...

Smart City Project Using MQTT

– Interaction Design –

July 3, 2020

Springer Nature

Contents

Chapter 1
User in Smart Cities and new HMI

written by
Katrin Glöwing, matriculation number: 2170348

Abstract This project is about controlling some system adjustments in the car by using hand gesture recognition.
For that project it is necessary to train a neuronal network and implement a program which uses the camera as an input device to recognise and detect the hand gestures. This problem will be solved with an VGG-16 model with four additional dense layers on top and a cross-validation of two different data sets.
This program will help drivers to be more concentrated on the road and could also be an approach for intelligent systems which understands sign language.

1.1 Introduction

In case of human machine interface is gesture control, means tending a hardware with gestures, particularly with hand gestures, growing in importance. The contact-free gesture control is aimed at a more intuitive interaction between user and machine [1]. Furthermore contact-free systems avoid the transmission of microbes. This is especially in the medical sector and in the care sector beneficial [2].
Figure 1.1 compares the protected global gesture recognition and touchless sensing market size in 2014 and 2020. This statistic was published in 2014. As it is shown in Figure 1.1, the market size worldwide was in year 2014 at 5,15 billion U.S. dollars while the projected market size in year 2020 is more than a quadruple of the size in 2014. The value of the projected global structure recognition and touchless sensing market amounts to 23,55 billion U.S. dollars.
The next section will presents some related works. The following section shows the concept including the goal, the requirements and the use cases. After that, the implementation is explained. First the used model and then follows some parts of the implementation with explanation. The results will be shown with the aid of a

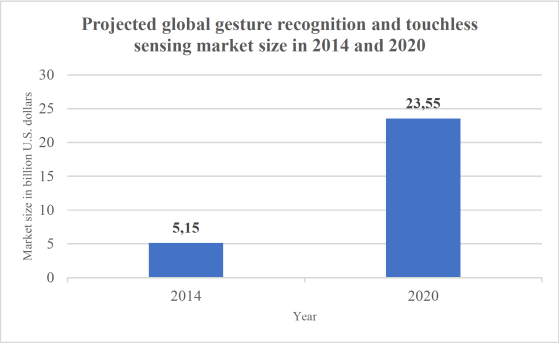

Fig. 1.1 Statistic: Projected global gesture recognition and touchless sensing market size [3].

program showcase and by showing which of the before defined requirements are implemented. In the end there will be the conclusion including some potential future works and finally the references.

1.2 Related works

Currently (2019) gesture control appears for example in the entertainment and the automotive sector. Until now is the gesture control among others recover in *Microsoft Kinect* and *Nintendo Wii* or also in TVs from *Samsung* [4].
With this hardware it is possible to control actions on the television screen or rather operate the TV by hand gestures from the user [4].
In case of using gesture control in medical sector, it is possible to instruct robotic mechanism to perform complex surgical procedures by using hand gestures as commands [5]. One approach, published in 2008, uses the hand gestures from the surgeons to take control over digital images. This means, it is possible to influence the digital images while an operation contact-free [6].
In the automotive industry *BMW* applies since 2016 in their *7-er series* for example gesture control for diverse functions [4].
This project corresponds to the automotive sector.

1.3 Concept

This report will explain a program for gesture recognition.

1.3.1 Goal of the project

The goal of the project is to implement a system, which recognizes hand gestures from the user to control some system adjustments in the car.

1.3.2 Requirements

Figure 1.2 represents the requirements of this project. In summary there are seven requirements, four functional (Type: F) and tree non-functional (Type: NF) requirements.
Functional requirements are requirements which describes what a system or product must execute [7] and non-functional requirements describes the fundamental basics of the systems architecture [8].

Type	ID.	Description
F	R1	The system must recognize **three different hand gestures**.
F	R2	One gesture must make a **request to the server** (calling police).
F	R3	The system must have trained a **neuronal network** (maybe convolutional).
F	R4	The **output** must be the **name of the gesture**.
NF	R5	The system must be written in **Python**.
NF	R6	The **input** device should be a **webcam**.
NF	R7	The system must use the **OpenCV library**.

