OSuRV Better Documentation

VEZBE 5

TOP-DOWN

Pristup od-gore-na-dole je metoda rešavanja problema i dizajna sistema u kojoj se složen sistem razbija na manje, lakše upravljive komponente.

Pristup počinje sa pregledom sistema na visokom nivou, a zatim postepeno razbija na sve manje delove, dok se pojedinačne komponente i njihove interakcije razumeju.

PWM

Pulse Width Modulation (PWM) je digitalni signal koji se koristi za kontrolu količine snage koja se daje opterećenju.

Signal je kvadratna talasna linija, gde je ciklus rada talasa ***DUTY*** (procentualno vreme kada je signal visok u odnosu na vreme kada je nisko) korišćen za kontrolu snage koja se daje opterećenju.

PWM se često koristi u aplikacijama kao što su kontrola brzine motora, smanjivanje svetlosti i kontrola snage.

---

open()

Funkcija ***open()*** se koristi da otvori fajl ili uređaj.

Ona prima dva argumenta:

- ime fajla ili uređaja
- režim u kom treba da se fajl ili uređaj otvori

Funkcija vraća deskriptor fajla, što je mali broj koji se koristi da se referiše na fajl ili uređaj u kasnijim operacijama.

---

read()

Funkcija ***read()*** se koristi da se pročitaju podaci iz otvorenog fajla ili uređaja.

Ona prima tri argumenta:

- deskriptor otvorenog fajla ili uređaja
- pokazivač na bufer gde će se podaci skladištiti
- broj bajtova koje treba pročitati.

Funkcija vraća broj pročitanih bajtova, ili -1 u slučaju greške.

---

write()

Funkcija ***write()*** se koristi da se podaci upisuju u otvoreni fajl ili uređaj.

Ona prima tri argumenta:

- deskriptor otvorenog fajla ili uređaja
- pokazivač na bufer koji sadrži podatke koje treba upisati

- broj bajtova koje treba upisati

Funkcija vraća broj upisanih bajtova, ili -1 u slučaju greške.

---

close()

Funkcija ***close()*** se koristi da zatvori otvoreni fajl ili uređaj.

Ona prima jedan argument:

- deskriptor otvorenog fajla ili uređaja

Funkcija vraća 0 ako je uspešna ili -1 ako dođe do greške.

---

small_4dof_arm_ws

"small_4dof_arm_ws" je ime radnog prostora u sistemu za upravljanje robotima (***ROS***) koji se koristi za razvoj i upravljanje robotskom rukom sa 4 stepena slobode (***DOF***).

ROS radni prostor je direktorijum na fajl sistemu gde su sve potrebne datoteke i paketi za ROS projekat smešteni.

Ime ***"small_4dof_arm_ws"*** ukazuje da je radni prostor za malu robotsku ruku sa 4 stepena slobode, što znači da ruka ima 4 zgloba koji se mogu kontrolisati nezavisno.

---

dmesg -w

Ova komanda se koristi da nam prikaže bafer poruka jezgra (kernela).

Nastavak -w nam omogućava da se ova komada izvršava u nadzornom režimu, što znači da će prikazivati nove poruke dok su dodavane u bafer.

Ovo je korisno da bi nadgledali poruke jezgra (kernela) u realnom vremenu.

---

pos_cmd & pos_fb

U ROS-u, vektori ***pos_cmd*** i ***pos_fb*** su promenljive koje se koriste da predstavljaju željenu poziciju (***pos_cmd***) i trenutnu poziciju odziva (***pos_fb***) čvorova robota.

---

catkin_make

Komandna linija ***catkin_make*** se koristi za izgradnju, testiranje i instaliranje paketa u radnom prostoru ***catkin***.

To je pogodan alat koji automatizuje proces pozivanja sistema za izgradnju ***CMake*** i drugih potrebnih alata za izgradnju.

Komanda se obično koristi u razvoju ROS (Robot Operating System).

---

source devel/setup.bash

Komanda ***"source devel/setup.bash"*** se koristi u ROS-u (Robot Operating System) da bi se ažurirale promenljive okruženja za trenutnu sesiju shell-a.

Ona dodaje potrebne putanje u promenljive okruženja za korisnika da koristi pakete koji su izgrađeni u catkin radnom prostoru.

Ova komanda se obično pokreće u korenu catkin radnog prostora i omogućava korisniku da koristi izvršne datoteke i biblioteke koje su izgrađene od strane catkin-a.

---

flow.sh

Zadati set naredbi se koriste za kontrolu i praćenje ruke u ROS-based sistemu.

1. ***"roslaunch s4a_main main.launch all_motors_sim:=false"***
   Pokreće glavni ROS čvor za kontrolu ruke.
   Postavlja ***"all_motors_sim"*** zastavicu na ***false***, što znači da će se kontrolisati stvarni motori na ruci umesto simuliranih motora.
2. ***"tail --follow roslaunch-logs/robot_hardware_interface__joints.log"***
   Se koristi za prikazivanje i pracenje logova stanja zglobova ručnog robota u realnom vremenu. Ovo može biti korisno za otklanjanje grešaka i praćenje rada ručnog robota.
3. ***"roslaunch common_teleop routines_teleop.launch"***
   Pokreće čvor za teleoperaciju koji omogućava kontrolu ručnog robota preko joystick-a ili drugog ulaznog uređaja.
4. ***"rostopic echo /tf_publish/beam00_base__to__beam2_hand_ee"***
   Se koristi za prikazivanje trenutnog transformacije (pozicije i orijentacije) krajnjeg izvršioca ručnog robota u realnom vremenu. Ovo može biti korisno za praćenje pozicije i orijentacije ručnog robota tokom rada.
5. ***"roslaunch s4a_teleop servo_teleop.launch"***
   Pokreće čvor za teleoperaciju koji omogućava kontrolu ručnog robota preko joystick-a ili drugog ulaznog uređaja.
   Specifično je za kontrolu servo motora na ruci i pretpostavlja da joypad mora imati analog on.

Zajedno, ove naredbe se koriste za kontrolu i praćenje ručnog robota u ROS-based sistemu.

TOPIC

U ROS-u (Robot Operating System), tema je imenovani tok podataka koji omogućava različitim čvorovima da međusobno komuniciraju šaljući i primajući poruke.

