Modell autó irányítása VerneMQ használatával

Jelen dokumentáció bemutatja, hogy hogyan készítettünk egy külön internetkapcsolattal rendelkező a világ bármely pontjáról irányítható modell autót, amely képes valós időben mozogni, illetve képet küldeni.

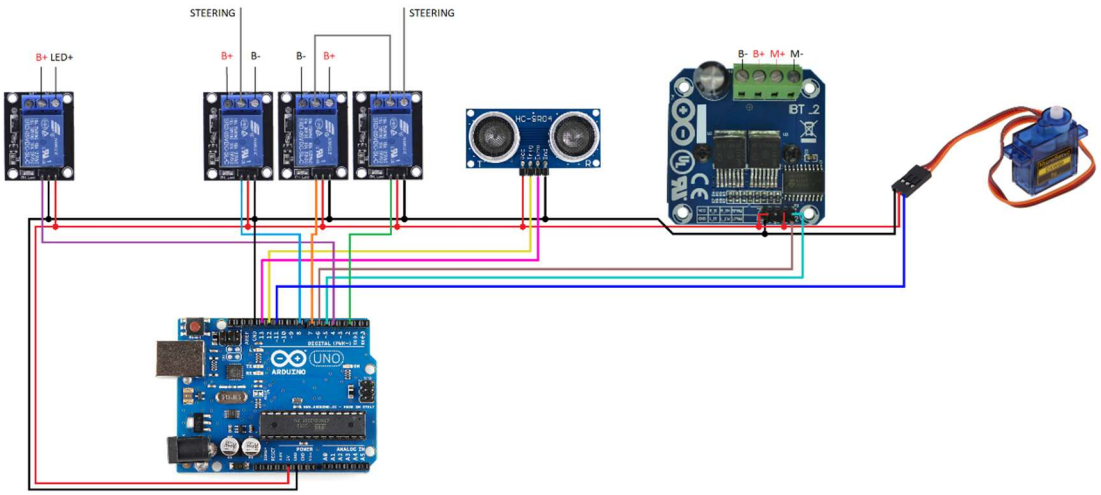
Készítette: Viczina Tamás, Bálint István

Arduino

Esetünkben az eszköz, amit irányítani szeretnénk hálózaton keresztül egy távirányítós autó, így annak helyi vezérléséhez ki kell váltanunk a gyárilag kapott kapcsolást egy általunk készített kapcsolásra, amely képes kommunikálni a Raspberry Pi-vel.

Ehhez legkézenfekvőbb megoldás az Arduino, amelyből számos variáns létezik, mivel mi egy R3-as típussal rendelkezünk így a projektben az kerül felhasználásra, az eszköz önmagában képes soros portos kommunikációra, amely segítségével USB porton keresztül képes kommunikálni a Raspberry Pi-vel.

Viszont az arduino önmagában nem elég a vezérléshez, ezért különböző külső eszközöket csatlakoztatunk hozzá. Reléket, motorvezérlőt, távolságérzékelőt, szervót, ezeknek áramfelvételét, illetve speciális igényeit figyelembe véve, esetünkben PWM-lábra van szüksége a motorvezérlőnek, hiszen az impulzusmoduláció segítségével tudjuk elérni, hogy a motor ne pusztán binárisan működjön, hanem szinteket alkalmazva a sebességét meg tudjuk adni 0-254-es tartományban.



Amint a kapcsolásunk készen áll szükségünk van egy jó sebességű optimalizált kódra, itt igyekeztünk kihasználni minden létező hardvert, így azokat a maximális támogatott üzenetküldési sebességen használjuk. (Amelyet az arduino IDE-ben található soros monitor is támogat)

Ez 2 000 000 bit továbbítására teszi alkalmassá az arduino soros portját 1 másodperc alatt.

A programban egy külső osztályt használunk, amely segítségével egyszerűbb megvalósítani az ultrahangos távolságérzékelő lekérdezését, ez a NewPing.h amely arduino IDE környezetben közvetlenül az alkalmazásból telepíthető (ESZKÖZÖK -> Könyvtárak Kezelése).

```
#include <NewPing.h>
#define FADE 10
#define LED_PIN 4
#define TRIGGER_PIN 12    //ULTRASONIC
#define SERVO_PIN 11
#define ECHO_PIN 13       //ULTRASONIC
#define MAX_DISTANCE 500

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

int camstate,ledstate,i;
int maxdistanceincm=15;
byte RPWM=5,LPWM=6,POL1=8,POL2=7, STEERINGEN=2,a=0;
String cmd;
char as;
bool sensorena=true,fwd,left,right;
```

Miután felvettük a szükséges változókat, segédváltozókat elkezdhetjük megírni a szükséges függvényeket, eljárásokat.

