TEKNIIKKA JA LIIKENNE

Sähkötekniikka

Elektroniikka ja automaatio

INSINÖÖRITYÖ

Robotti - jalkapallopelin PID - säätö

Työn tekijä: Lassi Räsänen
Työn ohjaaja: Kai Lindgren

Työ hyväksytty: ___. ___. 2011

Kai Lindgren
lehtori

ALKULAUSE

Tämä opinnäytetyö tehtiin Metropolia Ammattikorkeakoululle ja aiheen antoi lehtori Kai Lindgren. Opinnäytetyö oli haastava, sillä jalkapallopelin edelleen kehittäminen vaati perehtymistä Mikko Pasasen insinöörityöhön ja Merlin System Corp., Ltd:n Robot Soccer Engine - jalkapallopelin kehittämiseen. Vei runsaasti aikaa perehtyä tehtyihin muutoksiin alkuperäisestä.

Työn haastavimpia asioita oli ymmärtää strategiaohjelman toiminta. PID - säädön kehittäminen aikaisemman insinöörityön pohjalta oli olennainen osa tätä työtä. Muita merkittäviä osia työssäni oli PID - säädön simuloinnin kehittäminen ja komentojen lähettäminen ohjelmallisesti PC:stä robotille bluetooth linkin kautta.

Haluan kiittää lehtori Kai Lindgreniä työn ohjauksesta ja hyvästä yhteistyöstä.

Helsingissä 22.02.2011

Lassi Räsänen

TIIVISTELMÄ

Työn tekijä: Lassi Räsänen

Työn nimi: Robotti - jalkapallopelin PID - säätö

Päivämäärä: 29.04.2011 **Sivumäärä:** 50 s. + 4 liitettä

Koulutusohjelma: Sähkötekniikka

Suuntautumisvaihtoehto: Automaatiotekniikka

Työn ohjaaja: lehtori Kai Lindgren

Tämän insinöörityön tavoitteena oli dokumentoida robotti - jalkapallopeliä, jonka Metropolia Ammattikorkeakoulu oli hankkinut Merlin System Corp., Ltd:ltä. Työ kuuluu yhtenä osana koulun jalkapallopeli - projektiin, jossa tavoitteena on tulevaisuudessa mahdollisesti osallistua kansainvälisiin koulujen välisiin robotti - jalkapallo- kisoihin.

Työn tarkoitus oli kehittää pelistrategiaa sellaiseen suuntaan, että jalkapallopelin pelaaminen olisi sujuvampaa ja pallon haltuun ottaminen tarkempaa. PID - säädön kehityksessä käytettiin Microsoftin Visual Studio 2005 kehitysympäristöä. Ohjelmointikielinä käytettiin C:tä ja C++:aa.

Peli muodostaa erittäin monipuolisen ohjelmistokehitys - ympäristön opetuskäyttöön.

Avainsanat: Robotti - jalkapallopelin PID - säätö, Merlin Systems

SISÄLLYS

1 JOHDANTO

Tämä insinöörityö käsittelee strategiaohjelman PID - säätöä, robotti - jalkapallopelissä (kuva 1) ja simulaatiota. Työssä on keskitytty 1 (yhdelle) robotille luomaan PID - säätö, jolla robotti säätyy automaattisesti pelipallon liikkeen mukaan ja toimii itsenäisesti pelialustalla, sekä on tarkkuudeltaan kiitettävää luokkaa, eli osuu palloon joka kerta. Tämän voi laajentaa useamalle pelaajalle säätyväksi järjestelmäksi. Järjestelmään liittyen on aikaisemmin valmistunut kaksi työtä:

Mikko Pasanen: "ROBOTTI-JALKAPALLOPELI", Insinöörityö 2009.

Lassi Räsänen: "Robotti jalkapallon ohjelmallinen yhteys robotille", Projektityö, 2011.

Kuva 1. Robotti - jalkapallopeli

Tässä työssä esitetään vain lopputyön kannalta kiinnostavin osa - alue. Koodin omistavat Merlin Systems ja Metropolia Ammattikorkeakoulu. Koodin levittäminen Metropolian ulkopuolelle ei ole suotavaa, eikä osin edes luvallista. Koodia ei saa laittaa vapaasti internetistä saatavaksi.

2 PID SÄÄTÖ (TEORIA)

Usein esiintyy tilanteita, joissa säädöltä vaaditaan erittäin voimakasta ja välitöntä reagointia säätöpoikkeamaan. Esimerkiksi, jos mittausjärjestelmä on aikavakioltaan hidas verrattuna itse säädettävään järjestelmään ja/tai jos mittauksessa esiintyy viivettä ”näkee” säätöalgoritmi säädettävässä järjestelmässä ja sen todellisessa säätösuureessa (=suure, jota mittausjärjestelmä mittaa) tapahtuvat muutokset ikäänkuin myöhässä. Häiriön aiheuttama pienikin erosuureen muutos voi olla merkki siitä, että itse säädettävässä järjestelmässä on jo ehtinyt tapahtua paljon. Tällöin säädön pitää ottaa eräänlaista ennakkoa ja ohjata hyvin voimakkaasti järjestelmää oikeaan suuntaan.

edellä mainittu ennakointi voidaan toteuttaa lisäämällä säätöalgoritmiin derivoiva termi, joka muuttaa säätimeltä lähtevää ohjausta u(t) suhteessa erosuureen e(t) aikaderivaattaan eli muutosnopeuteen.

$$K_D \frac{d}{dt} e(t), K_D \geq 0, K_D \in R \tag{1}$$

Mikäli derivoiva termi lisätään PI - algoritmiin, saadaan säätö, joka reagoi nopeasti ja poistaa säätöpoikkeaman, joissain tapauksissa. Näin syntynyt PID - algoritmi voidaan kirjoittaa muotoon.

$$u(t) = K_P e(t) + K_1 \int_0^t e(\tau) d\tau + K_D \frac{d}{dt} e(t) \tag{2}$$

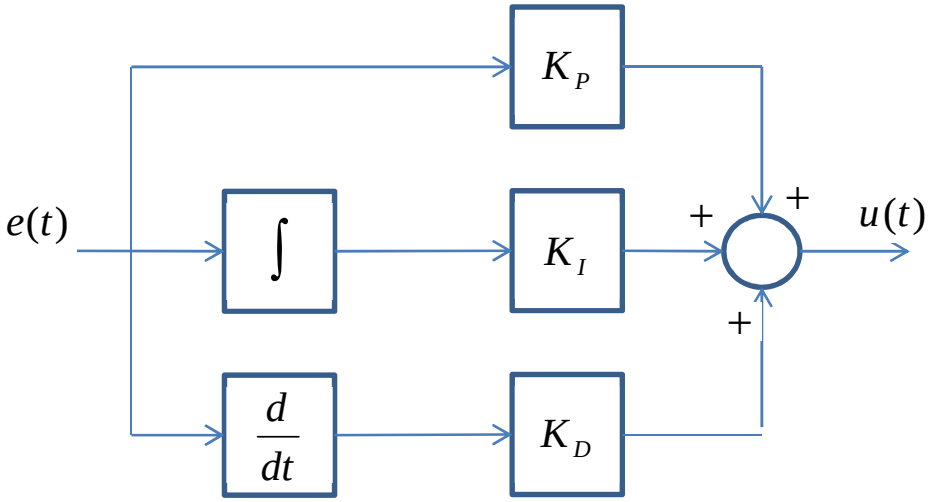
josta edelleen saadaan toinen yleisesti käytetty muoto

