Ivan Grdičak 0036491468
Tomislav Matulić 1191215272
Vito Papa 0036491772
Matteo Samsa 0069070580
Ivan Matković 0036492582

SEMINARSKI RAD - SPVP

Fakultet elektrotehnike i računarstva, Sveučilišta u Zagrebu
Zavod za elektroničke sustave i obradbu informacija
Sveučilište u Zagrebu

PetCare - skrb za kućne ljubimce

- Daljinska skrb i zabava za ljubimce
- Ugradbeni računalni sustavi
- Upravljanje pomoću aplikacije
- Komunikacija preko servera
- Raspberry Pi i Arduino
- Povezivanje podsustava

13. lipnja 2019

Sažetak

Sustav Petcare služi za daljinsku skrb o kućnim ljubimcima. Sustav omogućava vlasniku da nahrani, zabavi svog ljubimca te da putem kamere i detektora buke provjeri stanje u prostoriji gdje boravi ljubimac. Sustav je namijenjen korisnicima kojih često nema kod kuće i imaju kućnog ljubimca (potrebno je imati pristup internetu). Podsustav za zabavu kao i vremenski raspored hranjenja korisnik može koristiti i kada je kod kuće, pogotovo ako je teže pokretna osoba. U ovom radu su opisani podsustavi za hranjenje iz 3 spremnika hrane, detekciju trajanja buke i impulsa buke te zabavu uz dvije varijante (laser za mačke i izbacivač loptice za pse) ostvarenih na Arduinu. Opisana je Android mobilna aplikacija putem koje korisnik prima informacije od sustava (prikazuje korisniku obavijesti) i upravlja sustavom. Komunikacija od mobilne aplikacije do Arduina je ostvarena preko Raspberry Pi-a (FTP server) koji putem serijske veze USB-om komunicira s Arduinom, a preko Interneta s aplikacijom (FTP klijent). Prednosti sustava su mogućnost nadzora ljubimca, jednostavna i brza daljinska skrb i uvid u stanje, a nedostaci su što sustav ne omogućuje davanje vode i u slučaju da mobilni telefon ili server nema pristup Internetu sustav ne može raditi.

Sadržaj

Ovaj seminarski rad je izrađen u okviru predmeta „Sustavi za praćenje i vođenje procesa“ na Zavodu za elektroničke sustave i obradbu informacija, Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu.

1. Uvod

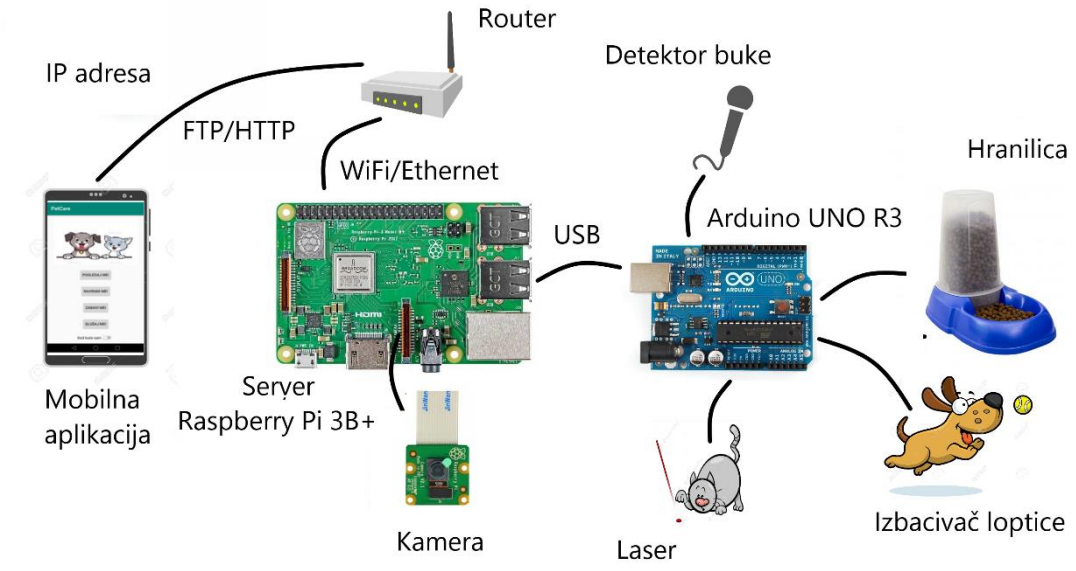
Projekt PetCare je sustav koji u današnjem svijetu predstavlja dio „pametne kuće" korisnicima koji posjeduju kućne ljubimce. Sustav omogućava daljinsku skrb o kućnim ljubimcima. Osobama koje posjeduju kućne ljubimce koji borave u unutarnjim prostorima, a zbog obaveza su često odsutne od kuće te se njihovo odsustvo ponekad produži od nekoliko sati do dva dana je potreban sustav PetCare. Osim osobama koje su često odsutne sustav PetCare je namijenjen i korisnicima koji su teže pokretni i posjeduju ljubimce te im je olakšana skrb o njima.

Sustav PetCare omogućava hranjenje ljubimca po unaprijed postavljenom rasporedu, hranjenje na zahtjev korisnika putem mobilne aplikacije. Hranilica može sadržavati 3 različite vrste hrane i korisnik može odabrati količinu i vrstu hrane putem aplikacije. Zabava za kućnog ljubimca je ostvarena upravljivim laserom (namijenjeno za mačke) i izbacivačem za loptice (namijenjeno za pse). Sustav sadrži podsustave za detekciju impulsa i trajanje buke te podsustav za video nadzor prostora pomoću kojih korisnik ima uvid u stanje ljubimca i prostora. Putem aplikacije korisniku stižu obavijesti o detektiranoj buci, o nedostatku hrane ili o drugoj greški sustava. Na primljenu obavijest ili vlastitom voljom korisnik može vizualno putem kamere pogledati stanje jer je vizualno najlakše utvrditi o kakvom se problemu radi i postoji li uopće problem.

Komunikacija sustava i aplikacije je ostvarena putem FTP servera kojem je za pristup potrebno korisničko ime i lozinka, što donekle jamči sigurnost sustava, odnosno samo korisnik svojim „login"-om može upravljati sustavom. FTP(*File Transfer Protocol*) server je podignut na Raspberry Pi-u i serijskom vezom je povezan s Arduinom koji upravlja sustavom. Korisnik aplikacijom može upravljati sustavom ako server i mobilni telefon imaju pristup Internetu. Ako korisnik nema pristup Internetu, ne može video nadzorom vidjeti ljubimca, ni upravljati laserom, nahraniti ljubimca ili primati obavijesti. Međutim sustav omogućava hranjenje po rasporedu (koji je korisnik prethodno postavio). Kako bi hranjenje po rasporedu radilo potrebno je samo napajanje i ostvarena serijska veza što je osnovna prednost sustava. Tako korisnik kod kuće na lokalnoj mreži može unaprijed podesiti raspored hranjenja. Greška sustava i detektirana buka će biti dojavljeni uspostavom Internetske veze. Video nadzor je dostupan putem HTTP zahtjeva te je mana što je moguće da netko zloupotrijebi tu činjenicu.

2. Opis cijelog PetCare sustava

Sustav se sastoji od 3 glavna upravljačka dijela, a to su Raspberry Pi 3B+, Arduino i Android mobilna aplikacija. Arduino upravlja hranilicom, detektorom buke, laserom i izbacivačem loptice. Arduino je serijskom vezom povezan s Raspberry Pi-em te njemu šalje informacije o stanju sustava, a od njega prima postavke i naredbe koje korisnik šalje putem aplikacije (server i aplikacija povezani su na Internet). Mobilna aplikacija je FTP i HTTP klijent, a Raspberry Pi je FTP server (datoteke koje se prenose su u JSON formatu i služe za upravljanje sustavom i nadzor sustava) i HTTP server (služi za video prijenos uživo). Korisnik većinom pristupa serveru kad nije na istoj lokalnoj mreži pa je u korisnikovom usmjerniku potrebno podesiti prosljeđivanje mrežnih vrata (FTP, HTTP i port po želji za kameru). Kamera korištena u projektu je Pi Camera Module v2 i povezana je direktno na Raspberry Pi. Na Slika 1 je prikazana shema cijelog sustava podijeljenog na podsustave. U sljedećim poglavljima je detaljnije opisan svaki od podsustava.

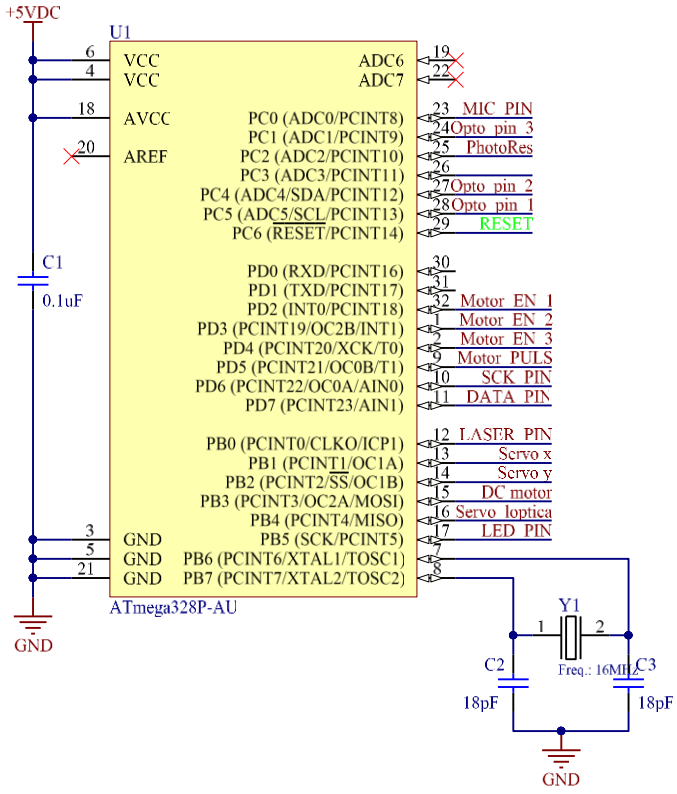


Slika 1 Shematski prikaz cijelog PetCare sustava

3. Sustav za praćenje i upravljanje

Mobilna aplikacija i Raspberry Pi razmjenjuju podatke preko FTP servera, a Raspberry Pi i Arduino podatke razmjenjuju preko serijske komunikacije.
Na Arduino razvojnu pločicu su povezani slijedeći funkcionalni dijelovi:

- Hranilica
- Detektor buke
- Laser igračka za mačke
- Izbacivač loptica za pse



Slika 2 Arduino s korištenim pinovima

3.1. Glavni program (Matteo Samsa)

Prilikom pokretanja sustava izvršava se inicijalizacija svih dijelova, nakon čega program ulazi u beskonačnu petlju – *loop()*. U svakoj iteraciji petlje program očita vrijeme, izraženo u milisekundama, koje je proteklo od pokretanja sustava pomoću funkcije *millis()* za potrebe izvršavanja određenih funkcija. Nakon toga, provjerava međuspremnik serijske

komunikacije, te u slučaju da ima novih podataka ih učita u polje tipa integer. Svaka poruka koju Arduino prima u sebi sadrži 5 bajtova: bajt za provjeru, identifikator, i tri bajta podataka. Više o porukama koje se razmjenjuju bit će navedeno kasnije u tekstu.

S obzirom na identifikator izvršava se određena grana *Switch-Case* funkcije. U svakoj grani se poziva funkcija za određenu funkcionalnost koja iskorištava jedan, dva ili sva tri bajta podataka iz poruke.

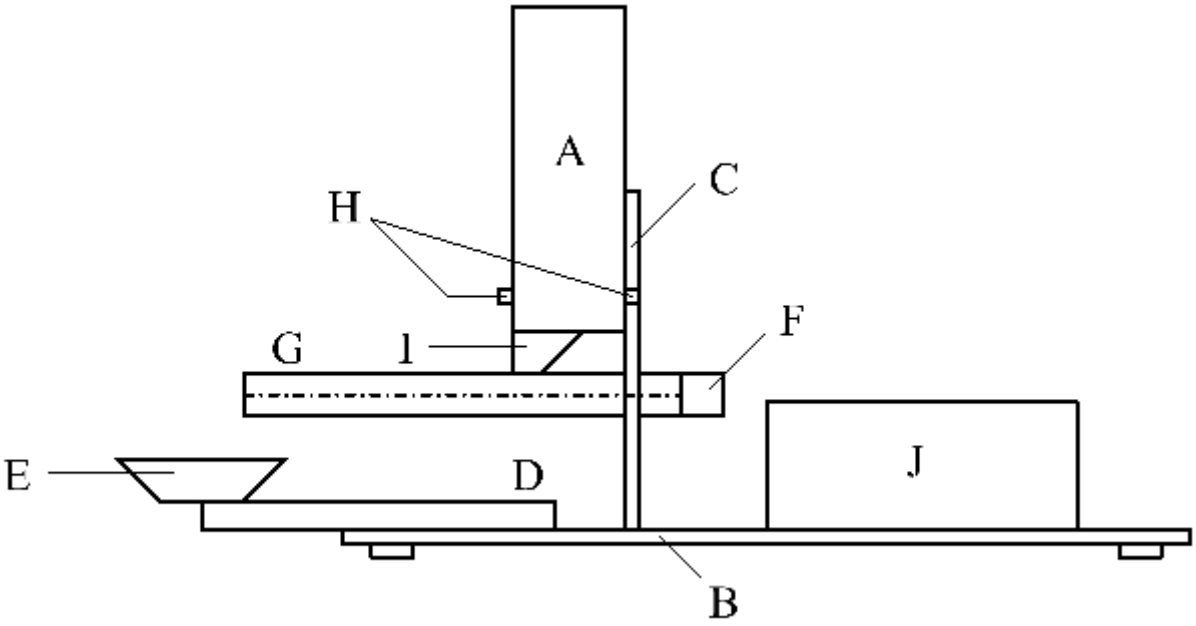
```
switch (msg[1]){
  case 1: write3(CHECK_BYTE, HRANILICA_ID_SEND, hranilica(msg[2], msg[3]));
          break;
  case 2: write3(CHECK_BYTE, DETEKTOR_ID_SEND, set_detektor(msg[2], msg[3],
msg[4]));
          break;
  case 3: laser(msg[2], msg[3], msg[4]);
          break;
  case 4: loptica(msg[2]);
          break;
  default: break;
}
```

Uz navedeno, u glavnom programu se pozivaju i funkcije koje se neprestano izvršavaju, na primjer, funkcija za detekciju buke.