Fig. 1.2 Requirements of the project.

The system must recognise three different hand gestures (ID.: R1) and show the name of the identified gesture (ID.: R4) by using a webcam as an input device (ID.: R6). One of these gestures must makes a request to the server to call the police (ID.: R2). The system must have trained a neuronal network (ID.: R3), must be written in the program language Python (ID.: R5) and must use the OpenCV library (ID.: R7).

1.3.3 Use Cases

Figure 1.3 shows the UML Use Case Model of the project. The project has three actors: the *car driver*, the *gesture recognition system* and the *server* of the smart city. The process is each time similar. The car driver *performs a hand gesture* and this includes that the gesture recognition system *recognizes the hand gesture* and runs the connected command. For example, when the car driver *performs the hand gesture "Fist"*, the gesture recognition system *recognizes the hand gesture "Fist"* and *turns the music on*.
One use case is connected to the server. If the car driver *performs hand gesture*

"L", the system *recognizes hand gesture "L"* and *calls the police*, the system will be *connected to the server* and will *send user data and reason (police) to the server*. The specified reason is the reason why the system would request to the server. The server *receives the data* and *replies to the system*. These answer could be *the distance value*, when everything works and the communication is successfully or otherwise an *error*.

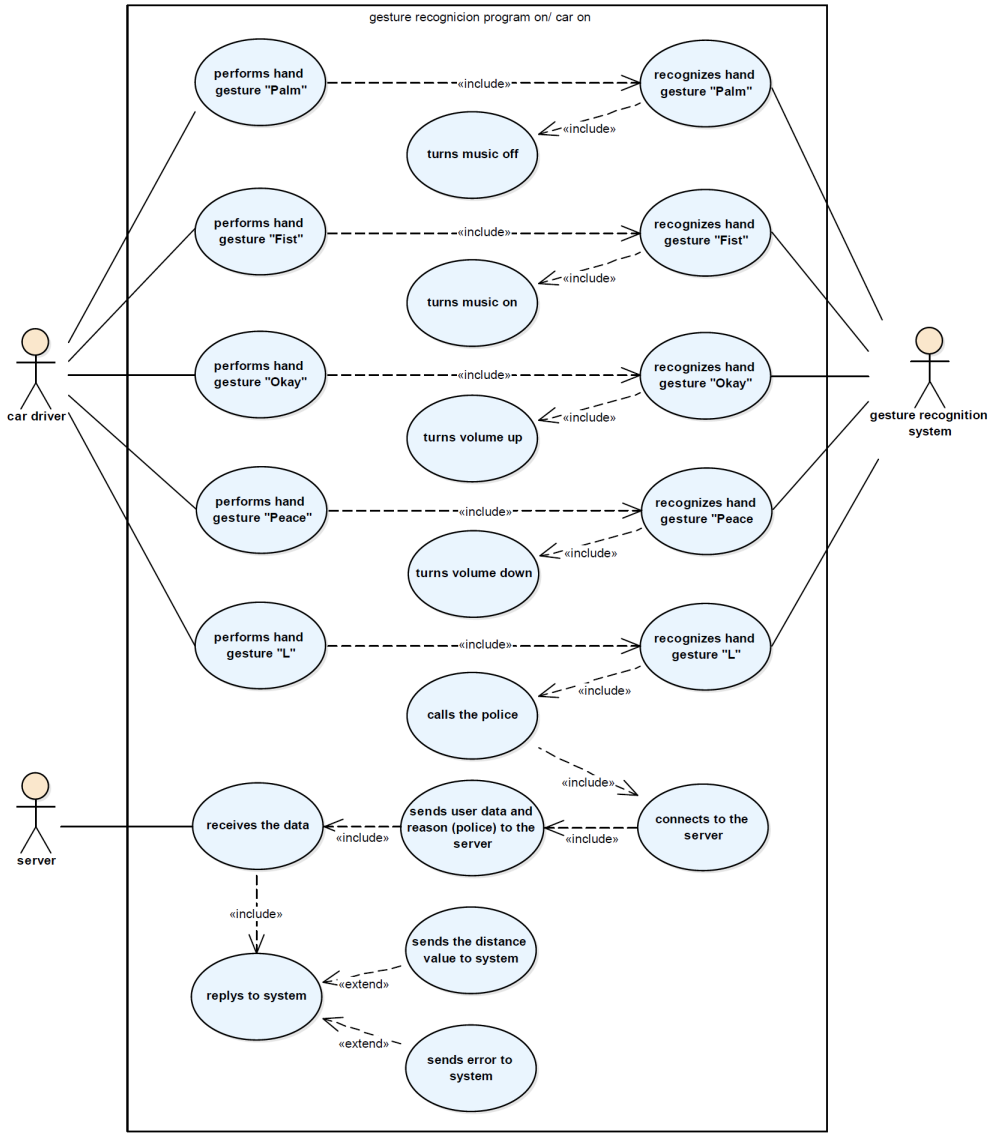

Fig. 1.3 Use Case Model.

1.4 Implementation

1.4.1 VGG-16 Model

The system uses the VGG-16 pre-trained model. The model is published in 2014 by Karen Simonyan and Andrew Zisserman in "Very Deep Convolutional Networks for Large-Scale Image Recognition" [9]. This model is a convolution neural net which enables handling a matrix, especially pictures which are painted as a matrix, as an input. Handling a matrix as an input has the advantage that the system is capable of recognizing idem objects as the same objects no matter in which position they are in the picture [10].

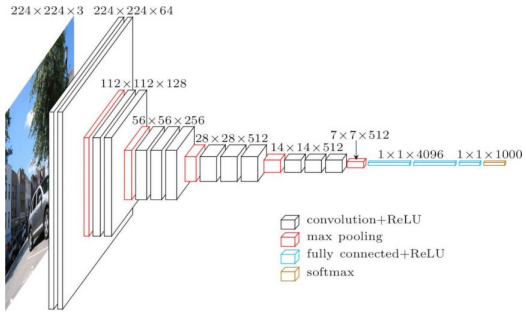

Fig. 1.4 Architecture of the VGG-16 Model [11].