Fontos az motorvezérlő kezelésénél hogy biztosítsuk, hogy az ellenkező irányban nem történik vezérlés ezért mindkét esetben ezen értékeket 0-ra állítjuk.

Majd a soros portról kapott vezérlési kulcsszót, illetve ahol szükséges ott a kulcsszó utáni értéket is kiolvassuk, majd kiírjuk az adott lábra.

Itt felmerülhet, hogy szükség lenne vizsgálatra, hogy az érték 0-254 tartományon belül legyen biztosan, de mivel a Python-ban készített kliensünk előre megadott értéktartományban küld majd üzenetet így erre itt nincs szükség.

```
void FORWARD() {
  fwd=true;
  analogWrite(LPWM,0);//ellenkező irány stop
  a=Serial.readStringUntil(' ').toInt(); //sebesség mértéke
  analogWrite(RPWM,a);
  }
 void BACKWARD(){
  fwd=false;
  analogWrite(RPWM,0);// ellenkező irány stop
  a=Serial.readStringUntil(' ').toInt(); //sebesség mértéke
  analogWrite(LPWM,a);
 }
 void STOP() {
  analogWrite(RPWM,0);
  analogWrite(LPWM,0);
 }
```

Az autó kanyarodását szervóval terveztük megoldani, de amely a megvalósításkor rendelkezésünkre állt nem működött egy viszonylag nagy értéktartományban, így az eredeti motorral oldottuk meg három darab relé használatával, amelyből kettő polaritást vált, és egy megszakít.

Ahhoz viszont hogy az autó így is egyenesen tudjon a közlekedni a manőverezés után, az ellenkező irányba egy rövidebb ideig tartó korrigálást is kellet implementálni.

```
void LEFT(){
  if (left==false)
  {
  digitalWrite(POL1,LOW);
  digitalWrite(POL2,LOW);
  digitalWrite(STEERINGEN, HIGH);
  delay(100);
  digitalWrite(STEERINGEN, LOW);
  left=true;
  }
 }
void LEFTRELEASED(){
  if (left==true)
  {
  digitalWrite(STEERINGEN, LOW);
  digitalWrite(POL1,HIGH);
  digitalWrite(POL2,HIGH);
  delay (100);
  digitalWrite(STEERINGEN, HIGH);
  delay (35);
  digitalWrite(STEERINGEN, LOW);
  left=false;
  }
 }

void RIGHT(){
  if(right==false)
  {
  digitalWrite(POL1,HIGH);
  digitalWrite(POL2,HIGH);
  digitalWrite(STEERINGEN, HIGH);
  delay(100);
  digitalWrite(STEERINGEN, LOW);
  right=true;
  }
 }
void RIGHTRELEASED(){
  if(right==true)
  {
  delay (200);
  digitalWrite(STEERINGEN, LOW);
  digitalWrite(POL1,LOW);
  digitalWrite(POL2,LOW);
  delay (100);
  digitalWrite(STEERINGEN, HIGH);
  delay (35);
  digitalWrite(STEERINGEN, LOW);
  right=false;
  }
}
```

A szervóra a Raspberry Pi kamerájának mozgatásakor van szükség, használatához készítettünk egy eljárást, amely képes átvenni PWM-értéket amelyet az üzenet tartalmaz.

A továbbiakban még készítettünk 3 előre beállított látószöget módosító eljárást.

```
 void SERVO() {
  a=Serial.readStringUntil(' ').toInt();
  analogWrite(SERVO_PIN,a);
  }
```

A SENSOR eljárással ki és be tudjuk kapcsolni, hogy az ultrahangos távolság érzékelő **maxdistanceincm** távolságnál kisebb távolság esetén közbelépjen-e, és megállítsa az autót.

