Līdzstrāvas elektrodzinēja vadība ar impulsa platuma modulāciju.

1. Informācijas avoti

1. Infineon Technologies TLE 5206-2 5A H-Bridge for DC-Motor Applications, http://www.farnell.com/datasheets/1651367.pdf

2. Texas Instruments LMD18245 3A, 55V DMOS Full-Bridge Motor Driver, http://www.farnell.com/datasheets/1771610.pdf

3. Electric Imp Platform, http://electricimp.com/product/

4. Vishay Semiconductors TCST1230Transmissive Optical Sensor with Phototransistor Output, http://www.vishay.com/docs/83765/tcst1230.pdf

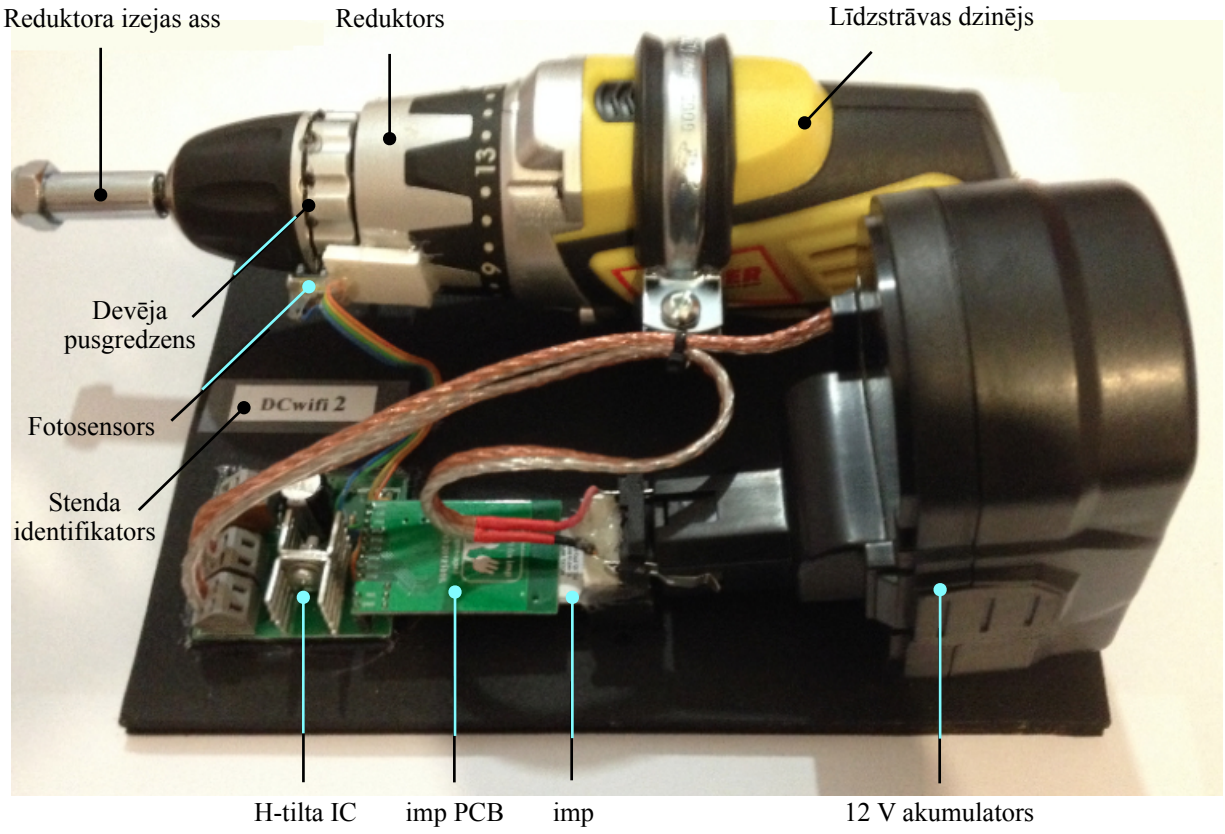
5.Bourns MF-R Series - PTC Resettable Fuses, http://www.farnell.com/datasheets/1719963.pdf

6. **imp** kods šī apraksta beigās - Squirrel valoda, http://squirrel-lang.org

`Darbu uzsākot, iepazīties un izprast avotos doto informāciju.`

2. Ievads

Laboratorijas darba stends attēlots Zīm.1.