Teme omogućavaju čvorovima da objave podatke (npr. čitanja senzora, komande kontrole) i da se drugi čvorovi pretplate na te podatke.

Svaka tema ima jedinstveno ime i tip podataka, a sve poruke objavljene na temi moraju biti istog tipa.

Teme se koriste za omogućavanje obrazaca komunikacije objavljivanja/pretplate između čvorova, što omogućava slobodno povezan sistem u kome se čvorovi mogu dodavati ili uklanjati bez uticaja na ostali deo sistema.

URDF

URDF stoji za ***Unified Robot Description Format.***
To je **XML format** koji se koristi za opis fizičkih i vizuelnih osobina robota i njegovih delova.
Opisuje zglobove, linkove i senzore robota, kao i njihovu kinematiku i dinamiku.
URDF datoteke se mogu učitati u simulatorima i vizualizatorima robota.

Ukratko, ***URDF*** datoteke opisuju fizičke osobine robota, dok ***SRDF*** datoteke opisuju semantiku robota.
Zajedno, one pružaju kompletan opis robota koji se može koristiti za simulaciju, vizualizaciju i manipulaciju.

SRDF

SRDF stoji za ***Semantic Robot Description Format.***
To je proširenje ***URDF***-a koji opisuje semantiku robota, kao što su konfiguracije robota i korisne pokrete.
SRDF datoteke sadrže informacije o grupama zglobova robota, end-efektorima i drugim semantičkim informacijama.
SRDF datoteke se mogu učitati u biblioteke za manipulaciju robota.

Ukratko, ***URDF*** datoteke opisuju fizičke osobine robota, dok ***SRDF*** datoteke opisuju semantiku robota.
Zajedno, one pružaju kompletan opis robota koji se može koristiti za simulaciju, vizualizaciju i manipulaciju.

CODE SNIPPETS

```
readJointNames();

  curr = State(n_joints);
  prev = State(n_joints);
  tmp_duty.resize(n_joints);
  acc_msg.data.resize(n_joints);


  ros::param::param("~all_motors_sim", all_motors_sim, false);
  ROS_INFO(
    "all_motors_sim = %s",
    all_motors_sim ? "true" : "false"
  );
```

Svrha ovog koda je da proveri da li su svi motori na robotu simulirani ili ne.
Ako je parametar ***"~all_motors_sim"*** postavljen na ***"true"***, to znači da su svi motori na robotu simulirani, u suprotnom je postavljen na ***"false"*** što znači da motori nisu simulirani.
Ova informacija se može koristiti za konfigurisanje ponašanja robota u skladu sa tim.

```
if(!all_motors_sim){
    // TODO Open driver.
    drv_fd = open(DEV_FN, O_RDWR);
  if(drv_fd < 0){
      ROS_WARN("Cannot open file", joints_log_fn.c_str());
}
}
```

Ovaj kod pokušava da otvori uređajsku datoteku koja se koristi za komunikaciju sa drajverom robota.

Ako je ***all_motors_sim*** postavljen na ***false***, to znači da motori nisu simulirani i kod treba da otvori file descriptor kako bi komunicirao sa stvarnim motorima.

Funkcija ***open*** se koristi da otvori uređajsku datoteku, a file descriptor se koristi da se komunicira sa drajverom.

```
for(int i = 0; i < n_joints+1; ++i){ // +1 for symetrical finger
    hardware_interface::JointStateHandle jsh(
      joint_names[i],
      &curr.pos_fb[i],
      &curr.vel_fb[i],
      &curr.eff_fb[i]
    );
    fb_if.registerHandle(jsh);

    hardware_interface::JointHandle jh(jsh, &curr.pos_cmd[i]);

    cmd_if.registerHandle(jh);
  }
  registerInterface(&fb_if);
```

```
  registerInterface(&cmd_if);

  ctrl_manager = new controller_manager::ControllerManager(this, nh);
  ros::Duration timer_period = ros::Duration(1.0/LOOP_HZ);
  non_realtime_loop_timer = nh.createTimer(
    timer_period,
    &RobotHardwareInterface::update,
    this
  );
```

Svrha ovog koda je da se podesi non-real-time kontrolni loop za robota.

Manager kontrolera je ROS paket koji pruža zajednički interfejs za interakciju sa različitim kontrolerima.

Manager kontrolera je odgovoran za učitavanje, isključivanje i prebacivanje kontrolera kao i ažuriranje kontrolera na određenoj frekvenciji.

Timer objekat se koristi da pozove ***"update"*** funkciju ***"RobotHardwareInterface"*** klase na frekvenciji od 1/LOOP_HZ sekundi.
Ova "update" funkcija je odgovorna za čitanje trenutnog stanja robota i ažuriranje kontrolera, tako da robot može da se kreće prema željenom ponašanju.
To omogućava hardverskom interfejsu robota da komunicira sa kontrolerima u non-real-time loop-u, a frekvencija ažuriranja je definisana varijablom ***LOOP_H.***

```
RobotHardwareInterface::~RobotHardwareInterface() {
  if(!all_motors_sim){
    // TODO Close driver.
    ROS_INFO("Closing driver");
    close(drv_fd);
  }
  delete ctrl_manager;
```

Svrha ovog koda je da očisti bilo koje resurse koji su alocirani tokom života objekta ***"RobotHardwareInterface"***.
Ako je ***"all_motors_sim"*** varijabla postavljena na false, to znači da je file descriptor otvoren da se komunicira sa drajverom robota i zatvara se pre nego što se objekat uništi.
Takođe, oslobađa memoriju koja je korišćena od strane ctrl_manager-a.

```
RobotHardwareInterface::State::State(size_t n_joints) {
  const double init_val = 0;
  pos_cmd.resize(n_joints+1, init_val);
  eff_fb.resize(n_joints+1, init_val);
  pos_fb.resize(n_joints+1, init_val);
  vel_fb.resize(n_joints+1, init_val);
  acc.resize(n_joints, init_val);
}
```

