OŠ Brinje Grosuplje
Ljubljanska cesta 40a, Grosuplje

Avtomatski sistem za hranjenje teličkov

Raziskovalna naloga

Področje: Elektrotehnika, elektronika in robotika

Avtorji: Svit Verhovšek, Svit Selan in Lian Kodre Klavžar
9. razred
Mentor: Matej Kastelic

Grosuplje, marec 2024

Vsebina

Kazalo slik

1. Povzetek

V raziskovalni nalogi smo razvili hranilnik teličkov: model avtomatiziranega vozila, ki vozi po obrobljeni poti ali tirnici in se ustavlja pri posameznih teličkih. Ker so različne starosti in velikosti, potrebujejo različno količino hranila, ki ga mora vozilo ob določenih časih pravilno dostaviti in zliti v vedro s cucljem, namenjeno posameznemu teličku.

Vozilo je sestavljeno iz komponent Lego Mindstorms EV3. Premika se s pomočjo močnega motorja, ki ga krmili miniračunalnik EV3 Brick z adapterjem Wi-Fi. S pomočjo senzorja barve vozilo ve, kje se nahajajo telički ob poti. Na Brick smo namestili operacijski sistem ev3dev in na njem zagnali našo aplikacijo, napisano v Pythonu, ki kot strežnik sprejema navodila za hranjenje. Vozilo s pomočjo pretočne hitrosti izračuna, koliko časa je potrebnega za točenje hranila glede na maso telička.

Navodila za hranjenje daje mikrokontroler Arduino Uno R4 Wi-Fi, podatki pa so shranjeni na računalniku Raspberry PI 3B v podatkovni bazi MySQL. Podatke Arduinu posreduje naš strežnik, napisan v Pythonu. Urnik se nahaja v preprosti tabeli v podatkovni bazi. Vozilo se po opravljenem delu vrne na začetni položaj.

2. Uvod

V raziskovalni nalogi bomo raziskali, kako razviti model vozila za hranjenje teličkov in poskušali razviti celotno rešitev. V pravem svetu so taka vozila ogromna, zato smo se odločili, da bo naš model poskušal simulirati hranjenje teličkov v pomanjšani simulaciji.

Naš model bo sestavljen iz:

- avtomatiziranega vozila,
- kontrolnega sistema in
- centralne podatkovne baze.

Vozilo bomo sestavili iz kompleta Lego Mindstorm EV3, ki smo ga že spoznali v dveh prejšnjih raziskovalnih nalogah. Za kontrolni sistem smo izbrali mikrokontroler Arduino Uno, ki smo spoznali pri predmetu Elektronika z robotiko v 9. razredu, za miniračunalnik pa Raspberry Pi 3B.

Poskušali bomo potrditi ali ovreči tri hipoteze.

Te hipoteze so:

- Ali je Lego Mindstorms EV3 s svojimi senzorji sposoben natančno izvajati svoje opravilo hranjenja, da pri tem telički ne ostanejo brez hrane?
- Ali je Arduino Uno sposoben kontrolirati EV3 na daljavo brezžično brez uporabe svojih PIN-ov?
- Ali je Raspberry PI 3B dovolj močan, da lahko deluje istočasno kot podatkovna baza in vstopna točka?

Pri raziskovalni nalogi bomo hipoteze potrdili ali ovrgli s pomočjo sestavljenega modela hranilnika teličkov.

Izvorno kodo bomo prenesli na portal Github: https://github.com/svitv/hranilnik.

Videoposnetek delovanja hranilnika bomo prenesli na Youtube: https://www.youtube.com/@mastersvitverhstudio7535/videos.

3. Sistem za hranjenje

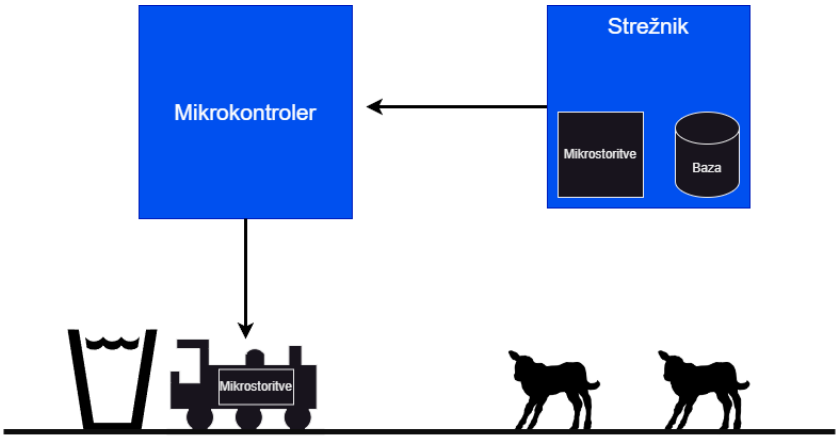
Model našega sistema za hranjenje (slika 1) bo sestavljen iz:

- vozila,
- ločenega kontrolnega sistema in
- miniračunalnika.

Vozilo bo sestavljeno iz delov Lego Mindstorms EV3, saj smo ga že uporabljali v prejšnji raziskovalni nalogi. Poskušali bomo preučiti, kateri senzorji bodo uporabni in kako bo vozilo videti. Na vozilu bo tekel operacijski sistem ev3dev (različica Debiana), programirali pa bomo v Pythonu, ki bo s pomočjo knjižnice Flask sprejemal ukaze kot strežnik.

