Turun ammattikorkeakoulu

AVR ja ARM harjorustyö, Ohjelmoinnin jatkokurssi

Jarmo Kivekäs 1302928

23. huhtikuuta 2015

Sisältö

1 Johdanto **2**

2 Järjestelmä arkkitehtuuri **2**

3 Käyttöliittymä **2**

3.1 USB käyttöliittymälaitteiden tulkitseminen 2

4 Ohjaus teoria **2**

4.1 Kineettinen malli 3

5 Kommunikointi protokolla **3**

6 Moottorien ohjaaminen **4**

7 Oppitulosket **4**

7.1 Rekisteriosoittimien käsittely 4

7.2 Struct teitorakenteiden esitys muistissa 4

7.3 Puskuroimaton kirjoitus päätteelle 4

7.4 Käytä oikeaa datalehteä 5

1 Johdanto

Tämä raportti käsittelee robottia joka toteutettiin harjoitustyönä Ohjelmoinnin perusteet kurssia varten. Harjoitustyön tarkoituksen oli oppia järjestelmäläheistä ohjelmointia ARM sekä AVR ympäristäissä. Raportin sekä siinä esitetyn materiaalin on laatinut Jarmo Kivekäs.

Monet (jopa suurin osa) työssä toteutetuista asioista olisi voinut toteuttaa yksinkertaisemmin käyttämällä valmiita kirjastoja. Oppimistavoitteiden saavuttamiseksi on keskitytty järjestelmän toteuttamiseen alhaisella abstraktio tasolla käyttäen paljon järjestelmäkohtaista koodia, sen sijaan että oltaisiin pyritty toteuttamaan monipuolinen järjestelmä. Työssä on käytetty lähinnä avr-libc:n toimintoja, omia otsikkotiedostoja sekä joitain otsikkotiedostoja jotka kuuluvat vaikona linuxin `gcc` asennukseen.

Työssä panostettiin myös vahvaan matemaattisenn oikaoppisuuteen ja välttäen empiiristä 'mitoitusta' kokeilemalla arvoja esim. ajastimen kalibrointia varten yms.

2 Järjestelmä arkkitehtuuri

3 Käyttöliittymä

Käyttöliittymä robotin ojausta varten toteutettiin RaspberryPi -alustalla. Robotin ohjaus käyttöliittymän kautta tapahtuu kokonaan RPI:n liitetyn ulkoisen USB näppäimistön kautta.

3.1 USB käyttöliittymälaitteiden tulkitseminen

Näppäimistön kautta annetut käskyt luetaan erityisestä `/dev/input/eventX` tiedostosta. Näppäinpainalluksia tulkitseva moduuli on toteutettu tiedostossa `src/hid_input.c`.

Näppämistöä tulkistseva koodia suoritetaan omassa ohjelma säikeessään. Säikeen aloittaminen on toteutettu käyttämällä POSIX säikeitä.

Uuden säikeen aloittavalle `pthread_create()` funktiolle annetaan kolmanneksi argumentiksi funkio-osoitin funktioon joka halutaan suorittaa uudessa säikeessä. Neljäntenä argumenttina annetaan yksi `void *` -osoitin jonka kautta kaikki funktion argumentit annetaan. Tässä toteutuksessa argumenttina on pelkkä `void *`:ksi muutettu merkkijonon osoitin (Tiedosto josta USB näppämimistöä tulkitaan).

```
pthread_t polling_thread_id;
pthread_create(&polling_thread_id, NULL, hid_polling_loop, (void *)hid_device);
```

Ohjelman säikeiden välillä siiretään dataa yhteisessä muistissa olevan muuttujan kautta. Yksi säkeistä aina muokkaa dataa ja toinen aina lukee, joten ei tarvittu toteuttaa mutex-järjestelmään tiedon eheyden varmistamiseksi.

4 Ohjaus teoria

Robotin toteuttamista varten sovellettiin useaa eri mateemaattista mallinnusta robotin oletetusta käyttäytymisestä. Mallit phojautuvat hyvin tunnettuihun ohjausmalleihin joiden oikea toimivuus on näyhty käytännön sovelluksissa jo ennestään.