Ha ezt a beállítást bekapcsolva hagyjuk, akkor egy esetleges hálózatvesztés bekövetkeztével képes megelőzni a frontális ütközést.

```
void SENSOR() {
  if(sensorena==true)
  {
    sensorena=false;
    Serial.println("Forward Collosion Prevention System: OFF");
  }
  else
  {
    sensorena=true;
    Serial.println("Forward Collosion Prevention System: ON");
  }
  }
```

Az előbbiekben ismertetett három eljárás, amely beállítja a kamera látószögét.

Itt fontos volt, hogy az állapottól függően ne túl gyorsan, hanem a kamerakép stabilitását figyelembe véve történjen a mozgás.

A szenzor kezdőpozíciójának ismeretével, amelyet a **camstate** segédváltozóban tárolunk.

```
void CAMTORIGHT() {
if (camstate==1)
{
  for (i=127;i<184;i++)
  {
    analogWrite(SERVO_PIN,i);
    delay(FADE);
  }
  camstate=2;
}
if (camstate==0)
{
  for (i=70;i<184;i++)
  {
    analogWrite(SERVO_PIN,i);
    delay(FADE);
  }
  camstate=2;
}
if (camstate==2)
{
      analogWrite(SERVO_PIN,184);
      camstate=2;
}
}
```

```
void CAMTOLEFT() {
  if (camstate==1)
{
  for (i=127;i>70;i--)
  {
    analogWrite(SERVO_PIN,i);
    delay(FADE);
  }
  camstate=0;
}
if (camstate==2)
{
  for (i=184;i>70;i--)
  {
    analogWrite(SERVO_PIN,i);
    delay(FADE);
  }
  camstate=0;
}
if (camstate==0)
{
      analogWrite(SERVO_PIN,70);
      camstate=0;
}
}

void CAMTOMIDDLE() {
  if (camstate==0)
{
  for (i=70;i<127;i++)
  {
    analogWrite(SERVO_PIN,i);
    delay(FADE);
  }
  camstate=1;
}
if (camstate==2)
{
  for (i=184;i>127;i--)
  {
    analogWrite(SERVO_PIN,i);
    delay(FADE);
  }
  camstate=1;
}
if (camstate==1)
{
      analogWrite(SERVO_PIN,127);
      camstate=1;
}
}
```

A HELP eljárás kiírja az összes funkciót a soros portra.

```
void HELP()
{
    delay(10);
    Serial.println("USER'S GUIDE");
    delay(10);
    Serial.println("For a left turn send 'l' character");
    delay(10);
    Serial.println("For a right turn send 'r' character");
    delay(10);
    Serial.println("For moving FORWARD use 'f' [0-254]");
    delay(10);
    Serial.println("For moving BACKWARD use 'b' [0-254]");
    delay(10);
```

```
    Serial.println("To STOP send 's'");
    delay(10);
    Serial.println("To TURN ON/OFF the Forward Collosion Preventing System
    send '+' ");
    delay(10);
    Serial.println("To LOOK LEFT WITH the CAMERA send '0'");
    delay(10);
    Serial.println("To LOOK RIGHT WITH the CAMERA send '2'");
    delay(10);
    Serial.println("To LOOK FORWARD WITH the CAMERA send '1'");
    delay(10);
    Serial.println("To TURN ON/OFF LED FRONT LIGHT send '3'");
}
```

A LEDSWITCH eljárás a jármű elején található LED-ek kapcsolását végzi, amely lehetővé teszi, hogy nem megfelelő látási viszonyok esetén ezt távolról is képesek legyünk elvégezni.

```
void LEDSWITCH()
{
  if (ledstate==0)
  {
    ledstate=1;
    digitalWrite(LED_PIN,HIGH);
  }
  else
  {
    ledstate=0;
    digitalWrite(LED_PIN,LOW);
  }
}
```

A setup eljárásban beállítjuk, hogy az arduino mely lábait használjuk ki vagy bementre, illetve alaphelyzetbe állítjuk a szervót.