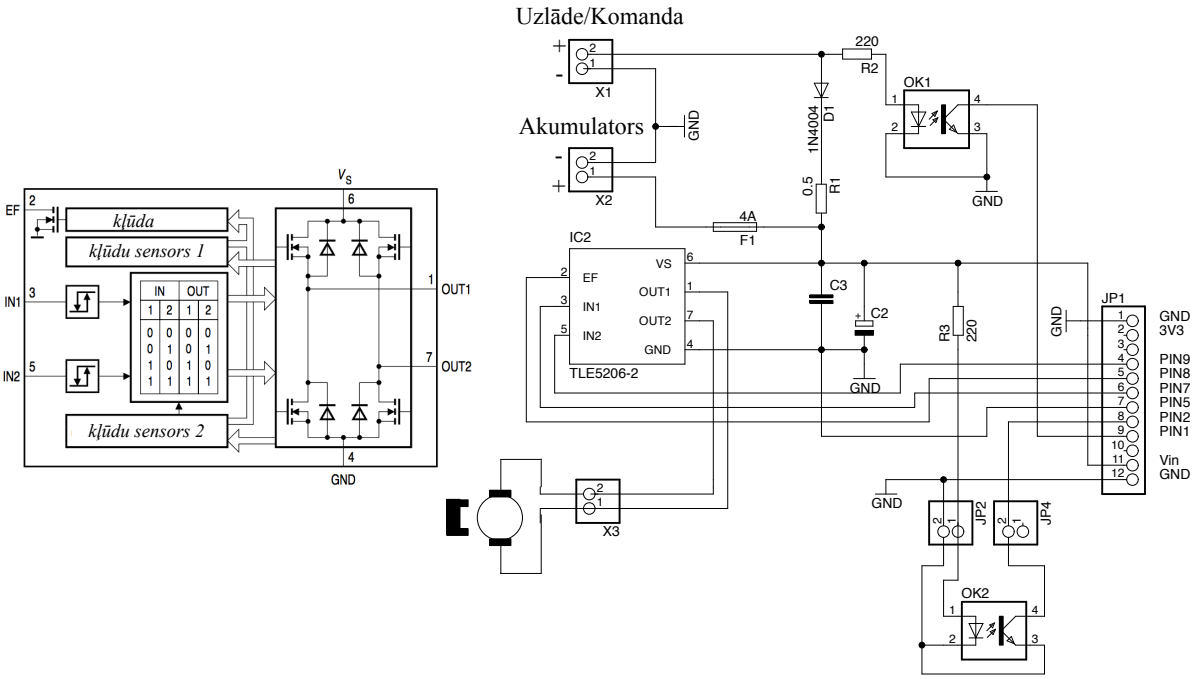
Zīm.1. Laboratorijas darba stends

Impulsa platuma modulāciju (IPM, *PWM*) plaši izmanto dažādu elektrotehnoloģisko iekārtu vadībā. Sekojošie darbi ir paredzēti labākai speciālitātes apguvei.