Naloga kontrolnega sistema bo zagon vozila ob vsakokratnem hranjenju, pri čemer bo sprožilec posredoval ustrezna navodila za vsakega telička. Sistem bo razvit na Arduino Uno R4 Wi-Fi mikrokontrolerju v programskem jeziku C in bo komuniciral preko brezžične povezave.

Na miniračunalniku Raspberry Pi 3B bomo namestili Raspberry Pi OS, na katerem bo tekla podatkovna baza MySQL. Na podoben način kot pri EV3 bo dostop do baze možen preko našega strežnika. Ta bo napisan v Pythonu, ki bo klical knjižnico Flask.



Slika 1: Model sistema za hranjenje teličkov

3.1 Hranjenje teličkov

Telički potrebujejo različno količino mlečnega nadomestka glede na njihovo maso in starost [1]. Priporočilo je, da mlečni nadomestek vsebuje 20–22 % surovih beljakovin, 17–18 MJ metabolne energije, največ 0,1 % surovih vlaknin in 16–20 % surove maščobe. Pri napajanju telet moramo poskrbeti za čistočo, da mlečni nadomestek dobro premešamo in da ne nastanejo grudice. Zagotoviti moramo ustrezno temperaturo mlečnega nadomestka (39–40 ˚C) ter ustrezno količino in koncentracijo mlečnega nadomestka, ki ga obvezno razdelimo na najmanj dva obroka dnevno.

Naša namišljena teleta bomo poimenovali Martin, Tele in Mirko.

Martin bo tehtal 7 kilogramov, Tele 5 kilogramov, Mirko pa 10 kilogramov.

Starost v tednih	Mleko ali mlečni nadomestek	Voda	Seno	Močna krma
1. teden	Mlezivo po volji (čim več) ali 4–5 l mleka ali ml. nadomestka	Po volji	–	–
2. teden	4–6 litrov (v 3 obrokih)	Po volji	Po volji	Po volji
3.–6. teden	6–8 litrov (2 obrokih)	Po volji	Po volji	Po volji
6.–8. teden	6–4 litrov (2 obrokih)	Po volji	Po volji	1,5 kg
8.–16. teden	postopno zmanjševanje iz 4 na 0 litrov	Po volji	Po volji	2 kg in več

Slika 2: Tabela količine hranjenja

3.2 Omejitve modela

Da bi realizirali hranjenje teličkov, smo se odločili za naslednje značilnosti in omejitve:

- vozilo bo potovalo le naprej in nazaj ter ne bo zavijalo,
- pot bo zaščitena z robniki ali utirjena, da bo ravna,
- s senzorji bomo ugotovili naslednje lokacije:
  a) lokacije teličkov, ko jih gre vozilo hranit, in
  b) začetno lokacijo, ko so vsi telički nahranjeni,
- prihod vozila k teličku bo sprožil mehanizem za hranjenje, ki bo:
  a) spustil roko s posodo s hranilom,
  b) počakal določen čas, potreben za dotok ustrezne količine hranila v vedro s cucljem, glede na pretok iz posode, in
  c) dvignil roko s posodo s hranilom,
- okolje za hranjenje bo imelo svojo vstopno točko Wi-Fi, preko katere vozilo, kontrolni sistem in miniračunalnik komunicirajo,
- hranilo bo tekoče in
- za simulacijo bomo določili, da je 1 minuta v simulaciji enaka 1 uri v resničnem času.

3.3 Lego Mindstorms EV3

Lego Mindstorms EV3 [2] so kocke za sestavljanje, ki omogočajo ustvarjanje različnih robotov in vozil z najnovejšo tehnologijo podjetja Lego. Je komplet za vse navdušence, ki se želijo naučiti osnov programiranja in sestavljanja robotov.

Škatla vsebuje približno 600 delov, miniračunalnik, imenovan Brick, ter tri motorje, daljinec za upravljanje robota, senzor za razdaljo, senzor za barve in senzor s tipko. Dodatno smo kupili še senzor za barvo in senzor za ugotavljanje smeri.

Slika 3: komplet Lego Mindstroms EV3

3.4 Arduino Uno Wi-Fi

Arduino [3] je italijansko podjetje za odprtokodno strojno in programsko opremo, ki načrtuje in izdeluje mikrokrmilnike na eni plošči. Njegovi izdelki strojne opreme so pod licenco CC BY-SA, medtem ko je programska oprema pod licenco LGPL, ki vsakomur dovoljuje proizvodnjo plošč in distribucijo programske opreme. Zasnove plošč Arduino uporabljajo različne mikroprocesorje in krmilnike. Plošče so opremljene z nizi digitalnih in analognih vhodno/izhodnih zatičev (angl. *pin*), ki jih je mogoče povezati z različnimi razširitvenimi ploščami (ščiti) ali testnimi ploščami (za izdelavo prototipov) in drugimi vezji. Plošče imajo serijske komunikacijske vmesnike, vključno z univerzalnim serijskim vodilom (USB) na nekaterih modelih, ki se uporabljajo tudi za nalaganje programov. Mikrokontrolerje je mogoče programirati z uporabo programskih jezikov C in C++. Projekt Arduino se je začel leta 2005 kot orodje za študente na Interaction Design Institute (Ivrea, Italija). Njegov namen je novincem in strokovnjakom omogočati poceni in enostaven način za ustvarjanje naprav.