$$u(t) = K_P (e(t) + \frac{1}{T_1} \int_0^t e(\tau) d\tau + T_D \frac{d}{dt} e(t)) \tag{3}$$

missä

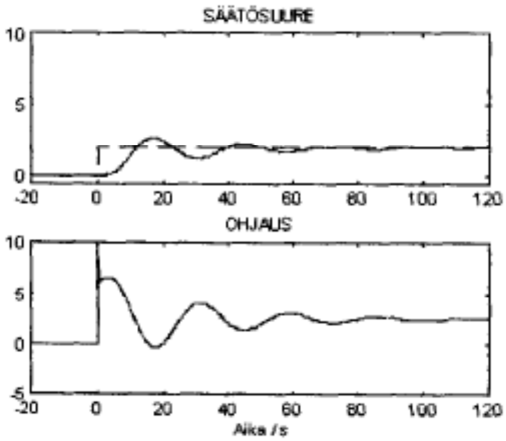
$$K_I = \frac{K_p}{T_I} \mid K_D = K_P * T_D \tag{4}$$

missä derivointitermin kerrointa T_D sanotaan derivointiaikavakioksi ja se ilmaisee karkeasti sen ajan, jonka kuluessa säätö saa eräänlaista lisävahvistusta nopean erosuuremuutoksen tapahduttua; jos erosuuremuutokset ovat hitaita, on derivointitermin vaikutus vähäistä. Derivointiaikavakion yksikkö on siis aikayksikkö. Kaavoja vastaavat lohkokaaviot esitetään seuraavaksi.



Kuva 2. PID - algoritmin lohkokaavio.

PID - algoritmilla toteutetun säätöjärjestelmän aikakäyttäytymistä on selvitetty kuvassa 3.



Kuva 3. PID - säädön käyttäytyminen ohjearvomuutoksessa. Ohjearvo merkitty katkoviivalla.

Nykyään kaikki sähköisesti toteutetut säätimet (tai automaatiojärjestelmien säätölohkot) sisältävät vähintäänkin täydellisen PID - algoritmin. Mikäli ei kuitenkaan haluta käyttää täydellistä PID - algoritmia, saadaan siitä PI - algoritmi virittämällä derivointiaikavakio T_D nollaksi tai vaikkapa P - algoritmi virittämällä edelleen integrointiaikavakio T1 äärettömäksi (= hyvin suureksi). /1/

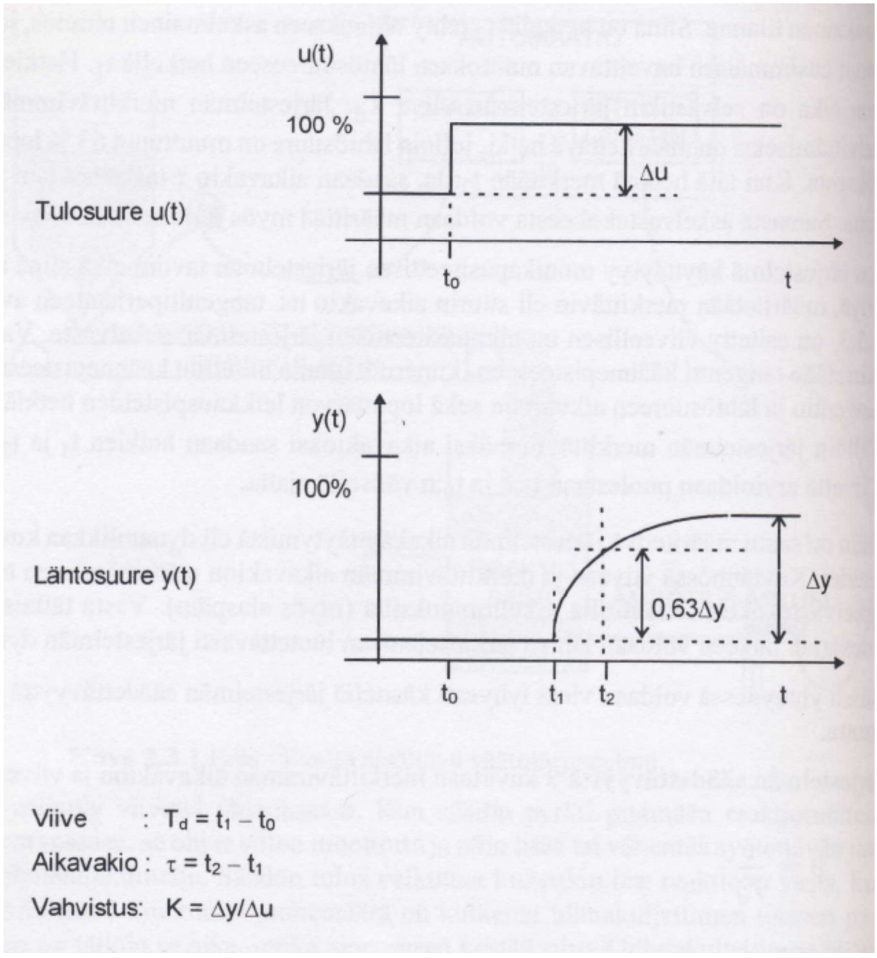
3 AIKAVAKIO JA VIIVE (TEORIA)

Järjestelmän aikavakiolla kuvataan sitä nopeutta, jolla sen energia- tai ainevarastot muuttuvat, kun tulosuureissa tapahtuu muutoksia. Mitä suurempi aikavakio, sitä hitaampi järjestelmä. Koska jokaiseen varastoon liittyy oma aikakäyttäytyminen, on järjestelmän aikavakioiden lukumäärä sama kuin sen kapasiteettiluku. Käytännön säätösuunnittelun kannalta riittää merkittävimmän, yleensä suurimman aikavakion määrittäminen; suurimman varaston tilavuus on ratkaisevaa.

Järjestelmän viiveellä (kuollut aika, horrosaika) tarkoitetaan sitä aikaa, joka kuluu tulosuureen muutoksesta ensimmäiseen tilan muutokseen. Viiveettömässä järjestelmässä tulosuureen muutos aiheuttaa välittömän tilan muutoksen, joka tapahtuu järjestelmän aikavakioiden kuvaamalla nopeudella. Viivettä saattaa esiintyä säätöjärjestelmässä säädettävän järjestelmän lisäksi ohjauksessa ja mittauksessa. Näitä tulisi kuitenkin mahdollisuuksien mukaan välttää.

