Tallinna Tehnikaülikool
Automaatikainstituut

Taivo Lints
990849LAS

Mootori juhtimine PC-ga

Aruanne õppeaines "Automaatjuhtimissüsteemid – projekt" (LAS3722)
Õppejõud: Andres Rähni

2003. aprill

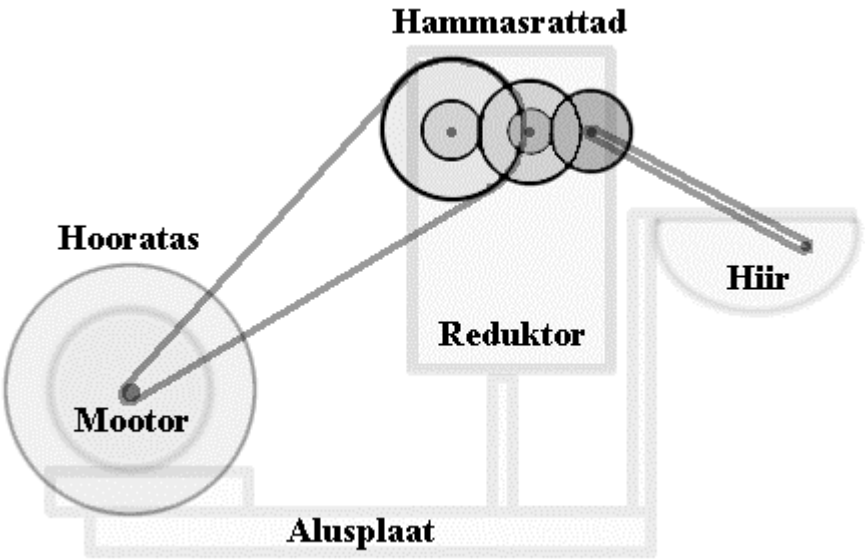
Eesmärgid

Käesoleva töö üldisemaks eesmärgiks oli omandada kogemus mingi reaalse masina ühendamiseks personaalarvutiga ja tema juhtimiseks enda poolt kirjutatud programmi abil. Selles töös oli masinaks elektrimootor ning juhtimisülesandeks mootori pöörete hoidmine etteantud väärtuse ümbruses.

Esimene etapp – juhitava seadme konstrueerimine

See etapp omab antud töös kõige suuremat tähtsust ning võttis seetõttu ka põhiosa töö tegemise ajast. Minu jaoks tulenes tema tähtsus sellest, et ma ei ole varem kunagi midagi isetehtut arvuti külge ühendanud ja seetõttu igasugune eelnev sellealane kogemus puudus.

Esmalt tuli välja mõelda seadme konstruktsioon. Esialgne idee oli panna mootori külge üks ketas, milles on auk, mis valgusdioodi ja anduri vahelt möödudes tekitab iga pöörde ajal ühe impulsi, mida seejärel on võimalik mõnest arvuti pordist sisse lugeda, et pöörlemiskiirust teada saada. Aga kuna tekkis kahtlus, et selle sisselugemise programmeerimine võib osutuda liiga keeruliseks (ühtegi impulssi ei tohiks vahele jätta, aga mootor pöörleb üsna kiiresti), siis otsustasin veidi lihtsama variandi kasuks. Nimelt kasutasin pöörlemiskiiruse kohta info saamiseks tavalist arvutihiirt, kuna hiirelt tuleva info lugemiseks vajalikud vahendid on vabalt kättesaadavad. Töö jaoks loodud maketi põhimõtteskeem on toodud järgneval joonisel.



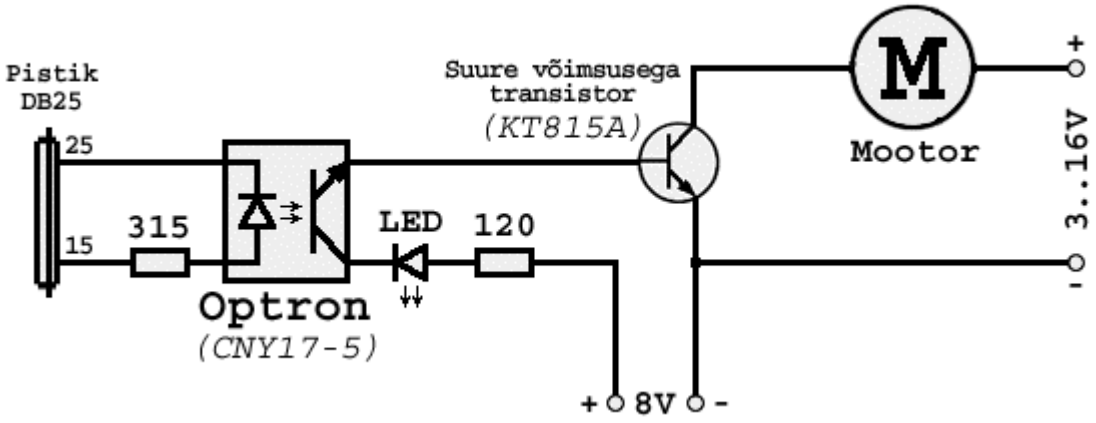
Joonis 1. Loodud seadme tööpõhimõte

Alusplaat ja tugistruktuurid on ehitatud ühe vana vene konstruktorikomplekti detailidest. Mootoriks on tavalisest mikromootorist veidi võimsam elektrimootor, mis pärineb elektrilise kirjutusmasina seest. Mootori teljele on kinnitatud hooratas (saadud magnetofonist), et pöörlemine oleks veidi ühtlasem. Mootori poolt tekitatud pöörlemine kantakse rõngaskummi abil üle reduktorile, mille ülesandeks on vähendada pöörlemiskiirus hiirele vastuvõetavale tasemele. Reduktoriks on kassettpleieri ülekandemehhanism (mille originaalülesandeks oli pleieri mootori kiire pöörlemise teisendamine aeglaseks aga tugevaks

lindivedaja pöörlemiseks). Ja lõpuks kantakse aeglustunud pöörlemine reduktorilt jällegi rõngaskummi abil hiire sees asuvale rullikule (antud juhul sellele, mis edastab hiire külgliikumist e. x-telge).