4.1 Kineettinen malli

Mekaanisesti robotti on rakennettu niin, että liikkuminen toteutetaan ensisijaisesti kahdella laitteen sivuilla sijaitsevan renkaan avulla. Robotti liikkuu käyttäen kahta ns. differentiaalista ajo-moottoria. Käytännössä tämä tarkoittaa sitä, että sivulla olevia moottoerita voidaan ohjeta toisitaan riippumatta. Robotti kääntyy kun renkaat pyörivät eri nopeutta suhteessa toisiinsa. Lisäksi robotilla on vapaasti liikkuvia tukipyöriä jotta se pytyy tasapainossa.

Robotin ohjaamista varten käytetty matemaattinen malli on seuraavanlainen:

$$\begin{cases} \dot{x} = \frac{R}{2}(v_r + v_l)\cos(\phi) \\ \dot{y} = \frac{R}{2}(v_r + v_l)\sin(\phi) \\ \dot{\phi} = \frac{R}{L}(v_r - v_l) \end{cases} \tag{4.1}$$

Robotin oletetaan kulkevan tasaisen tason pinnalla. x ja y kuvaavat laitteen paikkaa tasolla, ϕ robotin etuosan osittamaa suuntaan, v_r ja v_l ovat oikean sekä vasemmanpuolisen renkaan pyörimisnopeudet. Lisäksi mallissa esiintyy vakio R joka on ohjaukseen käytettyjen renkaiden säde. Vakio L on renkaiden etäisyys toisistaan.

—

Yllä esitelty malli toimii hyvin ohjausta varten. Ohjausalgorimejä kehittäessä päädyttiin kuitenkin siihen tulokseen, että järjestelmää kannattaa käsitellä yksinkertaisemalla mallilla. Robotin liikkeitä on hankala hahmottaa ajattelemalla pelkästään renkaiden pyörimisnopeutta.

Ohjausalgoritmien kehittämistä varten käytetty matemaattinen malli on seuraavanlainen:

$$\begin{cases} \dot{x} = v\cos(\phi) \\ \dot{y} = v\sin(\phi) \\ \dot{\phi} = \omega \end{cases} \tag{4.2}$$

Mallin avulla voidaan määrittää robotin liikke käyttäen hyväksi pelkästään sen translaatio nopeutta v sekä kulmanopeutta ω.

Soveltamalla malleja (4.1) ja (4.2) saadaan robotin translaation sekä rotaation (v ja ω) ja renkaiden pyörimisnopeuksien (v_r ja v_l) välille korrelaatio joka on helppo toteutta C-kielellä:

$$\begin{cases} v &= \frac{R}{2}(v_r + v_l) \\ \omega &= \frac{R}{L}(v_r - v_l) \end{cases} \qquad \begin{cases} v_r &= \frac{2v + \omega L}{2R} \\ v_l &= \frac{2v - \omega L}{2R} \end{cases} \tag{4.3}$$

5 Kommunikointi protokolla

Sarjaväylän kommunikaatio toteutettiin AVRllä käyttäen mikro-ohjaimen tuottamaa ISR -funktiokutsua joka tapahtuu aina kun sarjavälälle on saapunut tavu dataa. `printf()` -perheen funktioiden vaatima `stdio` puskuri toteutettiin myös, mutta funktioita ei kuitenkaan päädytty käyttämään suuren ROM vaatimuksien takia (yli 1000 tavua pelkän `printf()` käyttämistä varten).

Robotti tallentaa viimeksi saadut komennot, joten uusi komentoja tarvitsee lähettää sarjaväylän yli vain sillon kun niihin on tapahtunut muutos.

Lähetetty viesti koostuu yhden tavun pituisesta alkumerkistä, jonka jälkeen tulee kaksi `uint16_t` arvoa. Arvot kertovat kuinka nopeasti renkaiden kuulu pyöriä (kuinka monta ajastimen aikayksikköä jokaisen askelmoottorin askeleen välillä).