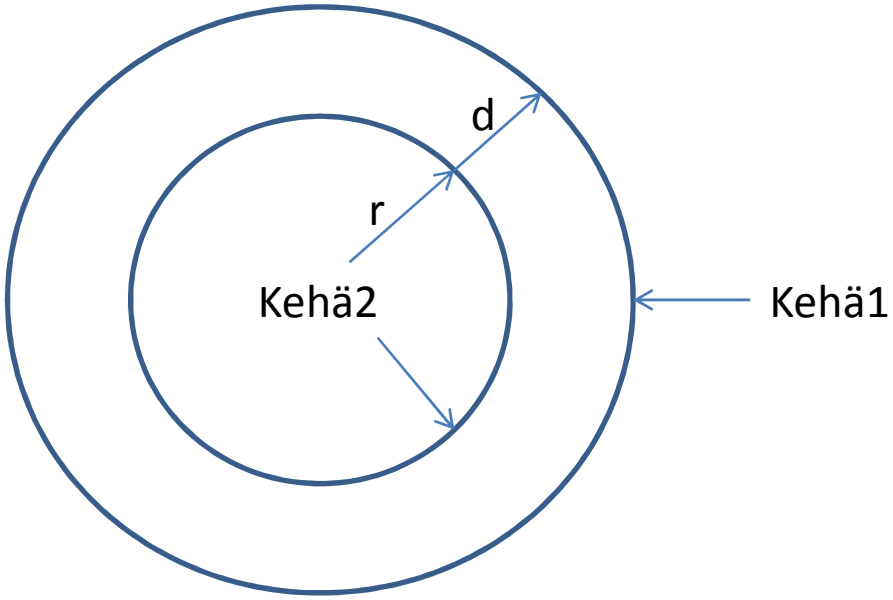
Jos järjestelmän voidaan katsoa käyttäytyvän kuten yksikapasiteettinen järjestelmä, määritetään sen aikavakio ns. 63 %:n säännön avulla. Kuvassa 4 sivulla 5 on esitetty askelvastekokeen mukainen tilanne. Siinä on hetkellä t_0 tehty ohjaukseen askelmainen muutos, joka on aiheuttanut ensimmäisen havaittavan muutoksen lähtösuureeseen hetkellä t_1. Järjestelmän merkittävimmän aikavakion selvittämiseksi on määritettävä hetki, jolloin lähtösuure on muuttunut 63 % lopullisesta muutoksesta. Kun tätä hetkeä merkitään t_2:lla, saadaan aikavakio τ laskettua t_2:n ja t_1:n erotuksena. Samasta askelvastekokeesta voidaan määrittää myös järjestelmän vahvistus.

Näin on saatu määritettyä järjestelmän aikakäyttäytymistä eli dynamiikkaa kuvaavat ominaisuudet. Käytännössä viiveen ja merkittävimmän aikavakion määrittämiseen tarvitaan useita askelvastekokeita erisuurilla askelmuutoksilla (myös alaspäin). Vasta tällaisen laajemman koesarjan jälkeen voidaan lähteä tarkastelemaan luotettavasti järjestelmän dynamiikkaa. /1/



Kuva 4. Yksikapasiteettisen järjestelmän viiveen, aikavakion ja vahvistuksen määrittäminen.

4 ROBOTIN KÄÄNTÖNOPEUDEN SELVITTÄMINEN



Kuva 5. Robotin kääntösäde ja halkaisija

Kehä1 lasketaan ympyrän säteen ja halkaisijan summalla, joka on yhden mukainen kiihtyvyyteen verrattuna ajan funktiona. Kehä2 lasketaan ympyrän säteen mukaan, joka on yhden mukainen kiihtyvyyteen verrattuna ajan funktiona.

$$Kehä1 = 2\pi(r + d) = V_1 * T \tag{5}$$

$$Kehä2 = 2\pi r = V_2 * T \tag{6}$$

Kehä1 ja kehä2 erotuksesta saadaan kiihtyvyys yhtälöt V_1 ja V_2 aikaan verrannollisina, jotka vastaa ympyrän halkaisijaa.

$$Kehä1 - Kehä2 = (V_1 - V_2)T = 2\pi d \tag{7}$$

Ratkaistaan aika yllä olevasta lauseesta ja saadaan halkaisija jaettuna kiihtyvyydellä.

$$\Rightarrow T = \frac{2\pi d}{V_1 - V_2} \tag{8}$$

Säteen ja halkaisijan välinen yhteys suhteessa kiihtyvyyteen. Kehä2 ratkaisu halkaisian ja säteen suhde kiihtyvyyteen.

$$2\pi r = 2\pi d\frac{V_2}{V_1 - V_2} \Rightarrow \frac{r}{d} = \frac{V_2}{V_1 - V_2} \tag{9}$$

Säteen ja halkaisijan välinen yhteys suhteessa kiihtyvyyteen. Kehä1 ratkaisu halkaisian ja säteen suhde kiihtyvyyteen.

$$2\pi(r + d) = 2\pi d\frac{V_1}{V_1 - V_2} \Rightarrow 1 + \frac{r}{d} = \frac{V_1}{V_1 - V_2} \tag{10}$$

Kehä2 säteen ja halkaisijan välinen yhteys ratkaistuna.

$$\frac{d}{r} = \frac{V_1 - V_2}{V_2} = \frac{V_1}{V_2} - 1 \tag{11}$$

Kehä1 säteen ja halkaisijan välinen yhteys ratkaistuna.

$$\frac{d}{r + d} = \frac{V_1 - V_2}{V_1} = 1 - \frac{V_2}{V_1} \tag{12}$$

Etäisyyksien suhde kiihtyvyyteen.

$$\frac{r + d}{d} = \frac{V_1}{V_1 - V_2} \tag{13}$$

$$1 + \frac{r}{d} = \frac{V_1}{V_1 - V_2} \tag{14}$$

$$1 + \frac{V_2}{V_1 - V_2} = \frac{V_1}{V_1 - V_2} \tag{15}$$

Muodostetaan yhtälöpari kahdesta seuraavasta yhtälöstä.

$$r = d * \frac{V_2}{V_1 - V_2} \;\; ; \; r + d = d * \frac{V_1}{V_1 - V_2} \tag{16}$$

$$\frac{\Delta t}{T} = \frac{\Delta \varnothing}{360^\circ} \Rightarrow T = 360^\circ * \frac{\Delta t}{\Delta \varnothing} \tag{17}$$

Kiihtyvyyksien erotus. Aika on ratkaistu ympyrän ja halkaisian suhteen.

$$V_1 - V_2 = \frac{2\pi d}{T} = \frac{2\pi d}{360^\circ * \frac{\Delta t}{\Delta \varnothing}} = \frac{2\pi}{360^\circ} * d * \frac{\Delta \varnothing}{\Delta t} \tag{18}$$

Pythagoran kaavaa apuna käyttäen saadaan ratkaistua etäisyyksien neliöiden summa eli etäisyyden yhteys keskipisteeseen halkaisijasta..

$$s = \sqrt{dx^2 + dy^2}$$

Kiihtyvyys saadaan kun etäisyys jaetaan ajan muutoksella.

$$\frac{(V_1 + V_2)}{2} = V = \frac{s}{\Delta t} \Rightarrow V_1 + V_2 = \frac{2s}{\Delta t} \tag{19}$$

Ratkaistuna kehä1 kiihtyvyys matemaattisesti.

$$2V_1 = \frac{2\pi}{360^\circ} d \frac{\Delta \varnothing}{\Delta t} + \frac{2s}{\Delta t} \Rightarrow V_1 = \frac{\pi}{360^\circ} d \frac{\Delta \varnothing}{\Delta t} + \frac{s}{\Delta t} \tag{20}$$

Ratkaistuna kehä2 kiihtyvyys matemaattisesti.

$$2V_2 = -\frac{2\pi}{360^\circ} d \frac{\Delta \varnothing}{\Delta t} + \frac{s}{\Delta t} \Rightarrow V_2 = \frac{s}{\Delta t} - \frac{\pi}{360^\circ} d \frac{\Delta \varnothing}{\Delta t} \tag{21}$$

5 PI SÄÄTÖ PAIKALLAAN PYÖRIVÄLLE ROBOTILLE

Ohjelma pyörittää robottia PC:ssä kuvaruudulla ilman kommunikointia robotin kanssa, joka säätyy palloa kohti automaattisesti pallon liikkeen mukaan.

LassiRotateTo ohjelman kehitys lähti Mikko Pasanen insinöörityöstä robotti-jalkapallopelin aliohjelmia hyödyntäen ja kyseisessä työssä saatujen kokemuksien pohjalta tehtiin LassiRotateTo aliohjelma.