Darbā izmanto "nezināmu" 12 V līdzstrāvas dzinēju ar pastāvīgo magnētu ierosmi un iebūvētu reduktoru. Nominālais reduktora izejas ass griešanās ātrums ir 550 apgr/min (barošanas spriegums 12 V)

3. Principiālā elektriskā shēma.

Stenda principiālā elektriskā shēma un IC funkcionālā shēma ir attēlotas Zīm.2

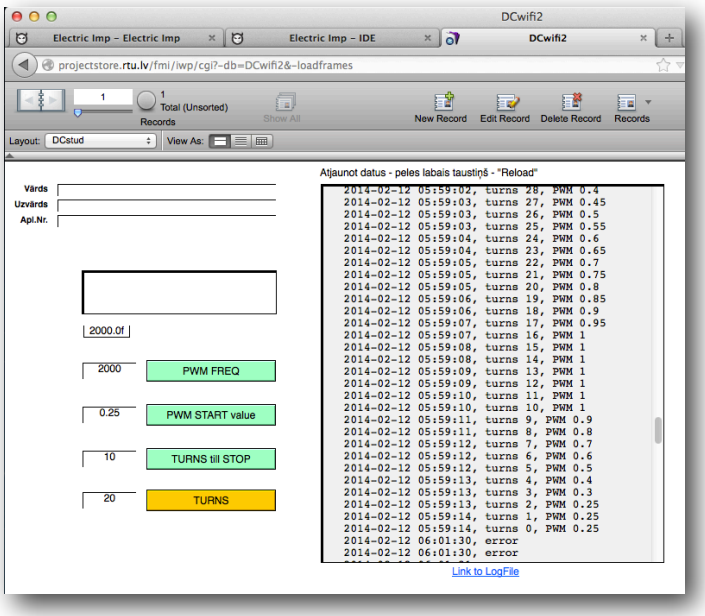


Zīm.2. Laboratorijas stenda principiālā elektriskā shēma (pa labi) in IC funkcionālā shēma (pa kreisi).

Dzināja vadību realizē ar pilno tiltu (*H-bridge, full bridge*), kuru vada electricimp (turpmāk vienkārši **imp**) mikrokontrolleris un nosako tilta darbību, kā arī realizējot darbības kļūdu (tilta pārslodzes) analīzi. Savukārt, **imp** vada ar WiFi bezvadu komunikācijas palīdzību, izmantojot interneta pārlūkprogrammu.

H-tilta funkcionālā shēma attēlo IC funkcijas. Tilta slēdžus veido MOSFET lauktranzistori, kuri ir šuntēti ar atpakaļdiodēm RL slodzes gadījumien.

Kļūdas signāls "0" IC2 (TLE5206) izejā EF parādās, ja strāva caur tilta slēdžiem pārsniedz 6A vai ir temparatūras pārslodze - skat. IC infomatīvo lapu.



Zīm.3. Līdzstrāvas motora vadības ekrānattēls - darbavirsma.

Interneta pārlūkprogramma attēlo stenda vadības-mērījumu aplikācijas izklājumu (layout). Ekrānattēls ir redzams Zīm3.

Aplikācija ir veidota izmantojot datubāzes programmu FileMaker un tās Instant Web Publishing (IWP) dzini un tās interneta adrese:

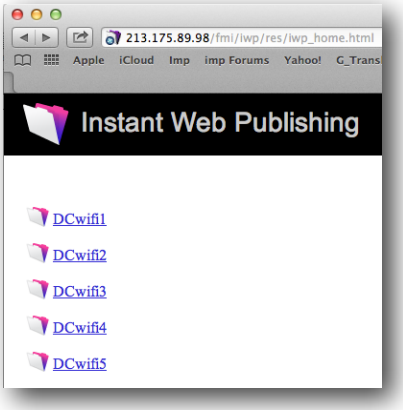
www.projectstore.rtu.lv (http://213.175.89.98/fmi/iwp/res/iwp_home.html)

Izvēlas atvērt failu, kura nosaukums atbilst nosaukumam uz stenda. Piemēram, ja stends ir DCwifi1, tad atver atbilstošo failu DCwifi1 (Zīm.4.).

Piekļuvei nepieciešamie:

Lietotājvārds (*Username*)- Student

Parole (*Password*)- etdv



Zīm.4. Atbilstošā faila izvēle, ekrānattēls.