Figure 1.4 shows the architecture of the used VGG-16 model. As you can see on the left side of the picture, the model needs 224x224 RGB image as an input. The third value is the number of channel of 3x3 kernel of the convolution layer [11].
A pile of convolution layers were traversed by the input image. The smallest receptive field size (3x3) to capture the outline sides and the center is used in this filters [11]. The first steps are two *convolution layers* followed by one *max pooling layer*. Then again two *convolution layers* followed by one *max pooling layer*. After that follows three *convolution layers* and one *max pooling layer*. The next layers are three *convolution layers* and one *max pooling layer* and again three *convolution layers* and one *max pooling layer*. Summarised the sequence that the model follows is, using multiple convolution layers (Figure 1.4, black layers), two or three, and after that max pooling (Figure 1.4, red layers) has to be done. The size of the picture is quickly getting reduce. It starts at 224, minimize next to 112, than it comes to 56, next 28, than 14 and finally the size of the picture is 7. Then follows three *fully connected dense layers* [11].
The size of the image is reducing because of the max pooling layers. These layers have the pool stride of 2, which means the size of this layers is 2x2 and this is the reason why the image size will be bisected every time after the max pooling layer [11].
The three consecutive *fully connected layers* (Figure 1.4, blue layers) collects the outputs of the previous *convolution layers* and *pooling layers*. In these *dense layers*

is everything connected to each single input and output [10].
Convolution layers and fully connected layers were brought into action through the *ReLU function*. This function raises each value which is below zero to zero and does not change each value which is higher than zero. Thereby is a better gradient propagation given. If this is not given and the vanishing gradient problem exist, it could be that the neuronal network will stop the further training [10].
The last layer, the *softmax layer* (1.4, yellow layer), results in a sum total of one by summing up all the previous outputs. This softmax layer also indicate the particular probability, if the class is true or false, of the outputs [10].