Parametrit: Kp = 0,1; Ki = 0,00025

Taulukko 1. Robotin säätyminen strategia ohjelmassa

1.	void LassiRotateTo(int des_rot , int speed, int id, Environment *env, bool forceForefront, bool first_time)
	{
	static Environment old_environment, old_env;
	
	old_env = &old_environment;
	
	static double sum = 0.0;
	if (first_time == true)
	{
	sum = 0.0;
	
	double max_speed = (double)speed
	int rob_rot = (int)env->home[id].rotation;
	
	double e;
		LassiRotateTo aliohjelma alkaa tästä.
		
		Jos tullaan tähän osioon ensimmäistä kertaa asetetaan kaikki tarvittavat muuttujat.

2.	if ((forceForefront != true) &&
	(Mikko_d(des_rot,rob_rot,id) >= 90. ||
	Mikko_d(des_rot,rob_rot,id) <= -90.))
	{
	rob_rot = Mikko_l(180 + rob_rot);
	e = Kp*Mikko_d(des_rot, rob_rot, id);
	
	env->home[id].velocityLeft = 0. - e;
	
	env->home[id].velocityRight = e;
	
	}
	else
		Seuraavaksi tehdään robotin liikunta alueet suunnat selviksi.
		Tehdään kenttä robotin takaosasta etuosaksi.
		Kulmanopeus e lasketaan virheestä Mikko_d aliohjelmalla.
		e on oikein kun se menee tangentiin väärin päin.
		Asetetaan pyörien nopeudet e arvon mukaan.

3.	{
	
	e = Kp*Mikko_d(des_rot, rob_rot, id);
		Robotti menee tangenttiin väärin pain eli takaperin.
		Asetetaan pyörien nopeudet kulmavirheen

	env->home[id].velocityLeft = 0. - e;
	
	env->home[id].velocityRight = e;
	
	}
	}
	else
		avulla e ja oikean pyörä vähennettynä nollasta.
		
		Jos tullaan ensin tähän else lauseseen on se false.

4.	{
	double max_speed = (double)speed;
	int rob_rot = (int)env->home[id].rotation;
	double e;
	
	double v_right_tobe = env->home[id].velocityRight;
	double v_left_tobe = env->home[id].velocityLeft;
	
	double d = 3, delta_t = 0.050;
	double delta_fii = Mikko_d(des_rot,rob_rot,id);
	double dx = env->home[id].pos.x - old_env->home[id].pos.x;
	double dy = env->home[id].pos.y - old_env->home[id].pos.y;
	
	double s = sqrt(pow(dx,2)+ pow(dy,2));
	
	double v_right_measured = (PI/360) * d * (delta_fii/delta_t) + s/delta_t;
	double v_left_measured = -1 * (PI/360) * d * (delta_fii/delta_t) + s/delta_t;
		Esitellään uudet muuttujat jotka toimii tässä aliohjelmassa.
		
		Nämä kaksi viimeistä muuttujaa on automaattisia muuttujia.

5.	if ((forceForefront != true) &&
	(Mikko_d(des_rot,rob_rot,id) >= 90. ||
	Mikko_d(des_rot,rob_rot,id) <= -90.))
	{
		Varmistetaan että robotti on oikein päin palloon nähden.

	rob_rot = Mikko_l(180 + rob_rot);
	
	sum = sum +
	(v_left_measured - env->home[id].velocityLeft)
	-
	(v_right_measured - env->home[id].velocityRight);
	
	e = Kp*Mikko_d(des_rot, rob_rot, id)+ Ki*sum;
	
	env->home[id].velocityLeft = 0. - e;
	env->home[id].velocityRight = e;
	
	}
	else
	
	{
		Tehdään robotin takaosasta etuosa. Summataan yhteen ja vähennetään pyörien nopeudet toisistaan.
		
		Kulmanopeus laskettu yhteen Ki vakiolla kerrotaan pyörien nopeudet toisistaan laskettu sum muuttujalla.

6.	sum = sum +
	(v_left_measured - env->home[id].velocityLeft)
	-
	(v_right_measured - env->home[id].velocityRight);
	
	e = Kp*Mikko_d(des_rot, rob_rot, id) + Ki*sum;
	
	env->home[id].velocityLeft = 0. - e;
	
	env->home[id].velocityRight = e;
	
	}
	
	}
	
	#ifdef DEBUGFILE
		Jos robotti liikkuu etupää edellä tehdään seuraavaa eli vähennetään mitattu pyörän nopeus todellisesta nopeudesta.
		
		proportionaalinen säätö tässä osiossa ja integraali määritelty tässä osiossa. Mikko_d aliohjelman avulla.

```
                        if (id ==
debug_rob)

                        {

                        fprintf(debugfile,
                "robot %d in MikkoRotateTo,
                des_rot: %d, e: %f  \n",
                id,des_rot,e );

                        }
                #endif


                 old_environment = *env;


                        }
```

1. Esitellään tarvittavat muuttujat aliohjelmalle.
2.
3.
4.
5
6.

6 LIIKKUVAN ROBOTIN PID - SÄÄTÖ

Ongelman lähtö kohtana oli että robotti ei osunut palloon riittävän tarkasti että pallon ja robotin välistä kulmaa oli saatava osuvammaksi palloon nähden. Robotin PID algorytmi syntyy LassiMoveToOld aliohjelmalla. Liikkuvan robotin kommunikointiin pc:n kanssa, yhdistettynä kamerakuvilla, jotka säätyvät pallon liikkeen mukaan pelikentällä. Minne pallo ikinä meneekin niin robotti seuraa pallon liikettä ja säätyy automaattisesti pc:n ja kamerakuvien mukaan, sekä muuttaa oikean, että vasemman pyörän nopeuksia tarvittaessa.

LassiMoveToOld aliohjelman kehitys sai alkunsa LassiRotateTo aliohjelman kehityksen pohjalta, josta lähdettiin toteuttamaan liikkuvalle robotille

ohjelmaa. LassiRotateTo:n pohjalta otettiin samanlainen koodi LassiMoveToOld:iin ja kehitys lähti siitä.

LassiMoveToOld:n säätöjärjestelmä yliohjautuu, kun vakiot menevät liian isoiksi arvoiksi. PID - säätöön tulee virhettä mukaan enemmän askelvasteen yliohjautumisesta eli yli menevät heilahdukset kuvassa 3 sivulla 3 ovat virheitä.

Parametrit: Kp = 0,1; Kd = 0,0025 ; Ki = 0,00025

Robotin viritys ohje: Robotin kulma asettaa 45 asteen kulmaan jalkapalloon nähden ja katsoa, että robotti lähtee kohti jalkapalloa. Muussa tapauksessa parametrit vääriä arvoja.

Taulukko 2. Stategia ohjelmassa pyörivä pid säätö

1.	void LassiMoveToOld(int des_rot , int speed, int id, Environment *env ,bool forceForefront, double distancevar)
	
	{
	static Environment old_environment, *old_env;
	old_env = &old_environment;
	static double sum = 0.0;
	static double virhe = 0.0;
	
	if (first_time == true)
	{
	
	sum = 0.0;
	
	virhe = 0.0;
		Tästä alkaa lassimovetoold aliohjelma ja esitellään tarvittavat muuttujat.
		
		Staattisten muuttujien esittelyt ja nollaukset.
		
		Jos tullaan ensimäistä kertaa tähän osioon asetetaan first_time todeksi.
		
		Asetetaan sum muuttuja nollaksi, joka laskee integraalin sisällön.
		