Slika 4: Mikrokontroler Arduino Uno R4 Wi-Fi

3.5 Raspberry PI 3B

Raspberry PI [4] je miniračunalnik na eni plošči (SBC), ki jih je v Združenem kraljestvu razvila fundacija Raspberry PI v sodelovanju z Broadcomom. Prvotno je bil računalnik namenjen spodbujanju poučevanja osnov računalništva v šolah, kasneje pa se je močno razširil med računalniškimi navdušenci.

Ta miniračunalnik ima naslednje lastnosti:

- 64-bitni štiri jedrni procesor ARM Cortex-A53 (1,2 GHz),
- 1 GB RAM,
- 4 USB vrata (na ena smo priklopili ključek Wi-Fi),
- HDMI za monitor,
- 32 GB kartica SD kot disk in
- Ethernet.

Slika 5: Miniračunalnik Raspberry Pi 3B

Običajno na takem miniračunalniku teče izpeljanka operacijskega sistema Linux (npr. Raspberry PI OS).

4. Raspberry PI

Miniračunalnik Raspberry PI smo uporabili za podatkovno bazo in za strežnik z mikrostoritvami za dostop do te baze. Naš strežnik smo programirali v Pythonu (priloga C). Izvorno kodo smo urejali z urejevalnikom nano.

4.1 Namestitev operacijskega sistema na Raspberry PI 3B

Na miniračunalnik Raspberry PI 3B namestimo RaspBerry PI OS, ki ga najdemo na spletni strani RaspBerry PI. Namešča se preko aplikacije RaspBerry PI Imager. To aplikacijo namestimo in zaženemo.

Slika 6: Aplikacija za pripravo namestitvene kartice SD

Izbrali smo RaspBerry PI OS with Desktop. Aplikacija nam je na kartico SD zapekla sliko operacijskega sistema. Kartico SD smo vstavili v miniračunalnik in ga zagnali.

Za napajanje smo uporabili prenosno powerbank baterijo z 10.000 mAh. Miniračunalniku smo določili naslov IP 192.168.178.69 in mu vklopili strežnik SSH.

4.2 Namestitev baze MySQL na Raspberry Pi 3B

Pred namestitvijo baze smo najprej nadgradili OS na najnovejše popravke:

```
sudo apt update
sudo apt full-upgrade
```

Sledila je namestitev baze MySQL:

```
sudo apt install mariadb-server
sudo mysql_secure_installation
```

Ker je za bazo potreben novi uporabnik, smo ga dodali:

```
sudo su postgres
createuser pi -P –interactive
```

Na vprašanje, ali bo uporabnik imel vlogo superuporabnika, smo odgovorili z DA (Y).

Bazo smo nato zagnali:

```
sudo systemctl start postgresql
```

Nato smo zagnali odjemalca za bazo z imenom psql in kreirali bazo:

```
psql
create database raziskovalna;
\connect raziskovalna;
```

Na bazi smo kreirali tabelo telicki:

```
CREATE TABLE telicki
(
    id int,
    ime varchar(255),
    teza double,
    naslednji int,
    cas timestamp
);
```

Na koncu smo vpisali tri testne teličke, kjer ID predstavlja vrstni red hranjenja in mora biti unikaten:

```
insert into telicki (id, ime, teza, naslednji, cas) values (1, 'Martin', 10, 6,
now());
insert into telicki (id, ime, teza, naslednji, cas) values (2, 'Tele', 15, 4, now());
insert into telicki (id, ime, teza, naslednji, cas) values (3, 'Mirko', 7, 3, now());
```

Da so bili podatki res vpisani, smo naredili poizvedbo:

```
MariaDB [raziskovalna]> select * from telicki;
+------+--------+------+-----------+---------------------+
| id   | ime    | teza | naslednji | cas                 |
+------+--------+------+-----------+---------------------+
|    1 | Martin |   10 |         6 | 2024-03-01 20:38:01 |
|    2 | Tele   |   15 |         4 | 2024-03-01 20:36:01 |
|    3 | Mirko  |    7 |         3 | 2024-03-01 20:38:20 |
+------+--------+------+-----------+---------------------+
3 rows in set (0.012 sec)
```

4.3 Razvoj mikrostoritev na Raspberry Pi 3B

Pri namestitvi OS je bil Python že naložen.

Najprej smo namestili knjižnico Flask in knjižnico za dostop do baze MySQL:

```
sudo apt-get install python3-flask
python -m pip install mysql-connector-python
```

Na miniračunalniku smo razvili preprosti strežnik za mikrostoritve, ki se nahaja v datoteki server.py.

Na začetku programiranja smo naložili knjižnice za Flask in za delo z bazo:

```
from flask import Flask, request
import mysql.connector
```

Za pravilno delovanje kontrolnega sistema smo morali razviti dve mikrostoritvi:

- storitev, ki vrne spisek vseh teličkov in podatek, ali jih je potrebno nahraniti, in
- storitev, ki vsem teličkom, ki bodo ravnokar nahranjeni, izračuna čas naslednjega hranjenja.

