

AVR ja ARM harjorustyö, Ohjelmoinnin jatkokurssi

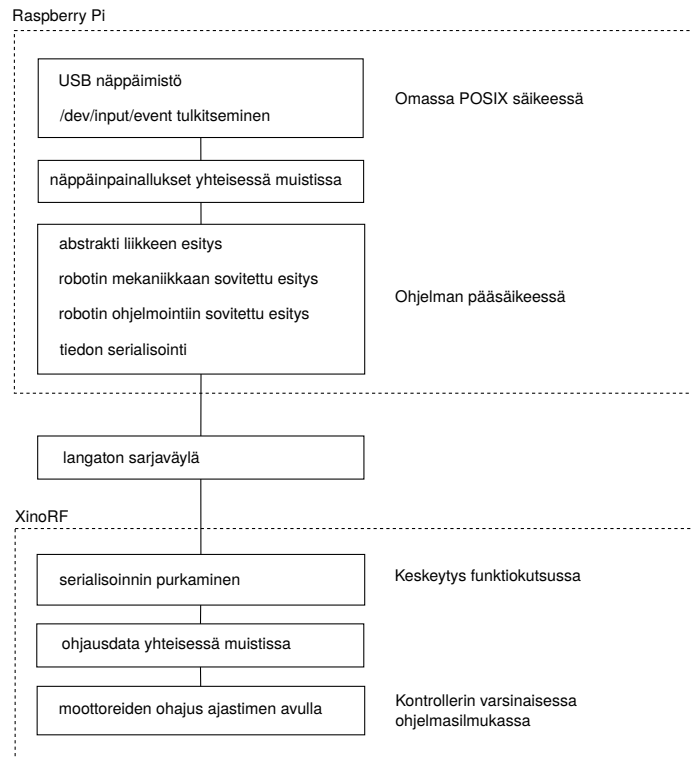
Jarmo Kivekäs

25. huhtikuuta 2015

Sisältö

1 Johdanto	2
2 Järjestelmän kuvaus	2
2.1 Käyttöliittymä	2
2.1.1 USB käyttöliittymälaitteiden tulkitseminen	2
2.2 Kineettinen malli	3
2.3 Kommunikointi protokolla	4
2.3.1 Viesin rakenne	4
2.4 Moottorien ohjaaminen:	5
3 Oppitulosket	5
3.1 Rekisteriosoitimien käsittely	5
3.2 Struct teitorakenteiden esitys muistissa	5
3.3 Puskuroimaton kirjoitus päätteelle	5
3.4 Käytä oikeaa datalehteä	6

Kuva 2.1: Järjestelmän arkkitehtuur



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 vaikkona käytettyyn linuxin gcc asennukseen.

Työssä panostettiin myös vahvaan matemaattiseen oikaoppisuuteen välttämällä empiiristä 'mitoitusta' kokeilemalla arvoja esim. ajastimen esijakajan kalibrointia varten yms.

2 Järjestelmän kuvaus

2.1 Käyttöliittymä

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

2.1.1 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 funktio-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 näppäimistöä tulkitaan).

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

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

2.2 Kineettinen malli

Robotin toteuttamista varten sovellettiin useaa eri mateemaattista mallinnusta robotin oletetusta käyttäytymisestä. Mallit pohjautuvat hyvin tunnettuihin ohjausmalleihin joiden oikea toimivuus on nähty käytännön soveluksissa jo ennestään.

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:

$$\begin{cases} \dot{x} = \frac{R}{2}(v_r + v_l) \cos(\phi) \\ \dot{y} = \frac{R}{2}(v_r + v_l) \sin(\phi) \\ \dot{\phi} = \frac{R}{L}(v_r - v_l) \end{cases} \quad (2.1)$$

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. Ohjausalgoritmejä kehittäessä päädyttiin kuitenkin siihen tulokseen, että järjestelmää kannattaa käsitellä yksinkertaisemalla mallilla. Robotin liikkeitä on hankala hahmottaa ajatteleamalla pelkästään renkaiden pyörimisnopeutta.