		Asetetaan virhe muuttuja nollaksi, joka laskee derivaatan sisällön.
		
		Esitellään muuttujia.

	double max_speed = (double)speed;
	int rob_rot = (int)env->home[id].rotation;
	
	double e;
		Esitellään kulmanopeus muuttuja.

	if ((forceForefront != true) && (Mikko_d(des_rot,rob_rot,id) >= 90. ||Mikko_d(des_rot,rob_rot,id) <= -90.))	forceForefront jos se ei ole tosi ja Mikko_d aliohjelma antaa robotin tiedot ja jos ne on yhtäsuuret tai suurempi kuin 90 tai Mikko_d aliohjelma antaa pienemmän tai yhtä suuren kuin -90. mennään lauseeseen.
	{	Jos robotti liikkuu takaperin tehdään seuraavaa. Tehdään takaosasta etuosa.
	rob_rot = Mikko_l(180 + rob_rot); // make back to front	Kulmanopeus saa proportionaalisen vahvistuksen, joka suoritetaan Mikko_d aliohjelman avulla.
	e = Kp*Mikko_d(des_rot, rob_rot, id);	
	env->home[id].velocityLeft = 0. - e;	Asetetaan nopeus vasemmalle pyörälle.
	env->home[id].velocityRight = e;	Asetetaan nopeus oikealle pyörälle.
	}	
	else	
	{	Kun robotti liikkuu

	e = Kp*Mikko_d(des_rot, rob_rot, id);
	// des_rot - rob_rot
	
	env->home[id].velocityLeft = 0. - e;
	
	env->home[id].velocityRight = e;
	
	}
	}
		eteenpäin tehdään seuraavaa.
		
		Asetetaan kulmanopeus proportionaalisella asetuksilla.
		
		Asetetaan vasemman pyörän nopeus.
		
		Asetetaan oikean pyörän nopeus eli kulmanopeus.

2.	else //first_time == false
	{
	
	double max_speed = (double)speed;
	
	int rob_rot = (int)env->home[id].rotation;
	
	double e;
	
	double v_right_tobe = env->home[id].velocityRight;
	double v_left_tobe = env->home[id].velocityLeft;
	
	double d = 3, delta_t = 0.050;
		max_speed:iin asetetaan nopeus speed.
		
		Kääntyminen asetetaan rob_rot:lla.
		
		Esitellään kulmanopeus tässä osiossa.
		
		Oikean pyörän - nopeus.
		
		Vasemman pyörän - nopeus.
		
		Muutos aijaksi asetetaan 0.050.

	double delta_fii = Mikko_d(des_rot,rob_rot,id);
	
	double dx = env->home[id].pos.x - old_env->home[id].pos.x;
	
	double dy = env->home[id].pos.y - old_env->home[id].pos.y;
	
	double s = sqrt(pow(dx,2)+ pow(dy,2));
		Tässä lasketaan missä pallo sijaitsee.

	double v_right_measured = (PI/360) * d * (delta_fii/delta_t) + s/delta_t;
	
	double v_left_measured = -1 * (PI/360) * d * (delta_fii/delta_t) + s/delta_t;
		Laskettu oikean pyörän nopeus.
		
		Laskettu vasemman pyörän nopeus.

	if ((forceForefront != true) && (Mikko_d(des_rot,rob_rot,id) >= 90. || Mikko_d(des_rot,rob_rot,id) <= -90.))
	{
	
	rob_rot = Mikko_l(180 + rob_rot);
		Tehdään takaosasta etuosa.

	sum = sum + (v_left_measured - env->home[id].velocityLeft) - (v_right_measured - env->home[id].velocityRight);
	
	virhe = virhe + (v_left_measured - env->home[id].velocityLeft) + (v_right_measured - env->home[id].velocityRight);
		Lasketaan integraali nopeudet pyörien nopeuksista.
		
		Lasketaan derivaatan nopeudet pyörien nopeuksista.

	e = Kp*Mikko_d(des_rot, rob_rot, id)+ Ki*sum + Kd*virhe;	Lasketaan kulmanopeus e proportinaali ja integraali sekä derivaatta termien avulla.

	env->home[id].velocityLeft = 0. - e;
	
	env->home[id].velocityRight = e;
		Asetetaan vasemman pyörän nopeus.
		
		Asetetaan oikean pyörän nopeus.

	}
	
	else
	
	{
		Robotti kulkee eteenpäin.

	sum = sum + (v_left_measured - env->home[id].velocityLeft) - (v_right_measured - env->home[id].velocityRight);	Lasketaan integraali nopeudet pyörien nopeuksista.
	virhe = virhe + (v_left_measured - env->home[id].velocityLeft) + (v_right_measured - env->home[id].velocityRight);	Lasketaan derivaatan nopeudet pyörien nopeuksista.
	e = Kp*Mikko_d(des_rot, rob_rot, id)+ Ki*sum + Kd*virhe;	Lasketaan kulmanopeus e proportinaali ja integraali sekä derivaatta termien summasta.
	env->home[id].velocityLeft = 0. - e;	Asetetaan vasemman

	env->home[id].velocityRight = e;
	}
	}
	}
		pyörän nopeus.
		Asetetaan oikean pyörän nopeus.
		first_time on epätosi

1. Esiteltiin muuttujat ja tehtiin robotin takaosasta etuosa.

2. Nopeuden laskentaa sekä proportionaali, integraali ja derivaatta termien laskentaa.

7 TYÖKONEIDEN VASTAMOMENTTI

Työkoneella tarkoitetaan kaikkia koneenrakennusteknillisiä laitteita, joita käytetään tuotantoelämän palveluksessa materian käsittelyyn mitä erilaisimmin tavoin: kuljettamiseen, muokkaamiseen, pilkkomiseen ym.

Kuljetustehtävissä työtehtävän vaatima käyttömomentti moottoriakselilla muodostuu etupäässä liikkeen muuttamiseen tarvittavasta momentista ($-T_A$), varsinkin kun itse like ja liikenneväline pyritään suunnittelemaan siten, että niiden vaatima käyttömomentti on pieni.

Muokkaamis- ja pilkkomistehtävissä työkoneen käyttömomentti koostuu pääosinitse muokkaustyön vaatimasta muodonmuutostyöstä sekä erilaisista kitkasta ja voiteluaineiden viskositeeteista johtuvista liikevastuksista.

Työkoneiden vastamomentti (käyttömomentti vakionopeudella) ilmoitetaan sähkökäytön suunnittelua ja mitoitusta varten tavallisesti nopeuden funktiona. Työkoneen liikkeelle saamista voi kylmissä käyttöolosuhteissa vaikeuttaa suuri lähtömomentti.

… luokitellaan usein nopeuseksponentin perusteella.