Järgmisena oli vaja luua elektriskeem, mille abil arvutiprogrammi poolt mõnda porti kirjutatav juhtinfo jõuaks mootorini. Esmalt tuli otsustada, mis põhimõttel juhtimine üldse toimuma hakkab – kas kasutada näiteks digitaal-analoogmuundurit või hoopis pulsilaiusmodulatsiooni *(pulse-width modulation)*. Valisin pulsilaiusmodulatsiooni, sest selle jaoks tarvilikku elektriskeemi on ilmselt lihtsam realiseerida. Tõsi küll, miinuspooleks on mootori töö suurem ebaühtlus (pulseerimine).

Elektriskeemi ehitamine oli üsna pikk protsess. Esiteks ei ole mul eriti suuri elektroonikaalaseid teadmisi ja teiseks piiras valikuvõimalusi komponentide kättesaadavus. Sellest tulenevalt seisnes skeemi ehitamine arvukate katsete tegemises erinevate ühendusviiside ja erinevate komponentidega (millest enamuse hankisin oma keldrist lammutatud elektroonikaseadmete trükkplaatidelt). Lõpptulemusena saadud kasutuskõlblik elektriskeem on toodud järgneval joonisel.



Joonis 2. Elektriskeem

Kasutatavaks arvutipordiks valisin paralleelpordi, sest selle kontaktide olekuid on lihtne muuta. Oma skeemi ühendamiseks pordiga kasutan *Parallel Laplink* kaablit (mida joonisel 2 ei ole näidatud). Selle kaabli täielikku infokaarti ei hakka siin välja tooma, aga antud skeemi jaoks vajalik info on järgmine: 2 • 15; 25 • 25. See tähendab siis seda, et paralleelpordi kontakt nr. 2 (*Data 0*, mille olekut minu programm muudab) jõuab kaabli teises otsas pistikukontaktile nr. 15; ja pordi kontakt nr. 25 *(Ground)* on ka teises otsas kontaktil nr. 25.

Niisiis on minu skeemil olemas pistik DB25 *(female)*, ehk 25 kontaktiga pistikupesa (vt. joonis 2, vasakul), mille külge käib *Parallel Laplink* kaabel.

Kuna kõige olulisem nõue, mille oma skeemile seadsin, oli "mitte rikkuda arvutiporti" (sest ma ei kasutanud eraldi odavat sisend-väljundkaarti, vaid otse emaplaadil asuvat paralleelporti ja selle riknemine oleks olnud küllaltki kulukas), siis järgmiseks komponendiks on optron, mis täielikult isoleerib kogu ülejäänud skeemi arvutipordist. Seejuures tuleb muidugi silmas pidada, et ka juhtmete jootekohad peavad olema kaetud isoleeriva materjaliga – vastasel korral võib optronist hoolimata tekkida elektriline ühendus välisskeemi ja arvuti sisendite vahel. Samuti peab jälgima, et optron ise porti läbi ei laseks.

Selle ärahoidmiseks lisasin takistuse 315 Ù,mis hoiab voolutugevuse ca.5 mA juures, mida minu port peaks normaalselt välja kannatama. Aga kuna erinevate arvutite paralleelpordid on erinevad (näiteid erinevatelt infolehtedelt: *Sink/Source 6mA, Source 12mA/Sink 20mA, Sink 16mA/Source 4mA, Sink/Source 12mA*), siis tuleb alati ettevaatlik olla ja valida konkreetsele pordile sobivad komponendid.

Joonise 2 keskmisel kolmandikul asub kahte äärmist kolmandikku vahendav lülitusahel, millel on eraldi toide ja mis sisaldab ka ühte valgusdioodi, et oleks võimalik visuaalselt jälgida, mis skeemis toimub (LED põleb, kui lülitusahelas on vool, st. "lüliti" on "sees"). Voolutugevuse piiramiseks on jällegi lisatud üks takisti, antud juhul 120 Ù.Lülitusahela toide on 8 volti lihtsalt sellepärast, et mul oli üks adapter, mis väljastas 8 volti. Tõenäoliselt oleks saanud skeemi teha ka selliselt, et lülitusahelal eraldi toide puudub – kasutatakse ära mootori toide – kuid antud juhul valisin eraldi toitega variandi, kuna selle tööd oli lihtsalt kergem ette kujutada (mootori toidet kasutades oleks lülitusahela toitepinge väärtus olnud kõikuv – sõltuv niimootoritöörežiimist kuikamootoritoiteallikast, mis on muudetava väljundpingega. Kuna aga lülituselemendiks on transistor, mitte relee, siis oleks need kõikumised omakorda mõjutanud mootori toidet... Minu elektroonikateadmised ei ole eriti sügavad, seetõttu valisingi esialgu mõistetavama variandi.).