Svrha ovog koda je da se inicijalizuje stanje robota. Stanje robota uključuje poziciju, brzinu i ubrzanje zglobova robota.
Ove vrednosti su smeštene u članske varijable ***"pos_cmd", "eff_fb", "pos_fb", "vel_fb"*** i ***"acc".***
Konstruktor uzima broj zglobova na robotu kao argument i promenjuje veličinu ovih varijabli da bi mogli da sadrže stanje svih zglobova.
Inicijalizujući ih na 0, stanje robota se postavlja na podrazumevano, poznato stanje.

```
bool operator!=(
  const RobotHardwareInterface::State& a,
  const RobotHardwareInterface::State& b
) {
  return
    a.pos_cmd != b.pos_cmd ||
    a.eff_fb != b.eff_fb ||
    a.pos_fb != b.pos_fb ||
```

```
    a.vel_fb != b.vel_fb ||
    a.acc != b.acc;
}
```

Svrha ove funkcije je da omogući lakše upoređivanje dva objekta ***"RobotHardwareInterface::State"*** klase kako bi se proverilo da li su različiti.
Ovo može biti korisno za proveru da li se stanje robota promenilo, na primer u kontrolnom ciklusu ili za potrebe debugga.

```
void RobotHardwareInterface::update(const ros::TimerEvent& e) {

  ros::Time now = ros::Time::now();
  ros::Duration elapsed_time = ros::Duration(e.current_real - e.last_real);
  read(elapsed_time);
  ctrl_manager->update(now, elapsed_time);
  write(elapsed_time);

  log_joint_states_fun(now);

  prev = curr;
}
```

Ova funkcija je deo kontrolnog loop-a robota, poziva se periodično i ažurira kontrolere i zapisuje stanje zglobova.
Čita i piše podatke sa robota, i ažurira prethodno stanje sa trenutnim stanjem.

```
void RobotHardwareInterface::read(ros::Duration elapsed_time) {
  double dt = elapsed_time.toSec();


  if(all_motors_sim){
    for(int id = 0; id < n_joints; id++){
      // rad = % * s * rad/s
      curr.pos_fb[id] += prev.eff_fb[id]/100*dt*SPEED;
    }
  }else{
    // TODO Read from driver.
    if(offset_channels){
      int o = lseek(drv_fd, SEEK_SET, sizeof(tmp_duty[0])*N_CH_OFFSET);
      if(o < 0){
        ROS_WARN(
          "lseek failed! errno = %d -> %s",
          errno, strerror(errno)
        );
      }
    }
    const int s = sizeof(tmp_duty[0])*n_joints; // Symetrical not read.
    int r = ::read(drv_fd, (char*)tmp_duty.data(), s);
    if(r != s){
      ROS_WARN("read went wrong!");
    }
    for(int id = 0; id < n_joints; id++){
      curr.pos_fb[id] = (tmp_duty[id] - 25)*(M_PI/100) - M_PI_2;
    }
  }

  for(int id = 0; id < n_joints; id++){
    double pos_cmd;
    if(motors_en){
      pos_cmd = curr.pos_cmd[id];
    }else{
      pos_cmd = 0;
    }
    // Estimate effort.
    curr.eff_fb[id] = sym_clamp(
      // Distance left / speed = time left
      // time left / dt * 100 = percent left.
      (pos_cmd - curr.pos_fb[id])/SPEED/dt * 100,
      100.0
    );
```

```
    curr.vel_fb[id] = (curr.pos_fb[id] - prev.pos_fb[id])/dt;
    curr.acc[id] = (curr.vel_fb[id] - prev.vel_fb[id])/dt;
  }
```

Funkcija prima ***"ros::Duration"*** objekat nazvan ***"elapsed_time"*** koji se koristi da izračuna ***promenu vremena (dt)*** od poslednjeg ažuriranja.
Funkcija onda proverava da li su svi motori robota simulirani ili ne (***all_motors_sim***).
Ako jesu, funkcija ažurira trenutno povratno stanje pozicije svakog zgloba, dodajući prethodno povratno stanje napora, podeljeno sa 100, pomnoženo sa ***dt*** i ***SPEED***.
Ako motori nisu simulirani, funkcija treba da čita iz drajvera.

```
void RobotHardwareInterface::write(ros::Duration elapsed_time) {
  if(!all_motors_sim && motors_en){
    // Write to driver.
    if(offset_channels){
      int o = lseek(drv_fd, SEEK_SET, sizeof(tmp_duty[0])*N_CH_OFFSET);
      if(o < 0){
        ROS_WARN(
          "lseek failed! errno = %d -> %s",
          errno, strerror(errno)
        );
      }
    }
    // -pi/2=2.5%=25 +pi/2=12.5%=125
    for(int id = 0; id < n_joints; id++){
      tmp_duty[id] = (curr.pos_cmd[id] + M_PI_2)*(100/M_PI) + 25;
    }
    const int s = sizeof(tmp_duty[0])*n_joints; // Symetrical not writen.
    int r = ::write(drv_fd, (char*)tmp_duty.data(), s);
    if(r != s){
      ROS_WARN("write went wrong!");
    }
  }
}
```

Ova funkcija je odgovorna za slanje vrednosti komande položaja za svaki zglob drajveru.
Ona se samo nastavlja ako su motori omogućeni i nisu u simulaciji.
Ako je flag ***offset_channels*** postavljen na ***true***, koristi se ***lseek()*** funkcija da se postavi pozicija u drajver fajlu na offset, slično kao ***read()*** funkcija.

Funkcija onda pretvara vrednosti komande položaja za svaki zglob u vrednost ciklusa rada između 25 i 125.
Ovo se radi tako što se pi/2 dodaje komandi položaja, rezultat se množi sa 100/pi, a zatim se dodaje 25.