Prva storitev se imenuje *status*. Prebere iz baze vse teličke in vrne podatke v CSV:

```
@app.route("/status", methods=['GET'])
def status():
    baza        =        mysql.connector.connect(host="localhost",        user="pi",
password="raspberry", database="raziskovalna", autocommit="true")
    print(baza)
    kurzor = baza.cursor()
    kurzor.execute("select ime, teza, case when cas < now() THEN 1 ELSE 0 end from
telicki order by id;")
    telicki = kurzor.fetchall()
    rezultat = ""
    for x in telicki:
        rezultat += x[0] + "," + str(x[1]) + "," + str(x[2]) + "\n"
    return rezultat
```

Vrnemo ime telička, maso in podatek, ali je čas za hranjenje. Čas za hranjenje ugotovimo tako, da izračunani čas primerjamo s trenutnim časom.

Druga storitev se imenuje *popravi*. Vsem nahranjenim teličkom izračuna čas naslednjega hranjenja tako, da trenutnemu času prišteje vrednost kolone *naslednji*:

```
@app.route("/popravi", methods=['GET'])
def popravi():
    baza        =        mysql.connector.connect(host="localhost",        user="pi",
password="raspberry", database="raziskovalna", autocommit="true"))
    print(baza)
    kurzor = baza.cursor()
    kurzor.execute("update telicki set cas = now() + INTERVAL naslednji MINUTE WHERE
cas < NOW();")
    kurzor.commit()
    return "OK"
```

Povezava na bazo sama samodejno shranjuje spremembe, zato je *autocommit* nastavljen na *true*.

Na koncu zaženemo mikrostoritev:

```
if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

Program zaženemo na roke:

```
python server.py
```

4.4 Ugotovitve

Raspberry PI je deloval zelo hitro. Podatkovna baza je bila odzivna. Strežnik je deloval hitro. Prav tako je bil razvoj na urejevalniku nano hiter brez časovnega zamika.

5. Arduino Uno R4 Wi-Fi

Mikrokontroler Arduino Uno R4 Wi-Fi smo uporabili za izvajalca urnika hranjenja teličkov, pri čemer iz miniračunalnika dobi informacije o hranjenju (čas in masa), vozilu pa pošlje zahtevo po hranjenju. Naš mikrokontroler smo programirali v programskem jeziku C. Za urejevalnik izvorne kode smo uporabljali Arduino IDE.

5.1 Razvoj aplikacije na Arduino Uno R4 Wi-Fi

Program se preko IDE naloži na Arduino in tam ostane, dokler ga ne povozimo z drugim programom.

Ko se Arduino vklopi, se samodejno zažene tudi program.

Najprej smo v programu naložili potrebne knjižnice:

```
#include <WiFiS3.h>
#include <WiFiSSLClient.h>
#include <IPAddress.h>
#include <SPI.h>
#include <ArduinoHttpClient.h>
#include <string.h>
```

Nato smo določili globalni spremenljivki za povezavo Arduina na Wi-Fi:

```
char ssid[] = "TP-Link_453C";
char pass[] = "94809046"
```

V funkciji setup(), ki se zažene ob vklopu Arduina, smo vzpostavili povezavo na Wi-Fi:

```
  WiFi.begin(ssid, pass);
  delay(3000);
  IPAddress ip = WiFi.localIP();
```

Nato pa smo v funkciji loop na vsaki 2 minuti naredili naslednje korake:

1. Iz miniračunalnika preberemo potrebne podatke o hranjenju.
2. Na miniračunalnik pošljemo zahtevo, da se ponovno izračuna čas hranjenja teličkov.
3. Pripravimo ukaz JSON za vozilo.
4. Pošljemo ukaz.

Na miniračunalnikovo mikrostoritev pošljemo zahtevo HTTP, ki nam vrne spisek teličkov z ustreznimi podatki v obliki CSV:

```
WiFiClient c;
HttpClient http(c, "192.168.178.69", 5000);

int err = http.get("/status");

if (err == 0) {
    err = http.responseStatusCode();
```

```
    if (err == 200) {
      String str = http.readString();
      …
    }
}
```

Na enak način obvestimo miniračunalnik, da naredi novi izračun časa hranjenja, a le za tiste teličke, ki so bili nahranjeni:

```
WiFiClient c2;
HttpClient http2(c2, "192.168.178.69", 5000);
err = http2.get("/popravi");

if (err == 0) {
    err = http2.responseStatusCode();

    if (err == 200) {
        String str2 = http2.readString();
        …
    }
}
```

Nato sestavimo ukaz JSON:

```
String body = "{\"ukaz\" : \"nahrani\", \"telicki\": [";
char *token = strtok((char*)str.c_str(), "\n");
bool ok = false;
bool first = true;

while (token != NULL) {
    if (ok) {
        if (first) {
            first =false;
        } else {
            body += ",";
        }