1.4.2 Gesture Recognition System

The implementation of this project is written in Python and uses the OpenCV library [12].

```
3  import copy # erlaubt Kopiervorgänge
4  import cv2 # Bibliothek zur Lösung von vision Problemen
5  import numpy as np # Daten werden in numpy arrays gespeichert
6  from keras.models import load_model # Keras API verwenden für DL
7  import time # ermöglicht viele zeitbezogene Funktionen
8  import paho.mqtt.client as mqtt
9  import json
```

Fig. 1.5 Importing the necessary packages [12].

At first all the necessary packages has to be imported, as it is shown in Figure 1.5 [12]:

- The system needs *copy* to allow copy processes [12].
- *Cv2* is a library which solves vision problems and is used in our program among other things for using the camera and the keyboard [12].
- With the *numpy as np* importation it is possible to save the images as arrays, in this case in arrays with three channel (RGB) [12].
- The *keras.models import model* command, makes it possible to load and use the VGG-16 model [12].
- The *time* importation brings a lot of time functions with it [12].
- *Paho.mqtt.client as mqtt* is needed to connect this system to the local mqtt broker.
- With the importation of *json* is it possible to connect this system to the server and to send and receive data from the server.

```
23 gesture_names = {0: 'Fist',
24                  1: 'L',
25                  2: 'Okay',
26                  3: 'Palm',
27                  4: 'Peace'}
28
29 # Öffne gespeichertes Modell / Gewichte aus der .h5-Datei
30 model = load_model('C:/Users/katri/Documents/Gesture-Recognition/VGG_cross_validated.h5')
```

Fig. 1.6 Defining the gesture names and loading the model [12].

Figure 1.6 shows the determination of the gesture names and the command that loads the used model [12].

There will be five gestures: *fist*, *L*, *Okay*, *Palm* and *Peace*. In section 1.5 are the associated pictures of this gesture names [12].
The used *model* is the already explained VGG-16 Model (see section 1.4.1) with four additional dense layers on top [12].

```
def predict_rgb_image_vgg(image):
    # Bild in ein NumPy-Array umwandeln
    image = np.array(image, dtype='float32')
    image /= 255

    # ein Array von Wahrscheinlichkeiten zurückgeben
    pred_array = model.predict(image)
    print(f'pred_array: {pred_array}')
    # den Index der höchsten Wahrscheinlichkeit erfassen
    result = gesture_names[np.argmax(pred_array)]
    print(f'Result: {result}')
    print(max(pred_array[0]))
    score = float("%0.2f" % (max(pred_array[0]) * 100))
    print(result)
    return result, score
```

Fig. 1.7 Transforming the data [12].

Figure 1.7 shows the function *predict_rgb_image_vgg(image)* which is used to transform the data. The image is grabbed from the screen and resized. The model has to understand the input and that is why the image is transformed into a NumPy array in the next step [12].

```
# Hintergrundsubtraktion
def remove_background(frame):
    # Lernrate zum Aktualisieren des Hintergrundmodells
    fgmask = bgModel.apply(frame, learningRate=learningRate)
    kernel = np.ones((3, 3), np.uint8)
    # erodiert die Grenzen des Vordergrundobjekts
    fgmask = cv2.erode(fgmask, kernel, iterations=1)
    res = cv2.bitwise_and(frame, frame, mask=fgmask)
    return res
```

Fig. 1.8 Extracting the gesture [12].

To recognize the gesture it is necessary to subtract the background (*remove_background(frame)*) as it is shown in Figure 1.8. First step is to create a mask (*fgmask*) of the background by capturing the background before the hand of the user is in the frame. If the user now appears his or her hand in the frame, the mask will only show the hand and eliminate everything else [12].
The next step, also shown in Figure 1.8, is to change the background to black and the hand, which is what the system should analyze, to white by using binary threshold values. This is necessary to recognize every hand, no matter which skin color the hand has and also for using this program with colourful gloves. This step is also needful to extract the gesture from the background clearly [12].
While the camera is open, which means also while the system is running, the user can control the system with the keyboard. The shortcuts were defined in Figure 1.9 [12]:

- If the user presses the escape key (*if k == 27*), all the windows will be exit at any time [12].
- If the user presses the b-button on the keyboard (*k == ord('b')*), the background will be captured [12].

```
    # Keyboard OP
    k = cv2.waitKey(10)
    if k == 27:  # press ESC to exit all windows at any time
        break
    elif k == ord('b'):  # press 'b' to capture the background
        bgModel = cv2.createBackgroundSubtractorMOG2(0, bgSubThreshold)
        time.sleep(2)
        isBgCaptured = 1
        print('Background captured')
    elif k == ord('r'):  # press 'r' to reset the background
        time.sleep(1)
        bgModel = None
        triggerSwitch = False
        isBgCaptured = 0
        print('Reset background')
    elif k == 32:
        # Wenn die Leertaste gedrückt wird
        cv2.imshow('original', frame)
        # copies 1 channel BW image to all 3 RGB channels
        target = np.stack((thresh,) * 3, axis=-1) # Stapel
        target = cv2.resize(target, (224, 224)) # Größe ändern
        target = target.reshape(1, 224, 224, 3) # Umformung
        prediction, score = predict_rgb_image_vgg(target) # Vorhersage
```

Fig. 1.9 Using the keyboard to control the system [12].

- When the r-button will be pressed (*k == ord('r')*), the system will reset the background [12].
- When the space-bar will be pressed by the user, the output will be the recognized gesture [12].

There is also one option, which is not shown in Figure 1.9, to turn the tracker on. This is for expanding the data set [12].

```
    if prediction == 'Palm':
        action = "music off"
    elif prediction == 'Fist':
        action = "music on"
    elif prediction == 'Okay':
        action = "volume up"
    elif prediction == 'Peace':
        action = "volume down"
    elif prediction == 'L':
        action = "calling police"
```

Fig. 1.10 Defining the corresponding actions of the well defined gestures [12].

Figure 1.10 shows what the system does when the gesture is recognized [12]:

- If the prediction is *'Palm'*, the system will turn the music in the car off [12].
- If the prediction of the gesture is *'Fist'*, the system will turn the music on [12].
- If the system predict the gesture *'Okay'*, the volume will be turned up [12].
- If the prediction is *'Peace'*, the volume will be turned down [12].
- If the system predict the gesture *'L'*, the police will be called by connecting this system to the server of the smart city.

In Figure 1.11 is shown the connection to the server, which inside the "*elif prediction == 'L'*" block. The Gesture Recognition System sends the *"driver_name"*, the *"location"* of the car, the *"reasons"* why the system would like to connect the server and the *"id"* of the car to the server.

```
    elif prediction == 'L':
        action = "calling police"
        #Methode um Daten zu senden
        def sendData():
            topic = "/hshl/users/" #Das Topic in dem gesendet werden soll
            # Die Daten die gesendet werden sollen
            call = {
                "driver_name": "Katrin",
                "location": [51.67, 8.36], # Array mit Koordinaten
                "reasons": "police", # Array mit reasons
                "id": "k80"
                }
            client.publish(topic, json.dumps(call))
            print("sendet")
        def on_connect(client, userdata, flags, rc):
            client.subscribe('/hshl/users/')
            print("subscribed to the users topic!")
            sendData()  # Aufruf der Senden Methode

        # Dont change anything from here!!
        BROKER_ADDRESS = "mr2mbqbl71a4vf.messaging.solace.cloud"  # Adresse des MQTT Brokers
        client = mqtt.Client()

        client.on_connect = on_connect  # Zuweisen des Connect Events
        # Benutzernamen und Passwort zur Verbindung setzen
        client.username_pw_set("solace-cloud-client", "nbsse0pkvpkvhpeh3ll5j7rpha")
        client.connect(BROKER_ADDRESS, port=20614)  # Verbindung zum Broker aufbauen

        print("Connected to MQTT Broker: " + BROKER_ADDRESS)
        #client.loop_forever()  # Endlosschleife um neue Nachrichten empfangen zu können
        #time.sleep(4)
        #client.loop_stop()
        #client.disconnect() # disconnect when get message
        if prediction == 'L':
            client.loop_forever()
        else:
            client.loop_stop()
```

Fig. 1.11 Connecting the system to the server.

1.5 Results

1.5.1 Program Showcase

Start the program by pressing the execute bottom in the individual Python development environment. After that the system will start the webcam and will use it as an input device (Figure 1.12).