Na kraju, funkcija upisuje ove vrednosti ciklusa rada u drajver koristeći ***write()*** funkciju, sa obradom grešaka za slučaj kada se neuspelo pisanje, što će ispisati poruku upozorenja.
Upis se vrši na celu memorijsku lokaciju, ne ograničava se na određeni zglob.

```
void RobotHardwareInterface::motors_en_cb(
  const std_msgs::Bool::ConstPtr& msg
) {
  motors_en = msg->data;
  ROS_INFO(
    "Motors %s", motors_en ? "enabled" : "disabled"
  );
}
```

Ovo je funkcija koja se poziva kada se primi poruka na određenom topic-u.
Ova promenljiva se koristi da se odredi da li su motori omogućeni ili onemogućeni.

```
static shared_ptr<srdf::Model> readSRDF() {
  bool all_ok = true;
  bool ok;

  string urdf_xml_string;
  ok = ros::param::get("/robot_description", urdf_xml_string);
  if(!ok){
    all_ok = false;
    ROS_WARN("Cannot read \"/robot_description\" param!");
  }

  string srdf_xml_string;
  ok = ros::param::get("/robot_description_semantic", srdf_xml_string);
  if(!ok){
    all_ok = false;
    ROS_WARN(
      "Cannot read \"/robot_description_semantic\" param! "\
        "Maybe demo.launch is not started."
    );
  }

  if(!all_ok){
    ROS_WARN("Cannot read SRDF!");
    return nullptr;
  }


  ScopedLocale l("C");

  urdf::ModelInterfaceSharedPtr u = urdf::parseURDF(urdf_xml_string);
  shared_ptr<srdf::Model> s = std::make_shared<srdf::Model>();
  s->initString(*u, srdf_xml_string);

  return s;
}
```

Ova funkcija se koristi za čitanje ***URDF*** i ***SRDF*** robota iz ROS parametara i njihovo parsiranje u memoriji.

```
void RobotHardwareInterface::readJointNames() {
  shared_ptr<srdf::Model> s = readSRDF();
  if(!s){
    ROS_ERROR("Cannot read joint names!");
  }

  for(auto& g: s->getGroups()){
    for(int i = 0; i < g.joints_.size(); i++){
      auto jn = g.joints_[i];
      if(jn != "virtual_joint"){
        joint_names.push_back(jn);
      }
    }
  }
  //joint_names.pop_back(); // Remove symetrical finger.

  n_joints = joint_names.size()-1;
  if(n_joints == 0){
    ROS_ERROR("Cannot read joint names!");
  }
}
```

Funkcija prvo poziva funkciju ***readSRDF()*** da pročita ***SRDF*** datoteku, i smešta povratnu pokazivačku vrednost u objekat klase ***srdf::Model*** pod nazivom ***s***.
Ako se vrati pokazivač koji je null, funkcija vraća poruku o grešci "Cannot read joint names!"

Funkcija onda prolazi kroz sve grupe zglobova u ***SRDF*** modelu, proverava ime svakog zgloba, ako nije "***virtual_joint***", onda dodaje to ime u ***vektor*** pod nazivom ***joint_names.***

Na kraju, postavlja promenljivu ***n_joints*** na veličinu vektora ***joint_names-1***, i proverava da li je n_joints jednak nuli.

Ako jeste, vraća poruku o grešci "Cannot read joint names!".

Ukratko, ova funkcija se koristi da pročita imena zglobova robota iz ***SRDF*** datoteke i smesti ih u vektor pod nazivom joint_names i postavlja vrednost promenljive ***n_joints*** na broj zglobova.

VEZBE 6

DUTY

Duty ciklus signala je odnos vremena tokom kog je signal u određenom stanju (visok ili nizak) u odnosu na ukupni period signala.

Obično se izražava kao procenat ukupnog perioda, sa duty ciklusom od 0% što označava da je signal uvek u niskom stanju, i duty ciklusom od 100% što označava da je signal uvek u visokom stanju.

U slučaju PWM-a, duty ciklus predstavlja količinu vremena PWM signala u visokom (uključenom) stanju tokom jednog perioda.

BLDC (Brushless Direct Current)

Da bismo kontrolisali ***BLDC*** motor u ROS-u, tipično bismo koristili kontroler motora koji se vezuje sa ROS sistemom preko serijskog ili USB veze.

Kontroler motora bi bio odgovoran za kontrolu snage i brzine motora na osnovu komandi koje su primljene od ROS sistema.

U ROS-u, može se napisati čvor koji se povezuje sa kontrolerom motora i šalje komande ***BLDC*** motoru. ROS nudi biblioteke kao što je ***robot_hardware_interface*** koji omogućava interakciju sa hardverom na unificiran način, što znači da možemo kontrolisati ***BLDC*** motore zajedno sa drugim tipovima hardvera, kao što su senzori i aktuatori, koristeći zajednički interfejs.

SW/Test/test_app/flow.sh

Komande:

- ***./waf build && ./build/test_bldc***

waf build je slican kao make komanda, koristi se za izgradnju ***test_bldc*** aplikacije gde sa argumentima ***+-20*** postavljamo brzinu motora.

- ***./waf build && ./build/test_servos w 0 25***

Ova komanda nam sluzi da izgradimo ***test_servos*** aplikaciju. Argument ***w 0 25*** nam postavlja jedan od zglobova robotove ruke na željenu poziciju gde nam ***"w"*** implicira na funckiju ***write.***

---

motor_ctrl_write()

```
static ssize_t motor_ctrl_write(
  struct file* filp,
  const char *buf,
  size_t len,
  loff_t *f_pos
) {
  int i;

  //TODO copy_from_user() -> duty
  if(copy_from_user(duty, buf, len)==0)
  {
    //TODO bldc__set_dir()
    if(duty[0]==1)
      bldc__set_dir(BLDC__CH_0, CW);
```

```
    else
      bldc__set_dir(BLDC__CH_0, CCW);

    //TODO hw_pwm__set_threshold(ch, abs_duty << 1);
    //hw_pwm__set_threshold(HW_PWM__CH_0,
    for(i=0; i<2; i++){
      duty[i]=abs(duty[i]);
      duty[i]=duty[i]<<1;
    }
    hw_pwm__set_threshold(HW_PWM__CH_0, duty[0]);
    hw_pwm__set_threshold(HW_PWM__CH_1, duty[1]);
    return len;
  }
  else
      return -EFAULT;
}
```

Ova funkcija se koristi za kontrolu motora podešavanjem smera i pragova BLDC kanala i PWM kanala odgovarajuće. To čini kopiranjem podataka iz bafera i korišćenjem istih za podešavanje smera i pragova (prag označava najmanju potrebnu snagu za pokretanje motora) motora.