Ar šo aplikāciju iespējams:

- iestatīt noteiktu apgriezienu skaitu. Apgriezieni ir 0, pēc barošanas sprieguma padošanas. Sekojoša apgriezienu iestatīšana ir attiecināma pret šo vērtību. Piemēram, ja pēc barošanas sprieguma pieslēgšanas nākamā vērtība ir 50, motors veic šo apgriezienu skaitu un apstājas, ja nākamā vērtība ir 30, tad dzinējs veic 50-30=20 apgriezienus pretējā virzienā.
- mainīt impulsa -platuma modulācijas (IPM) frekvenci - "PWM FREQ". Frekvencei ir fiksētas vērtības: 500, 750, 1000, 1250, 1500, 1750, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000 Hz, t.i. robežās 500...2000 Hz ar soli 250 Hz, robežās 2000...10000 Hz ar soli 1000 Hz.
- IPM sākuma vērtību, palaižot motoru. IPM sākuma vērtības "PWM START value" ir fiksētas: 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0, t.i. robežās 0...1 ar soli 0.05.
- mainīt apgriezienu skaitu pirms apstāšanās "TURNS till STOP", kuros IPM vērtība pakāpeniski samazinās - bremzēšanas apgriezienu skaitu.
- reģistrēt datus reģistrācijas failā (logfile).

4. Dzinēja darba algoritmns:

Dzinēja palaišanas un darbības algoritms ir sekojošs:

Sākot no iestatītās PWM START value, programma palielina PWM SATRT value par 0.05 ik 0.75 sekundes. Kad dzinējs ir veicis vienu apgriezienu (sācis rotēt), PWM vērtība palielinās par 0.05 ar katru apgriezienu. Kad dzinējs ir veicis TURNS - TURNS till STOP apgriezienu skaitu, sākas tā bremzēšana - PWM samazina par 0.1 ar katru nākošo apgriezienu.

Šeit ar "dzinējs" ir domāta dzinēja reduktora izejošā ass.

5. Darba uzdevumi:

Darbu uzsākot pievieno 12 V akumulatoru, ievērojot polaritāti. Darbu beidzot akumulatoru atvieno.

Ja nepieciešams restartēt **imp** - atvieno akumulatoru un pievieno pēc 5...10 sek.

Laiks starp komandas nosūtīšanu un **imp** reakciju var sasniegt dažas sekundes atkarībā no interneta pieslēguma datu apmiņas ātruma un noslodzes.

Iestatītās sākumvērtības (padodot **imp** barošanas spriegumu, t.i. pieslēdzot akumulatoru): PWMFREQ - 8000 Hz, PWM START value - 0, TURNS till STOP - 5.

Rezultātu aplūkošanai atjaunot interneta pārlūka logu, izmantot "Reload" .

1.uzdevums. Noteikt IPM minimālo frekvenci: PWM FREQ min.

Ja IPM frekvence ir zemāka par kādu noteiktu vērtību, motors nespēj uzsākt griešanos. Ja vērtība ir augsta, palielinās komutācijas zudumi tiltā.

TURNS - 20 vai 0, atkarībā no iepriekšējās vērtības.

PWM FREQ noteikt dažādām PWM START value vērtībām robežās 0.0 ... 0.35.

2.uzdevums. Noteikt IPM maksimālo sākuma vērtību.

Šī vērtība nosaka laiku, kurā dzināja rotācijas ātrums saniedz maksimālo (pwm = 1). Lielāka sākuma vērtība samazina šo laiku, bet var apgrūtināt dzinēja palaišanu.

PWM FREQ - iepriekš izvēlētā + 1000 Hz (tuvākā iespējamā iestatījuma vērtība).

3.uzdevums. Noteikt optimālo apgriezienu skaitu TURNS till STOP, lai pēdājais uzrādītais apgriezienu skait (turns) ir vienāds ar uzdoto.

Noteikt šo vērtību dažādām PWM START value vērtībām robežās 0.0 ... 1.0.