A setup egyik leglényegesebb pontja hogy a soros port időzítését csökkentsük az alapértelmezettről 100ms-ra. Így már ezzel a változtatással érezhetően gyorsan reagáló eszközt kapunk, ami kulcsfontosságú, hiszen kísérletünk célja hogy minél kisebb legyen a késleltetés.

```
void setup()     {
  for(int i=5;i<9;i++)
  {
   pinMode(i,OUTPUT);
  }
  for(int i=5;i<9;i++)
  {
   digitalWrite(i,LOW);
  }
   delay(1000);
   Serial.begin(2000000);
   pinMode (2,OUTPUT);
   pinMode (SERVO_PIN,OUTPUT);
   digitalWrite(STEERINGEN, LOW);
   Serial.setTimeout(100);
   delay(50);
   Serial.println("SEND 'h' for HELP");
   ledstate=0; left=false; right=false;
   pinMode (LED_PIN,OUTPUT);
   analogWrite(SERVO_PIN,127);
   camstate=1;
}
```

A loopban ha van soros port, akkor az érkező karaktert egy switch-case szerkezettel vizsgáljuk meg, és itt hívjuk meg az eljárásokat, valamint ezen kívül fut még az ultrahangos távolságszenzor értékét figyelő if, amely ha a sensorena igaz, akkor nem engedi elindulni előre az autót, illetve megállítja azt, ha akadályt érzékel.

```
void loop() {
if (Serial.available())
    {
      cmd =Serial.readStringUntil(' '); //feladat betűjele
      as=cmd.charAt(0); //karakter átadása
    switch (as){
    break;
    case 'r':
    RIGHT();
    break;
    case 'l':
    LEFT();
    break;
    case 'f':
    FORWARD();
    break;
    case 'b':
    BACKWARD();
    break;
    case 's':
    STOP();
    break;
    case '+':
    SENSOR();
    break;
    case 'c':
    SERVO();
    break;
    case '0':
    CAMTORIGHT();
    break;
    case '2':
    CAMTOLEFT();
    break;
    case '1':
    CAMTOMIDDLE();
    break;
    case 'h':
    HELP();
    break;
    case '3':
    LEDSWITCH();
    break;
    case '4':
    LEFTRELEASED();
    break;
    case '5':
    RIGHTRELEASED();
    break;
    default:
    break;
    }
    }
    unsigned int distance = sonar.ping_cm();
    if(distance<maxdistanceincm&&sensorena==true&&fwd==true)
    {
      STOP();
    }
    }
```

VerneMQ Broker

Az adatforgalom lebonyolításához egy a két eszköz hálózati forrásától független szerverre van szükségünk, amely a mi esetünkben a Contabotól bérelhető VPS szerver, amelyen egy VerneMQ Brokert futtatunk.

A Brokeren minimális konfigurációra volt szükségünk, pusztán egy felhasználónevet és jelszót állítottunk be, hogy az alapvető biztonságot megteremtsük.

Ehhez a VerneMQ hivatalos segédletét használtuk.

A Broker alapvetően csatornákon fogadja az üzeneteket, amely később továbbküldésre kerül az adott csatornára feliratkozott klienseknek.

Ami számunkra kulcsfontosságú paraméter üzenetküldésnél az a QoS ami az angol Quality of Service rövidítése, itt tudjuk megadni, hogy az üzenetet tulajdonképpen milyen protokollal küldje.

Három lehetőségünk van:

- 0 ami maximum egyszer küldi el de itt nincs garancia
- 1 ami legalább egyszer elküldi és erre van garancia
- 2 ami pontosan egyszer küldi el

A három opció közül az első a legköltségmentesebb, de ha biztosan át akarjuk juttatni az üzenetet egyszer, akkor az első a legjobb választás, viszont a pluszban érkező üzeneteket mind a kibocsájtó mind a fogadó oldalon tolerálnunk kell. Mi ezt a laptopon futtatott programban segédváltozókkal oldottuk meg, illetve kihasználtuk a tkinter, bind függvényének lehetőségeit, így kényelmesen el lehetett különíteni a billentyűzet gombjainak lenyomását, illetve felengedését.

A QoS értékét a publish függvény paramétereként tudjuk megadni.

A mi Python kódjainkban az mqtt kezelésére a paho.mqtt.client könyvtárat használtuk.

A Raspberry-n két Pythonban írt program fut, az egyik felelős az mqtt brokertől egy előre definiált csatornán kapott üzenet továbbításáért a soros portra, a másik pedig folyamatosan készít fényképeket és base64-es kódolással továbbítja, egy szintén előre definiált mqtt csatornára.