Ona prima četiri parametra:

- **struct file*** filp: pokazivač na strukturu datoteke, koristi se za pristup datoteci koja se čita ili piše
- **const char*** buf: pokazivač na bafer u koji se upisuju podaci
- **size_t** len: dužina podataka u baferu
- **loff_t** f_pos: pokazivač na poziciju datoteke

1. Koristi funkciju ***copy_from_user()*** da kopira podatke iz bafera na koji pokazuje buf u promenljivu duty.
2. Ako je kopiranje uspešno, izvršiće sledeće:
3. Proverava prvu vrednost promenljive duty, ako je jednaka 1, postavlja smer BLDC kanala 0 na smer kazaljke na satu (***CW***), u suprotnom postavlja ga na suprotni smer kazaljke na satu (***CCW***) pozivanjem funkcije ***bldc__set_dir()***.
4. Zatim koristi for petlju da postavi apsolutnu vrednost promenljive duty, i onda bitno levo pomera za 1, što povećava prag BLDC motora.
5. I postavlja prag PWM kanala 0 i 1 odgovarajuće koristeći funkciju ***hw_pwm__set_threshold***().
6. Funkcija zatim vraća dužinu podataka.
7. Ako kopiranje nije uspelo, funkcija vraća -***EFAULT***, što je kod greške koji označava da se desila greška tokom poziva sistema.

Notes

- Korisnik šalje željene podatke u prostor jezgra, gde se koriste za kontrolu količine snage koja se isporučuje opterećenju putem BLDC (bezkontaktni direktni strujni) motora.
- Kernel korišćenjem ovih podataka kontroliše stanje PWM (modulacija širine pulsa) signala, što zauzvrat kontroliše snagu koja se isporučuje BLDC motoru.
- Ovo omogućava korisniku da kontroliše brzinu i smer BLDC motora.

---

motor_ctrl_read()

```
static ssize_t motor_ctrl_read(
  struct file* filp,
  char* buf,
  size_t len,
  loff_t* f_pos
) {
```

```
  motor_ctrl__read_arg_fb_t a;
  int i;

  for(i=0; i<2; i++)
  {
      a.pos_fb[i]=duty[i];
      bldc__get_pulse_cnt(i, &a.pulse_cnt_fb[i]);
  }



  if(copy_to_user(buf, &a, len)!=0)
  return -EFAULT;

  return len;
}
```

Ova funkcija se koristi da pročita povratne informacije sistema za kontrolu motora. To čini kopiranjem vrednosti određenih promenljivih i čitanjem određenih vrednosti korišćenjem specifičnih funkcija, a zatim kopiranjem podataka u pruženi bafer.

Ona prima četiri parametra:

- **struct file*** filp: pokazivač na strukturu datoteke, koristi se za pristup datoteci koja se čita ili piše
- **char*** buf: pokazivač na bafer u koji se skladište podaci
- **size_t** len: dužina podataka u baferu
- **loff_t** f_pos: pokazivač na poziciju datoteke

1. Funkcija počinje kreiranjem varijable **a** tipa ***motor_ctrl__read_arg_fb_t***.
2. Zatim koristi for petlju da kopira vrednosti niza **duty** u polje ***pos_fb*** varijable ***a***.
3. Zatim poziva funkciju ***bldc__get_pulse_cnt()*** da dobije broj pulsa BLDC kanala i smešta ga u polje ***pulse_cnt_fb*** varijable a.
4. Zatim koristi funkciju ***copy_to_user()*** da kopira podatke iz varijable ***a*** u bafer na koji pokazivač ***buf*** pokazuje (u user space).
5. Ako kopiranje uspe, ona vraća dužinu podataka, inače vraća ***-EFAULT.***

Notes

- ***motor_ctrl_read()*** iterira samo 2 puta, jer čita povratne informacije za samo 2 kanala (za poziciju i kontrolu pulsa)
- Info sa 2 kanala se smešta u ***a.pos_fb*** i ***a.pulse_cnt_fb***

---

motor_ctrl_ioctl()

```
static long motor_ctrl_ioctl(
  struct file* filp,
  unsigned int cmd,
  unsigned long arg
) {
  motor_ctrl__ioctl_arg_moduo_t a;
  a = *(motor_ctrl__ioctl_arg_moduo_t*)&arg;

  //TODO Check cmd

  //TODO hw_pwm__set_moduo()
  hw_pwm__set_moduo(a.ch, a.moduo);
  return 0;
}

static struct file_operations motor_ctrl_fops = {
  open           : motor_ctrl_open,
  release        : motor_ctrl_release,
```

```
  read           : motor_ctrl_read,
  write          : motor_ctrl_write,
  unlocked_ioctl : motor_ctrl_ioctl
};
```

Ova funkcija služi za upravljanje PWM modulom (periodom PWM signala) korišćenjem ioctl komande.

Ona prima 3 parametra:

- **struct file*** filp: Ovaj parametar je pokazivač na strukturu datoteke, koristi se za pristup datoteci koja se čita ili piše.
- **unsigned int** cmd: Ovaj parametar sadrži komandu koju funkcija treba da izvrši.
- **unsigned long** arg: Ovaj parametar sadrži bilo koji dodatni argument potreban za izvršavanje komande.

1. Preko komande ***IOCTL_MOTOR_CLTR_SET_MODUO***, funkcija prima argument ***'arg'*** koji sadrži informacije o kanalu (***ch***) i moduo vrednosti (***moduo***) za PWM modul.
2. Ukoliko je kanal manji od broja hardverskih kanala (***HW_PWM__N_CH***), funkcija poziva ***hw_pwm__set_moduo()*** funkciju koja postavlja moduo vrednost za dati hardverski kanal.
3. Ukoliko je kanal veći od broja hardverskih kanala, funkcija poziva ***sw_pwm__set_moduo()*** funkciju koja postavlja moduo vrednost za dati softverski kanal.

---

motor_ctrl_exit()

```
void motor_ctrl_exit(void) {
  printk(KERN_INFO DEV_NAME": Removing %s module\n", DEV_NAME);

  sw_pwm__exit();
  hw_pwm__exit();

  gpio__exit();

  unregister_chrdev(DEV_MAJOR, DEV_NAME);
}
```

Koristi se kao funkcija za čišćenje kada se modul drajvera uklanja iz sistema.

Ona oslobađa resurse koji su alocirani od strane drajvera i izvodi neophodne zadatke za čišćenje pre nego što drajver izađe.