3.2. Hranilica (Vito Papa)

Hranilica je dio sustava zaslužna za hranjenje kućnog ljubimca. Sastoji se od fizičkog dijela i programskog koda potrebnog za rad. Fizički dio sastoji se od postolja na kojemu se nalazi senzor za mjerenje mase te tri jedinice za dostavu i pohranu hrane. Svaka jedinica sastoji se od nosača, spremnika za hranu (na kojemu se nalaze senzori za provjeru količine hrane), prijenosne jedinice (za dostavu hrane), drivera za motor i motor.

Programski kod napravljen u Arduino IDE te je namijenjen za Arduino razvojne pločice.



Slika 3 Shema fizičke izvedbe hranilice

Tablica 1 Komponente prikazane na shemi

A	Spremnik hrane
B	Postolje
C	Nosač
D	Senzor mase
E	Zdjelica
F	Motor
G	Zupčanik u kućištu
H	Senzor hrane
I	Lijevak
J	Pomoćna elektronika

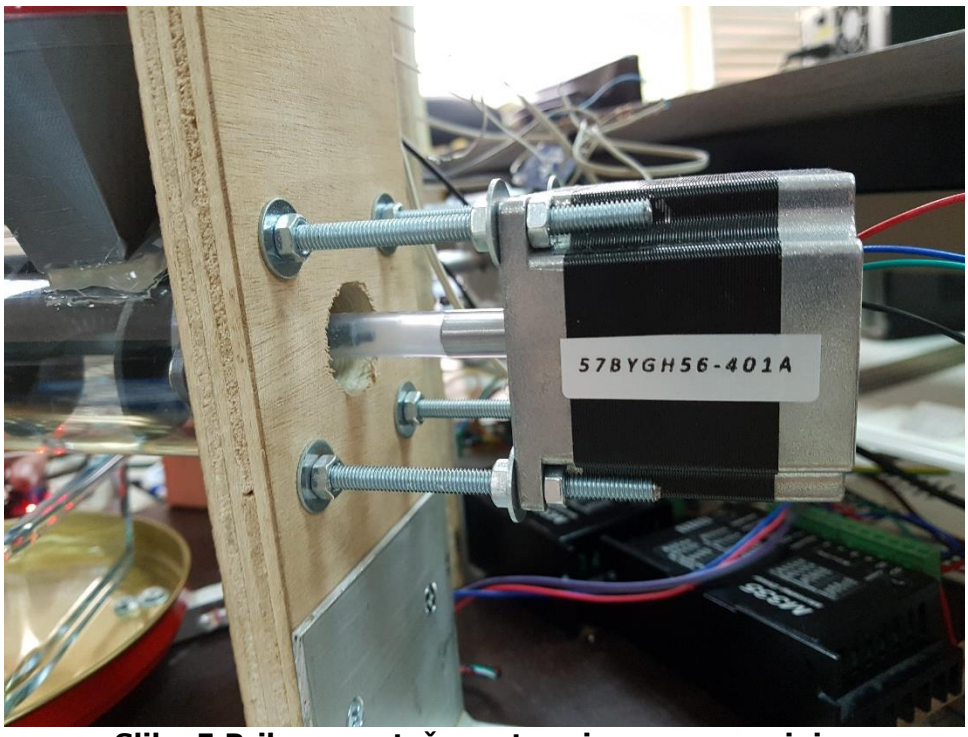
3.2.1. Driver za motor i motor

Korišteni su Nema 23 stepper motori sa M335 drajverom. Drajver služi kako bi upravljački krug galvanski odijelio od kruga za napajanje i kontrolu motora. Svaki drajver ima priključak za napajanje, priključak kojim je spojen na motore (B-, B+, A-, A+) i priključak za signale sa upravljačke jedinice. Osim toga, preko Motorima je moguće upravljati s 3 signala: PUL, DIR, ENA. Na rastući brid signala PUL motor će se okrenuti za 1 korak. Korak kao i još neke Signali DIR i ENA označavaju smjer okretanja i omogućavanje rada motora. Svi signali na drajveru imaju diferencijalni ulaz, npr. PUL+ i PUL-. Ako je PUL+ na razini kao i PUL-,

logička razina koju detektira drajver je niska, a ako je razlika veća od 4V, razina je visoka.

Slika 4 Drajver korišten u projektu

Sam motor je preko 4 navojne šipke učvršćen na nosač te učvršćene podloškama i maticama. Osovina motora spojena je gumenom cijevi (spojnicom) na osovinu svrdla radi smanjivanja vibracija.



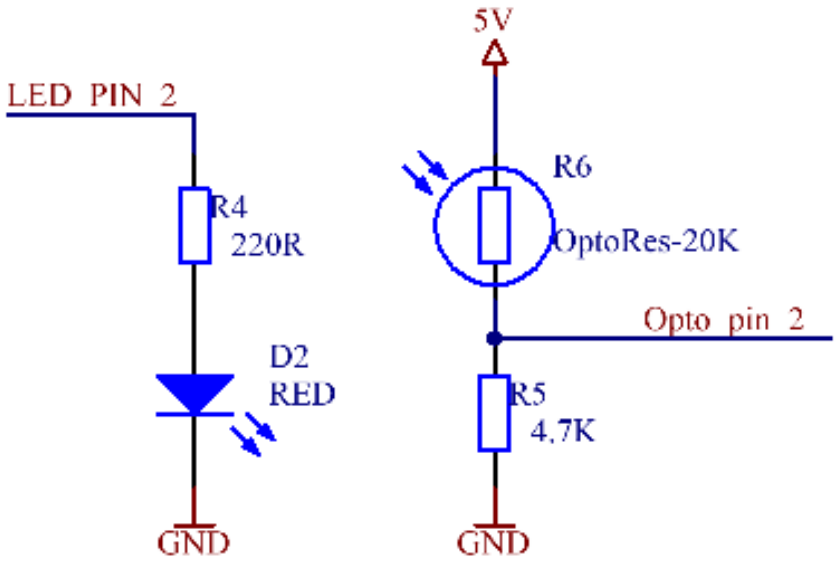
Slika 5 Prikaz montaže motora i gumena spojnica

3.2.2. Spremnik hrane i detektor nestanka hrane

Spremnik hrane načinjen je od pakiranja Pringles-a i lijevka za usmjeravanje hrane. Pakiranje Pringlesa odabrano je zbog inherentne namijenjenosti za prehrambene proizvode i jednostavnosti punjenja spremnika (jednostavan poklopac).

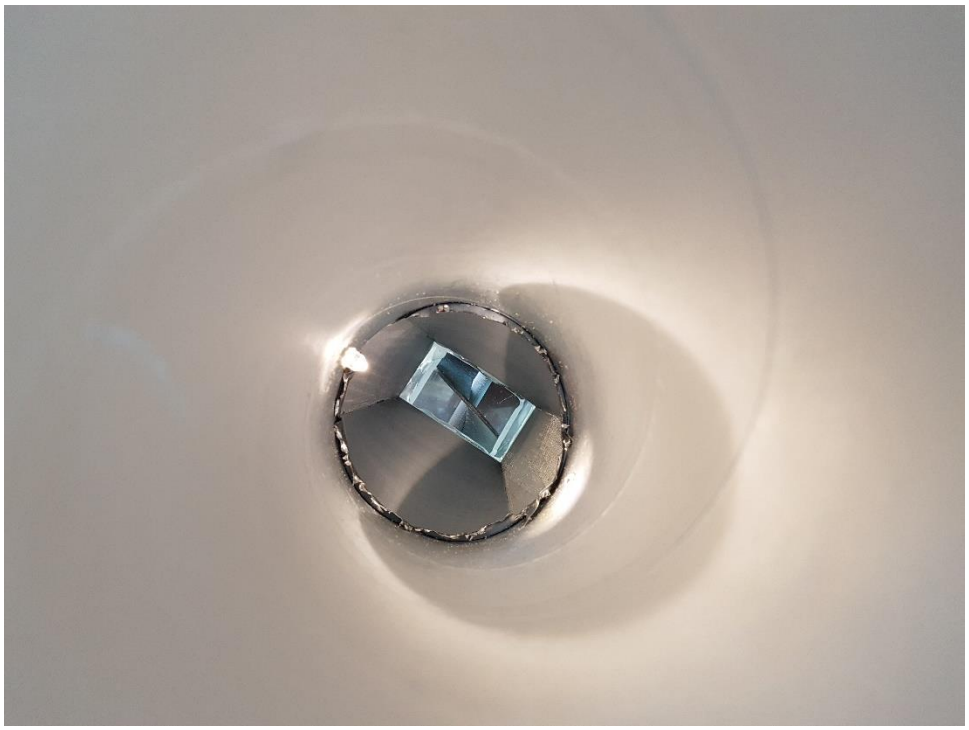
Slika 6 Prikaz spremnika hrane načinjen od pakiranja Pringles-a

Svaki spremnik hrane pri dnu ima senzor za slanje signala prilikom nestanka hrane u spremniku. Senzor je realiziran kao fotootpornik s jedne strane spremnika te led diode s druge. Spojna shema realizirana je kao na slici :



Slika 7 Shematski prikaz detektora hrane u spremniku

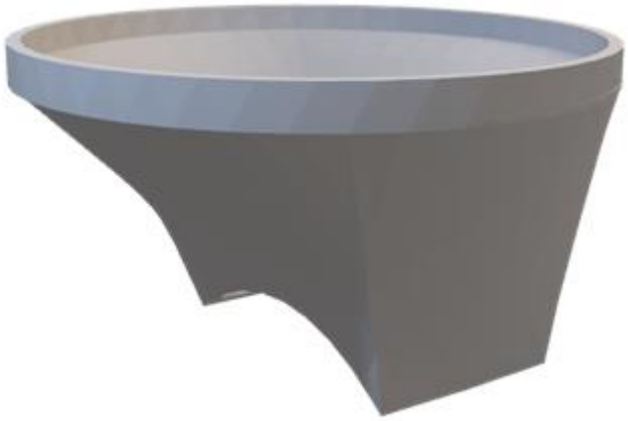
Kada razina hrane u spremniku padne dovoljno nisko da svjetlost može doprijeti do fotootpornika, njegov otpor se smanji. S obzirom na to da je Opto_pin spojen na analogni ulaz razvojne pločice Arduino Uno, moguće je detektirati tu promjenu i izvršiti potreban kod za obradu signala. Led dioda i fotootpornik osigurani su vrućim ljepilom.



Slika 8 Prazan spremnik, svjetlost pada na fotootpornik

Prazan spremnik, može se jasno vidjeti kako svjetlost pada na fotootpornik kada hrana ne sprječava put

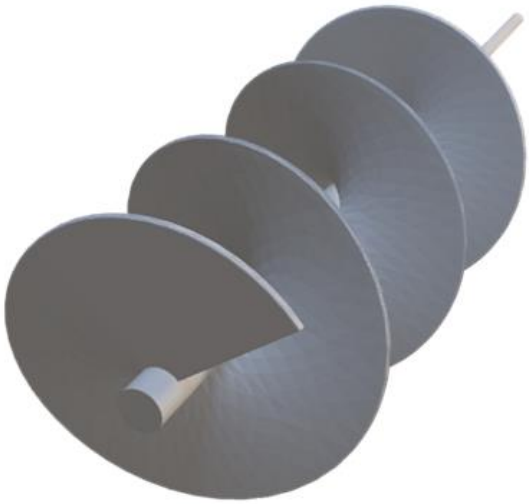
Konačno, izlaz spremnika hrane i ulaz cijevi sa zupčanikom spaja lijevak za usmjeravanje hrane. Lijevak je dizajniran da kvalitetno prianja uz obli rub cijevi, te da je moguće učvrstiti spremnik hrane u njega. Model je dizajniran u AutoCAD 2019 razvojnom okruženju. Nakon toga, izveden je u format pogodan za 3D printanje. 3D printanje izvedeno je pomoću Prusa mk2 printera, za dobivanje presjeka, skaliranje i orijentaciju objekata korišten je program PrusaSlicer. Lijevak je za spremnik hrane kao i za cijev sa zupčanikom spojen vrućim ljepilo.



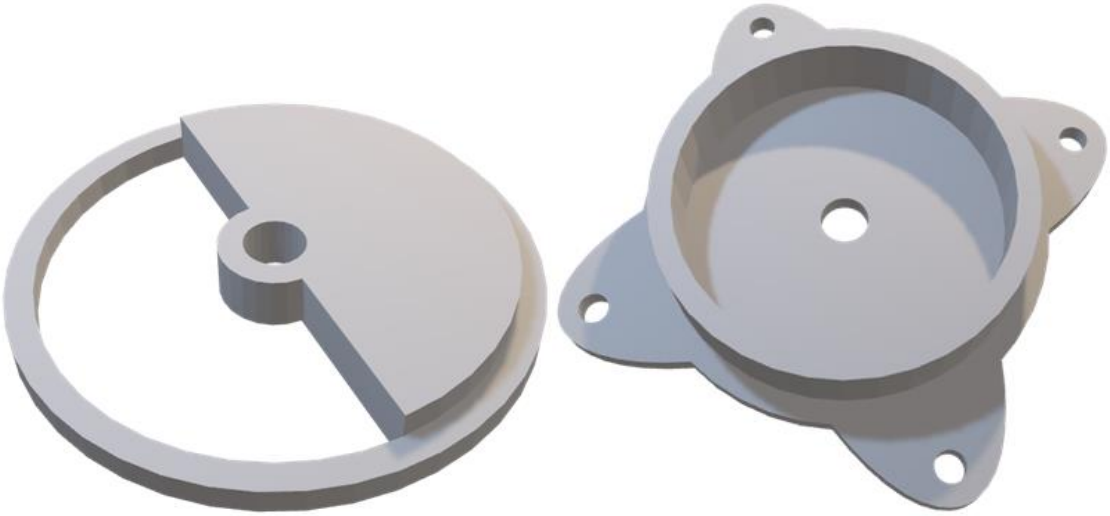
Slika 9 3D model lijevka za usmjeravanje hrane

3.2.3. *Prijenosnik hrane*

Prijenosnik hrane je dio sustava koji prenosi hranu iz spremnika u zdjelicu. Realiziran je kao prozirna cijev s izduženim zupčanikom (svrdlom). Tijekom utjecaja gravitacije, hrana iz spremnika hrane pada kroz lijevak u cijev i puni prostor između latica zupčanika. Kada mikrokontroler dobije naredbu i pokrene motore, zupčanik gura hranu do izlaza, te ona pada u zdjelicu. Zupčanik je preuzet sa https://grabcad.com/library/auger , no prilagođen je primjeni u AutoCAD 2019 razvojnom okruženju (podebljana je osovina i latice radi smanjena mogućnosti pucanja). Čepovi na krajevima cijevi samostalno su izrađeni U AutoCAD-u. Osim učvršćivanja cijevi za nosač, čepovi služe i kao ležajevi za osovinu zupčanika.