6. Secinājumi, rakstiski:

1. paskaidrot pilna tilta (H-tilta) darbību,
2. paskaidrot darbības alogoritmu - darbības un vadības loģiku,
3. pamatot darba gaitā izvēlētās vērtības - būtību.
4. pievienot reģistrācijas faila izdruku - copy no interneta pārlūka, paste teksta failā (paraksts, datums).

Mutiski - darba aizstāvēšana.

7. Programmas kods

Agent

```
//"Agent" nodrošina imp sasaisti ar internetu
// apstrādā pieprasījumu
http.onrequest(function(req, resp){
    if (req.method == "POST"){
// "izfiltrē" datus
        local body = http.urldecode(req.body)
        local value = (body.data);
//nosūta datus ierīcei
        device.send("need", value);
// pēc POST nosūta atbildi interneta pārlūkprogrammai
        resp.send(200, "OK " + value);
    }});
//saņem datus no ierīces
device.on("dc_val", function(s) {
```

//log faila servera adrese

```
local resp = http.post("http://xxx.xxx.xxx.xxx/imp/impDC/impDC2.php",
 {"Content-Type":"application/x-www-form-urlencoded"},
```

//no Agent datus pārsūta uz datu servera log failu

```
 http.urlencode({coord = s})).sendsync(); });
```

Device //"ierīce"

```
local turns = 0;
local value;
local coord = 0 ;
local coordSet = 0;
local pin1State;
local pin1StatePerv;
local pin2State;
local pin2StatePerv;
local pwm = 0.0;
local flag = 0;
local turnsON = 0;
local t=0;
local error = 1;
local errorPerv = 1;
local timeER;
sumstring <-"";
pwmstring <-"";
stringpwm <-"";
errorstring <-"";
local pwmfreq = 8000.0;
local pwmstart = 0.00;
local turnsTillStop = 5;
```

definē konstantes un maiņīgos

definē noklusējuma vērtības

//configurē imp PWM izvadus

```
hardware.pin7.configure(PWM_OUT, 1.0/pwmfreq, 0.0);
hardware.pin9.configure(PWM_OUT, 1.0/pwmfreq, 0.0);
```

// Funkcija **pin2change**- nosaka darbības, mainoties optiskā sensora stāvoklim. Šajā gadījumā - procedūras, kuras veic tikko karodziņš iziet ārpus optiskā sensora, izejā 0

```
function pin2change() {
    pin2State = hardware.pin2.read();
    if (pin2State == 0 && pin2State != pin2StatePerv) {
```