Mootori lülituselemendiks, nagu juba öeldud, on transistor. Arvestades sellega, et mootor tarbib üpris suuri voole (eriti sisselülitamisel ja pulseerimisel), peab ka transistor olema piisavalt võimas. Igaks juhuks kruvisin sellele transistorile veel ka jahutusradiaatori külge – võimsatel transistoridel on selle jaoks kruviauk olemas.

Mootori toiteallikas (joonisel 2 paremal) on reguleeritav, antud juhul vahemikus ca. 3..16 volti. See võimaldab tekitada häireid, et kindlaks teha süsteemi suutlikus nende häiretega toime tulla.

Teine etapp – programmi kirjutamine

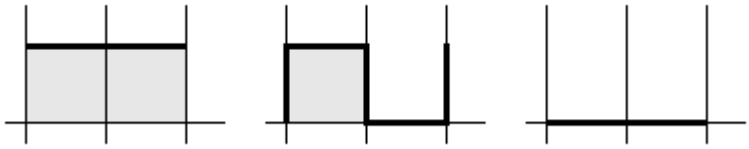
Programmeerimiskeeleks valisin C++, kuna valdan seda keelt teistest keeltest rohkem ja soovin oma oskusi veelgi täiendada. Pealegi on C/C++ antud töö jaoks igati sobiv – võimaldab teha kiiresti töötavat programmi ning on üleüldse üsna madala (riistvaralähedase) taseme keel. Tõsi küll, antud programm oli nii väike, et teda ei olnud mõtet kirjutada objektorienteeritult, vaid pigem tavalise C stiilis. Klaviatuuri, hiire, taimerite jms. jaoks kasutasin Allegro't *(game programming library by Shawn Hargreaves)*, mis on *giftware*, st. teda võib vabalt mistahes eesmärgil kasutada. Programmikood on tervikuna toodud käesoleva aruande lisas. Järgnevalt on lühidalt kirjeldatud tema tööpõhimõtet.

Juhtimisalgoritmi enda põhimõte on äärmiselt lihtne. Võrreldakse kasutaja poolt etteantud seadesuurust (mitu pööret minutis peaks mootor tegema) tegeliku pöörlemiskiirusega, mida arvutatakse hiirelt saadud info põhjal. Kui tegelik on väiksem, suurendatakse mootori seesolekuaega ja samavõrra vähendatakse väljasoleku aega, nii et lülitamise sagedus ei muutu (standartne pulsilaiusmodulatsiooni põhimõte). Kui tegelik kiirus on suurem, toimitakse vastupidiselt. Sees / väljas suhet muudetakse alati sama ühiku võrra, mitte proportsionaalselt veaga. See teeb algoritmi reaktsiooni küll aeglasemaks, aga kuna hiirelt saadav info reaalse pöörlemiskiiruse kohta on VÄGA hüplik (nagu järgnevatelt graafikutelt võib näha), põhjustaks proportsionaalse muutuse kasutamine ka juhtsignaali suurt kõikumist, mis ei oleks hea.

Kõikvõimalikud parameetrid nagu sisendinfo töötlemise sagedus, väljundimpulsside sagedus, sees / väljas suhte muutmise kiirus jms. määrasin kindlaks katsetamise teel – vaadeldes, kuidas nende muutused süsteemi käitumist mõjutavad.

Sisendinfo töötlemise sagedus: liiga väike sagedus tähendab vana informatsiooni kasutamist ning põhjustab pöörlemiskiiruse võnkumaminekut; liiga suur sagedus tekitab valeandmeid – hiire rullik ei pruugi olla jõudnud piisavalt pöörduda ning programm arvab, et pöörlemiskiirus on null.

Väljundimpulsside sagedus: liiga väike sagedus muudab mootori töö väga ebaühtlaseks (töötab, ei tööta, töötab, ...); liiga suur sagedus piirab oluliselt erinevate sees / väljas suhete arvu. Nimelt on programmis ära määratud väikseim ajaühik (antud juhul üks millisekund), mille jooksul väljund ei saa muutuda. Seetõttu näiteks väljundimpulsside 2 millisekundise perioodi korral on suhtel vaid kolm erinevat võimalust: 2 / 0, 1 / 1 ja 0 / 2:



Joonis 3. Diskreetse aja mõju.

Tegelikult on see minimaalne ajaühik reaalselt võib-olla isegi suurem kui 1 ms, tulenevalt arvuti piiratud töökiirusest.

Sees / väljas suhte muutmise kiirus: liiga suur muutmiskiirus põhjustab ebaühtlast tööd, sest sisendinfot loetakse küllaltki harva ning ebatäpselt ja seetõttu ei ole mõttekas selle mitte eriti täpse info järgi kogu aeg suuri muutusi teha; liiga väike muutmiskiirus teeb juhtimissüsteemi reaktsiooni häiretele või seadesuuruse muutmisele liiga aeglaseks.

Kuna lisas toodud programmikoodi juurde kirjutasin samuti üsna palju kommentaare, siis selles peatükis koodist rohkem ei räägi. Illustratsiooni mõttes on järgnevalt veel ära toodud programmi ekraanipilt – prinditakse mõningate muutujate hetkeväärtusi, mis parajasti programmi töö jälgimiseks olulised tundusid.

```
mouse_check = 1
mx = -28
rps = 15.677491
rpm = 940.649475
activity_ratio = 0.398669




target_rpm = 1000
```

Joonis 4. Ekraanipilt (värvid muudetud printimise jaoks sobivamaks).