        String vrstica(token);
        int pos1 = vrstica.indexOf(",");
        int pos2 = vrstica.indexOf(",", pos1 + 1);
        String ime = vrstica.substring(0, pos1);
        String teza = vrstica.substring(pos1 + 1, pos2);
        String nahrani = vrstica.substring(pos2 + 1);
        body += "{\"ime\":\"" + ime + "\", \"teza\":" + teza + ", \"nahrani\":";

        if (nahrani.equals("1")) {
            body += "true";
        } else {
            body += "false";
        }

        body += "}";
    }

    if (strlen(token) < 5) {
        ok = true;
    }

    token = strtok(NULL, "\n");
}

body = body + "]}";
```

Na koncu ukaz JSON pošljemo na EV3:

```
WiFiClient c3;
HttpClient http3(c3, "192.168.178.44", 5000);
err = http3.post("/ukaz", "application/json", body.c_str());

if (err == 0) {
    err = http3.responseStatusCode();

    if (err == 200) {
        String str3 = http3.readString();
    }
}
```

5.2 Ugotovitve

Arduino Uno R4 Wi-Fi je prav tako deloval zelo hitro. Program je deloval brezhibno, povezava Wi-Fi je bila stabilna in PIN-ov nismo potrebovali.

6. Lego Mindstorms EV3

Za izdelavo vozila smo uporabili Lego Mindstorms EV3. Komplet ima svoj miniračunalnik, ki se imenuje Brick. Programirali smo v Pythonu. Izvorno kodo smo urejali z urejevalnikom nano.

6.1 Vozilo

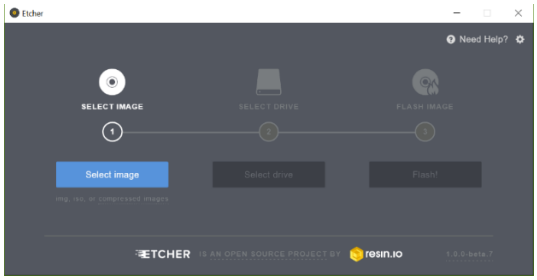
Na vozilo smo dali:

- dva velika motorja za poganjanje vozila naprej in nazaj,
- en srednji motor za premikanje roke s posodo s hranilom in
- en senzor za barve.

6.2 Namestitev operacijskega sistema na Lego Mindstorms EV3

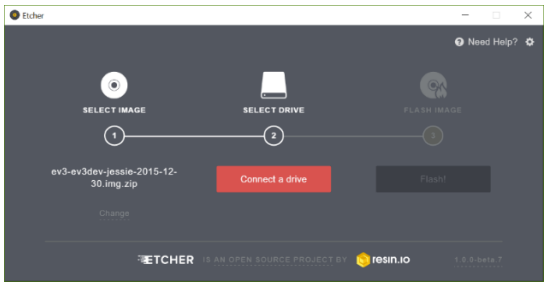
Centralna enota na Lego Mindstorms EV3 se imenuje Brick in je v našem vozilu računalnik, ki pošilja ukaze motorjem in bere vrednosti senzorjev. Na njem smo namestili operacijski sistem Linux (različico EV3 Micropython – ev3dev), ki smo ga dobili na spletni strani EV Micropython. Datoteko s sliko operacijskega sistema smo nato odpakirali na disk.

Na spletni strani programa Etcher smo naložili in namestili aplikacijo, s katero pripravimo kartico SD z Linuxom za naš računalnik. Potem smo jo zagnali in izbrali našo sliko EV3.



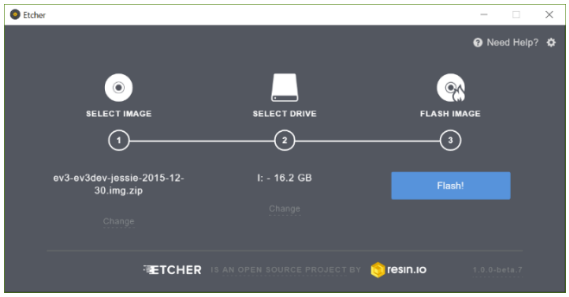
Slika 7: Aplikacija za pripravo namestitvene kartice SD – prvi korak

Sledila je izbira spominske kartice SD.



Slika 8: Aplikacija za pripravo namestitvene kartice SD – drugi korak

Ko smo pritisnili na gumb Flash!, se je izvedel zapis na spominsko kartico.



Slika 9: Aplikacija za pripravo namestitvene kartice SD – tretji korak

Pripravljeno spominsko kartico SD smo vstavili v enoto Brick in zagnali računalnik. Ko se je sistem naložil, smo mu nastavili Wi-Fi in mu dodelili naslov IP 192.168.178.44.

Preko računalniškega terminala (mRemoteNG) smo se povezali na EV3 preko povezave SSH. Uporabniško ime je bilo *robot*, geslo pa *maker*.

```
Using username "robot".
Keyboard-interactive authentication prompts from server:
End of keyboard-interactive prompts from server
Linux ev3dev 4.14.117-ev3dev-2.3.5-ev3 #1 PREEMPT Sat Mar 7 12:54:39 CST 2020 armv5tejl
            _____     _
   _____   _|___ /  __| | _____   __
  / _ \ \ / / |_ \ / _` |/ _ \ \ / /
 |  __/\ V / ___) | (_| |  __/\ V /
  \___| \_/ |____/ \__,_|\___| \_/

Debian stretch on LEGO MINDSTORMS EV3!
Last login: Sun Oct 31 17:18:24 2021 from 192.168.178.22
robot@ev3dev:~$ python
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

Slika 10: Dostop do miniračunalnika preko terminala

Programski jezik Python je že del tega operacijskega sistema.