Ohjausalgoritmien kehittämistä varten käytetty matemaattinen malli on seuraavanlainen:

$$\begin{cases} \dot{x} = v \cos(\phi) \\ \dot{y} = v \sin(\phi) \\ \dot{\phi} = \omega \end{cases} \quad (2.2)$$

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

Soveltamalla malleja (2.1) ja (2.2) saadaan robotin translaation sekä rotaation (v ja ω) ja renkaiden pyörimisnopeuksien (v_r ja v_l) välille korrelaatio joka on helppo toteuttaa C-kielellä:

$$\begin{cases} v &= \frac{R}{2}(v_r + v_l) \\ \omega &= \frac{R}{L}(v_r - v_l) \end{cases} \quad \begin{cases} v_r &= \frac{2v + \omega L}{2R} \\ v_l &= \frac{2v - \omega L}{2R} \end{cases} \quad (2.3)$$

2.3 Kommunikointi protokolla

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

Robotti tallentaa viimeksi saadut komennot muistiin, joten uusi komentoja tarvitsee lähettää sarjaväylän yli vain silloin kun niihin on tapahtunut muutos. Näin vältetään toistuvilta samanlaisilta viesteiltä.

2.3.1 Viesin rakenne

Vestit ovat kiinteän mittaisia, ja sisältävät aina samat kentät. Viesti voidaan esittää C-kiellä seuraavanlaisella tietorakenteella:

```
struct command_message {
    uint16_t id;
    uint16_t tick_period_right;
    uint16_t tick_period_left;
    char direction;
    char checksum;
} __attribute__((packed));
```

Nimi	start	id	right_tick_period	left_tick_period	direction	checksum
koko	8 bit	16 bit	16 bit	16 bit	8 bit	8 bit

start: kiinteästi määritelty bittikuvio 0x61 joka vastaa ascii merkkiä 'a'. Merkin tarkoitus on välttää sitä, että vastaanottaja yrittää tulkita kaikkea sarjaväylään saapuvaa dataa. Johtuen siitä, että data tulee radion kautta, 'turhaa' dataa joka on tarkoitettu jonkin muun laitteen vastaanotettavaksi saapuu melko paljon. Viestiä ei ruveta purkamaan ennen kuin merkki on vastaanotettu.

id: 16 bitin pituinen laitekohtainen tunnistuskoodi. Koodin tehtävänä on mahdollistaa usean samanlaisen robotin ohjaamisen samalla alueella, ilman että järjestelmien viestiliikenteet häiritsevät toisiaan. Koodi on tarkoitus tallentaa robotin pitkäaikaiseen EEPROM muitsiin jotta se säilyy vaikka laite sammutetaan (toistaiseksi toiminto on viellä toteuttamatta).

right- sekä left_tick_period: kertoo kuinka monen ajatimen aikayksikön välein robotin askelmoottoriden tulee ottaa askel. Tarkemmin kohdassa 2.4.

direction: 8 bitin pituinen bittijono, josta alimmat kaksi bittiä on käytössä.

2.4 Moottorien ohjaaminen:

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

3 Oppitulosket

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

USB käyttöliitymlaitteiden raavan datan tulkistamista varten ei löytynyt kovin perusteellista oppimateriaalia. Suuri osa työskentely- ja oppimisprosessia koostui eri asiaankuuluvien C otsaketiedostojen lukemisesta, koska tietoa ei suurikaan muualta löytynyt helposti.

3.1 Rekisteriosoittimien käsittely

Muita kiinnostavia asioita oli esimerkiksi 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;
};
```

3.2 Struct teitorakenteiden esitys muistissa

Struct tietorakentien 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 muistiosoitteiden jaollisuudesta välitä. Tämä johtaa siihen, että structiin pakattua dataa ei voida turvallisesti siirtää suoraan ARM arkkitehtuurilta AVR arkkitehtuurille ilman että erikseen vamiistetaan että tietorakenteiden esitys todella on sama molemmilla arkkitehtuureilla. Kirjoittaessa koodia jota on tarkoitus suorittaa molemmilla arkkitehtuureilla pitää pitää mielessä että myös datatyyppejen 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 datatyypppejä kuten `uint8_t` ja `int32_t`

3.3 Puskuroimaton kirjoitus päätteelle

Virheen korjaus viestejä kannattaa 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 suoritettiin se itsenäisesti vai `strace` analysoimana.

3.4 Käytä oikeaa datalehteä

Ojelmoitaessa PWM lähtöä (ei käytössä lopullisessa järjestelmässä) 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.