Kolmas etapp – katsete tegemine koos tulemuste registreerimisega

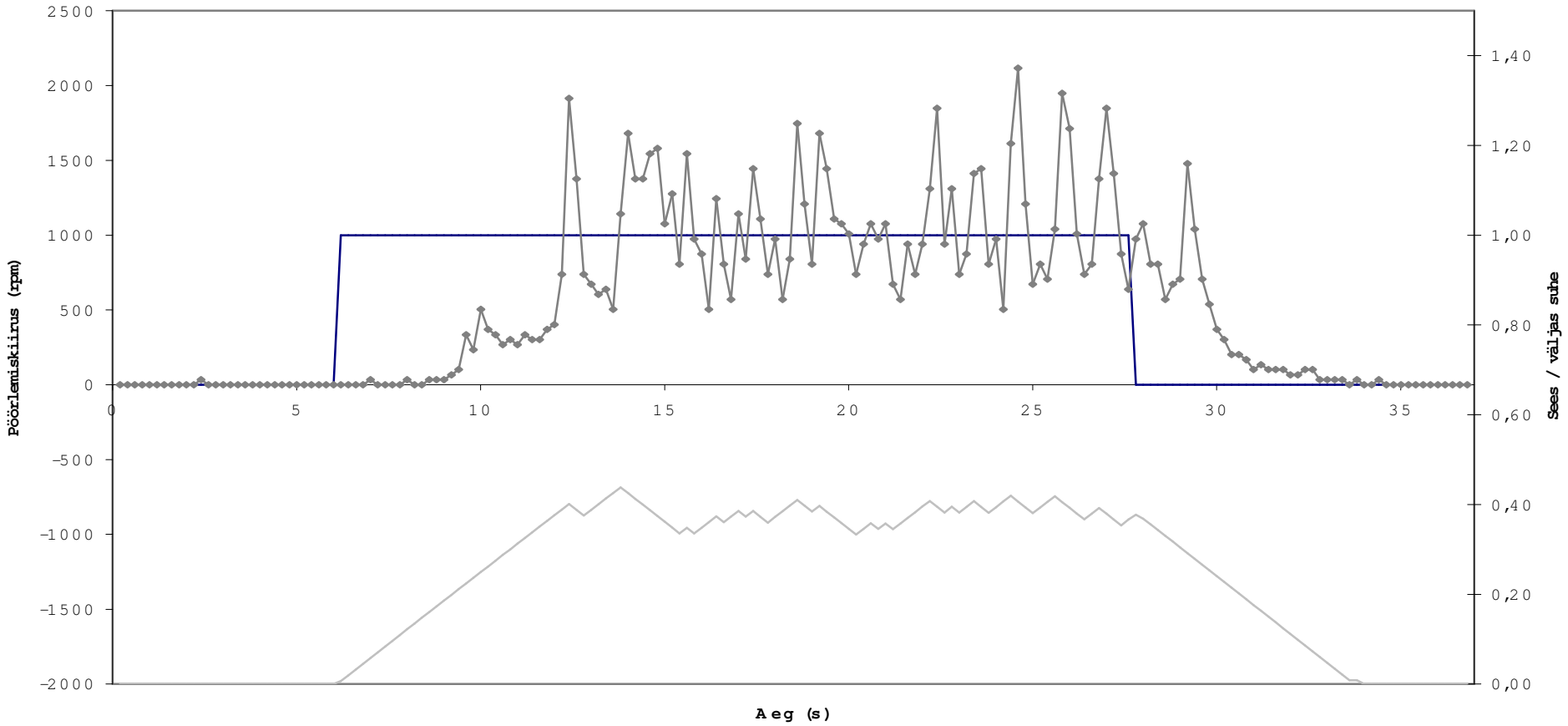
Katsetulemuste registreerimise eesmärgil väljastab programm mõningaid parameetreid tekstifaili, mis näeb välja selline:

```
bps = 5
timestep = 1
activity_ratio_change = 2e-08
cycle = 75

global_time  target_rpm  rpm  activity_ratio

205  0  0  0
405  0  0  0
605  0  0  0
805  0  0  0
1005  0  0  0
1205  1000  0  1.22e-06
1405  1000  0  0.00549992
1605  1000  0  0.0167797
1805  1000  0  0.0280306
2005  1000  0  0.0393176
2205  1000  0  0.0505892
2405  1000  0  0.0618976
2605  1000  0  0.0731791
```