//nosaka kad būs 1 apgrieziens. Līdz 1 apgriezienam PWM pieaug pēc laika, tad pēc katra apgrieziena:

```
        ++turnsON;
// nosaka kad jāsākl bremzēt - samazināt PWM:
        if(math.abs(turns - coordSet) <= turnsTillStop) {
            if(flag == 1) {pwm = pwm - 0.1;}
            if(pwm <= pwmstart) pwm = pwmstart;}
        else if(pwm < 1.0) {pwm += 0.05;}
// neļauj PWM būt lielākam par 1:
        if(pwm >= 1.0) {pwm = 1.0}
// kad PWM =1, "uzliek karodziņu" - tas neļauj kļūdaini sākt bremzēt uzsākot rotāciju
        if(pwm == 1.0) {flag = 1;}
        if(turns < coordSet) { ++turns;}
        else if(turns > coordSet) {--turns;}
        function _tostring(turns);
        function _tostring(pwm);
//veido informācijas string, kuru nosūta uz log failu
        sumstring  = "turns " + turns + ", PWM " + pwm;
        server.log(sumstring);
//nosūta uz Agent un tas, savukārt, uz log file
        agent.send("dc_val",sumstring);}
    pin2StatePerv = pin2State;
}
//kinfigurē pin2 tā, ka funkcija pin2change tiek palaista pie katras optosensora stāvokļa izmaiņas
hardware.pin2.configure(DIGITAL_IN_PULLUP, pin2change);
```

```
// Funkcija pin8change- nosaka darbības, mainoties IC error flag EF stāvoklim
function pin8change() {
    if(math.abs(turns - coordSet) > 4) {
    error = hardware.pin8.read();
    if (error == 0 && error != errorPerv) {
        timeER = hardware.millis();
        function _tostring(pwmfreq);
        function _tostring(pwmstart);
        errorstring = "error"+"pwmFREQ: "+pwmfreq+", pwmSTART: "+pwmstart;
        server.log(errorstring);
        agent.send("dc_val",errorstring);
        pwm = pwmstart;
        if(turns < coordSet) {
            hardware.pin9.write(pwm);
            hardware.pin7.write(0.0);}
        else if(turns > coordSet) {
```

```
            hardware.pin7.write(pwm);
            hardware.pin9.write(0.0); }
        while(hardware.millis() - timeER < 1000) {;}
        turnsON = 0;
        flag=0;}
    errorPerv = error;}
}
hardware.pin8.configure(DIGITAL_IN_PULLUP, pin8change);
```

```
// Funkcija motor- dzinēja griešanās virzienu  un PWM pieaugumu/samazinājumu
function motor() {
// PWM pieaugums "pēc laika", kamēr nav bijis 1 apgrieziens
    if(turnsON < 2 && turns != coordSet) {
        if(hardware.millis() - t > 750.0) {
            pwm += 0.05;
            server.log("PWMstartup " + pwm);
            function _tostring(pwm);
            stringpwm = " ," + "PWMstartup " + pwm;
            agent.send("dc_val",stringpwm);
            t = hardware.millis();
            if(pwm >= 1.0) pwm = 1.0;}
    }
// nosaka nepieciešamo griešanās virzienu
    if(turns < coordSet) {
            hardware.pin9.write(0.0);
            hardware.pin7.write(pwm);}
    else if(turns > coordSet) {
            hardware.pin7.write(0.0);
            hardware.pin9.write(pwm); }
// dzinēja rotors ir īsslēgts, ja ir sasniegts vajadzīgais apgriezienu skaits
    else {
            hardware.pin7.write(0.0);
            hardware.pin9.write(0.0);
            turnsON = 0;
            pwm = pwmstart;
            flag = 0;}
// funkciju motor palaiž ik pēc 10 ms
    imp.wakeup(0.01, motor);
}
motor();
```

```
server.log("DC motor Started");
```

```
// saņem datus no Agent
agent.on("need", function(value) {
    t = hardware.millis();
// konfigurē PWM frekvenci (vērtība saņemta ar interneta starpniecību)
    if (value.slice(-1) == "f"){
        local getInteger = value.slice(0, -1).tointeger();
        if (getInteger != 0){
            pwmfreq = getInteger;
            hardware.pin7.configure(PWM_OUT, 1.0/pwmfreq, 0.0);
            hardware.pin9.configure(PWM_OUT, 1.0/pwmfreq, 0.0);
            local pwm7 = 1/hardware.pin7.getperiod();
            function _tostring(pwm7);
            local pwm9 = 1/hardware.pin9.getperiod();
            function _tostring(pwm9);
            pwmstring = "PWM_freq " + pwm7 + ", " + pwm9;
            server.log(pwmstring);
            return;}}
// konfigurē PWM sākuma vērtību (vērtība saņemta ar interneta starpniecību)
    else if (value.slice(-1) == "p"){
        local getFloat = value.slice(0, -1).tofloat();
            pwmstart = getFloat;
            pwm = pwmstart;
            server.log("pwmstart: " + pwmstart);}
// konfigurē apgriezienu skaitu pie kura jāuzsāk bremzēšana pirms apstāšanās (vērtība saņemta ar interneta starpniecību)
    else if (value.slice(-1) == "s"){
        local getInt = value.slice(0, -1).tointeger();
            turnsTillStop = getInt;
            server.log("turnsTillStop: " + turnsTillStop);}
    else {
        coordSet = value.tointeger();
        server.log("coordSet: " + coordSet);}
    motor();
});
```