Az mqtt-n érkező adatok soros portra való továbbításához szükséges. Python kód:

```
#!/usr/bin/env python3

import paho.mqtt.client as mqtt
import time
import serial


def callback( client, userdata, msg ):
    x = msg.payload
    #print( msg.topic + " " + str( x ) );
    ser.write( x )

def on_connect(client, userdata, flags, rc):
    #print(rc)
    client.subscribe( "/car/controls", 0 )
    client.message_callback_add( "/car/controls",  callback );
    client.publish( "/car/startup", "started" )

ser = serial.Serial(

   port='/dev/ttyACM0',
   baudrate = 2000000,
   parity=serial.PARITY_NONE,
   stopbits=serial.STOPBITS_ONE,
   bytesize=serial.EIGHTBITS,
   timeout=1
)
time.sleep( 3 )

mqttc =  mqtt.Client()
mqttc.on_connect = on_connect
mqttc.username_pw_set( "username", "password" )
mqttc.connect( "host", 1883, 60 )
mqttc.loop_forever()
```

A kamerával képet készítő, valamint továbbító Python kód:

A képeket a már konfigurált VerneMQ broker segítségével továbbítottuk, ezeket egy virtuális meghajtóra írtuk, majd onnan beolvasva base64-es kódolással küldtük tovább.

```
#!/usr/bin/env python3

import picamera
import base64
import time
import paho.mqtt.client as mqtt



def send(client,camera):
    camera.capture("/mnt/kep.png",format="png")
    f = open("/mnt/kep.png","rb")
    f.seek(0,2)
    size=f.tell()
    f.seek(0,0)
    data=f.read(size)
    encoded=base64.b64encode( data ).decode( 'ascii' )
    client.publish( "/car/camera", encoded,qos=1 )

def on_connect(client, userdata, flags, rc):
    print(rc)

camera = picamera.PiCamera()
camera.resolution=(200,200)
camera.vflip=True
camera.hflip=True

mqttc =  mqtt.Client()
mqttc.on_connect = on_connect
mqttc.username_pw_set( "username", "password" )
mqttc.connect( "host", 1883, 60 )
mqttc.loop_start()

while True:
    send(mqttc,camera)
    time.sleep(0.1)
```

A két program automatikus futtatása, illetve a virtuális fájlrendszer létrehozása miatt az alábbi módosításokat kellett még végrehajtanunk:

- Egy indító szkriptet kell írnunk, ami elindítja a két programot amennyiben a rendszer rendelkezik hálózati hozzáféréssel.
- A crontab-ban minden újraindításkor le kell futtassuk a két program indításáért felelős szkriptet.
- A /etc/fstab-ban kellett felcsatolnunk ramfs-t, hogy ide mentsük a képeket, elkerülve azt, hogy a gyakori írás/olvasás műveleteket az sd kártyán hajtsuk végre.

A szkript:

Mivel a két programunk helyes működéséhez hálózati hozzáférés szükséges, így elkerülhetetlen, hogy ellenőrizzük, rendelkezésünkre áll-e internet hozzáférés és csak ebben az esetben indítsuk el a két programot.

```
#!/bin/bash
echo -n "Waiting for usb0: " > /home/pi/log.log
ping -c 1 google.com
while test $? -ne 0
do
sleep 1s
echo -n "." >> /home/pi/log.log
ping -c 1 google.com
done
echo " done" >> /home/pi/log.log
echo -n "Start controls.py: " >> /home/pi/log.log
/home/pi/Desktop/controls.py > /home/pi/out.log 2> err.log &
sleep 1s
echo -n "Start camera.py: " >> /home/pi/log.log
/home/pi/Desktop/camera.py > /home/pi/out.log 2> err.log &
echo "PID: $!" >> /home/pi/log.log
```