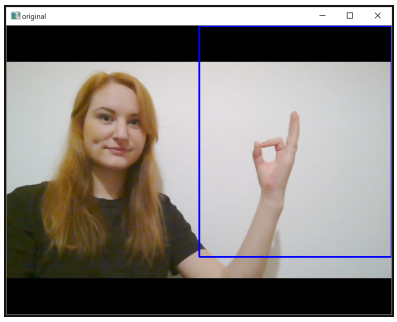

Fig. 1.12 Webcam frame shown on the screen.

Next the user has to press "b" while no hand is in the blue frame, to capture the background. If this is done, the frames which are shown in Figure 1.13 will be opened. After that the user can show a gesture and press the space-bar to name the recognized gesture and the prediction score in the ORI frame. This is shown in Figure 1.14. As it is shown, the gesture prediction is right and the probability is very high (99.98%). The resulting action is "volume up".

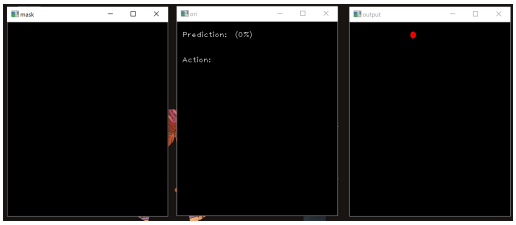

Fig. 1.13 Mask, ORI and output frame shown on the screen.

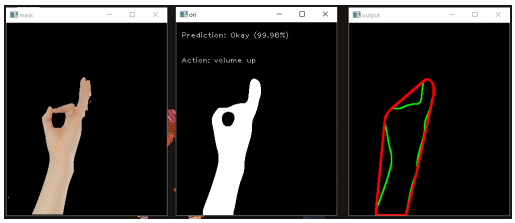

Fig. 1.14 Mask, ORI and output after showing the gesture and pressing the space-bar.

Fig. 1.15 Implemented gestures [12].

Which kind of gestures the driver could show is presented in Figure 1.15.

If the user would like to call the police, he or she has to present the "L" gesture and if this happens, the system will recognize the gesture and connect to the server and sends the data (see Figure 1.16).

Fig. 1.16 The panel is showing that the system sends data to the server.

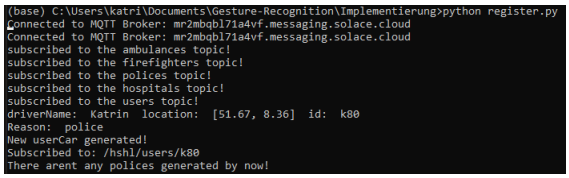

Fig. 1.17 The command prompt is showing that the server receives the data.

Figure 1.17 shows the command prompt from the server. The request is successfully received.

1.5.2 Implemented Requirements

Each requirement, functional and non-functional, is implemented (Figure 1.2).
There are five different hand gestures which can be recognized by this system (Figure 1.2, R1), two more than determined before.
The "L" gesture calls the police by making a request to the server (Figure 1.2, R2).
The requirement R3 "*The system must have trained a neuronal network*" is also fulfilled. As it is described in section 1.4.2 the system uses the VGG-16 Model with four additional dense layers [12]. The system is already trained by Brenner Heintz [12] in the cloud with AWS and by cross-validating his model with data from Kaggle from Benen Harrington [13]. For more information on this topic, follow the corresponding URL-Links in the references.
The output is the name of the gesture and is shown on the top at the ORI frame (Figure 1.2, R4) including the corresponding probability and the following action of the system.
The system is written in Python (Figure 1.2, R5) and uses the OpenCV library (Figure 1.2, R7).
The input device for the hand gestures is the webcam (Figure 1.2, R6) and the additional input device for controlling the system is the keyboard.