Najprej smo naredili nadgradnjo sistema:

```
sudo apt-get update
```

Nato smo namestili Flask, ki je ogrodje (ang. *framework*) za razvoj spletnih mikrostoritev za Python. Z njim pišemo spletne storitve, ki Arduinu omogočajo, da zahteva novo nalogo hranjenja. Namestili smo ga z ukazom:

```
sudo apt-get install python3-flask
```

Sledila je namestitev Python namestitvene aplikacije PIP:

```
sudo apt install python3-pip
```

Na koncu smo s to namestitveno aplikacijo namestili še knjižnico za delo z EV3:

```
pip3 install pybricks
```

6.3 Razvoj mikrostoritev na Lego Mindstorms EV3

Na Lego Mindstorms EV3 smo razvili preprost strežnik za mikrostoritve, ki se nahaja v datoteki vozilo.py. Strežnik smo napisali z uporabo programskega jezika Python (priloga A). Pri tem smo uporabili knjižnico Flask. Naš strežnik posluša na vratih 5000.

Na začetku naložimo knjižnice za Flask in za vozilo:

```
from flask import Flask, request
from ev3dev.ev3 import *
from threading import Thread
from ev3dev2.sensor.lego import ColorSensor
from time import sleep
from ev3dev2.sound import Sound
import json
```

Da lahko delamo z vozilom, ustvarimo objekte za delo z barvnim senzorjem, zvokom ter dvema motorjema. Prav tako imamo zastavico, ki pove, ali se vozilo premika. Za delo s Flaskom naredimo spremenljivko app.

```
color_sensor = ColorSensor()
sound = Sound()
premikanje = True
levi_motor = LargeMotor('outA')
desni_motor = LargeMotor('outB')
srednji_motor = MediumMotor('outC')
app = Flask(__name__)
```

Motorja smo priklopili na izhoda A in B.

Slika 11: Zasedenost vrat na enoti Brick EV3

Naš program zažene strežnik Flask in še eno nit, ki skrbi za nadzor nad premikanjem.

```
if __name__ == '__main__':
    thread = Thread(target=zanka)
    thread.start()
    app.run(host="0.0.0.0")
```

Za premikanje vozila kličemo funkcijo run_forever(), ki sprejme hitrost kot parameter. Hitrost je lahko pozitivna ali negativna številka, ker se lahko vrti v obe smeri.

Da se vozilo začne premikati naprej, moramo nastaviti negativno vrednost –100:

```
levi_motor.run_forever(speed_sp=-100)
desni_motor.run_forever(speed_sp=-100)
```

Da se vozilo začne premikati nazaj, moramo nastaviti pozitivno vrednost 100:

```
levi_motor.run_forever(speed_sp=100)
desni_motor.run_forever(speed_sp=100)
```

Da se vozilo ustavi, moramo nastaviti vrednost na 0:

```
levi_motor.run_forever(speed_sp=0)
desni_motor.run_forever(speed_sp=0)
```

Imamo štiri mikrostoritve, ki poganjajo vozilo:

- naprej, ki začne premikati vozilo naprej,
- nazaj, ki začne premikati vozilo nazaj,
- stop, ki vozilo ustavi, in
- ukaz, ki sprejme spisek teličkov v obliki JSON.

Primer ukaza, ki pokliče http://192.168.178.44:5000/ukaz in pošlje zahtevo kot ukaz POST:

```
{
    "ukaz" : "nahrani",
    "telicki": [
        {
            "ime": "Belka",
            "teza": 7,
            "nahrani": true
        },
        {
            "ime": "Miro",
            "teza": 5,
            "nahrani": false
        },
        {
            "ime": "Martin",
            "teza": 10,
            "nahrani": true
        }
    ]
}
```

Trenutno poznamo le en ukaz, in to je *nahrani*. Poslan je spisek teličkov s podatki:

- ime,
- masa in
- zastavica, ki pove, ali je telička potrebno nahraniti.

Vozilo z barvnim senzorjem prepozna telička ob rdeči barvi, se pri njem ustavi in ga nahrani.

Če vozilo pride do telička, ki se ga trenutno ne hrani, nadaljuje pot do naslednjega telička.

Pri teličku vozilo naredi naslednje:

1. spusti posodo s hranilom,
2. počaka nekaj časa, da se vedro s cucljem napolni, in
3. dvigne posodo s hranilom.

Ko vozilo pride do zadnjega telička, je zadnji korak vožnja nazaj do začetne točke, kjer posodo s hranilom po potrebi spet napolnimo.

Začetno točko prepoznamo po modri barvi.

Posebna nit skrbi za prepoznavanje teličkov in izvede ustrezne korake.

Premik roke:

```
if spisek[pos]["nahrani"] == True:
    srednji_motor.run_forever(speed_sp=-20)
    time.sleep(4)
    srednji_motor.run_forever(speed_sp=0)
    time.sleep(1 * spisek[pos]["teza"])
    srednji_motor.run_forever(speed_sp=20)
    time.sleep(4)
    srednji_motor.run_forever(speed_sp=0)
```

Masa telička pove, koliko hranila potrebuje oz. koliko časa mora hranilo teči v vedro s cucljem. Zato potrebujemo podatek o pretoku iz posode s hranilom.