6 Moottorien ohjaaminen

Moottorieden ohjausta varten käytetään 16-bittistä ajastinta. Ajastimen esijakaja on asetttu olemaan 1024 ja mikro-ohjaimen kellotaajuus on 16 MHz. Tästä saadaan että kello jonka tarkkuus (yhden aikayksikön pituus) on 1024/16 MHz = $64\mu s$. Ajastimen laskin kierähtää takaisin nollaan $2^{16} * 64\mu s \approx 4.2s$ välein.

7 Oppitulosket

Projektin tarkoituksena oli syventyä järjestelmäläheiseen C-kilen ohjelmointiin AVR sekä ARM prosessoriarkkitehtuureilla ja yelisemmin kielen eri ominaisuuksiin.

USB käyttöliitymälaitteiden raa'an datan tulkistemista varten ei löytynyt kovin perusteellista oppimateriaalia. Suuri osa työskently- ja oppimisprosessia koostui eri asiaankuuluvien C otsaketiedostojen lukemisesta, koska tietoa ei suurikaan muualta löytynyt helposti.

7.1 Rekisteriosoittimien käsittely

Muita kiinnostavia asioita oli esimerkiski keskiä millä datatyypillä AVR:n erikoistoimintorekisterit (SFR) niin kuin `PORTB` ovat esitetty. Esitytavan tunteminen oli olennaista jotta pystyi kirjoittamaan tehokkaita funktioita joille annetaan I/O -nasta argumenttina. Arduino/wiring kirjastoja käyttäessä kuluu paljon laskutehoa siihen, että siirrytään Arduinon nastojen numeroinnista AVR yhteensopivaan esitystapaan. Käsittelemällä rekisterejen osoittajia saadaan huomattavasti nopeammin suoritettavaa koodia.

Esitystapaa tuli käytettyä esim stepperimoottoreiden määrittelyssä:

```
struct stepper_state_machine {
    uint8_t phaseA_pin;
    uint8_t phaseB_pin;
    volatile uint8_t *phaseA_port;
    volatile uint8_t *phaseB_port;
    char state;
};
```

7.2 Struct teitorakenteiden esitys muistissa

`Struct` tietorakentiden vertaileminen eri arkkitehtuureilla oli myös kiinnostavaa. RPI:n 32-bittin ARM arkkitehtuuri pyrkii asettamaan datan muistiosotteisiin jotka ovat neljällä jaollisia. 8-bittinen AVR ei kuitenkaan muistiosoitteien jaollisuudesta välitä. Tämä johtaa siihen, että structiin pakattua dataa ei voida turvallisesti siirtää suoraan ARM arkkitehtuurilta AVR arkkitehtuurille ilman että erikseen vamistetaan että tietorakenteiden esitys todella on sama molemmilla arkkitehtuuriella. Kirjoittaessa koodia jota on tarkoitus suorittaa molemilla arkkitehtuureilla pitää pitää mielessä että myös datatyyppien koot eroavat toisistaan. AVR:n `int` muuttuja on 16-bittinen, kun 32-bittisen ARM:n datatyyppi on 32-bittinen. Yleispätevää koodia saa kuitenkin kätevästi kirjoitettua käyttämällä `<inttypes.h>` määrittämiä datapyyppejä kuten `uint8_t` ja `int32_t`

7.3 Puskuroimaton kirjoitus päätteelle

Virheen korjaus viestejä kannataa tulostaa päätteellä käyttämällä `fprintf(stderr, ...)` tavallisen `printf()` sijasta, varsinkin kun työstää aika-kriittistä koodia. Puskuroidun tulostuksen sivuvaikutus tuli esiin, kun ohjelma RPIllä suoritettava ohjelma käyttäytyi eri tavall riippuen siitä, että suoritettiinko se itsenäisesti vai `strace` analysoimana.

7.4 Käytä oikeaa datalehteä

Ojelmoitaessa PWM lähtöä (ei käytössä lopullisessa järjestlmässä) moottoreiden ohjaamista varten oli vahingossa ATmega8 datalehti käytössä ATmega328 datalehden sijaan. Mikrokontrollert ovat melkein samanlaiset, mutta niissä ei kuitenkaan ole identtiset ajastimet, eikä koodi toiminut.