1.6 Conclusion

Towards the end, this Gesture Recognition System is clearly useful. Detecting the hand gesture by using the pre-trained VGG-16 model with four additional dense layers on top with B. Heintz's and B. Harrington's data sets works smooth.
In the introduction (section 1.1) is described that tending a hardware with gestures is growing in importance in case of human machine interface. The Figure 1.1 shows only a *projected* global gesture recognition and touchless sensing market size for the year 2020 but in section 1.2 it is proved, that there are some new gesture control systems since 2014.
Section 1.3 defined the goal of the project, the requirements and the use cases.
The following section (section 1.4) describes the used VGG-16 model in detail and some implementations like importing the necessary packages, defining the gesture names, loading the model, transforming the data, extracting the gesture, defining the corresponding actions of the well defined gestures and connecting the system to the server.
The results are demonstrated in section 1.5. The program showcase (section 1.5.1) proves, that the system works and how the system works from the users point of

view. The section 1.5.2 shows that each requirement is implemented including two more hand gestures and the keyboard as one additional input.
It would be operative to implement this hand gesture recognition system into real cars, to control some car activities by showing hand gestures. This will helps car drivers to keep focused on the road because it is not necessary to find bottoms on the centre consol display in the car. The entries to control the car activities will be done by hand gestures. An advantage over using speech input is that if its too loud for the system to recognize what the driver said, the voice control does not work. For the gesture recognition system is loudness no problem.
This system is not only for cars, it can also be installed in different vehicles like trains or motor trucks.
Another option which can this system be a base for, is for detecting sign language. Some of the used gestures are already some gestures of the sign language. The *fist gesture* represents an *A*, the *palm gesture* stands for an *B*, the *peace sign* represents in the sign language the letter **V** or *K*, it depends on how the fingers are crossed, and the *L gesture* is actually the letter *L* in the sign language.

References

1. G. Sommer, N. Krüger, and C. Perwass, *Mustererkennung 2000 - 22. DAGM-Symposium. Kiel, 13.–15. September 2000*. Berlin Heidelberg New York: Springer-Verlag, 2013.
2. T. Keiser, O. Höß, B. Klein, J. Neuhüttler, H. Schneider, and T. Vetter, *Gestensteuerung im Pflegeumfeld – Das Projekt GeniAAL: Grundlagen, Anwendungsfelder, Technologien und Erfahrungen*. Books on Demand, 2015. [Online]. Available: https://books.google.de/books?id=qerVBgAAQBAJ
3. T. I. Partners. (2019) Global biometric technologies market revenue from 2018 to 2027 (in billion u.s. dollars). [Online]. Available: https://www.statista.com/statistics/1048705/worldwide-biometrics-market-revenue/
4. F. Courtney Kennedy. Gestensteuerung: Don't touch this. [Online]. Available: https://www.elektroniknet.de/elektronik/distribution/don-t-touch-this-168250.html
5. H. Zhao, S. Wang, G. Zhou, and D. Zhang, "Gesture-enabled remote control for healthcare," in *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2017, pp. 392–401.
6. J. P. Wachs, H. I. Stern, Y. Edan, M. Gillam, J. Handler, C. Feied, and M. Smith, "A gesture-based tool for sterile browsing of radiology images," *Journal of the American Medical Informatics Association*, vol. 15, no. 3, pp. 321–323, 2008.
7. S. Robertson and J. Robertson, *Mastering the Requirements Process: Getting Requirements Right*. Pearson Education, 2012. [Online]. Available: https://books.google.de/books?id=yE91LgrpaHsC
8. L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE Software*, vol. 30, no. 2, pp. 38–45, 2013.
9. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.
10. R. Becker. (2019) Convolutional neuronal networks – aufbau, funktion und anwendungsgebiete. [Online]. Available: https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/
11. M. ul Hassan. (2018) Vgg16 – convolutional network for classification and detection. [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/pll$_s witcher$
12. B. Heintz. (2018) Training a neural network to detect gestures with opencv in python: How i built microsoft kinect-like functionality with just a webcam and a dream. [Online]. Available: https://towardsdatascience.com/training-a-neural-network-to-detect-gestures-with-opencv-in-python-e09b0a12bdf1
13. B. Harrington. (2018) Hand gesture recognition database with cnn. [Online]. Available: https://www.kaggle.com/benenharrington/hand-gesture-recognition-database-with-cnn/notebook