Slika 10 3D model zupčanika



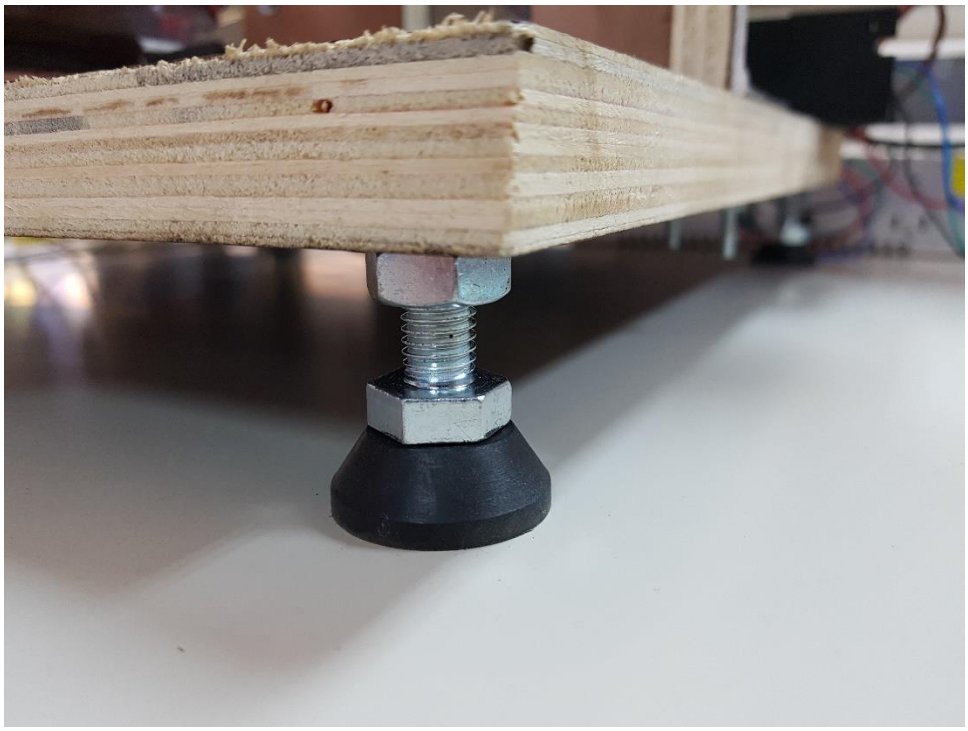
Slika 11 Čepovi za krajeve cijevi

Slika 12 Kompletan prijenosnik hrane

Slika 13 Izlazni čep

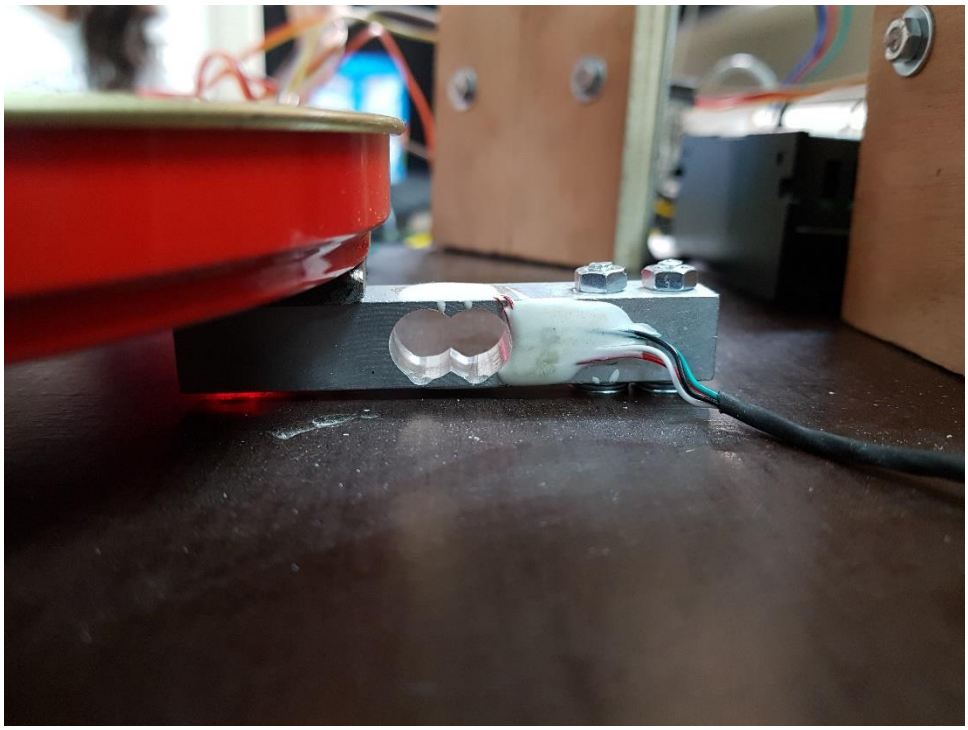
3.2.4. Postolje, nosači i senzor mase

Postolje je napravljeno od ploče iverice 35cm x 35 cm. Radi smanjenja buke i zaštite, na ploču su ugrađene gumene noge.



Slika 14 Gumene noge

Na postolje ugrađen je senzor mase i postolje za zdjelicu. Senzor mase je YZC-133 izrađen na principu tenzometarskog mosta. Četiri izlazna signala senzora spojena su na pojačalo HX711. Obrađeni podaci šalju se pinom SDA na takt prijenosa SCK. Da bi senzor mogao nesmetano raditi, od podloge je odignut podloškama. Vijcima je spojen za postolje. Ispod postolja za zdjelicu učvršćena je čelična pločica debljine 2mm kako bi se onemogućilo uvijanje postolja za zdjelicu (mekan lim).



Slika 15 Senzor mase

Slika 16 Postolje za zdjelicu

Nosači su drvene letvice učvršćene za postolje aluminijskim L profilima, vijcima i maticama. Svakom nosaču izbušene su četiri rupe za navojne šipke i jedna veća za osovinu.

Kompletan sustav može se vidjeti na slici:

Slika 17 Kompletan sustav

Većina komponenti nije kupljena, nego upotrijebljena iz zaliha. Popis (bitnijih) komponenti može se vidjeti u sljedećoj tablici :

Naziv	Opis	Dobavljač	Cijena	Količina
Nema 23	Stepper motor	STEPPERONLINE	190.76 kn	3
M335	Driver motora	Makerstore	164.45 kn	3
Arduino Uno -R3**	Mikrokontroler	Arduino	147.81 kn	1
S-360-24**	Izvor napajanja	AliExpres	105.25 kn	1
MP1584EN	Izvor napajanja	Amazon	52.62 kn	1
Postolje	*	*	*	*
Zupčanik i njegovo kućište	*	*	*	*
MCCFR0S2J0	Otpornik	Farnell	0.45 kn	3
MCF 2W 330	Otpornik	Farnell	1.06 kn	3
Spremnici za hranu	*	*	*	*
NSL 19M51.	Fotootpornik	Farnell	6.64 kn	3

YZC-133	Senzor pritiska	Diykits	57.50 kn	1
HX711	Driver senzora	Diykits	29.00 kn	1

* - Komponente su samostalno izrađene
** - Koponente se koriste i u drugim djelovima projekta
S obzirom na to da je u pitanju izrada prototipa, korišteni su dostupne komponente, bez obzira na njihovu cijenu i prekoračenje potrebnih tehničkih specifikacija.

3.2.5. Programska potpora

Programska potpora, kao što je već spomenuto, pisana je u Arduino IDE okruženju u Arduino C programskom jeziku. Potpora za hranilicu osmišljena je kao zaseban modul sa svojim .ino i .h datotekom koji daljnjem programu na uporabu daje samo dvije funkcije. Jednu za inicijalizaciju i jednu za davanje rane Ostatak funkcija potrebnih za rad izveden je u samom modulu i nije namijenjen korištenju na višim razinama programske potpore.

```
void hranilica_init(void)
{
  pinMode(motor_pulse_pin, OUTPUT);
  pinMode(motor_1_EN, OUTPUT);
  pinMode(motor_2_EN, OUTPUT);
  pinMode(motor_3_EN, OUTPUT);

  delay(10);

  digitalWrite(motor_1_EN, LOW);
  digitalWrite(motor_2_EN, LOW);
  digitalWrite(motor_3_EN, LOW);

  massOffset = readMass();

// Timer2 init
  TIMSK2 = (TIMSK2 & B11111110) | 0x01;
  TCCR2B = (TCCR2B & B11111000) | 0x05;
}
```

Funkcija init_hranilica inicijalizira potrebne pinove na mikrokontroleru, pokreće brojilo koje određuje okretaje motora i kompenzira pomak nule na masenom senzoru.

```
ISR(TIMER2_OVF_vect){
   digitalWrite(motor_pulse_pin, !digitalRead(motor_pulse_pin));
}
```

Prekidna rutina brojila je jednostavna i samo invertira stanje puls pina (puls pin spojen je na sve motore, rad motora određuje enable pin na svakom pojedinom motoru).

Funkcija hranilica prima dva parametra, grami hrane , te spremnik iz kojeg će se hrana dati. Povrati parametar je jedan bajt koji opisuje ima li greške u sustavu. Prilikom pozivanja funkcije, određeni motor se pali i radi dok se masa ne promijeni za zadanu vrijednost ili, ako prođe previse vremena, ugasi I pošalje gršku.
Funkcije motorOn i motorOff jednostavne su funkcije za omogućavanje rada motora preko enable pina.
Funkcija container_error provjerava jesu li spremnici prazni očitavajući očitanu vrijednost napona na djelilu fotootpornika i otpornika te uspoređivajući je s referentnom

```
char hranilica (int grams, int container)
{
  error = 0x00;
  int i;
  long int startingTime = millis();
  float startingMass = readMass();

  if (startingMass < 0 || startingMass> MAX_MASS)
  {
   error = error | 0x08;
  }

  motorOn(container);
  while (readMass() < startingMass + grams){
      if (millis() > startingTime + MOTOR_ERROR_TIME)
      {
        motorOff(container);
        error = error | 0x10;
        break;
      }
   }
  motorOff(container);
  //deb4(grams, container, massError, motorError);
  container_error ();
  return error;
}
```

```
void container_error (void)
{
   if (analogRead(photo_res_1_pin) >
PHOTO_RES_THRESHOLD)
     {error = error | 0x01;}
   if (analogRead(photo_res_2_pin) >
PHOTO_RES_THRESHOLD)
     {error = error | 0x02;}
   if (analogRead(photo_res_3_pin) >
PHOTO_RES_THRESHOLD)
     {error = error | 0x03;}
}
```

3.3. Detektor buke (Matteo Samsa)

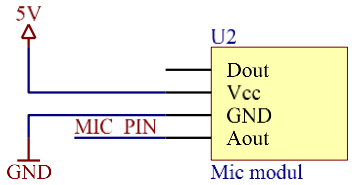
Detektor buke služi za praćenje razine buke u okolini uređaja te javljanje prekomjerne razine i trajanja putem mobilne aplikacije. Detektor je realiziran pomoću mikrofon modula i Arduino Uno razvojne pločice. Programski kod je napisan u programskom okruženju Arduino IDE. Mogućnosti koje su programirane:

- Promjenjiva razina detekcije
- Promjenjivo vrijeme trajanja buke

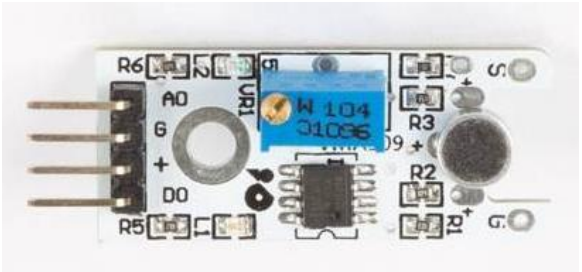
Vrijednosti se mijenjaju putem mobilne aplikacije. S obzirom na vrijednosti tih parametara korisnik će putem mobilne aplikacije primati obavijesti.

3.3.1. Sklopovlje

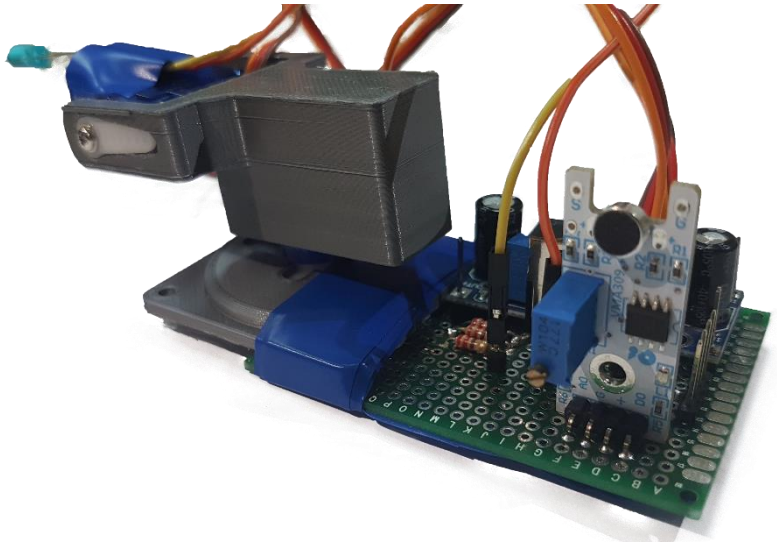
Senzor koji se koristi za mjerenje razine buke je mikrofon modul VMA309. Senzor zvuka ima četiri pina: napajanje (Vcc), masu (gnd), te analogni i digitalni izlaz iz senzora. Na digitalnom izlazu će biti logička jedinica ako je amplituda generiranog signala iz senzora zvuka veća od granične koja se namješta pomoću trimera na samom modulu. Digitalni izlaz se u ovom projektu neće koristiti. Koristit će se analogni izlaz koji daje analognu vrijednost napona kao funkciju glasnoće koju detektira mikrofon. Pomoću te razine moguće je odrediti ima li buke u okolini detektora.