1. Funkcija počinje sa ispisom poruke koja ukazuje da se modul uklanja.
2. Zatim poziva exit funkcije drugih modula koji su inicijalizovani od strane drajvera, kao što su ***sw_pwm__exit()***, ***hw_pwm__exit()*** i ***gpio__exit()***.
3. Ove funkcije oslobađaju resurse koji su korišćeni od strane ovih modula i izvode neophodne zadatke za čišćenje.
4. Na kraju, ona deregistruje uređaj pozivanjem funkcije ***unregister_chrdev()*** sa glavnim brojem uređaja i nazivom uređaja. Ovo deregistruje uređaj i čini ga nedostupnim za buduću upotrebu.

SW/Test/test_app/test_bldc.c

Fajl ***SW/Test/test_app/test_bldc.c*** sadrži test aplikacije koje se koriste za testiranje i kontrolu BLDC motora u ROS-u.

Ovaj fajl može biti koristan za razumevanje kako kontrolisati BLDC motore i testirati ih u ROS okruženju.

ROS/arm_and_chassis_ws/src/wc_main/src/simple_ackermann_steering_controller.

Fajl ***ROS/arm_and_chassis_ws/src/wc_main/src/simple_ackermann_steering_controller.cpp*** sadrži kod koji implementira upravljački sistem za robota u ROS-u.

Ovaj fajl je koristan za razumevanje kako kontrolisati i pogonske i upravljačke motore robota, kao i razumevanje ackermann upravljačkog modela.

---

hw_pwm__set_moduo(ch, moduo)

Ova funkcija postavlja vrednost modula za kanal hardverskog PWM-a.

Funkcija uzima dva argumenta, "***ch***" koji je broj kanala i "***moduo***" koji je vrednost modula koja se postavlja.

Funkcija koristi konstantni niz ***"offsets"*** da odredi offset adresu u memoriji za određeni PWM kanal.

Offset adresa u memoriji se odnosi na lokaciju određenog komada podataka unutar većeg bloka memorije. U kontekstu PWM (modulacije širine impulsa), offset adresa u memoriji bi se mogla odnositi na lokaciju registara za kontrolu PWM-a unutar memorije.

Zatim koristi ***iowrite32*** funkciju da upiše vrednost modula u lokaciju u memoriji registra modula PWM kanala. Ovo efektivno postavlja vrednost modula za PWM kanal.

---

bldc__set_dir(ch, dir)

Ova funkcija se koristi za postavljanje smera BLDC (bezkontaktni DC) motora.

Funkcija prima dva argumenta:

- Kanal (***ch***) BLDC motora
- Smer (***dir***) u kom bi motor trebao da se okrene

Smer može biti u smeru kazaljke na satu (***CW***) ili suprotno kazaljci na satu (***CCW***). Funkcija prvo proverava da li je kanal koji je prosleđen unutar opsega dostupnih BLDC kanala (***BLDC__N_CH***).

Ako jeste, funkcija postavlja smer BLDC motora na tom kanalu na prosleđeni smer i postavlja odgovarajući pin za smer (***dir_pin***) BLDC motora na visoko ili nisko zavisno od prosleđenog smera.

Ako je kanal koji je prosleđen izvan opsega, funkcija ništa ne radi.

---

hw_pwm__set_threshold(ch, threshold)

Ova funkcija postavlja prag (***threshold***) za kanal (***ch***) hardverskog PWM-a (***hw_pwm***).

Korišćenjem offseta za kanal koji se prosleđuje kao parametar, funkcija pristupa adresi u virtuelnoj memoriji na kojoj se nalazi registar ***DAT*** (privremeni registar za skladištenje podataka) odgovarajućeg kanala i upisuje vrednost threshold-a.

VEZBE 7

servo_fb.c

Implementacija modula za povratnu informaciju servo motora za Linux sistem.

Modul je dizajniran da pruža Pulse Width Modulation (PWM) povratnu informaciju za servo motor.

On koristi **Linux kernel API** da interaktuje sa GPIO sistemom, i koristi **Linux kernel vremenske funkcije** da izmeri trajanje PWM signala.

Modul koristi strukturu ***"servo_fb_t"*** da bi sačuvao broj pin-a, broj ***irq*** (interrupt request), vremensku tačku padajuće ivice i atomsku ***T_on*** (vreme u stanju rada) promenljivu za svaki kanal servo motora.

Kada je pozvana funkcija ***servo_fb__init()*** ona inicijalizuje sve kanale sa povratnim informacijama servo motora.

Ona podešava ***pinmux*** određenog pina kao ulazni i traži ***irq*** (Interrupt Request) za taj pin.

Ona takođe postavlja f-ju obrade ***irq*** na ***fb_isr***.

Funkcija ***fb_isr()*** se poziva kada ima rastući ili padajući ivični trenutak na pin-u.

Funkcija zapisuje vremensku tačku padajuće ivice u promenljivu ***t_fe*** i postavlja atomsku promenljivu ***T_on*** na prethodno zapisanu vremensku tačku padajuće ivice.

Kada je pozvana funkcija ***servo_fb__exit()*** ona isključuje ***irq*** i oslobađa irq i ***gpio*** za sve kanale povratne informacije servo motora.

---

gpio__steer_pinmux()

```
void gpio__steer_pinmux(uint8_t pin, gpio__pinmux_fun_t pinmux_fun) {
  uint8_t reg;
  uint8_t shift;
  uint32_t tmp;

  if(check_pin(pin)){
    return;
  }

  get_gpfsel_offsets(pin, reg, idx);

  // Read whole register.
  tmp = ioread32(virt_gpio_base + reg);

  // Clear 3b field.
  tmp &= ~(0b111 << shift);

  // Set 3b with appropried function
  tmp |= pinmux_fun << shift;

  // Write back updated value.
  iowrite32(tmp, virt_gpio_base + reg);
}
```

1. Inicijalizuje GPIO pin za kanal povratne informacije servo-a (***servo_fb***).
2. Prva linija postavlja pinmux pin povezan sa trenutnim kanalom (***servo_fb[ch]***) na mod ulaza (***GPIO__IN***).
3. Sledeći blok koda zahteva da kernel Linux-a rezerviše pin (***servo_fb[ch].pin***) za korišćenje kao ulazni pin i dodeljuje mu oznaku.
4. Ako zahtev za rezervisanjem pina ne uspe, kod štampa poruku o grešci i vraća 1.

Steering pinmux je funkcija koja omogućava korisniku da konfiguriše višestruko korišćene ulazno-izlazne (I/O) pinove uređaja da izvrše specifičnu funkciju.