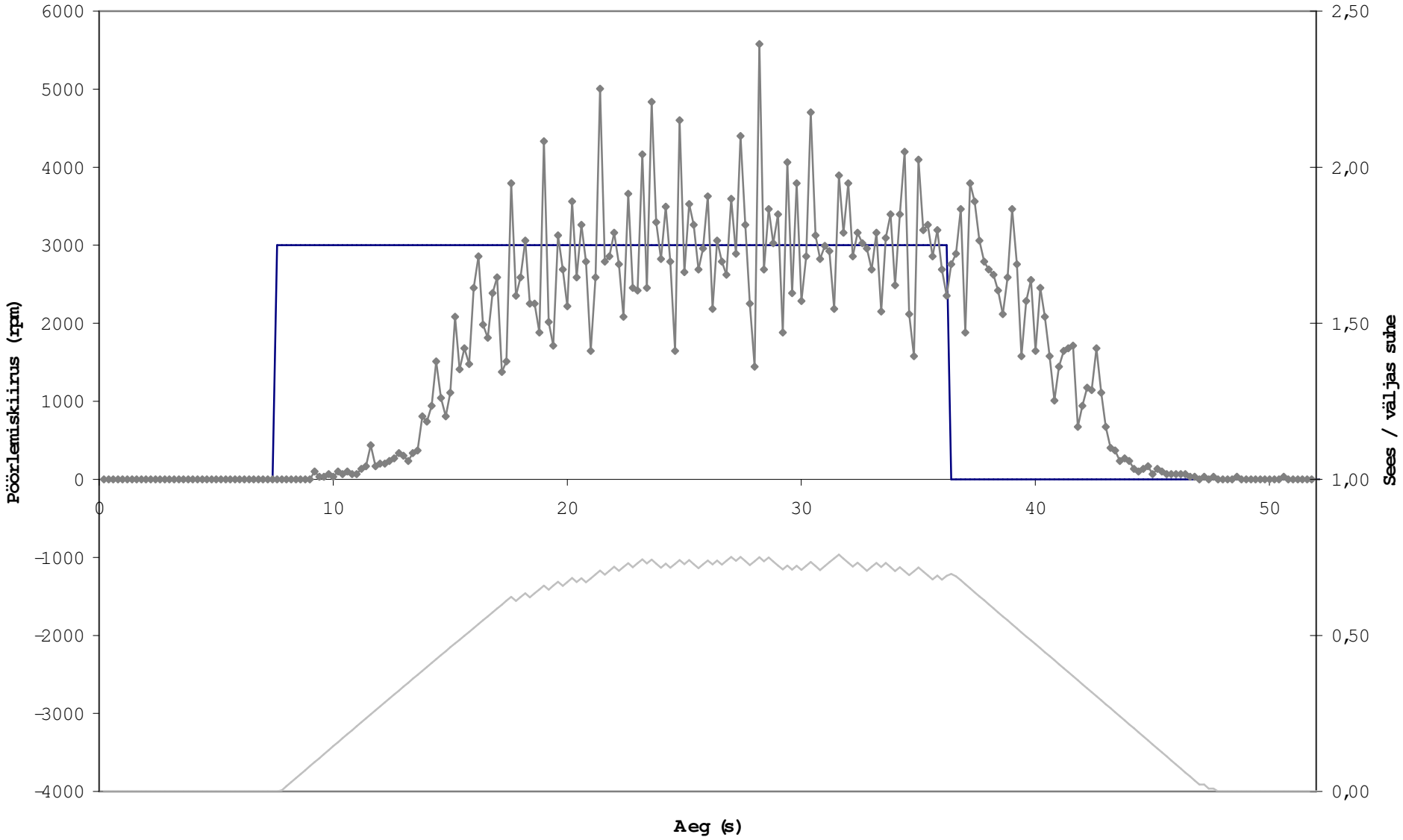
Sellest saab kerge vaevaga näiteks Excel'i abil graafikuid valmistada. Järgnevas ongi sooritatud mõningad katsed ja nende tulemus visualiseeritud. Kõikvõimalikud programmi parameetrid olid katsete käigus samad, mis lisas toodud programmikoodiski. Mootori toitepinge oli 10V.



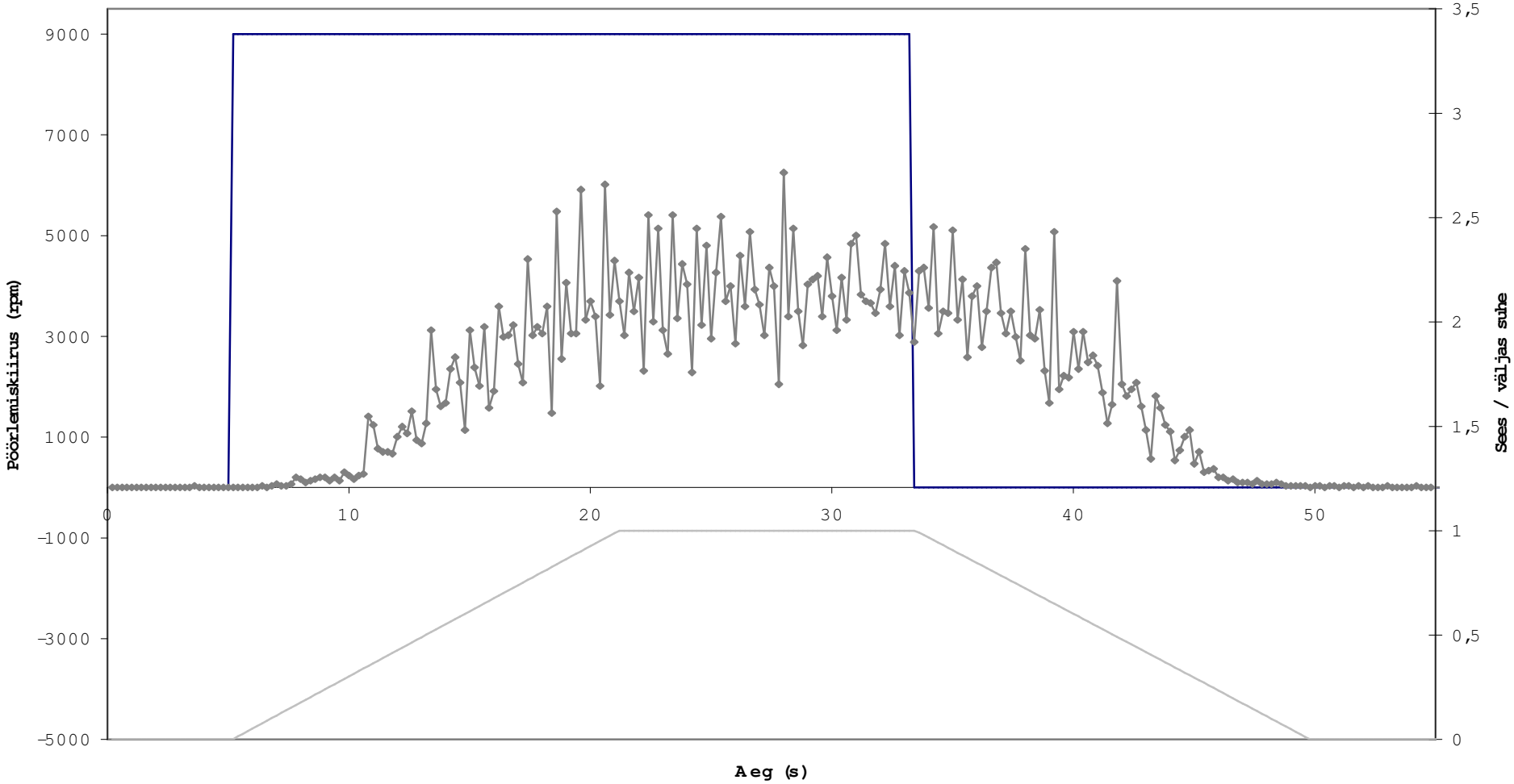
Graafik 1. Seadesuuruse hüpe 0 • 1000 • 0.