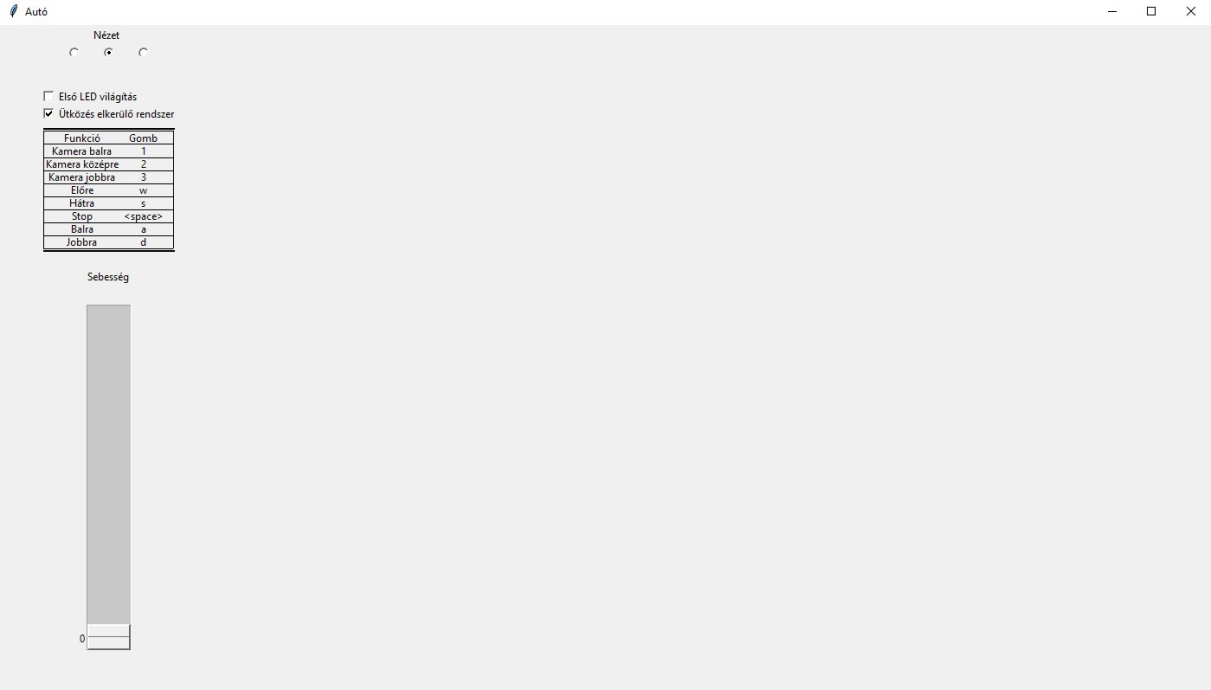
A módosított crontab:

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 * * * *root    cd / && run-parts --report /etc/cron.hourly
25 6 * * *root test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.daily )
47 6 * * 7root test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.weekly )
52 6 1 * *root test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.monthly )
@reboot pi /usr/bin/carremote.sh &
#
```

A módosított /etc/fstab:

```
proc            /proc           proc    defaults          0       0
PARTUUID=ca868177-01  /boot           vfat    defaults          0       2
PARTUUID=ca868177-02  /               ext4    defaults,noatime  0       1
ramfs /mnt  ramfs defaults,mode=777     0     0
# a swapfile is not a swap partition, no line here
#   use  dphys-swapfile swap[on|off]  for that
```

Grafikus felület (Beérkező kép nélkül):



A program kódja:

Amit külön szükséges előre telepíteni: paho.mqtt, urllib, tkinter

```
from tkinter import *
import paho.mqtt.client as mqtt
import time
import urllib.request
import base64
import numpy
```

Létrehozzuk a tkinter segítségével az ablakot, illetve a változókat, amikre a továbbiakban szükségünk lesz.

A bind() függvényt a gomb lenyomására illetve felengedésére is meghívjuk, de mivel a folyamatosan lenyomva tartott gomb is lenyomó eseményeket generál, ezért segédváltozókkal korlátozzuk a parancs kiadását kanyarodás esetén, hogy az mindenképp csak egyszer legyen elküldve felengedés elött.

```
window = Tk()
var=IntVar()
global left
left=IntVar()
global right
right=IntVar()
left=0
right=0
chk_state = BooleanVar()
chkled_state = BooleanVar()
chk_state.set(True)
chkled_state.set(False)
```

Minden eseményre írnunk kell egy meghívható függvényt vagy eljárást. Az elsők között egy olyan függvényre van szükségünk, amely abban az esetben, ha nincs internetkapcsolat nem engedi tovább futni a programunkat a további kapcsolódási hibák elkerülése érdekében.

```
def wait_for_internet_connection():
    while True:
        try:
            response =
urllib.request.urlopen('http://google.com',timeout=1)
            return
        except urllib.request.URLError:
            pass
```

Az on_connect() függvényben megadjuk, hogy milyen csatornákra iratkozzon fel, és a már fent említett QoS is meg kell adnunk, ami alapértelmezetten 0.

```
def on_connect(client,userdata,flags,rc):
    client.subscribe( "/car/camera",  qos=0 )
    print(rc)
```

Az on_message() függvényben megadhatjuk, hogy mi történjen abban az esetben, ha üzenet érkezik a fent említett csatornák egyikén. Esetünkben a csatornán egy base64-ben kódolt kép érkezik, amelyet beállítunk egy Label hátterének.