Pinmux se koristi za dodeljivanje specifičnih funkcija datom pinu, kao što su digitalni ulaz, digitalni izlaz, analogni ulaz ili periferna funkcija.

---

gpio_request_one()

Funkcija u kernelu Linux-a koja se koristi za zahtevanje i konfigurisanje jednog GPIO pina.

Ona prima tri argumenta:

- Broj GPIO pina koji zahtevate

- Oznake koje opisuju smer i ponašanje pina
- Oznaka koja će biti povezana sa zahtevanim pinom

Ova funkcija će rezervisati navedeni pin za korišćenje pozivaoca, konfigurisati ga prema oznakama i vratiti 0 u slučaju uspeha, u suprotnom će vratiti negativan kod greške.

---

gpio_to_irq()

Funkcija u kernelu Linux-a koja konvertuje broj GPIO pina u odgovarajući broj IRQ-a.

Ona prima jedan argument:

- Broj GPIO pina za koji želite da dobijete broj IRQ-a

Ova funkcija se koristi da se dobije broj IRQ-a za dati GPIO pin, tako da možete zahtevati i konfigurisati prekid na tom pinu.

Jednom kada imate broj IRQ-a, možete ga koristiti da registrujete obrađivač prekida sa kernelom, koji će biti pozvan kada se dogodi događaj prekida na tom pinu.

Vraća odgovarajući broj IRQ-a u slučaju uspeha, u suprotnom će vratiti kod greške.

---

request_irq()

Funkcija u kernelu Linux-a koja se koristi za registrovanje obrađivača prekida za specifični IRQ, tako da kada se dogodi prekid na tom IRQ-u, poziva se registrovana funkcija obrađivača.

Ona prima pet argumenata:

- Broj IRQ-a za koji želite da registrujete obrađivač
- Pokazivač na funkciju obrađivača prekida koja će se pozvati kada se dogodi prekid na tom IRQ-u
- Oznake koje navode tip prekida i ponašanje obrađivača
- String koji se koristi kao naziv prekida, koji će se prikazati u datoteci ***/proc/interrupts***
- Pokazivač na strukturu uređaja koja je povezana sa prekidom

Ova funkcija će registrovati navedeni obrađivač prekida za dati broj IRQ-a, konfigurisati ga prema oznakama i vratiti 0 u slučaju uspeha, u suprotnom će vratiti negativan kod greške.

---

fb_isr()

```
static irqreturn_t fb_isr(int irq, void* data) {
  u64 t = ktime_get_ns();
  servo_fb_t* p = (servo_fb_t*)data;
  bool s = gpio__read(p->pin);
  if(s){
    // It was rising edge.
    atomic64_set(&p->T_on, p->t_fe);
  }else{
    // It was falling edge.
    p->t_fe = t;
  }
  //TODO calc duty
  u64 high_time = atomic64_read(&p->T_on) - p->t_fe;
  u64 period = p->t_fe - p->t_prev_fe;
  double duty_cycle = (double)high_time / (double)period;

  p->t_prev_fe = t;

  return IRQ_HANDLED;
}
```

Funkcija **Interrupt Service Routine (ISR)** u datoteci **servo_fb.c**, koja se aktivira prilikom promene stanja ulaznog pina.

Aktivira se prilikom uspona ili pada nivoa na pinu.

Kada se ISR aktivira, dobija se trenutno vreme u nanosekundama koristeći ***ktime_get_ns()***.

Čita se trenutno stanje pina koristeći ***gpio__read(p->pin)*** i postavlja se vrednost **T_on** na **t_fe** (vreme padajuće ivice) ako je pin visok, u suprotnom se postavlja ***t_fe*** (vreme padajuće ivice) na trenutno vreme.

Jedan način da se izračuna duty ciklus je da se koriste vremenske vrednosti koje se čuvaju u promenljivim 'p->T_on' i 'p->t_fe'.

Možete oduzeti vrednost **'p->T_on'** od **'p->t_fe'** da biste dobili trajanje visokog stanja, a zatim podelite to sa ukupnim periodom (što bi bilo 'p->t_fe' minus prethodni padajući rub vremena) da biste dobili duty ciklus kao decimalnu vrednost između 0 i 1.

ISR vraća ***IRQ_HANDLED*** kada završi.

timer_callback()

```
static enum hrtimer_restart timer_callback(struct hrtimer* p_timer) {
  //TODO sw_pwm_t* ps = container_of(p_timer, sw_pwm_t, timer);
  sw_pwm_t* ps = (sw_pwm_t*)p_timer;
  unsigned long flags;

  ps->on = !ps->on;
  if(ps->on){
    // Changing interval at the end of period.
    spin_lock_irqsave(&ps->interval_pending_lock, flags);
    ps->on_interval = ps->on_interval_pending;
    ps->off_interval = ps->off_interval_pending;
    spin_unlock_irqrestore(&ps->interval_pending_lock, flags);

    hrtimer_forward_now(&ps->timer, ps->on_interval);
    gpio__set(ps->pin);
  }else{
    hrtimer_forward_now(&ps->timer, ps->off_interval);
    gpio__clear(ps->pin);
  }

  return HRTIMER_RESTART;
}
```

Funkcija u datoteci ***sw_pwm.c***, to je funkcija povratnog poziva koja se aktivira kada istekne tajmer.

Postavlja se trenutno stanje pina (uključeno ili isključeno) koristeći ***gpio__set(ps->pin)*** i ***gpio__clear(ps->pin)*** i menja se interval vremena uključenja i isključenja koristeći ***ps->on_interval*** i ***ps->off_interval.***

Takođe se ažuriraju nove vrednosti intervala uključenja i isključenja koristeći ***ps->on_interval_pending*** i ***ps->off_interval_pending*** i vraća se ***HRTIMER_RESTART*** kada završi.

hrtimer_forward_now()

hrtimer_forward_now(&ps->timer, ps->on_interval);

Koristi se za podešavanje vremena isteka tajmera na trenutno vreme plus vreme koje je sačuvano u ***ps->on_interval***, tako da će tajmer isteći nakon tog vremena.

bldc__set_dir()

```
void bldc__set_dir(bldc__ch_t ch, dir_t dir) {
  if(ch >= BLDC__N_CH){
    return;
```

```
  }
  bldc[ch].dir = dir;
  if(dir == CW){
    gpio__set(bldc[ch].dir_pin);
  }else{
    gpio__clear(bldc[ch].dir_pin);
  }
}
```

Služi za postavljanje smera rada BLDC motora.