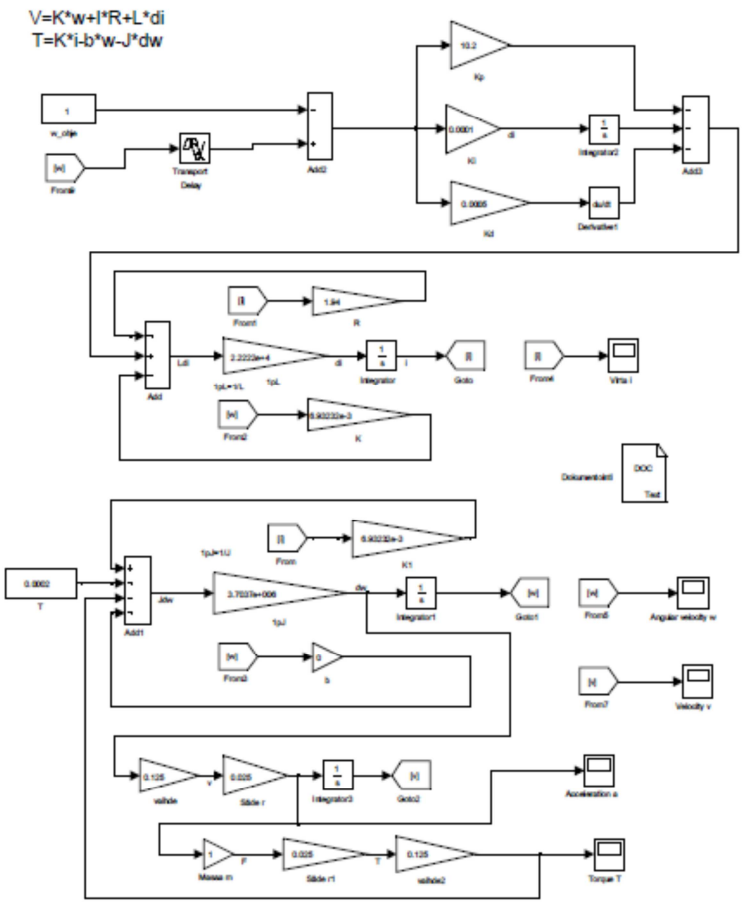
Periaatteessa työkoneen vastamomentti voidaan tarkasti ilmoittaa käyttönopeuden potenssisarjana:

$$T_{(Wm)} = \sum_{-\infty}^{\infty} T_{wn} \left(\frac{\omega_m}{\omega_{mN}} \right)^n \tag{5}$$

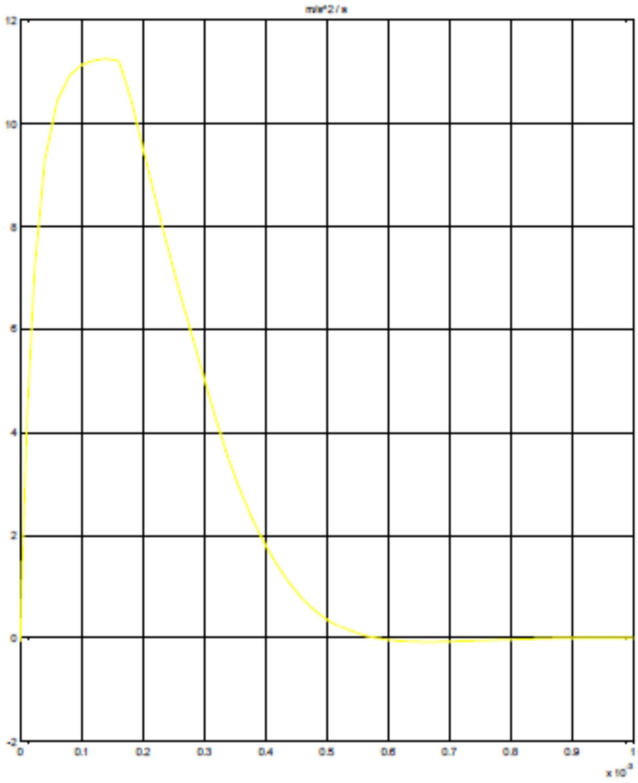
/2/

8 MATLAB SIMULAATIO

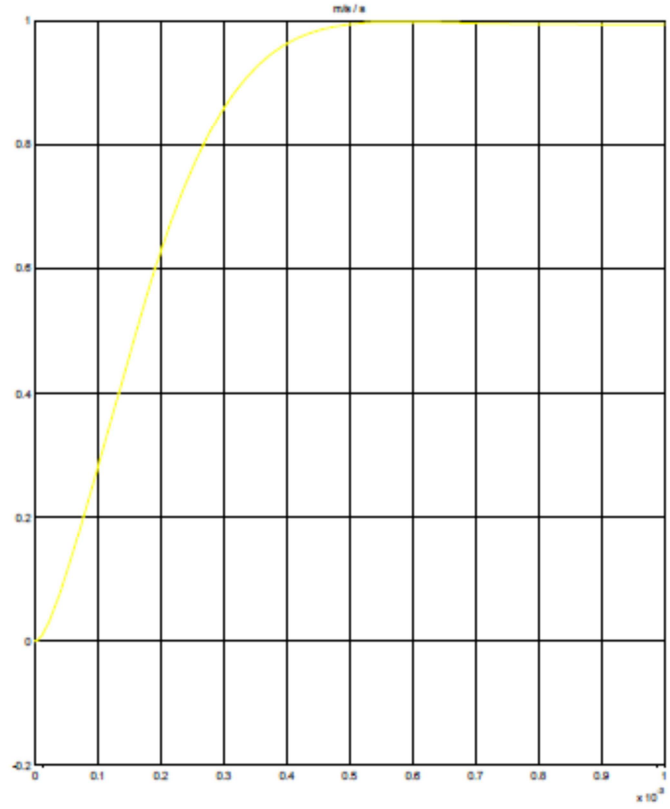
Seuraavaksi annetaan kulmanopeus ohjearvollisesti 1 ja proportionaalinen vahvistus on 0.2. Integraalinen vahvistus on 0.0001. Derivaatan vahvistus on 0.0005. Piirin viiveeksi on laitettu sekunneissa 0.00015. Momentin laskennan suhteena on käytetty robotin hammastuksen suhdetta, joka oli kymmenkertainen toisiinsa nähden ja ne olivat 42 ja 12.



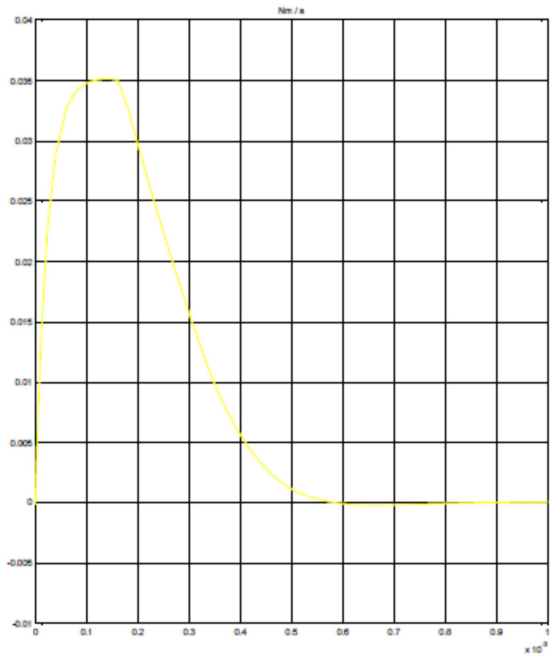
Kuva 6. Simulaatio ympäristö ja minimi arvot