Strežnik vedno odgovori z sporočilom JSON:

```
{"status": 1}
```

Program zaženemo na roke:

```
python vozilo.py
```

6.4 Ugotovitve

Lego Mindstorms EV3 je že zelo star Lego komplet, za katerega se je izkazalo, da je zelo počasen. Vse terminalske akcije so vsakih 10 sekund zamrznile za nadaljnjih 10 sekund. Že samo urejanje na urejevalniku nano je bilo zelo počasno. Zagon našega Python programa je trajal več kot minuto. Kljub temu smo bili presenečeni, da so senzorji delovali v realnem času. Vseeno se je po določenem času aplikacija sesula.

7 Zaključek

Raziskovalna naloga je bila uspešno zaključena in uspelo nam je razviti model hranjenja teličkov s pomočjo kompleta Lego Mindstorms EV3, mikrokontrolerja Arduino Uno R4 Wi-Fi in mikroračunalnika Raspberry PI 3B. Programski jezik Python smo spoznali že v 6. razredu pri pouku predmeta Računalništvo, zato programiranje na Lego Mindstorms EV3 in Raspberry PI ni bilo težavno. Prav tako smo v 9. razredu programirali na Arduinu. Tako da smo vsa ta naša znanja uporabili v tej nalogi.

Na Raspberry PI smo naložili Raspberry PI OS, ustrezne knjižnice za Python in podatkovno bazo MySQL, ki smo jo prvič uporabljali in se po novem imenuje MariaDB. Za delo z bazo smo uporabljali stavke SQL (INSERT, UPDATE in SELECT). Prav tako smo uporabili knjižnico Flask za razvoj mikrostoritev.

Na Lego Mindstorms EV3 smo naložili izpeljanko Linuxa z imenom ev3dev. Na njej smo uporabili knjižnico za delo z lego motorji in senzorji, prav tako smo dodali knjižnico Flask za razvoj mikrostoritev.

Pri nalogi smo poskušali ugotavljati veljavnost treh hipotez:

- Ali je Lego Mindstorms EV3 s svojimi senzorji sposoben natančno izvajati svoje opravilo hranjenja, da pri tem telički ne ostanejo brez hrane?
- Ali je Arduino Uno sposoben kontrolirati EV3 na daljavo brezžično brez uporabe svojih PIN-ov?
- Ali je Raspberry PI 3B dovolj močan, da lahko deluje istočasno kot podatkovna baza in vstopna točka?

Ker Lego Mindstorms EV3 ne deluje dovolj hitro in stabilno, je bila prva hipoteza ovržena.

Ostali dve hipotezi sta bili potrjeni, kar dokazuje, da sta Arduino in Raspberry PI odlična produkta.

Pri raziskovalni nalogi sta bili dve hipotezi potrjeni, ena pa ovržena.

Priloga A – Viri

[1] Vzreja telet – PDF dokument . Dosegljivo: https://lj.kgzs.si/Portals/1/A-Splet2020/TL090%20-%20Vzreja%20telet%20-%202020.pdf.

Zadnji popravek: 2023. [Dostopno: 12.2.2024]

[2] Lego Mindstorms EV3 - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Lego_Mindstorms_EV3.

Zadnji popravek: 2023. [Dostopno: 15.2.2024]

[3] Arduino – Wikipedia . Dosegljivo: https://en.wikipedia.org/wiki/Arduino.

Zadnji popravek: 2023. [Dostopno: 18.2.2024]

[4] RaspBerry PI – Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Raspberry_Pi.

Zadnji popravek: 2023. [Dostopno: 1.3.2024]