Funkcija prima dva argumenta:

- "***ch***" koji predstavlja kanal BLDC motora
- "***dir***" koji predstavlja smer rada

U funkciji se prvo sprema smer rada u nizu "***bldc***" na odgovarajućem indeksu kanala.

Zatim se proverava da li je smer rada "**CW**" ili "**CCW**" i prema tome se na odgovarajućem pinu (***bldc[ch].dir_pin***) postavlja logička jedinica ili logički nula.

---

bldc_init()

```
int bldc__init(void) {
  int r;
  bldc__ch_t ch;

  for(ch = 0; ch < BLDC__N_CH; ch++){
    gpio__steer_pinmux(bldc[ch].dir_pin, GPIO__OUT);
    bldc__set_dir(ch, CW);


    gpio__steer_pinmux(bldc[ch].pg_pin, GPIO__IN);
    // Initialize GPIO ISR.
    r = gpio_request_one(
      bldc[ch].pg_pin,
      GPIOF_IN,
      bldc[ch].pg_label
    );
    if(r){
      printk(
        KERN_ERR DEV_NAME": %s(): gpio_request_one() failed!\n",
        __func__
      );
      goto exit;
    }

    bldc[ch].pg_irq = gpio_to_irq(bldc[ch].pg_pin);
    r = request_irq(
      bldc[ch].pg_irq,
      pg_isr,
      IRQF_TRIGGER_FALLING,
      bldc[ch].pg_label,
      &bldc[ch]
    );
    if(r){
      printk(
        KERN_ERR DEV_NAME": %s(): request_irq() failed!\n",
        __func__
      );
      goto exit;
    }
  }

exit:
  if(r){
    printk(KERN_ERR DEV_NAME": %s() failed with %d!\n", __func__, r);
    bldc__exit();
  }

  return r;
}
```

Ova funkcija inicijalizuje bezkontaktni DC motor (BLDC) izvršavanjem sledećih zadataka za svaki kanal BLDC-a:

Inicijalizuje pin za smer čišćenjem pina 17 i postavljanjem ***pinmux-a*** za ovaj pin da bude izlaz.

Inicijalizuje tajmer za softversku modulaciju širine impulsa (PWM) postavljanjem pina, čišćenjem istog i podešavanjem pinmux-a za pin da bude izlaz.

Inicijalizuje IRQ (Interrupt request).

Vraća 0 ako funkcija uspešno završi.

---

sw_pwm.c

Datoteka pod nazivom ***"sw_pwm.c"*** u okruženju ROS (Robot Operating System) sadrži izvorni kod za implementaciju Pulse Width Modulation (PWM) na osnovu softvera.

PWM je tehnika koja se koristi za kontrolu količine snage koja se dostavlja uređaju, kao što su motor ili LED, tako što se snaga brzo uključuje i isključuje.

"sw" u nazivu datoteke ukazuje da se radi o softverskoj implementaciji PWM-a, a ne o hardverskoj implementaciji.

---

duty_permille

Ciklus rada, takođe poznat kao odnos rada ili širina impulsa, je mera koliko dugo signal bude u "uključenom" stanju u odnosu na ukupni period signala.

Obično se izražava kao procenat, sa vrednošću između 0 i 100, gde 0 odgovara signalu uvek u "isključenom" stanju, a 100 odgovara signalu uvek u "uključenom" stanju.

Duty_permille je samo još jedna predstava istog koncepta, ali umesto procenta, predstavljen je u permilima (‰) ili na hiljade.

---

bldc__get_pulse_cnt()

```
void bldc__get_pulse_cnt(bldc__ch_t ch, int64_t* pulse_cnt) {
  if(ch >= BLDC__N_CH){
    return;
  }

  int64_t pulse_cnt2 = bldc[ch].pulse_cnt;

  if(pulse_cnt2 != bldc[ch].pulse_cnt){
    pulse_cnt2 = bldc[ch].pulse_cnt;
  }
  *pulse_cnt = pulse_cnt2;
}
```

Ova funkcija čita brojač pulsa za određeni kanal bldc (brushless DC) motora.

Ako se prosledi indeks kanala koji je veći od ukupnog broja kanala, funkcija se neće izvršiti.

Inače, funkcija će skladištiti trenutni broj pulsa za taj kanal u varijablu ***"pulse_cnt2"*** i onda ce je kopirati u varijablu ***"pulse_cnt"*** koja je prosleđena kao argument.

---

atomic64_add()

Funkcija koja se koristi za atomsko dodavanje vrednosti u 64-bitnu varijablu.

Ova funkcija osigurava da se operacija dodavanja izvrši kao jedinstvena operacija, bez mogućnosti da se drugi procesi ili niti mešaju u taj proces u toku izvršavanja.

To znači da se vrednost varijable ne može promeniti od strane drugih procesa ili niti dok se ova operacija izvršava.

Pokretanje ROS

- ***roslaunch wc_main main.launch all_motors_sim:=false***

Pokreće datoteku pokretanja pod nazivom ***"main.launch"*** iz paketa ***"wc_main"***.

Komanda takođe postavlja oznaku pod nazivom ***"all_motors_sim"*** na ***"false"***, što se može koristiti u datoteci pokretanja za konfigurisanje sistema da koristi stvarne motore umesto simuliranih motora.

- ***roslaunch common_teleop routines_teleop.launch***

Pokreće datoteku pokretanja pod nazivom ***"routines_teleop.launch"*** iz paketa ***"common_teleop"***.

Ova datoteka pokretanja se može koristiti za pokretanje seta unapred definisanih rutina za robota, kao što su prethodno programirane sekvence kretanja ili zadaci.

- ***roslaunch wc_teleop manual_teleop.launch***

Pokreće datoteku pokretanja pod nazivom ***"manual_teleop.launch"*** iz paketa ***"wc_teleop"***.

Ova datoteka pokretanja se može koristiti za pokretanje seta čvorova koji omogućavaju ručno upravljanje robota, kao što su džojstik ili tastatura.