```
def on_message( client, userdata, msg):
    data = base64.b64decode( msg.payload )
    img2=PhotoImage(data=data)
    panel.configure(image=img2)
    panel.image = img2
    print( "message arrived.." )
```

Már csak meg kell adnunk, hogy a gomb lenyomásoknál, illetve felengedéseknél milyen függvényt hívjon meg az adott esemény.

Itt megadva a csatornát (amelyre a Raspberry Pi már feliratkozott) küldhetjük ki a parancsokat, ahol szükséges ott a motorvezérlő sebesség paraméterét a csúszkáról kérdezzük le.

```
def wpressed(event):
    mqttc.publish(topic="/car/controls", payload="f"+" "+str(var.get()),
qos=0, retain=False)

def spressed(event):
    mqttc.publish(topic="/car/controls", payload="b"+" "+str(var.get()),
qos=0, retain=False)

def apressed(event):
    global left
    if left<1:
        mqttc.publish(topic="/car/controls", payload="l", qos=0,
retain=False)
        left=1
```

```
def dpressed(event):
    global right
    if right<1:
        mqttc.publish(topic="/car/controls", payload="r", qos=0,
retain=False)
        right=1

def stoppressed(event):
    mqttc.publish(topic="/car/controls", payload="s", qos=0, retain=False)

def areleased(event):
    global left
    left=0
    mqttc.publish(topic="/car/controls", payload="4", qos=0, retain=False)

def dreleased(event):
    global right
    right=0
    mqttc.publish(topic="/car/controls", payload="5", qos=0, retain=False)


def view1pressed(event):
    mqttc.publish(topic="/car/controls", payload="0", qos=0, retain=False)

def viewleft():
    mqttc.publish(topic="/car/controls", payload="0", qos=0, retain=False)

def view2pressed(event):
    mqttc.publish(topic="/car/controls", payload="1", qos=0, retain=False)

def viewmiddle():
    mqttc.publish(topic="/car/controls", payload="1", qos=0, retain=False)

def view3pressed(event):
    mqttc.publish(topic="/car/controls", payload="2", qos=0, retain=False)

def viewright():
    mqttc.publish(topic="/car/controls", payload="2", qos=0, retain=False)
```

A LED, illetve a közelségérzékelő ki és bekapcsolása az alábbi két esemény segítségével történik:

```
def collosion():
    mqttc.publish(topic="/car/controls", payload="+", qos=0, retain=False)

def ledlight():
    mqttc.publish(topic="/car/controls", payload="3", qos=0, retain=False)
```

Miután megvannak az események, az első dolog, amit a programnak le kell kérdeznie az az, hogy rendelkezünk-e internetkapcsolattal.

```
wait_for_internet_connection()
```

Amint ez megvan, konfigurálhatjuk az mqtt-t.

```
mqttc = mqtt.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
```

```
mqttc.username_pw_set( "username", "password" )
mqttc.connect( "host", 1883, 60 )
```

Ezután pár alapbeállítást kell elvégeznünk például az ablak neve, mérete, illetve az objektumok létrehozása elhelyezése.

```
window.title("Autó")
window.geometry('1920x1080')
```

Az első ilyen objektum, amit elhelyezünk az a kamera képet megjelenítő Label lesz, itt alapvetően három függvénnyel helyezhetjük el pack(),grid() illetve place() amit mi is használunk.

```
#camera
panel = Label(window)
panel.place(x=250,y=70)
```

Készítettünk egy információs táblázatot, amely tartalmazza az autó irányításához szükséges összes információt.