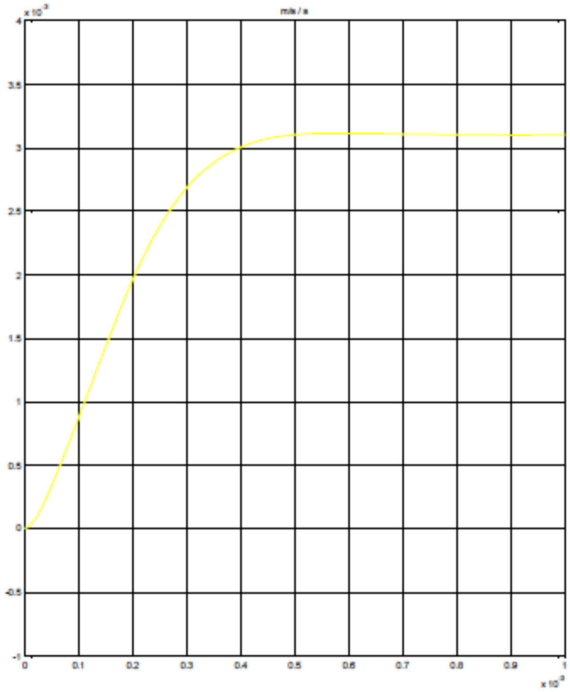
Kuva 7. kiihtyvyys minimi arvoilla



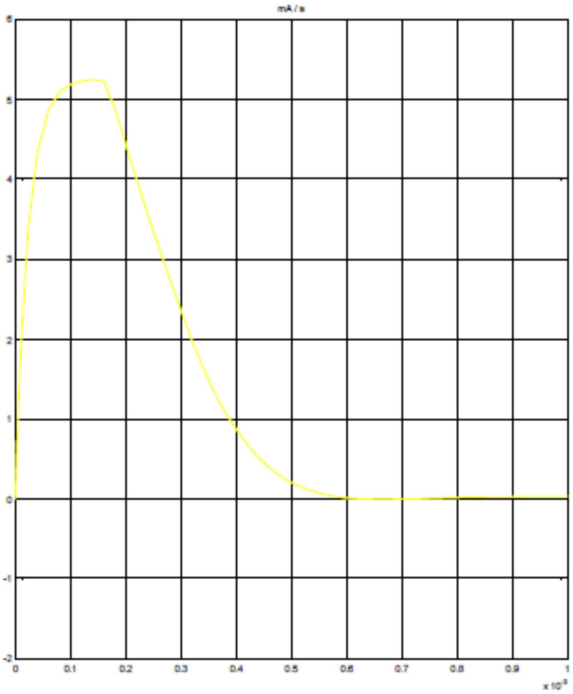
Kuva 8. Todellinen nopeus minimi arvoilla



Kuva 9. Momentti minimi arvoilla

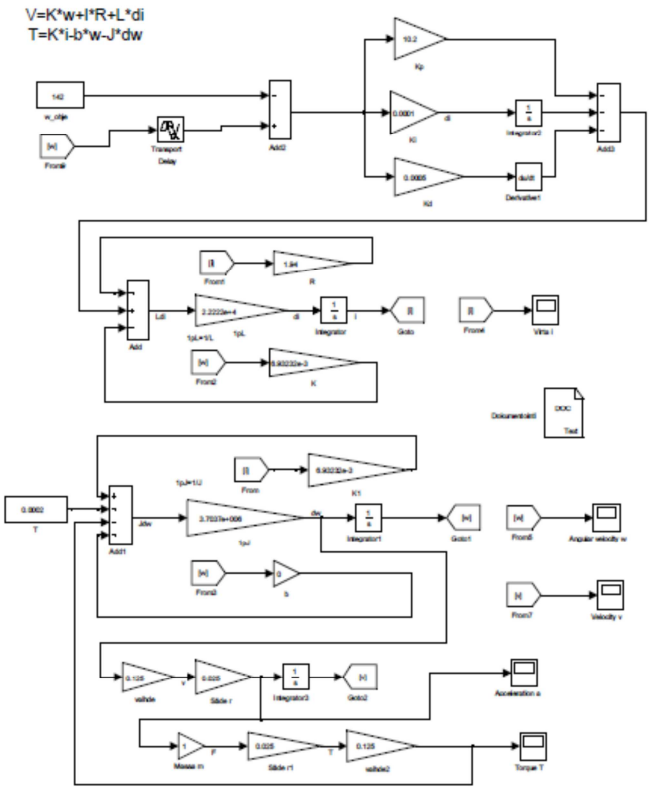


Kuva 10. Nopeus minimi arvoilla

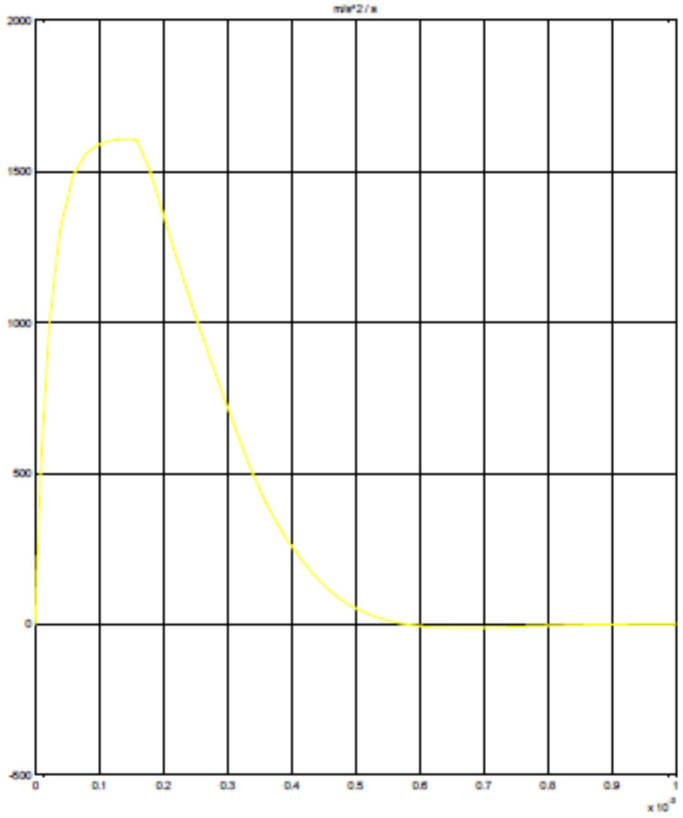


Kuva 11. Virta minimi arvoilla

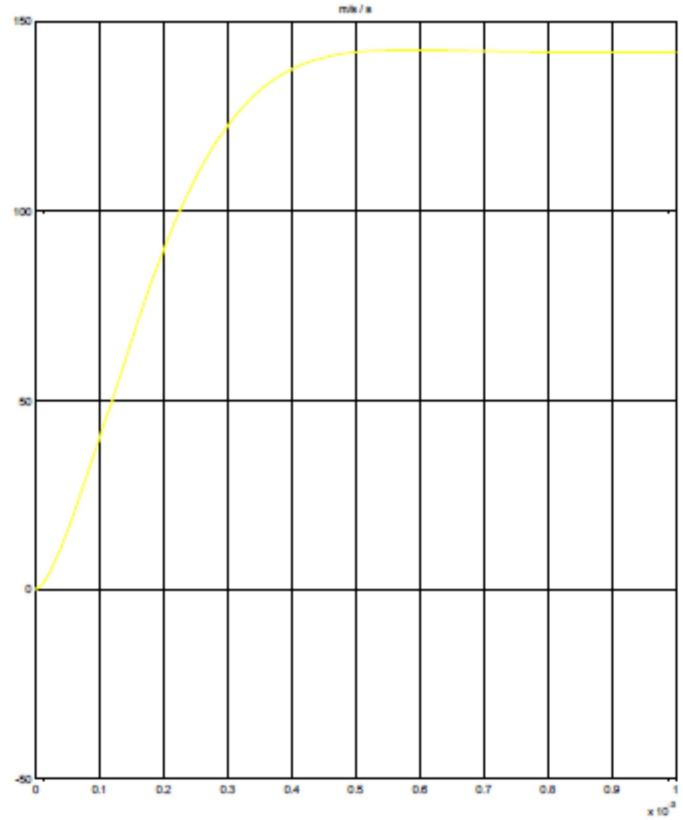
Seuraavaksi kulmanopeuden ohjearvo 142 ja proportionaalinen vahvistus on 10.2. Integraalinen vahvistus mallissa on 0.0001. Derivoinin vahvistus simulaatio mallissa 0.0005. Simulaatiossa käytetty kokonais viive sekuneissa 0.00015