Slika 18 Shematski prikaz mikrofon modula



Slika 19 Mikrofon modul



Slika 20 Modul zalemljen na pločicu uz laser igračku za mačke

3.3.2. Programska podrška

Program je pisan tako da su pri pokretanju razina detekcije i potrebno trajanje buke za slanje obavijesti postavljeni na predodređenu vrijednost. Postavljanje razine i potrebnog trajanja vrši se kroz aplikaciju na mobitelu.

```
int set_detektor(int enable_notification, int volume_threshold, int
duration_threshold){
  vol_threshold = VOLUME_DEF+volume_threshold-32;
  dur_threshold = duration_threshold*1000; //jer zapisuje u ms
  enable_detektor = enable_notification;
  return 1;
}
```

Prilikom svake iteracije koda provjerava se trenutna razina buke i uspoređuje sa zadanom ako je *enable_notification* jednak jedinici.

Ako je razina buke veća od zadane razine, od tog trenutka započinje brojanje vremena. Ako buka traje duže od zadanog poslat će se poruka putem serijske komunikacije na Raspberry Pi razvojnu pločicu te će taj podatak biti zapisan na web server kojeg će aplikacija na mobitelu pročitati. U tom trenutku na mobitelu se ispisuje obavijest. Nakon što prođe zadano vrijeme u stanju bez buke, poslat će se poruka da nema buke. Poruke se šalju samo pri promjeni stanja.

```
void detektor(unsigned long current_time){
  int volume = analogRead(MIC_PIN); //ocitanje senzora
  //nema buke, postavlja pocetak timera na trenutno vrijeme
  if ((volume < vol_threshold) && (vol_flag == 0)){
    dur_start = current_time;
    dur_stop = current_time;
  }
  //prvi puta detektira buku, zapocinje timer
  if ((volume > vol_threshold) && (vol_flag == 0)){
    dur_start = current_time;
    vol_flag = 1;
  }
  //ako još traje buka, zapisuje trenutno vrijeme kao kraj buke
  if ((volume > vol_threshold) && (vol_flag != 0)){
    dur_stop = current_time;
    vol_flag = 1;
  }
  //ako buka padne ispod zadane razine započinje brojati
  if ((volume < vol_threshold) && (vol_flag == 1)){
    downtime = current_time;
    vol_flag = 2;
  }
  //ako nije bilo buke za zadano trajanje ponovno postavlja pocetak detekcije
  if ((volume < vol_threshold) && (vol_flag == 2)){
    if((current_time - downtime) > dur_threshold){
      vol_flag = 0;
    }
  }
  //u slucaju da je bila buka i prestane, pošalje se poruka
  if ((dur_stop == dur_start) && (msg_flag != 0)){
    msg_flag = 0;
    write3(CHECK_BYTE, DETEKTOR_ID, msg_flag);
  }
  //u slucaju da je buka trajala više od zadanog trajanja, pošalje se poruka
  if (((dur_stop - dur_start) > dur_threshold) && (msg_flag != 1)){
    msg_flag = 1;
    write3(CHECK_BYTE, DETEKTOR_ID, msg_flag);
  }
}
```

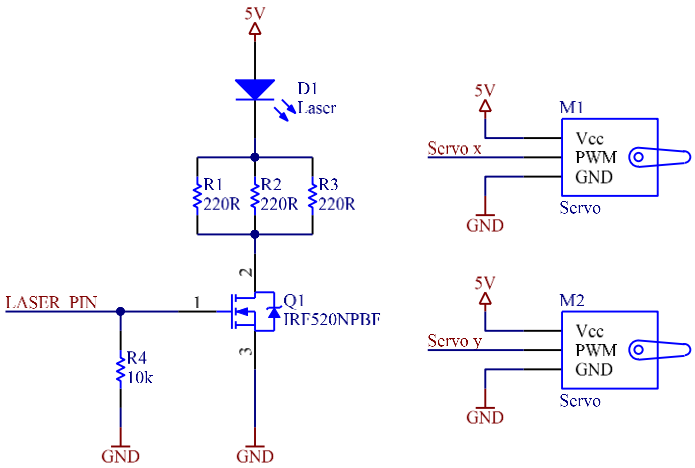
3.4. Laser igračka za mačke (Matteo Samsa)

Jedinica za zabavu mačaka je napravljena pomoću dva servo motora i laserskog pokazivača. Servo motori se koriste zbog njihove jednostavne i precizne upravljivosti. Motori su spojeni tako da pružaju upravljivo pokretanje lasera u trodimenzionalnom prostoru. Servo motori i laser su upravljani Arduino Uno razvojnom pločicom.

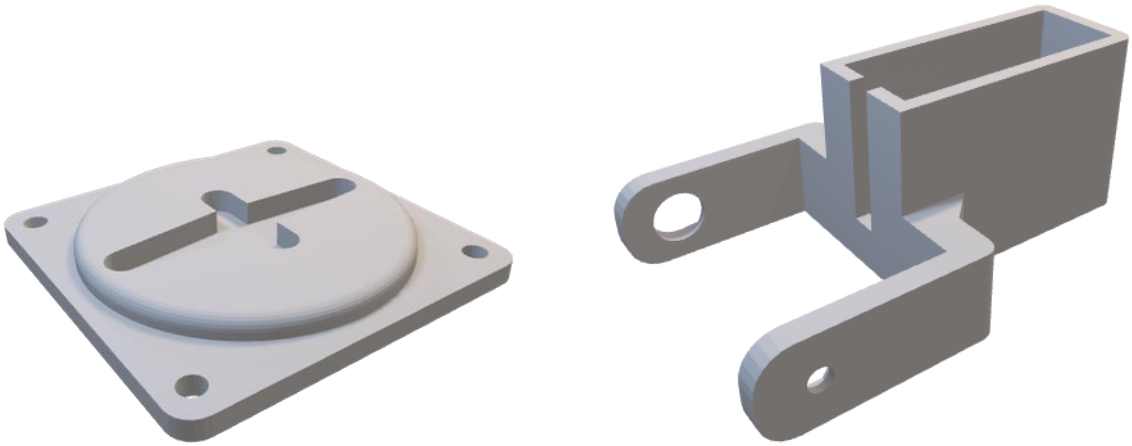
3.4.1. Sklopovlje

Korišteni su 9g servo motori (VMA600) koji se upravljaju pwm signalom. Motori su postavljeni tako da omogućuju pokretanje lasera u prostoru. Kućište koje je korišteno za takvo postavljanje motora je 3d printano iz PLA materijala.

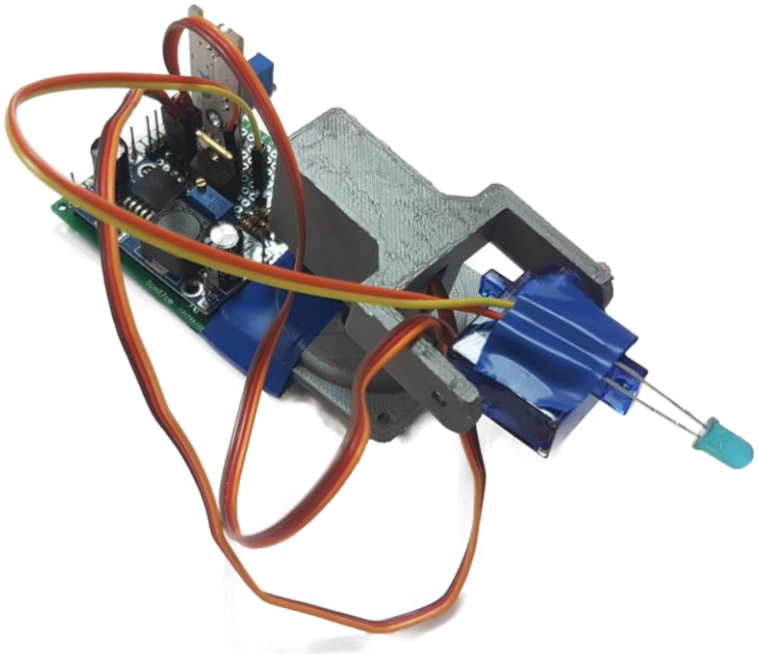
Laser dioda se upravlja putem N-kanalnog MOSFET-a IRF520 kako bi se omogućilo vanjsko napajanje sklopovlja umjesto napajanja preko Arduino pločice.



Slika 21 Shematski prikaz sklopa sustava



Slika 22 3D modeli držača dvaju servo motora



Slika 23 Sustav koji omogućava kretanje pokazivača u prostoru

3.4.2. Programska podrška

Program je pisan tako da omogućuje upravljanje položaja lasera kroz mobilnu aplikaciju. U funkciju se primaju 3 varijable, od kojih prva ukazuje je li laser uključen ili ne, a druge dvije predstavljaju x i y koordinate. S obzirom na to da korišteni servo motori imaju relativno veliki moment za korištenu primjenu, prilikom pokretanja i zaustavljanja pomiču cijelu platformu na kojoj se nalaze. Zbog toga su implementirani PID regulatori za oba motora. PID regulatori omogućuju precizan put s određenim karakteristikama s obzirom na željenu i trenutnu poziciju motora. U ovom slučaju, ako se motor nalazi daleko od željene koordinate kretat će se brzo, dok će se u blizini željene koordinate kretati sporo. Vrijednosti parametara potrebnih za PID regulatore su izračunati prema Ziegler-Nichols metodi uz preporuke za odziv bez prebačaja (engl. *overshoot*). Ziegler-Nichols metoda zahtjeva nalaženje proporcionalnog pojačanja koje dovodi sustav na granicu stabilnosti, odnosno odziv sustava su stabilne oscilacije, te period tih oscilacija. Uz pomoć tih dviju vrijednosti i jednadžbi dobivaju se parametri pojačanja proporcionalnog, integralnog i derivacijskog dijela PID regulatora. Iako motori nemaju iste parametre u sustavu jer su različito opterećeni, korišteni su isti parametri za oba motora kako bi imali jednake brzine.

Tablica 2 Kritično proporcionalno pojačanje i period oscilacija pri tom pojačanju

	Eksperimentalno određene vrijednosti
$K_{kr}K_{kr}$	0,80393
$T_{kr}T_{kr}$	0,21

Tablica 3 Parametri određeni prema Ziegler-Nichols preporuci za odziv bez prebačaja

	K_p	K_i	K_d
ZN preporuke za odziv bez prebačaja	$\frac{K_{kr}K_{kr}}{2\quad 2}$	$\frac{2}{5}\cdot\frac{K_{kr}}{T_{kr}}\frac{2}{5}\cdot\frac{K_{kr}}{T_{kr}}$	$K_{kr}\ \cdot\frac{T_{kr}}{15}K_{kr}\ \cdot\frac{T_{kr}}{15}$
Vrijednosti	0,160786	1,53129	0,01125502

Prilikom primanja poruke koja uključuje laser, postavlja se globalna varijabla *enable_laser* u 1, uključuje se laser, te postavljaju željene koordinate u prostoru. S obzirom na to da rad PID regulatora nije trenutan, dio koda za pomicanje lasera na željene koordinate se izvršava u glavnom dijelu programa do namještanja lasera. Prilikom namještanja lasera, prilikom svake iteracije se učita trenutna pozicija te postavi slijedeći položaj prema PID funkciji sve dok se laser ne postavi na željenu poziciju.