```
#infobox
info1 = Label(window, text="Funkció\nKamera balra\nKamera középre\nKamera
jobbra\nElőre\nHátra\nStop\nBalra\nJobbra",anchor=W)
info1.place(bordermode=INSIDE,x=50,y=120)
info2 = Label(window, text="Gomb\n1\n2\n3\nw\ns\n<space>\na\nd",anchor=W)
info2.place(bordermode=INSIDE,x=140,y=120)
infobox1frame = Frame(window,bg='black',height=2,width=152)
infobox1frame.place(x=50,y=119)
infobox2frame = Frame(window,bg='black',height=2,width=152)
infobox2frame.place(x=50,y=260)
infoboxline1frame = Frame(window,bg='black',height=1,width=150)
infoboxline1frame.place(x=50,y=137)
infoboxline2frame = Frame(window,bg='black',height=1,width=150)
infoboxline2frame.place(x=50,y=153)
infoboxline3frame = Frame(window,bg='black',height=1,width=150)
infoboxline3frame.place(x=50,y=168)
infoboxline4frame = Frame(window,bg='black',height=1,width=150)
infoboxline4frame.place(x=50,y=183)
infoboxline5frame = Frame(window,bg='black',height=1,width=150)
infoboxline5frame.place(x=50,y=198)
infoboxline6frame = Frame(window,bg='black',height=1,width=150)
infoboxline6frame.place(x=50,y=213)
infoboxline7frame = Frame(window,bg='black',height=1,width=150)
infoboxline7frame.place(x=50,y=228)
infoboxline8frame = Frame(window,bg='black',height=1,width=150)
infoboxline8frame.place(x=50,y=243)
infoboxline9frame = Frame(window,bg='black',height=1,width=150)
infoboxline9frame.place(x=50,y=258)
infoboxline10frame = Frame(window,bg='black',height=1,width=150)
infoboxline10frame.place(x=50,y=122)
infoboxside1frame = Frame(window,bg='black',height=136,width=1)
infoboxside1frame.place(x=50,y=122)
infoboxside2frame = Frame(window,bg='black',height=136,width=1)
infoboxside2frame.place(x=200,y=122)
info3 = Label(window, text="Sebesség",anchor=W)
info3.place(bordermode=INSIDE,x=98,y=280)
```

Ezek után létrehoztuk a gombokat, jelölőnégyzeteket és a csúszkát.

```
#buttons,scales,checkboxes
thrtlscale = Scale(window, from_=254,
to=0,variable=var,resolution=1,activebackground="grey",length=400,width=50)
view = Label(window, text="Nézet")
view.grid(column=0, row=0)
cleft = Radiobutton(window, value=1,command=viewleft) #Camera to left
cmiddle = Radiobutton(window, value=2,command=viewmiddle)#Camera to middle
cright = Radiobutton(window, value=3,command=viewright)#Camera to right
safetysys= Checkbutton(window, text='Ütközés elkerülő rendszer',
var=chk_state,command=collosion)
ledstate= Checkbutton(window, text='Első LED világítás',
var=chkled_state,command=ledlight)
```

Valamint elhelyeztük azokat.

```
#design
thrtlscale.place(bordermode=INSIDE, x=75,y=320)
safetysys.place(bordermode=INSIDE, x=45,y=90)
ledstate.place(bordermode=INSIDE, x=45,y=70)
cleft.place(bordermode=INSIDE,x=75,y=20)
cmiddle.place(bordermode=INSIDE,x=115,y=20)
cright.place(bordermode=INSIDE,x=155,y=20)
view.place(bordermode=INSIDE,x=105,y=1)
```

Végül a bind() függvényekkel megadjuk, hogy mely gomb lenyomásakor, illetve felengedésekor melyik függvény/eljárás kerüljön meghívásra.

```
#keyboard
window.bind("w", wpressed)
window.bind("a", apressed)
window.bind("<KeyRelease-a>", areleased)
window.bind("<KeyRelease-d>", dreleased)
window.bind("s", spressed)
window.bind("d", dpressed)
window.bind("1", view1pressed)
window.bind("2", view2pressed)
window.bind("3", view3pressed)
window.bind("<space>", stoppressed)

window.mainloop()
```

A programunk így képes fogadni és megjeleníteni a Raspberry Pi-től kapott képeket, illetve elküldeni a megfelelő utasítást gyorsabban, mint amire a projekt elkészítése elött számítottunk.

A kamera képét hiába tudjuk sűrűbben küldeni, MQTTfx-el monitorozva 2FPS-re tehetnénk szert mivel két képet kapunk másodpercenként, azonban a programunk tesztelése során nem tudta megjeleníteni a képeket ilyen gyorsan. Habár nem értük el az MQTT elméleti maximális adattovábbítási méretét, ami 268MB-os üzenetet is képes elküldeni, kijelenthetjük, hogy az általunk küldött 250x250-es kép 500ms-onkénti elküldése nem okoz látványos lassulást a rendszerben.

Az elkészült autó