Ülemisel graafikul: sirge joon on seadesuurus, hüplev joon on sisendist loetud info põhjal leitud pöörlemiskiirus. Tegelikult mootori pöörlemine muidugi nii ebaühtlane ei olnud – suurem osa müraga sarnanevast veast tuleb pöörlemiskiiruse määramise süsteemi ebatäpsustest: mehaaniliste ülekannete ebaühtlus, hiireinfo lugemise ajastamisprobleemid jms.

Alumisel graafikul: porti saadetavat pulsilaiusmodulatsiooni iseloomustav sees / väljas suhe ("sees" = väljund aktiivne). Loomulikult saab see suurus olla ainult vahemikus 0..1, aga graafiku kahte ossa jaotamise nimel tuli parempoolsele teljele lisada ka ühest suuremaid väärtusi.

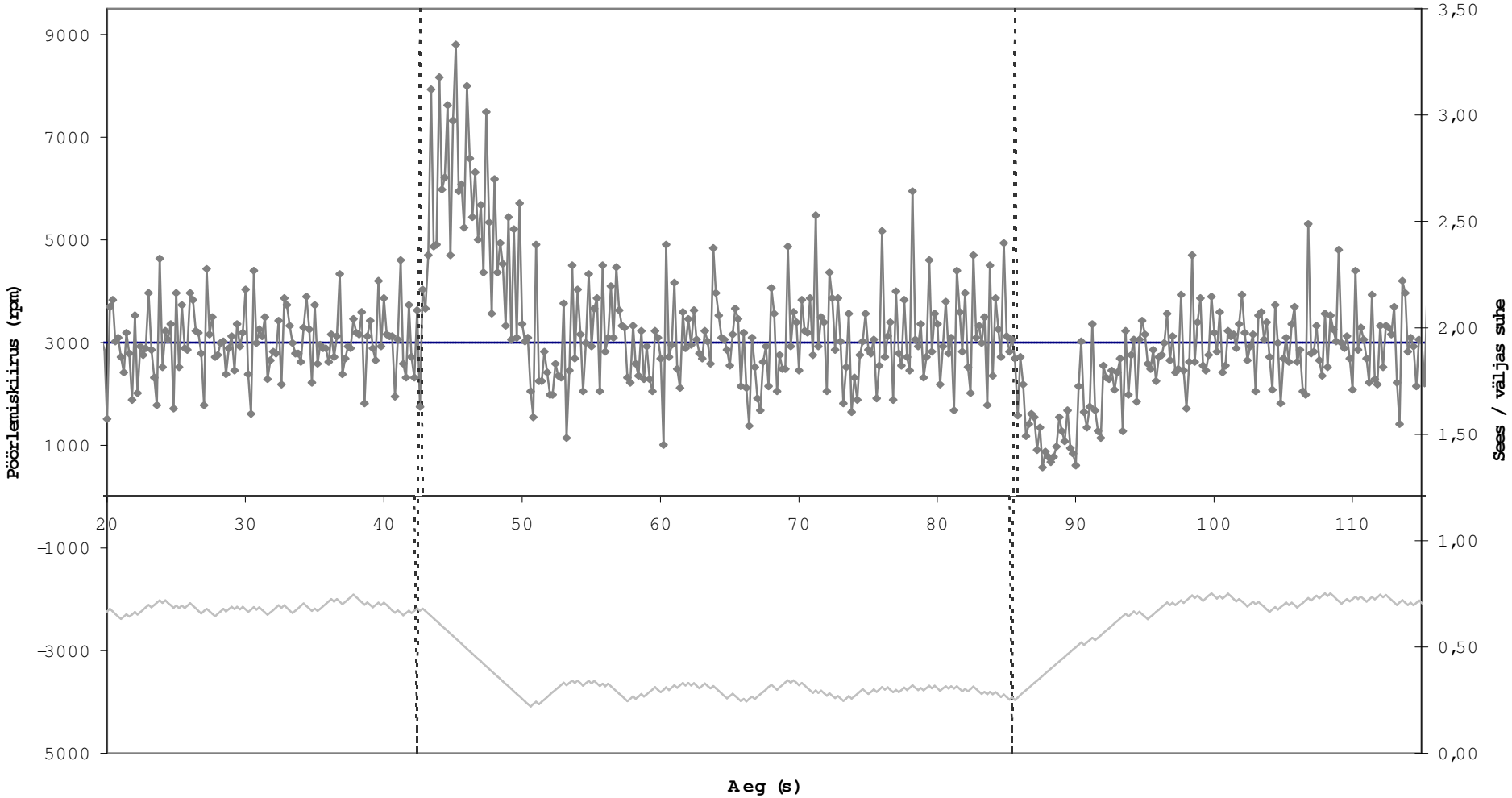


Graafik 2. Seadesuuruse hüpe 0 • 3000 • 0.



Graafik 3. Seadesuuruse hüpe 0 • 9000 • 0.

Selles katses on ette antud seadesuurus, mida mootor 10V toitepinge korral ei suuda saavutada. See graafik illustreerib ka pöörlemiskiiruse lugemise vea olemasolu – kui sees / väljas suhe on 1, siis on mootor kogu aeg sisse lülitatud ja mingit pulseerimist ega võnkumist ei toimu, aga lugem on ikkagi äärmiselt hüplik.



Graafik 4. Häire mõju süsteemile.

Algul on mootori toitepinge 10V, esimese markeri juures keeratakse järsult 16 voldile, teise markeri juures tagasi 10 voldile. Mõlemal juhul suudab juhtsüsteem häirele mõistlikult reageerida. Keskmises osas võib märgata teatavat võnkumist, mis tekib, kui toitepinge on suur, aga nõutav pöörlemiskiirus suhteliselt väike ja seetõttu väikesed muutused juhtsignaalis põhjustavad suuri muutusi mootori kiiruses (süsteem ei ole lineaarne). Sellises režiimis oleks ilmselt vaja muuta programmiparameetreid.

Kokkuvõte

Käesoleva töö peamine eesmärk – saada kogemus reaalse seadme ühendamiseks arvutiga – sai edukalt täidetud. Konkreetsem ülesanne – mootori pöörete hoidmine etteantud väärtuse lähedal – õnnestus samuti. Loomulikult on võimalik kõike seda teha paremate vahenditega ja täpsemalt, kuid eesmärgiks ei olnudki niivõrd täpsuse taga-ajamine, kuivõrd lihtsalt ühe töötava süsteemi loomine. Tööstuslikku väärtust valmistatud süsteem küll ei oma, kuid õppimise seisukohalt on sellise asja ehitamine väga kasulik kogemus.

```
#include "allegro.h"
#include "pc.h"
#include <fstream>
using namespace std;


// --- Global initialization ------------------------------------

// LPT address (the most common one)
const int port_address = 0x378;

// Input related constants
const int bps = 5;  // How often shall we check the input (beats per second)
                    // Less frequent checking means more integration and
                    // therefore less noise. [Also affects file output frequency]
const int bpm = 60 * bps;  // Just a conversion from bps to bpm

