because the infrastructure of the project is kept expandable, additional features can easily be added.

A future project could add the following features: Autonomously starting the process. To do so, a REST-Api connecting to the global controller was already set up. Dealing with multiple destinations and crossings. The existing interface for processing the detection of an intersection can be used. In the intersection guide a map needs to be set up manually or automatically. The existing structure expects for each destination id a list of the consecutive direction commands for each intersection. Further extensions can be: Implementing autonomous turning back after the fire is extinguished and autonomous parking after the fire station is reached. Implementing the detection of obstacles and according steering around them. Implementing better aiming of the water pump by additionally centering the firetruck in front of the fire symbol so the center of the fire is hit. And

lastly, implementing the detection and corresponding extinguishing of different types of fires.

[1] D. Hanik, “Aufbau und Implementierung eines Elektrofahrzeugs mit REST API zur Steuerung und Kamerabildübertragung,” Universtiy of Applied Sciences Hof, 2023.

[2] “Getting Started With Jetson Nano Developer Kit,” *NVIDIA Developer*, Mar. 05, 2019. https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro

[3] kangalow, “Jetson Xavier NX - Run from SSD,” *JetsonHacks*, May 29, 2020. https://jetsonhacks.com/2020/05/29/jetson-xavier-nx-run-from-ssd/ (accessed Feb. 11, 2024).