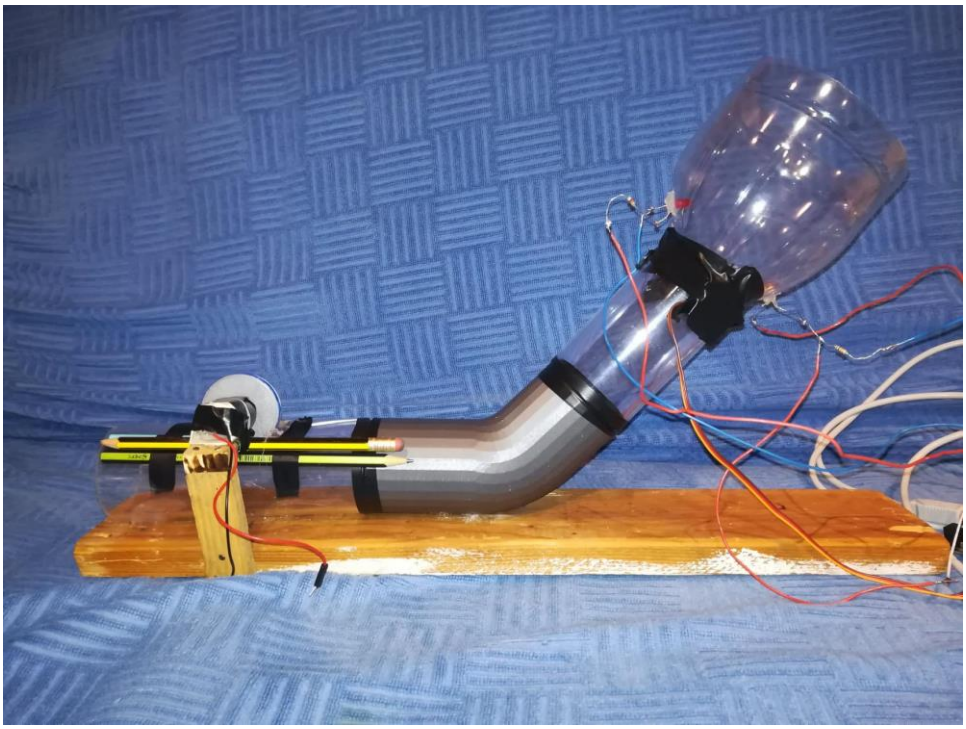
```
void laser(int E, int x, int y){
  enable_laser = E;
  if (E == 1){
    digitalWrite(LASER_PIN, HIGH);
    Setpointx=(double)(180-x);
    Setpointy=(double)y;
  }
  if (E == 0){
    digitalWrite(LASER_PIN, LOW);
    //vracanje lasera u pocetno stanje
    Setpointx=SERVO_X_DEF;
    Setpointy=SERVO_Y_DEF;
    while(Inputx!=SERVO_X_DEF or Inputy!=SERVO_Y_DEF){
      pid_x();
      pid_y();
    }
  }
}
```

```
int pid_x(void){
  if((Setpointx-err_bar)<=Inputx && (Setpointx+err_bar)>=Inputx) return 0;
  Inputx = servo_x.read();
  myPIDx.Compute();
  servo_x.write(Outputx);
  //deb3( Setpointx, Inputx,Outputx);
}
```

```
if(enable_laser==1){ //pozivanje iz glavnog programa
      pid_x();
      pid_y();
}
```

3.5. Izbacivač loptice (Ivan Matković)

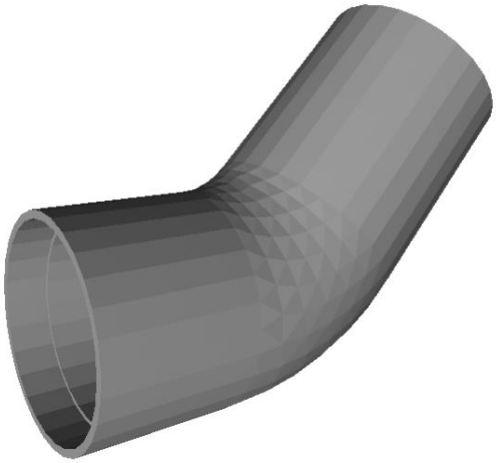
Izbacivač loptice je dio jedinice za zabavu PetCare sustava, a sastoji se od mehaničkog, sklopovskog i programskog dijela. Sustav je realiziran pomoću DC motora, drivera za DC motor, servo motora i senzora za prisustvo loptice, a upravljan je Arduino Uno razvojnom pločicom. Programski kod je napisan u programskom okruženju Arduino IDE. Paljenje i gašenje sustava moguće je putem mobilne aplikacije. Sustav je prikazan na Slika 24.



Slika 24 Sustav za izbacivanje loptice

3.5.1. Mehanički dio sustava

Mehanički dio sustava sastoji se od ljevkaste strukture na kojoj se nalazi senzor za prisustvo loptice. Na ljevkastu strukturu nastavlja se cijev s pričvršćenim servo motorom koji ima ulogu aktuatora. Poslije cijevi slijedi koljeno za preusmjeravanje loptice (napravljeno pomoću 3D printera) na koje se ponovno nastavlja cijev s pravokutnim izrezom na vrhu namijenjenim za kotač koji svojim okretanjem izbacuje lopticu. Model koljena za 3D printanje nacrtan je u AutoCAD-u i prikazan je na Slika 25.



Slika 25 3D model koljena izbacivača loptice

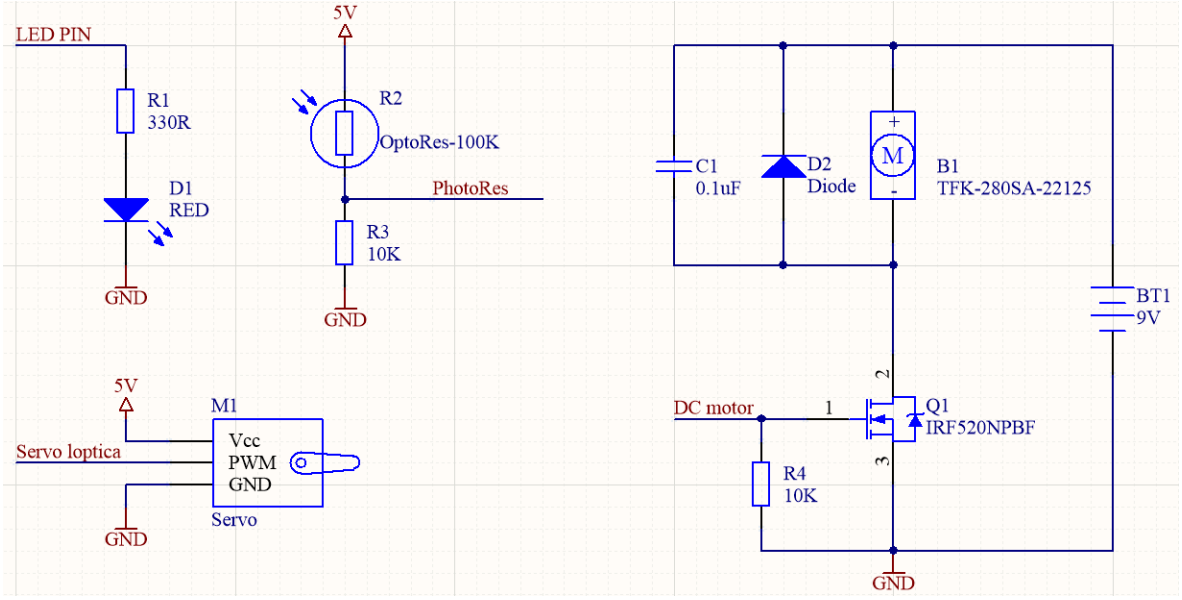
Kotač je, kao i koljeno, napravljen pomoću 3D printera kako bi odgovarao zupčaniku koji se nalazi na osovini DC motora. Model kotača za 3D printanje prikazan je na Slika 26. Držač za DC motor napravljen je pomoću drvene L strukture. Koljeno, donji dio cijevi i L struktura pričvršćeni su na drveno postolje kako bi sustav mogao stabilno stajati.

Slika 26 3D model kotača

3.5.2. Sklopovlje

Senzor za prisustvo loptice izveden je pomoću serijski spojenih fotootpornika i otpornika od 10 kΩ te LED diode kojoj je serijski spojen otpornik od 330 Ω. Napon djelila fotootpornika i otpornika doveden je na analogni pin kako bi se u programskom dijelu mogla detektirati prisutnost loptice. Kada loptica nije u sustavu, na izlazu senzora je visoka naponska razina, a kada loptica prekrije fotootpornik, na izlazu senzora je niska

naponska razina. LED dioda je spojena kako bi se smanjila promjena intenziteta svjetlosti na fotootporniku zbog različitih vanjskih uvjeta. Serija LED diode i otpornika dovedena je na digitalni pin kako bi se moglo upravljati njenim paljenjem i gašenjem za slučajeve kada sustav radi i ne radi zbog smanjenja potrošnje. Kao aktuator za propuštanje loptice koristi se servo motor 9g (VMA600) upravljan pwm signalom. Za ispaljivanje loptice koristi se DC motor koji je upravljan driverom. Driver se sastoji od n kanalnog MOSFET-a IRF520, diode, keramičkog kondenzatora od 0,1 μF i otpornika od 10 kΩ. Električna shema cijelog sklopovskog dijela prikazana je na slici. U ovom prototipu, motor je napajan iz baterije od 9 V, negativna stezaljka baterije spojena je na masu arduina, a pozitivna stezaljka na pozitivnu stezaljku motora. Motoru su paralelno spojena dioda radi zaštite i kondenzator radi smanjenja mehaničkog šuma motora. Negativna stezaljka motora spojena je na odvod tranzistora, između uvoda i upravljačke elektrode spojen je otpornik kako bi držao upravljačku elektrodu na niskoj razini kada Arduino ne šalje signal (osigurava da je uvijek poznato stanje), a na upravljačku elektrodu doveden je pwm signal za upravljanje brzinom vrtnje motora. Shema cijelog sklopovskog dijela prikazana je na Slika 27.



Slika 27 Električna shema sklopovskog dijela sustava za izbacivanje loptice

3.5.3. Program

Programski dio je pisan tako da je moguće upravljati sustavom putem mobilne aplikacije. Funkcija prima jednu varijablu koja javlja jel je sustav uključen ili isključen. Prilikom primanja poruke koja uključuje sustav, postavlja se globalna varijabla enable_loptica u 1 i zasvijetli LED dioda. Kada je sustav uključen, čita se vrijednost na senzoru. Kada se loptica detektira, pokrene se DC motor kako bi postigao zadanu brzinu

vrtnje. Dvije sekunde nakon, servo motor se zakrene i propusti lopticu i sekundu poslije se vrati u početni položaj. Sekundu nakon što se servo motor vratio u početni položaj, gasi se DC motor i sustav je spreman za ponovno izbacivanje loptice. Funckija koja izvršava opisani proces nalazi se ispod u tekstualnom okviru.

```
void loptica(int en) {
  enable_ball = en;

  if (en == 1) {
    digitalWrite(LED, HIGH);
    previousTime = millis();
  }
  else {
    digitalWrite(LED, LOW);
    analogWrite(DCMOTOR, 0);
  }
}

void timer_ball(unsigned long currentTime) {
  value = analogRead(PHOTORES);
  if (value < 300) {
    if (((currentTime - previousTime) > PERIOD) && (ball_released == 0)) {
      analogWrite(DCMOTOR, 255);
      ball_released = 1;
      previousTime = currentTime;
    }
  }
  if (((currentTime - previousTime) > PERIOD_MOTOR_ON) && (ball_released == 1))
{
      servo_ball.write(150);
      ball_released = 2;
      previousTime = currentTime;
  }
  if (((currentTime - previousTime) > PERIOD) && (ball_released == 2)) {
    servo_ball.write(105);
    ball_released = 3;
    previousTime = currentTime;
  }
  if (((currentTime - previousTime) > PERIOD) && (ball_released == 3)) {
    analogWrite(DCMOTOR, 0);
    ball_released = 0;
    previousTime = currentTime;
  }
}
```

4. Server i video nadzor

4.1. Kamera (Ivan Gredičak)

Korisnik najbolje može procijeniti stanje svojeg ljubimca i doma ako vizualno pogleda prostor. Iz tog razloga neizostavan dio sustava je video kamera kojom korisnik može uživo pogledati stanje svoga doma i kućnog ljubimca. U sljedećim potpoglavljima je opisan način realizacije video prijenosa uživo pomoću HTTP zahtjeva.

Kamera korištena u projektu PetCare je Pi Camera Module V2 (Slika 28). Modul se sastoji od Sony IMX219 8-megapikselnog senzora. Kamera je jednostavna za korištenje i ima predviđeni port (CSI) za spajanje na Raspberry Pi (u originalnom pakiranju dolazi sa 15 cm ribbon kabela). Navedeni modul je vrlo često korišten u primjeni sigurnosti doma (video nadziranja). Modul je podržan na svim modelima Raspberry Pi 1,2 i 3 i za modul je izrađen značajan broj biblioteka. U okviru projekta PetCare korištena je biblioteka Picamera Python.

Slika 28 Pi Camera Module v2 spojen na odgovarajući konektor na Raspberry Piju

4.1.1. HTTP server

U projektu PetCare modul je korišten za video prijenos uživo na zahtjev korisnika. Navedena funkcionalnost je realizirana preuzetom i prilagođenom Python skriptom. Skripta otvara HTTP server na određenom portu (u projektu PetCare 8081). Korisnik može pristupiti javnoj IP adresi routera (u routeru je potrebno omogućiti prosljeđivanje mrežnih vrata odnosno *engl. port forward*) ili lokalnoj IP adresi ako je pametni telefon spojen na istu lokalnu mrežu kao i Raspberry Pi. Pristup kameri nije ograničen i može mu pristupiti svatko tko pošalje HTTP zahtjev na odgovarajuću IP adresu i port što znači da je sigurnost ugrožena (u budućnosti dodati mogućnost prijave i ograničiti pristup).

Odabrani format video prijenosa uživo je MJPEG (osim njega razmatran je i format h264) jer je taj format najlakši za implementaciju i otklanjanje pogrešaka budući da većina današnjih web preglednika podržava MJPEG (kao i mobilna aplikacija što je primarni uvjet). Odabrani framerate za navedenu primjenu je 10 što se pokazalo dovoljnim da korisnik može utvrditi stanje svojeg ljubimca, a istovremeno podatkovni

promet nije prevelik. Rezolucija video prijenosa uživo je postavljena na 320x240 što ne daje zadovoljavajuću sliku na web pregledniku na računalu, ali daje na ekranu mobilnog telefona putem aplikacije. Cijela skripta za pokretanje HTTP servera s video prijenosom uživo na mrežnim vratima 8081 je prikazana u nastavku:

```
import io
import picamera
import logging
import socketserver
from threading import Condition
from http import server

PAGE="""\
<html>
<head>
<title>Raspberry Pi - Surveillance Camera</title>
</head>
<body>
<center><h1>Raspberry Pi - Surveillance Camera</h1></center>
<center><img src="stream.mjpg" width="320" height="240"></center>
</body>
</html>
"""

class StreamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()

    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
            # New frame, copy the existing buffer's content and notify all
            # clients it's available
            self.buffer.truncate()
            with self.condition:
                self.frame = self.buffer.getvalue()
                self.condition.notify_all()
            self.buffer.seek(0)
        return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.send_response(301)
            self.send_header('Location', '/index.html')
            self.end_headers()
        elif self.path == '/index.html':
            content = PAGE.encode('utf-8')
            self.send_response(200)
```

```
            self.send_header('Content-Type', 'text/html')
            self.send_header('Content-Length', len(content))
            self.end_headers()
            self.wfile.write(content)
        elif self.path == '/stream.mjpg':
            self.send_response(200)
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache, private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-replace;
boundary=FRAME')
            self.end_headers()
            try:
                while True:
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
                    self.wfile.write(b'--FRAME\r\n')
                    self.send_header('Content-Type', 'image/jpeg')
                    self.send_header('Content-Length', len(frame))
                    self.end_headers()
                    self.wfile.write(frame)
                    self.wfile.write(b'\r\n')
            except Exception as e:
                logging.warning(
                    'Removed streaming client %s: %s',
                    self.client_address, str(e))
        else:
            self.send_error(404)
            self.end_headers()

class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
    allow_reuse_address = True
    daemon_threads = True

with picamera.PiCamera(resolution='320x240', framerate=10) as camera:
    output = StreamingOutput()
    #Uncomment the next line to change your Pi's Camera rotation (in degrees)
    camera.rotation = 180
    camera.start_recording(output, format='mjpeg')
    try:
        address = ('', 8081)
        server = StreamingServer(address, StreamingHandler)
        server.serve_forever()
    finally:
        camera.stop_recording()
```