// Port output related constants
const int timestep = 1;  // Smallest unit of time (in milliseconds)
const int min_cycle = 2 * timestep;  // Smallest allowed output cycle (msec)
const float cycle_change = 0.001; // How much Left and Right keys change the cycle duration
const double activity_ratio_change = 0.00000002;  // How fast can activity ratio change
const int target_rpm_change = 1;  // How much Up and Down keys change the target rpm value

// Timer variables (must be declared volatile so the optimiser doesn't mess up)
volatile int mouse_check = 0;
volatile int global_time = 0;
volatile int output_time = 0;



// --- Timer interrupt handlers ---------------------------------

// This function is for timing the input
void inc_mouse_check(void)
{
   mouse_check++;
}

END_OF_FUNCTION(inc_mouse_check);

// This function is for timing the port output
void inc_time(void) {
   global_time += timestep;
   output_time += timestep;
}

END_OF_FUNCTION(inc_time);
```

```
// --- Main function --------------------------------------

int main() {

   // Initializing the environment (Allegro)
   allegro_init();
   install_keyboard();
   install_mouse();
   install_timer();

   set_gfx_mode(GFX_SAFE, 320, 200, 0, 0);
   text_mode(0);

   int textcolor = makecol(0, 200, 0);

   // All variables and code used inside interrupt handlers must be locked
   LOCK_VARIABLE(mouse_check);
   LOCK_FUNCTION((void*)inc_mouse_check);

   LOCK_VARIABLE(output_time);
   LOCK_FUNCTION((void*)inc_time);

   // Installing the timers
   install_int_ex(inc_mouse_check, BPS_TO_TIMER(bps));
   install_int_ex(inc_time, MSEC_TO_TIMER(timestep));

   // Initializing input related variables
   int mickey_x = 0,  // Mouse mickeys
       mickey_y = 0;
   float rps = 0,     // How fast the motor is spinning
         rpm = 0;

   // Initializing port output related variables
   float cycle = 75;   // Output cycle (in msec)
   double activity_ratio = 0;  // Ratio: output high / cycle time.
                               // In range [0..1]
   int activity_time = cycle * activity_ratio;  // Conversion from ratio to time
   bool flag_high = false;   // Just a useful flag (for port output)

   int target_rpm = 0;  // How fast we want the motor to be spinning

   // Initializing other variables
   ofstream outfile("outfile.txt");
   int tabtime = 0;  // Just a helper variable for file output

   // Saving some information
   outfile << "bps = " << bps << endl;
   outfile << "timestep = " << timestep << endl;
   outfile << "activity_ratio_change = " << activity_ratio_change << endl;
   outfile << "cycle = " << cycle << endl << endl;

   outfile << "global_time  target_rpm  rpm  activity_ratio" << endl << endl;
```