Priloga B – Program za EV3: vozilo.py

```
#!/usr/bin/env python3
from flask import Flask, request
from ev3dev.ev3 import *
from threading import Thread
from ev3dev2.sensor.lego import ColorSensor
from time import sleep
from ev3dev2.sound import Sound
import json

spisek = {}
color_sensor = ColorSensor()
sound = Sound()
premikanje = True
levi_motor = LargeMotor('outA')
desni_motor = LargeMotor('outB')
srednji_motor = MediumMotor('outC')
app = Flask(__name__)

def zanka():
    global spisek, premikanje
    pos = 0
    nazaj = False

    while True:
        if len(spisek) == 0:
            continue

        color = color_sensor.color
        text = ColorSensor.COLORS[color]

        if nazaj == True:
           if color == 2:
               levi_motor.run_forever(speed_sp=0)
               desni_motor.run_forever(speed_sp=0)
               pos = 0
               nazaj = False
               spisek = {}
               continue
           else:
               levi_motor.run_forever(speed_sp=100)
               desni_motor.run_forever(speed_sp=100)
               continue

        if color == 5 and premikanje == True:
                premikanje = False
                levi_motor.run_forever(speed_sp=0)
                desni_motor.run_forever(speed_sp=0)
                if spisek[pos]["nahrani"] == True:
                    srednji_motor.run_forever(speed_sp=-20)
                    time.sleep(4)
                    srednji_motor.run_forever(speed_sp=0)
```

```
                    time.sleep(1 * spisek[pos]["teza"])
                    srednji_motor.run_forever(speed_sp=20)
                    time.sleep(4)
                    srednji_motor.run_forever(speed_sp=0)
                pos += 1
                if pos == len(spisek):
                   nazaj = True
                   continue
                for i in range(600):
                    if pos != len(spisek):
                        levi_motor.run_forever(speed_sp=-100)
                        desni_motor.run_forever(speed_sp=-100)
                premikanje = True

@app.route("/ukaz", methods=['POST'])
def ukaz():
    global spisek, premikanje
    print("Ukaz")
    data = request.get_json()
    spisek = data["telicki"]
    premikanje = True
    levi_motor.run_forever(speed_sp=-100)
    desni_motor.run_forever(speed_sp=-100)
    return "{\"status\":1}"
@app.route("/naprej")
def naprej():
    print("Naprej")
    levi_motor.run_forever(speed_sp=-100)
    desni_motor.run_forever(speed_sp=-100)
    return "{\"status\":1}"

@app.route("/nazaj")
def nazaj():
    print("Nazaj")
    levi_motor.run_forever(speed_sp=100)
    desni_motor.run_forever(speed_sp=100)
    return "{\"status\":1}"

@app.route("/stop")
def stop():
    print("Stop")
    levi_motor.run_forever(speed_sp=0)
    desni_motor.run_forever(speed_sp=0)
    return "{\"status\":1}"

print("Started")
if __name__ == '__main__':
    thread = Thread(target=zanka)
    thread.start()
    app.run(host="0.0.0.0")
```

Priloga C - Program za Arduino: telicki.ino

```
#include <WiFiS3.h>
#include <WiFiSSLClient.h>
#include <IPAddress.h>
#include <SPI.h>
#include <ArduinoHttpClient.h>
#include <string.h>

char ssid[] = "TP-Link_453C";
char pass[] = "94809046";

void setup() {
  Serial.begin(9600);

  Serial.println("Starting...");
  WiFi.begin(ssid, pass);

  Serial.println("Connecting");
  delay(3000);

  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  IPAddress ip = WiFi.localIP();
  Serial.print("IP: ");
  Serial.println(ip);
}

void loop() {
  Serial.println("Loop");
  WiFiClient c;
  HttpClient http(c, "192.168.178.69", 5000);

  int err = http.get("/status");

  if (err == 0) {
    err = http.responseStatusCode();

    if (err == 200) {
      String str = http.readString();

      WiFiClient c2;
      HttpClient http2(c2, "192.168.178.69", 5000);
      err = http2.get("/popravi");

      if (err == 0) {
        err = http2.responseStatusCode();

        if (err == 200) {
          String str2 = http2.readString();

          String body = "{\"ukaz\" : \"nahrani\", \"telicki\": [";
```

```
          char *token = strtok((char*)str.c_str(), "\n");

          bool ok = false;
          bool first = true;

          while (token != NULL) {
            if (ok) {
              if (first) {
                first =false;
              } else {
                body += ",";
              }

              String vrstica(token);
              int pos1 = vrstica.indexOf(",");
              int pos2 = vrstica.indexOf(",", pos1 + 1);

              String ime = vrstica.substring(0, pos1);
              String teza = vrstica.substring(pos1 + 1, pos2);
              String nahrani = vrstica.substring(pos2 + 1);

              body += "{\"ime\" : \"" + ime + "\", \"teza\" : " + teza
+ ", \"nahrani\" : ";

              if (nahrani.equals("1")) {
                body += "true";
              } else {
                body += "false";
              }

              body += "}";
            }

            if (strlen(token) < 5) {
              ok = true;
            }

            token = strtok(NULL, "\n");
          }

          body = body + "]}";
          Serial.println(body);

          WiFiClient c3;
          HttpClient http3(c3, "192.168.178.44", 5000);
          err = http3.post("/ukaz", "application/json", body.c_str());

          if (err == 0) {
            err = http3.responseStatusCode();

            if (err == 200) {
              String str3 = http3.readString();
              Serial.println(str3);
```

```
            }
          }
        }
      }
    }
  }

  delay(120000);
}
```

Priloga D – Program za Raspberry PI: server-py

```
#!/usr/bin/env python3
from flask import Flask, request
import mysql.connector
import json

app = Flask(__name__)

@app.route("/status", methods=['GET'])
def status():
    baza = mysql.connector.connect(host="localhost", user="pi",
password="raspberry", database="raziskovalna", autocommit="true")
    print(baza)
    kurzor = baza.cursor()
    kurzor.execute("select ime, teza, case when cas < now() THEN 1 ELSE
0 end from telicki order by id;")
    telicki = kurzor.fetchall()
    rezultat = ""

    for x in telicki:
        rezultat += x[0] + "," + str(x[1]) + "," + str(x[2]) + "\n"

    return rezultat

@app.route("/popravi", methods=['GET'])
def popravi():
    baza = mysql.connector.connect(host="localhost", user="pi",
password="raspberry", database="raziskovalna", autocommit="true")
    print(baza)

    kurzor = baza.cursor()
    kurzor.execute("update telicki set cas = now() + INTERVAL naslednji
MINUTE WHERE cas < NOW();")

    return "OK"

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```