4.2. Komunikacija između Raspberry Pi-ja i mobilne aplikacije (Tomislav Matulić)

Sva komunikacije između mobilne aplikacije i Raspberry Pi-ja ostvarena je čitanjem iz JSON datoteke i pisanjem u JSON datoteku. JSON (engl. *JavaScript Object* Notation) je format namijenjen za razmjenu podataka. Za jednostavnije čitanje i pisanje koristi se sedam različitih JSON datoteka, a one su: *buka_down.JSON*, *buka_up.JSON*, *hrana_down.JSON*, *hranisad.JSON*, *laser_up.JSON*, *lopta_up.JSON* i *vrijeme.JSON*. Čitanje JSON datoteke i pisanje u JSON datoteku jednostavno se ostvaruje koristeći programski jezik Python koji se pokazao kao jednostavan, a istovremeno efikasan programski jezik u kojem smo implementirali sve potrebne funkcionalnosti na Raspberry Pi-ju. U nastavku slijedi kod Python modula u kojem smo napisali funkcije za čitanje i pisanje JSON datoteka.

```
import json

def citaj(ime,x):
    with open(ime, "r+") as jsonFile:
        data = json.load(jsonFile)
    return int(data[x])

def pisi(ime,x,y):
    with open(ime,"r+") as jsonFile:
        data = json.load(jsonFile)
    tmp = data[x]
    data[x]=y
    with open(ime, "w+") as jsonFile:
        json.dump(data,jsonFile)
```

Funkcija *citaj* iz JSON datoteke čije je ime *ime* vraća vrijednost (engl. *value*) ključa (engl. *key*) *x*. Funkcija *pisi* u JSON datoteku čije je ime *ime* piše vrijednost *y* za ključ *x*. U kodu ne smije zaboraviti dodati *import.json*, modul koji omogućuje rad s JSON datotekom i podacima.

Veza između mobilne aplikacije i Raspberry Pi-ja ostvarena je korištenjem FTP servera na Raspberry Pi-ju koji se pokazao povoljnijim od SSH servera. Naime, SSH server nije podržan u internetskim preglednicima i teže je bilo uklanjati pogreške. Osim navedenog, komunikacija putem FTP-a bila je jednostavnija za implementaciju u mobilnoj aplikaciji. Zbog svega navedenog odabran je FTP server. Na Raspberry Pi je instaliran *ProFTPD*, besplatni i *open-source* FTP server. Prilikom rada sustava moramo na usmjerivaču proslijediti (engl. *forward)* port 21 na kojem se odvija komunikacija putem FTP-a. Nadalje, omogućen

je *Apache* HTTP Server te je pokrenuta skripta kojom je omogućen rad kamere na portu 8081 koji je također potrebno proslijediti. Detaljnije o kameri napisano je u predviđenom poglavlju. Prethodnim je razmatranjima uspješno ostvarena komunikacija između Raspberry Pi-ja i mobilne aplikacije.

Demonstracija rada sustava dana je koristeći *buka_down.JSON* datoteku u kojem se nalazi ključ "Flag". On može poprimiti vrijednost 0 ili 1. U slučaju da je omogućena detekcije buke te se dogodi buka Arduino putem serijske komunikacije pošalje Raspberry Pi-ju poruku. Arduino promjeni ključ "Flag" u vrijednost 1 koju čita mobilna aplikacija. U trenutku kada je obavijest o buku pročitana mobilna aplikacija postavlja ključ "Flag" u vrijednost 0 što omogućava ponovnu detekciju buku. U nastavku je odsječak koda koji prima poruku s Arduina i piše u JSON datoteku.

```
if read[0]==2:
    if read[1]==1:
        try:
            myjson.pisi("buka_down.json","Flag",1)
        except:
            pass
```

4.3. Komunikacija između Raspberry Pi-ja i Arduina (Tomislav Matulić)

U svrhu implementacije komunikacije između Raspberry Pi-ja i Arduina korištena je serijska komunikacija ostvarena pomoću USB priključaka dostupnih na Raspberry Pi-ju i Arduinu. Takav je način komunikacije najjednostavniji, ne zahtjeva dodatno sklopovlje, a ujedno služi kao izvor napajanja za Arduino Uno.

Raspberry Pi šalje na seriju poruke u formatu „*<Check_byte,id,data1,data2,data3>*", odnosno Arduino očekuje takav format za primanje sa serije. *Check_byte* je kontrolni bajt koji uvijek iznosi 218 dekadski, odnosno DA heksadekadski. Nakon njega slijedi *id* koji kazuje o kojem se sustavu radi. Tablicom 1 prikazana je veza pojedinog sustava i vrijednosti *id*-a. Podaci koji se nalaze u *data1, data2* i *data3* namijenjeni su sustavu *id*, a interpretacija podataka dana je tablicom 2.

Tablica 4: Veza između *id* primatelja i sustava

id	Sustav
1	Hranilica
2	Detektor buke
3	Laser
4	Izbacivanje loptice

Tablica 5: Veza između primatelja i interpretacija poruke

id	1	2	3	4
data1	Količina (u gramima)	Uključeno ili isključeno slanje detekcije	Uključen ili isključen laser	Uključeno ili isključeno izbacivanje loptice
data2	Spremnik (1,2,3)	Razina detekcije	Pomak u horizontalnom smjeru	Ne koristi se
data3	Ne koristi se	Trajanje buke prije slanja zahtjeva za obavijest	Pomak u vertikalnom smjeru	Ne koristi se

Radi preglednosti konačne Python skripte napisali smo Python modul za primanje podataka sa serije i slanje podataka na seriju.

```
import time
TOWRITEBEGIN='<' #60
TOWRITESTOP='>'#62
TOREADSTART='[' #ASCII = 91
TOREADSTOP=']' #ASCII = 93
CHECK_BYTE= 218
def salji(ard,idd,data1,data2,data3):
        ard.write(TOWRITEBEGIN.encode())
       zaslanje=",".join([str(CHECK_BYTE),str
       (idd),str(data1),str(data2),
       str(data3)])
         ard.write(str(zaslanje).encode())
         ard.write(TOWRITESTOP.encode())
         time.sleep(0.002)
def primime(ard,x):
    input = ard.read(7)
    re = input
    time.sleep(0.002)
    return re
```

Funkcija *salji* prima pet argumenata. Na koju serijesku vezu želimo poslati podatak vezan je argument *ard*. Argumenti *idd*, *data1*, *data2*, *data3* odgovaraju redom vrijednostima *id,data1,data2,data3* koji pripadaju formatu „*<Check_byte,id,data1,data2,data3>*" .
U nastavku je isječak koda napisan za Arduino koji se tiče serijske komunikacije.

```
void primi(void){
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char rc;
    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();
        if (recvInProgress == true) {
            if (rc != END_MARKER) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= NUM_CHARS) {
                    ndx = NUM_CHARS - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }
        else if (rc == START_MARKER) {
            recvInProgress = true;
        }
    }
}
```

Gornja po ruka nalazi u varijabli *msg*.

```
void parseData(void){
    char * strtokIndx;
    strtokIndx = strtok(tempChars, ",");
    msg[0] = atoi(strtokIndx);
    for(i=1; i<5; i++){
      strtokIndx = strtok(NULL, ",");
      msg[i] = atoi(strtokIndx);
    }
}
```

Slanje podataka s Arduina, odnosno primanje podataka na Raspberry Pi, formata je *„[Check_Byte,id,data]"*. Analogno slanju s Raspberry Pi-ja, *Check_Byte* iznosi 218 dekadski. U Tablici 3 vidljiv je koji

id pripada kojem sustavu, dok je u Tablici 4 napisana veza između pojedinog *id-a* i poruke koja se nalazi u *data.*

Tablica 6: Veza između *id* primatelja i sustava

id	Sustav
11	Hranilica
2	Detektor buke

Tablica 7: Veza između primatelja i interpretacija poruke

id	11	2
data	Dojava greške (0 ili 1)	Detektirana je buka (0 ili 1)

U prethodno napisanom Python modulu nalazi se i funkcija *primime* koja prima sedam bajtova sa serijske veze *ard*. U nastavku je dio Python skripte kojim se obrađuju pristigli podaci.

```
while ard.in_waiting:
        try:
            s=myfun.primime(ard,s)
            xx=struct.unpack('<BBBBBBB',s)
            if xx[0]==91 and xx[6]==93 and xx[1]==218:# [  ] DA
                read[0]=xx[3]
                read[1]=xx[5]
                   #OBRADA PRIMLJENIH PODATAKA
        except:
            s=0
```

Za slanje podataka s Arduina koristi se kod:

```
void write3(int data1, int data2, int data3){
  Serial.write(TO_TRANSMIT); //[
  Serial.write(data1);
  Serial.write(ZAREZ);
  Serial.write(data2);
  Serial.write(ZAREZ);
  Serial.write(data3);
  Serial.write(END_TRANSMIT);//]
  delay(10);
}
```

U prethodnom kodu *data1* odgovara *Check_Byte* varijabli, *data2* odgovara *id* varijabli, a *data3* odgovara *data* varijabli „[Check_Byte,id,data]" formata. Pomoću prethodno opisanih funkcija ostvarena je komunikacija između Raspberry Pi-ja i Arduina.

Sva su prethodna razmatranja implementirana u *loop()* potprogram Arduino koda te je napisana skripta koja se pokreče automatski prilikom paljenja Rapberry Pi-ja. Unutar Python skripte i Arduinovog *loop()* potprograma inicijalizira se serijska komunikacija na 19200 bauda. Automatsko pokretanje skripte moguće je ostvariti koristeći *terminal* i nardbu *sudo crontab -e.* Dodavanjem slijedećeg koda ostvaruje se automatsko pokretanje Python skripti prilikom paljenja sustava

```
@reboot python3 kamera.py
@reboot python3
```

5. Mobilna aplikacija (Ivan Grdičak)

Mobilna aplikacija služi korisniku kako bi u odsustvu od kuće mogao upravljati sustavom i pratiti stanje sustava i kućnog ljubimca. Aplikacija je izrađena u besplatnom programskom okruženju Android Studio, kod za grafičko sučelje je pisan u XML-u, a programski kod programskim jezikom Java (Android studio omogućava i programski jezik Kotlin). Minimalna verzija Androida koju pametni telefon korisnika mora imati je Android 4.4 KitKat (API 19). Aplikacija od korisnika zahtjeva pristup Internetu, dozvolu za obavijesti i dozvolu za pristup memoriji telefona. Komunikacija s PetCare sustavom je ostvarena FTP protokolom, a datoteke se šalju i primaju u JSON formatu. Video prijenos na zahtjev u aplikaciji se ostvaruje HTTP zahtjevom. U sljedećim potpoglavljima su dijelovi aplikacije detaljnije opisani.

5.1. Korisničko sučelje

Prilikom prvog otvaranja nakon instalacije aplikacija korisnika traži dozvolu korištenja unutarnje memorije telefona na koju korisnik mora potvrdno odgovoriti kako bi aplikacija mogla ispravno raditi. Dozvola od korisnika se traži pomoću objekta *AlertDialog*. Na svakom sljedećem otvaranju aplikacije korisniku se prikazuje kratka poruka *„Dopušten pristup memoriji"* (ako korisnik nije u Postavkama isključio dozvolu). Prikaz kratke poruke je ostvaren funkcijom *Toast.makeText().* Na vrhu početnog ekrana je slika u *png* formatu prikazana pomoću *ImageView* objekta. U datoteci *AndroidManifest.xml* su definirane dozvole koje aplikacija traži od korisnika, odsječak koda u kojem su definirane dozvole je prikazan u nastavku (neke od nabrojanih dozvola nisu potrebne za ispravan rad aplikacije, ali su korišteni zbog otklanjanja pogrešaka prilikom izrade):

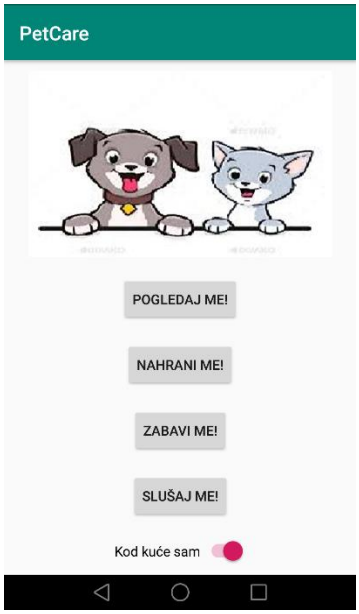
```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="android.permission.WRITE_INTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_INTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_OWNER_DATA" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Korisničko sučelje aplikacije se sastoji od ukupno 5 ekrana. S početnog ekrana korisnik ostalim ekranima pristupa pritiskom na odgovarajući gumb (u ovisnosti koju aktivnost želi). Na početnom ekranu je sklopka *„Kod kuće"* (Slika 29) kojom korisnik daje informaciju aplikaciji o tome treba li se spojiti na lokalnu IP adresu servera ili na javnu IP adresu. Korisnik mora ispravno postaviti sklopku inače se aplikacija može ponašati nepredvidljivo i „smrznuti se". U nastavku je primjer XML koda kojim se definiraju sklopke, gumbi i ostali objekti na ekranu (u nastavku rada svi objekti su definirani na taj način). Tim objektima se definira ID po kojem je jedinstven u aplikaciji, veličina i položaj na ekranu i ostali parametri koji nisu obavezni već opisuju ponašanje (npr. poziv funkcije na pritisak gumba ili promjenu stanja sklopke). U projektu PetCare najvažniji parametar je *onClick* kojim se definira funkcija koja će se pozvati pritiskom gumba.