Lisa: programmikood..

```
// --- Main loop -------

while(not key[KEY_ESC]) {

  // --- Processing input from motor (the feedback) and writing info to screen & file ---
  if(mouse_check > 0) {

    // NB! As retrieving mouse mickeys is done here, not in timer interrupt
    // handler, it is not happening at the very accurate time moment and
    // therefore introduces additional input error. But it's a too big
    // and inconvenient function for interrupt handler, so it must
    // be done here :(  (well, I'm not 100% sure, but still...)
    get_mouse_mickeys(&mickey_x, &mickey_y);

    // Conversion from mickey_x to rounds of motor
    // Constant 8.93 is taken from experiments (turning motor manually
    // and counting the cycles while summing mickeys... May be inaccurate)
    float rounds = - static_cast<float>(mickey_x) / 8.93;

    // Calculating rounds per second and rounds per minute
    rps = rounds * bps,
    rpm = rounds * bpm;

    // Printing the information on screen
    textprintf(screen, font, 20, 20, textcolor,"mouse_check = %d       ", mouse_check);
    textprintf(screen, font, 20, 35, textcolor,"mx = %d       ", mickey_x);

    textprintf(screen, font, 20, 50, textcolor,"rps = %f       ", rps);
    textprintf(screen, font, 20, 65, textcolor,"rpm = %f       ", rpm);

    textprintf(screen, font, 20, 80, textcolor,"activity_ratio = %f       ", activity_ratio);

    // Saving some info into a file
    outfile << global_time << "  " << target_rpm << "  " << rpm  << "  " << activity_ratio << endl;

    // Decreasing mouse_check flag back to zero (if remains higher,
    // then MUST decrease mouse check rate (in global constants))
    mouse_check--;
  }


  // --- Computing activity rate ---

  // If rpm too low, then increase activity ratio
  if(rpm < target_rpm) {
    activity_ratio += activity_ratio_change;
    if(activity_ratio > 1)
      activity_ratio = 1;
    activity_time = cycle * activity_ratio;
  }

  // If rpm too high, then decrease activity ratio
  if(rpm > target_rpm) {
    activity_ratio -= activity_ratio_change;
    if(activity_ratio < 0)
      activity_ratio = 0;
    activity_time = cycle * activity_ratio;
  }

  // --- Twiddling the port output ---

  // If activity time is over, then set output low
  if((output_time > activity_time) and flag_high) {
    outportb(port_address, 0);
    flag_high = false;
  }

  // If cycle is over, then set output high and reset time
  if(output_time >= cycle) {
    if(activity_ratio > 0)
      outportb(port_address, 1);
    flag_high = true;
    output_time = 0;
  }


  // --- User input processing ---

  // Increase target rpm value
  if(key[KEY_UP]) {
    target_rpm += target_rpm_change;
    textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
  }

  // Decrease target rpm value
  if(key[KEY_DOWN]) {
    target_rpm -= target_rpm_change;
    if(target_rpm < 0)
      target_rpm = 0;
    textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
  }

  // Set target rpm value to 0
  if(key[KEY_0]) {
    target_rpm = 0;
    textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
  }

  // Set target rpm value to 1000
  if(key[KEY_1]) {
    target_rpm = 1000;
    textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
  }
```

```
    // Set target rpm value to 3000
    if(key[KEY_3]) {
      target_rpm = 3000;
      textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
    }

    // Set target rpm value to 5000
    if(key[KEY_5]) {
      target_rpm = 5000;
      textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
    }

    // Set target rpm value to 9000
    if(key[KEY_9]) {
      target_rpm = 9000;
      textprintf(screen, font, 20, 180, textcolor,"target_rpm = %d       ", target_rpm);
    }

    // Writes a separator line to output file. Useful for marking some
    // real world event (e.g. the change of motor supply voltage level)
    // Note the limit: no more than 1 line per second (to avoid hundreds
    // of lines per one keypress)
    if(key[KEY_TAB])
      if((global_time - tabtime) > 1000) {
        outfile << "- - - Tab pressed - - -" << endl;
        tabtime = global_time;
      }

    // Increase cycle length
    if(key[KEY_RIGHT]) {
      cycle += cycle_change;
      activity_time = cycle * activity_ratio;
      textprintf(screen, font, 20, 95, textcolor,"cycle = %f       ", cycle);
    }

    // Decrease cycle length
    if(key[KEY_LEFT]) {
      cycle -= cycle_change;
      if(cycle < min_cycle)
        cycle = timestep;
      activity_time = cycle * activity_ratio;
      textprintf(screen, font, 20, 95, textcolor,"cycle = %f       ", cycle);
    }

  }
  // --- End of main loop -------

  // Clean-up
  outportb(port_address, 0);
}
```

Lisa: programmikood..