```
<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="slika_buka"
    android:text="@string/Buka"
    app:layout_constraintBottom_toTopOf="@+id/switch2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button3" />

<Switch
    android:id="@+id/switch2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="ip_chg"
    android:text="Kod kuće sam"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button4" />
```

Početni ekran se kao što je rečeno sastoji od 4 gumba na čiji se pritisak mijenja ekran koji korisnik odabire. Izgled početnog zaslona i veza s ostalim ekranima je prikazana na Slika 29



Slika 29 Snimka zaslona početnog ekrana aplikacije PetCare.

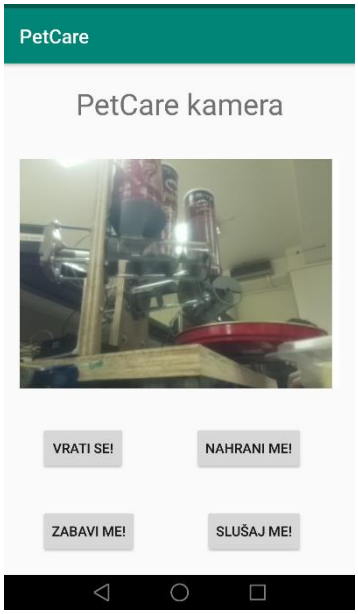
PetCare kamera ekran (otvara se pritiskom na gumb *„Pogledaj me!"* na početnom ekranu) omogućava korisniku video prijenos uživo pomoću HTTP zahtjeva na IP adresu i odgovarajuća mrežna vrata(*engl. port*) servera. HTTP zahtjev se ostvaruje objektom *WebView.* Adresa kojoj se pristupa se definira programski sljedećim kodom:

```
WebView webView=(WebView) findViewById(R.id.webID);

webView.getSettings().setLoadWithOverviewMode(true);
webView.getSettings().setUseWideViewPort(true);

webView.loadUrl("http://"+ipaddress+":"+port+"/stream.mjpg");
```

Kao i s početnog ekrana pritiskom gumba ispod prikaza video prijenosa (Slika 30) uživo se pristupa ostalim ekranima.

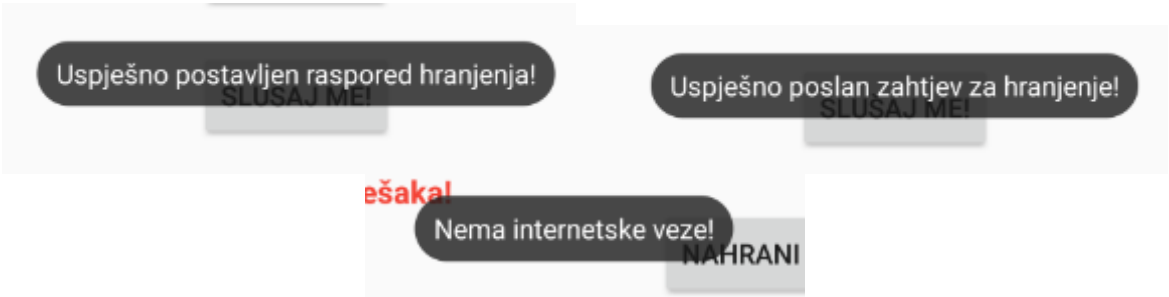


Slika 30 Snimka zaslona ekrana PetCare kamera.

Ekran Hranilica (otvara se otvara se pritiskom na gumb *„Nahrani me!"*) omogućava korisniku postavljanje do 5 termina hranjenja ljubimca, uz mogućnost definiranja količine hrane i definiranja spremnika te sklopkama kojima se pojedini termin može uključiti/isključiti. Sati i minute (minute se biraju u koracima od 5 minuta) se biraju pomoću padajućih izbornika koji se ostvaruju objektom *Spinner.* Osim definiranja rasporeda hranjenja korisnik može trenutno dozirati hranu ljubimcu. Masa (u gramima) i broj spremnika se definiraju ručnim unosom broja u objekt *EditText* (inicijalno stanje teksta u objektima je 100 grama i spremnik 1). U desnom donjem kutu se nalazi popis grešaka (ako postoje) koje mogu biti greška motora, vage i prazni spremnik () ostvaren objektom *TextView*. Ažuriranje rasporeda hranjenja i zahtjev za trenutno hranjenje korisnik ostvaruje pritiskom na gumbe *„Postavi vrijeme hranjenja"* i *„Nahrani sada!".* Kada broj spremnika nije ispravan prikazuje se poruka *„Neispravni podaci!"* Pri svakom novom otvaranju ekrana ažurira se popis grešaka i polja za unos su postavljena na vrijednost na koju ih je korisnik prethodno postavio (ovo vrijedi i za ekran Detektor buke koji je opisan u nastavku). Prikaz prethodnog (ili inicijalnog) stanja se izvršava pomoću *editText.setText(), spinner.setSelection() i switch.setChecked().*
Ako uređaj nema pristup Internetu tada se prikazuje kratka poruka „Nema internetske veze!" što znači da operacija nije izvršena. Kada spajanje na server nije uspješno prikazuje se poruka *„Nedostupan server"*. Navedene poruke se javljaju na svim ekranima kada korisnik pokuša pristupiti sustavu a nema Internetske veze ili je server nedostupan. Ekran Hranilica je prikazan na Slika 31, gdje su ujedno izlistane sve moguće greške koje se mogu dojaviti korisniku.

Slika 31 Snimka zaslona ekrana Hranilica.

Na Slika 32 su izlistani primjeri kratkih poruka koje se javljaju u slučaju da nema internetske veze ili da je zahtjev uspješno izvršen (slične poruke se javljaju i za zahtjeve korisnika vezane uz detektor buke, laser i izbacivač loptice).

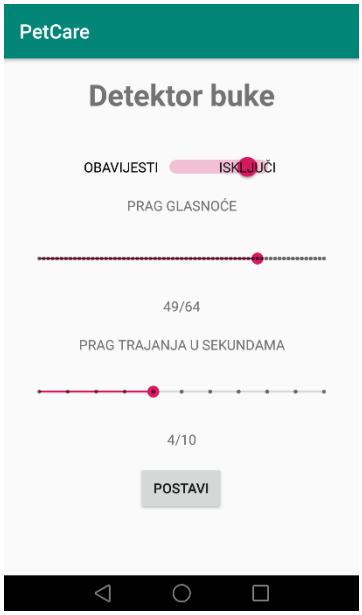


Slika 32 Primjeri kratkih poruka za informiranje korisnika.

Ekran PetCare kamera i laser (otvara se otvara se pritiskom na gumb „*Zabavi me!*“) omogućava korisniku jednako kao na jednom od prethodno opisanih ekrana video prijenos uživo. Korisnik pomoću sklopke u donjem desnom kutu uključuje/isključuje laser i to može vidjeti na video prijenosu. Poziciju lasera korisnik mijenja pritiskom na jedan od 4 gumba koji označavaju smjer promjene. Pritiskom na gumb za neki od smjerova u slučaju isključenog lasera prikazuje se kratka poruka „*Laser isključen!*“. Ekran uz kratku poruku uspješnog uključivanja lasera je prikazan na Slika 33.

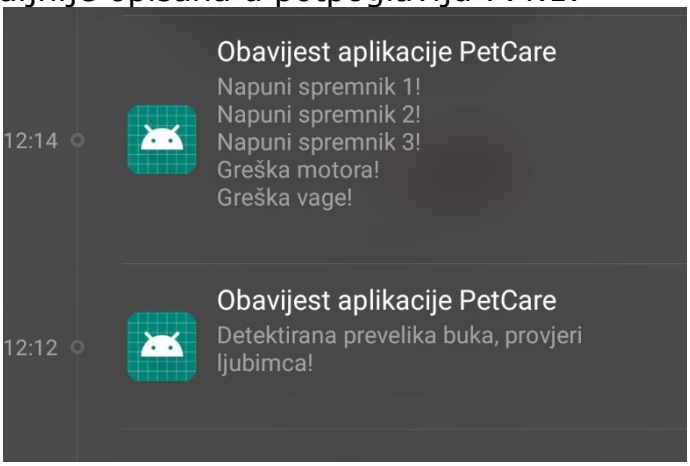
Slika 33 Snimka zaslona ekrana PetCare kamere i lasera(uz kratku poruku)

Ekran Detektor buke (otvara se otvara se pritiskom na gumb „*Slušaj me!*“) omogućava korisniku podešavanje postavki za detektor buke. Korisnik pomoću sklopke „*Obavijesti*“ bira želi li primati obavijesti od aplikacije u slučaju buke, a pomoću dva „klizača“ koji su ostvareni objektom *SeekBar* korisnik definira vrijeme trajanja buke i intenzitet buke nakon kojeg želi da mu stigne obavijest. Ispod svakog klizača je pomoću objekta TextView prikazana trenutna postavljenja vrijednost. Korisnik potvrđuje postvke pritiskom na gumb „*Postavi*“. Ekran je prikazan na Slika 34.



Slika 34 Snimka zaslona ekrana Detektor buke.

Korisniku stiže obavijest na Android pametni telefon kada dođe do promjene stanja sustava, odnosno kada je jedan od spremnika prazan, kada motor ili vaga ne rade ispravno ili je detektirana prevelika buka. Obavijest se prikazuje u okviru za obavijesti na vrhu ekrana pametnog telefona. Prikaz obavijesti prikazan je na Slika 35, a za ilustraciju su na tom primjeru prikazane sve moguće obavijesti. Realizacija prikaza obavijesti je detaljnije opisana u potpoglavlju 7.4.1.



Slika 35 Snimka zaslona s prikazanim obavijestima korisniku.

5.2. FTP klijent

Komunikacija sa serverom se ostvaruje FTP protokolom. FTP protokol omogućava veliku brzinu prijenosa datoteka i autori su ga odabrali jer je jednostavan za implementaciju u aplikaciji (klijent) i na serveru. U tu svrhu je potrebno implementirati FTP klijenta u PetCare aplikaciji. Server je realiziran na Raspberry Pi-u. Kako bi se klijent mogao povezati s FTP serverom potrebno je posjedovati korisničko ime i lozinku te ih definirati u aplikaciji. Za ostvarivanje FTP protokola preuzeta je klasa *MyFTP.class* (navedena klasa koristi biblioteku Apache Commons Net 3.6 API). Iz klase je korišteno nekoliko funkcija čije su deklaracije navedene u nastavku:

```
public boolean ftpConnect(String host, String username, String password,int port)
public boolean ftpDisconnect()
public boolean ftpDownload(String srcFilePath, String desFilePath)
public boolean ftpUpload(String srcFilePath, String desFileName, String
desDirectory, Context context)
```

Navedene funkcije su potrebne kako bi se FTP klijent povezao sa serverom i na kraju odspojio te kako bi mogao pohraniti datoteke i skinuti datoteke sa servera. Sve datoteke koje se šalju i primaju su u JSON formatu i spremaju se u memoriju telefona.

Zbog preuzimanja datoteka sa servera i pohranjivanja na memoriju telefona je potrebno dopuštenje korisnika za pristup memoriji. Za uspješnu komunikaciju sa serverom potrebna je Internetska veza, a kako bi se korisnika podsjetilo da uspostavi vezu u slučaju da je nema se koristi sljedeća funkcija:

```
public  boolean isInternetConnection()
{
   boolean connected = false;
   ConnectivityManager connectivityManager =
(ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);

if(connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).getStat
e() == NetworkInfo.State.CONNECTED ||

connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI).getState()
== NetworkInfo.State.CONNECTED) {
      connected = true;
   }
   else
      connected = false;
   return connected;
}
```

U slučaju da navedena funkcija kao izlaznu vrijednost daje '*false*', korisniku se prikaže kratka poruka „Nema internetske veze" koja ga podsjeća da nije moguće upravljati sustavom ni primati obavijesti dok se veza ne uspostavi.

5.3. Upravljanje sustavom

Upravljanje sustavom se odvija na zahtjev korisnika koji putem sučelja podešava vrijednosti koje je potrebno poslati serveru. Kao što je ranije navedeno u poglavlju 7.1 pritiskom na neki gumb korisnik potvrđuje postavke i one se šalju na server (uz internetsku vezu i povezivanje na server). Prije slanja podataka na server potrebno je pročitati korisnikove unesene vrijednosti i pohraniti ih u JSON datoteku. Kao što je ranije navedeno korisnik unosi vrijednosti u objekte EditText, Button, Switch, Spinner i SeekBar. Svaki od objekta je potrebno u programskom kodu definirati pomoću njihovog ID-a u slučaju da se želi pročitati ili promijeniti stanje, na primjer:

EditText = (EditText) findViewById(R.id.editTextId).

Primjeri poziva funkcija za čitanje vrijednosti objekata su navedene u nastavku:

```
editText.getText()                                    //String

switch1.isChecked()                                   //boolean

spinner.getSelectedItem()                             //String

seekBar.getProgress() // integer
```

Zbog jednostavnosti prilikom stvaranja i pisanja u JSON datoteku za slanje na server pročitani podaci su upisivani direktno u *String* varijablu, gdje su odmah definirani i nazivi pojedinih podataka u JSON-u. Nakon toga se definirana String varijabla upisuje u JSON datoteku (koja se nalazi u memoriji telefona), koja se zatim opisanim postupkom šalje na FTP server. Primjer prethodnog postupka je prikazan u kodu u nastavku:

```
String podaci = "{\"Hrani\":1, \"Grami\":" + editText1.getText() +
",\"Spremnik\":" + editText2.getText() + "}";
File file = new File(Environment.getExternalStorageDirectory() +
"/hranisad.json");

try (PrintStream out = new PrintStream(new
FileOutputStream(file.getPath().toString()))) {
    out.print(podaci);
} catch (FileNotFoundException e) {
    e.printStackTrace();
}

status = ftp1.ftpUpload(sada.getPath(), "hranisad.json", "/home/pi/", cntx); //
Context aplikacije, boolean
```

Rezultat prethodnog odsječka je JSON datoteka sljedećeg sadržaja:

```
{
'Hrana': 1,
'Grami': 100,
'Spremnik' 1
}
```

Razmjena podataka između Raspberry Pi-a i aplikacije je u potpunosti putem JSON datoteka. JSON (JavaScript Object Notation) je format datoteke (tekstualne) koja služi za prijenos i pohranu podataka. Podaci koji se prenose JSON datotekom mogu biti tipa Integer, String i Boolean te polja i strukture navedenih tipova podataka. Jedan podatak/objekt u JSON formatu se sastoji od ključa i od vrijednosti, koji su odvojeni dvotočkom dok su od ostalih objekata odvojeni zarezom. JSON format je čitljiviji od XML-a te je odabran za razmjenu podataka jer je jednostavnije implementirati čitanje i pisanje u takve datoteke kako u

programskom jeziku Java tako i u Python-u te je preglednije za analizu i otklanjanje pogrešaka. Primjer JSON datoteke korištene u okviru projekta PetCare je prikazana u nastavku (pod ključem „Hrana" se nalazi polje struktura podataka koje predstavljaju 5 termina hranjenja ljubimca, a svaki je termin opisan unutar strukture gdje su različite informacije pod različitim ključevima, navedeni JSON služi za prijenos informacija koje je korisnik unio na ekranu na Slika 31):

```
{
"Hrana":[
{
"Aktivno": 1,
"Sati":8,
"Minute":5,
"Grami":100,
"Spremnik":1
},
{
"Aktivno":0,
"Sati":10,
"Minute":20,
"Grami":100,
"Spremnik":1
},
{
"Aktivno":1,
"Sati":12,
"Minute":10,
"Grami":100,
"Spremnik":1
},
{
"Aktivno":1,
"Sati":13,
"Minute":45,
"Grami":100,
"Spremnik":1
},
{
"Aktivno":1,
"Sati":16,
"Minute":15,
"Grami":850,
"Spremnik":1
}
]
}
```

Osim prethodna dva primjera JSON-a u projektu, korišteno je još nekoliko JSON datoteka koje služe za prijenos postavki detektora buke,

upravljanje laserom i izbacivačem loptice. Podaci koje korisnik šalje serveru su opisani i prikazani na ekranima u potpoglavlju 7.1.

5.4. Primanje informacija i prikaz obavijesti

U prethodnom potpoglavlju je opisano slanje podataka (konfiguracija i upravljanje sustavom) s mobilne aplikacije pomoću korisničkog sučelja na server koji podatke šalje dalje sustavu. U nastavku je opisan način na koji mobilna aplikacija prima obavijesti i podatke od sustava i način na koji se upravlja tim podacima.

Kako bi mobilna aplikacija preuzela datoteke s FTP servera potrebno je da bude uspješno spojena na server kao FTP klijent. Nakon spajanja datoteku je potrebno preuzeti u memoriju telefona. Primjer navedenog procesa je prikazan u sljedećem odsječku koda:

```
File file = new File(Environment.getExternalStorageDirectory() +
"/hrana_down.json");
if (!file.exists()) {
    try {
        file.createNewFile();
    } catch (IOException e) {
        e.printStackTrace();
    }
status = ftp.ftpDownload("hrana_down.json", file.getPath());
```

Tim postupkom je iz trenutnog radnog direktorija (u projektu PetCare to je uvijek /home/pi) preuzeta datoteka *hrana_down.json* i pohranjena je u memoriju telefona pod istim imenom. Osim te datoteke za primanje podataka od sustava služi i datoteka *buka_down.json.* Nakon preuzimanja datoteke potrebno je pročitati podatke. Za čitanje podataka iz JSON datoteke potrebno je stvoriti *JSONObject*. Za rad s JSON podacima u programskom jeziku Java je korištena klasa funkcija *org.json.JSONObject*. Stvaranje JSON objekta zahtijeva kao ulazni parametar *String*. Čitanje sadržaja datoteke i pisanje u *String* je realizirano sljedećom funkcijom čiji je ulazni parametar podatak tipa *File*:

```
    // Funkcija koja služi za čitanje iz datoteke koja sadržaj vraća u Stringu
public static String getFileContents(final File file) throws IOException {
    final InputStream inputStream = new FileInputStream(file);
    final BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
    final StringBuilder stringBuilder = new StringBuilder();
    boolean done = false;
    while (!done) {
        final String line = reader.readLine();
        done = (line == null);
        if (line != null) {
            stringBuilder.append(line);
    reader.close();
    inputStream.close();
    return stringBuilder.toString();
}
```

Nakon što je pozivom prethodne funkcije u podatak tipa *String* pohranjen sadržaj JSON datoteke, taj podatak služi kao ulazni parametar za stvaranje objekta JSONObject. U nastavku su primjeri stvaranja objekta, čitanja i pisanja podataka u objekt:

```
content = getFileContents(file);
jsonObject = new JSONObject(content);
jsonObject.getInt("2"); // dohvaćanje Integer podatka pod ključem 2
jsonObject.put("Flag", 0); // upis podatka 0 pod ključ Flag
```

Podaci o stanju sustava su u JSON datotekama zapisani kao 0 ili 1. Dio podataka o stanju sustava korisnik može vidjeti na ekranu Hranilica gdje su prikazane greške u slučaju praznih spremnika, greške vage ili greške motora i podaci na tom mjestu se ažuriraju svaki put kad se ekran otvori. U slučaju da nema grešaka korisniku će pisati da nema grešaka ili u slučaju da ne može dohvatiti datoteku pokazat će se obavijest da server nije dostupan ili nema internetske veze. Za grešku hranilice u JSON datoteci je pod pripadajućem ključem upisan podatak 0, a ako nema greške podatak 1. JSON datoteka za prijenos podataka o buci ima dva ključa od kojih jedan daje informaciju postoji li buka, a drugi predstavlja zastavicu koju aplikacija spušta kada je zaprimila podatak da (ne)postoji buka kako ne bi korisniku iznova dolazila obavijest za istu buku. Stanje se svaki puta nakon obrade upisuje u tekstualnu datoteku *obavijesti.txt* koja se nalazi u unutarnjoj memoriji telefona i služi kako bi prilikom idućeg dohvaćanja sa servera stanje bilo uspoređeno s prethodnim i prikazala se obavijest samo u slučaju promjene stanja.

5.4.1. Obavijesti u pozadini

U dosadašnjem dijelu poglavlja je opisan način dohvaćanja datoteka o stanju sustava s FTP servera. U ovom dijelu će biti opisan način realizacije prikaza obavijesti u okviru za obavijesti pametnog telefona. Kako bi korištenje resursa pametnog telefona bilo racionalno za vrijeme dok aplikacija radi u pozadini ili ne radi uopće potrebno je odrediti vremenske korake u kojima će aplikacija preuzeti datoteke sa servera i provjeriti stanje te obavijestiti korisnika. U PetCare projektu je odabran vremenski korak od 75 sekundi. Vremenski određeno preuzimanje datoteka sa servera i prikaz obavijesti su realizirani pomoću klase funkcija *AlarmManager*. Prilikom svakog otvaranja aplikacije inicijalizira se vremenski brojač koji će nakon prolaska zadanog vremenskog intervala pokrenuti funkciju *onRecieve()* iz klase *AlarmReciever*. Funkcija za inicijalizaciju je prikazana u sljedećem odsječku koda:

```
public void startAlarm() {
    manager = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
    Intent intent= new Intent(this, AlarmReceiver.class);
    pendingIntent=PendingIntent.getBroadcast(this, 0, intent, 0);

    manager.setExact(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()+
1000*15, pendingIntent);

    ComponentName receiver = new ComponentName(this, MainActivity.class);
    PackageManager pm = this.getPackageManager();
    pm.setComponentEnabledSetting(receiver,
            PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
            PackageManager.DONT_KILL_APP);
}
```

Navedenom inicijalizacijom aplikacija svakih 75 sekundi pročita sadržaj datoteka stanja na serveru te zatim podatke obrađuje i u slučaju da je potrebno prikazuje korisniku obavijest (tekst obavijesti,vibracija i zvučni signal). Obavijesti hranilice i obavijesti buke dolaze kao dvije različite obavijesti, a hoće li se prikazati ovisi o prethodnom stanju sustava (*obavijesti.txt*) i trenutnom stanju. Tekst obavijesti se gradi kao podatak tipa *String* tako da se svaka nova obavijest koju je potrebno prikazati dodaje na kraj postojećeg teksta obavijesti. Naslov obavijesti je „Obavijest aplikacije PetCare“, a sve obavijesti su prikazane u prikazu velikog teksta obavijesti (najčešće potrebno potezom prsta proširiti obavijest). Definiranje i prikaz obavijesti realiziran je sljedećim odsječkom koda:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(arg0);
Notification notification = builder.setContentTitle("Obavijest aplikacije PetCare")
        .setContentText("Nove obavijesti").setAutoCancel(true)
        .setSmallIcon(R.drawable.lepo)
        .setStyle(new NotificationCompat.BigTextStyle()
                .bigText(content1))
        .setSound(Settings.System.DEFAULT_NOTIFICATION_URI)
        .setContentIntent(intent1).build();

NotificationManager notificationManager = (NotificationManager)
arg0.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(1, notification);
```

Završetkom prethodnog postupka potrebno je ponovo postaviti vrijeme sljedećeg čitanja sa servera u *AlarmManager*-u.

6. Zaključak

PetCare je sustav koji omogućava hranjenje, nadzor i zabavu kućnog ljubimca putem mobilne aplikacije. Osim navedenih funkcija sustav dojavljuje aplikaciji (korisniku) putem servera informacije o stanju sustava kao što su nedostatak hrane u spremnicima, prevelika buka ili grešku sustava. Kako bi sustav uspješno izvršavao sve svoje funkcionalnosti potrebno je da server i mobilni telefon imaju pristup Internetu ili da server i telefon budu spojeni na istu lokalnu mrežu. Hranjenje po unaprijed postavljenom rasporedu omogućuje da ljubimac dobiva hranu čak i bez pristupa Internetu, samo je potrebno da korisnik taj raspored postavi dok ima uspostavljenu Internetsku vezu ili dok je na istoj lokalnoj mreži kao server. Kako bi sustav ispunio svoje funkcionalnosti korisniku koji nije kod kuće potrebno je većinu vremena imati pristup Internetu kako bi mogao primiti obavijesti o stanju sustava (buka i nedostatak hrane) i time na vrijeme spriječiti problem. Sustav je posebno pogodan u slučaju nepredviđenog odsustva od kuće, gdje korisnik osim po rasporedu može i trenutno nahraniti ljubimca i po potrebi ga zabaviti. Ako sustav koristi osoba koja je teže pokretna i nalazi se u istom prostoru kao i sustav tada nije potreban video nadzor te je vjerojatno najvažniji dio koji se odnosi na zabavu ljubimca (video nadzor je koristan ako je osoba u drugoj prostoriji npr. u krevetu) i prehranu (spremnike je potrebno rjeđe puniti nego što bi bio slučaj s davanjem hrane ručno).

Potrebno je osigurati pristup video nadzoru i ograničiti ga samo na korisnike koji posjeduju korisničko ime i lozinku ili na neki drugi način zabraniti pristup ostalim osobama. U slučaju da je u prostoru mračno (bilo da je noć ili spuštene rolete i sl.) korisnik neće moći jasno vidjeti situaciju pa je u sustav potrebno uključiti osvjetljenje ili omogućiti upravljanje osvjetljenjem u kući. Općenito ako tijekom noći korisnik nije kod kuće potrebno je i ljubimcu omogućiti neki izvor svjetlosti. Sustav ne daje rješenje za davanje vode ljubimcu, u budućnosti bi sustav trebao biti proširen mogućnošću davanja vode ljubimcu koja bi bila odgovarajuće temperature (pogotovo za vrijeme ljetnih mjeseci). Osim video nadzorom korisnik se ne može uvjeriti da je ljubimac cijelo vrijeme u stanu (kamerom nisu pokriveni svi dijelovi stana), potrebno je ostvariti lokacijsko praćenje ljubimca kako se ne bi udaljio od doma, odnosno kako bi korisnik na vrijeme reagirao. U cjelokupnom projektu „pametne kuće" potrebno je razmišljati i o sigurnosti samog ljubimca (spajanjem s projektom sigurnog doma), za sada je korisniku omogućen video nadzor i detekcija buke, dok ostale nezgode koje predstavljaju opasnost nisu uračunate.

7. Literatura

[1] Getting started with PiCamera , 2017., URL: https://projects.raspberrypi.org/en/projects/getting-started-with-picamera ing-started-with-picamera, (2019-05-16)

[2] Stack Overflow, URL: https://stackoverflow.com/, (2019-05-16)

[3] Developers guide, službeni vodič za razvoj aplikacije u programskom paketu Android studio, URL: https://developer.android.com/guide , (2019-05-19)

[4] Android developers, službene reference za razvoj aplikacije u programskom paketu Android studio, URL: https://developer.android.com/reference, (2019-05-19)

[5] Android tutorials, 2019. URL: https://www.tutorialspoint.com/android/ (2019-06-01)

[6] Programming knowledge, URL: http://programmingknowledgeblog.blogspot.com/ (2019-06-02)

[7] Python tutorial, URL: https://www.w3schools.com/python/default.asp, (2019-06-01)

8. Pojmovnik

Pojam	Kratko objašnjenje	Više informacija potražite na
White paper	Kratak dokument koji daje uvid u neko područje, tehniku, politiku, proizvod, metodu, standard i sl.	en.wikipedia.org/wiki/White_paper
JSON	JavaScript Object Notation, format za jednostavnu razmjenu podataka	www.json.org/
FTP	File Transfer Protocol, aplikacijski protokol za razmjenu datoteka između klijenta i servera	en.wikipedia.org/wiki/File_Transfer_Protocol
HTTP	Hypertext Transfer Protocol, najčešći aplikacijski protokol na world wide webu	en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
CSI MJPEG	Camera Serial Interface Motion JPEG, video format koji komprimira svaki frame kao jednu JPEG sliku	en.wikipedia.org/wiki/Camera_Serial_Interface en.wikipedia.org/wiki/Motion_JPEG
PWM	Pulsno-širinska modulacija	en.wikipedia.org/wiki/Pulse-width_modulation
PID	Proporcionalni, integrabilni, derivabilni regulator	en.wikipedia.org/wiki/PID_controller