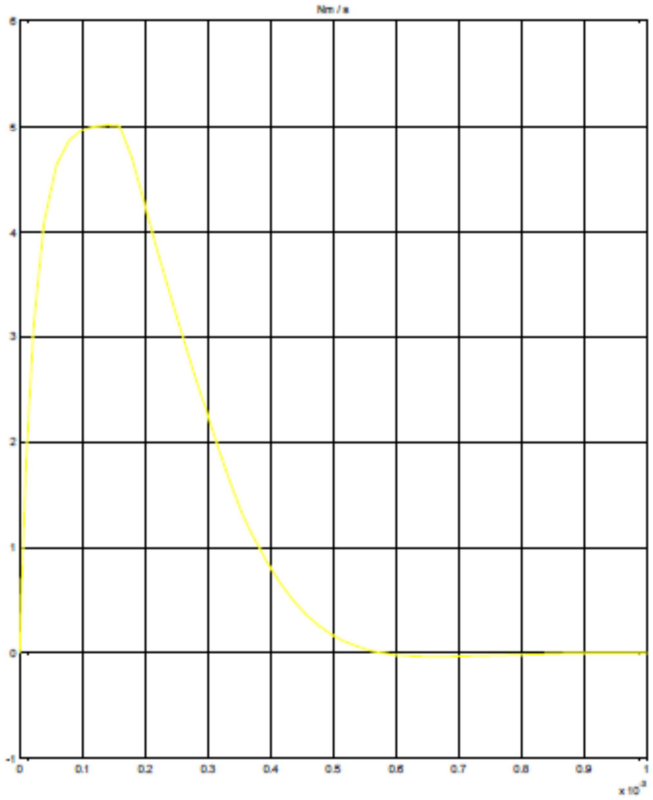
Kuva 12. Simulaatio ympäristö maximi arvoilla



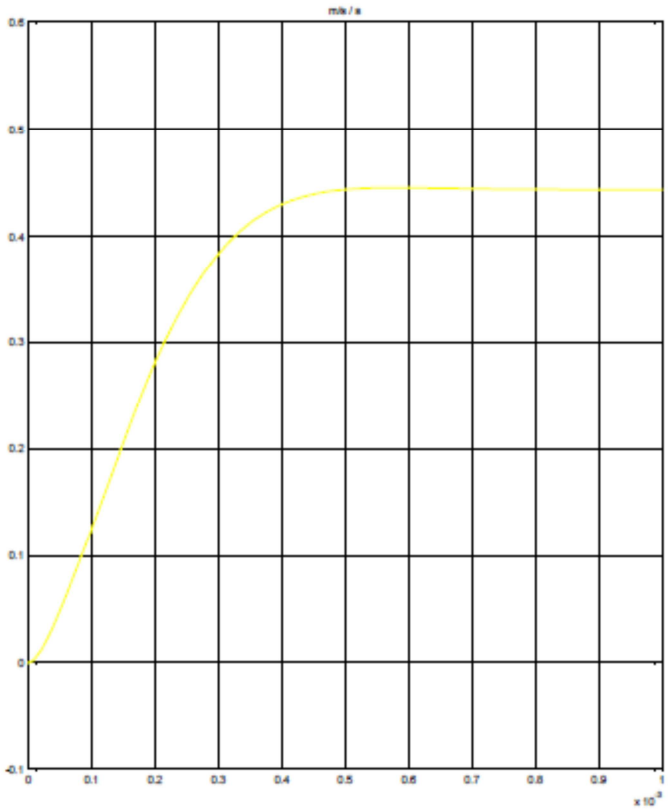
Kuva 13. Kiihtyvyys maximi arvoilla



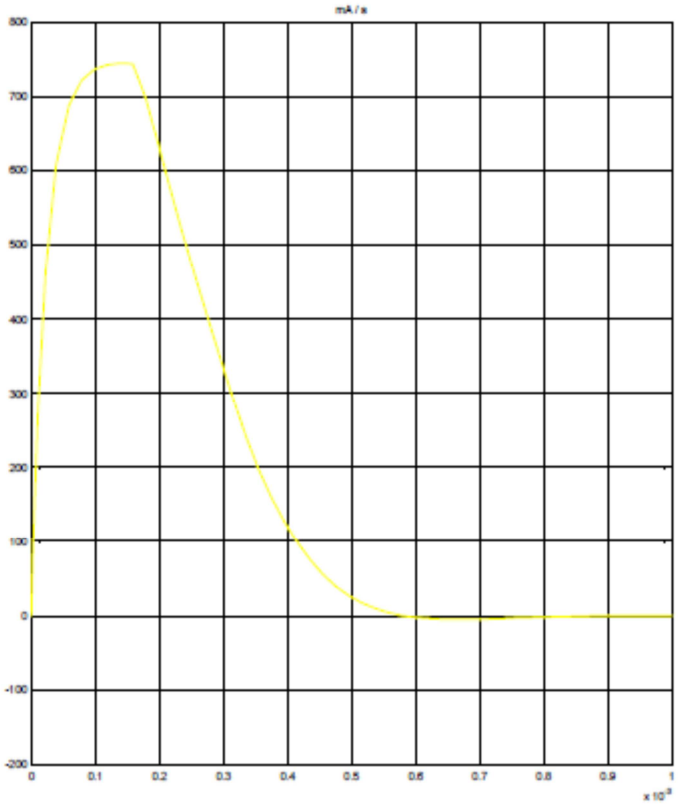
Kuva 14. Todellinen nopeus maximi arvoilla



Kuva 15. Momentti maximi arvoilla



Kuva 16. Nopeus maximi arvoilla



Kuva 17. Virta maximi arvoilla

8.1 Matlab simulaatio kaavat

V=K*w+I*R+L*di
T=K*i-b*w-J*dw

Faulhaber DC-Micromotors
Precious Metal Commutation
4.2 Watt
Encoders:IE2

w = radians/s.
r [rpm] = r * 2pii radians/min = r*2pii radians/(60 sec) =
= r*(2pii/60) radians/sec
=> w = (2pii radians/60 sec)*r tai
=> r = [60 sec/(2pii radians)]*w.

Back-EMF constant 0,725 mV/rpm
= 0.725 mV/(2pii radians/(60 sec))
= 0.725*60/(2pii)mV/(radians/sec)=6.9232 mV/(radians/sec)
= 6.93232E-3V/(radians/sec).
==========
=> Total-back-emf = r * 0,725 mV = w * x
=> x = (r/w) * 0.725 mV = [60 sec/(2pii radians)]*0.725 mV
= 6.9232 mV*sec/radians = = 6.9232E-3 V*sec/radians.

```
================================================
============

L=45e-6 H => 1/L = 2.2222e+4
================================================

J=2.7g*cm^2=2.7/1000 kg*(0.01m)^2=2.7*0.0001/1000 kg*m^2
2.7e^-7 kg m^2
=> 1/J = 3.7037e+006 /(kg m^2)
```

9 YHTEYSPALVELIMEN SELVITYS

Edellä on kommentoitu yhteyden muodostus robotille strategia ohjelmaa käyttäen. Pelin aikana on mahdollista vaihtaa robotin omia parametrejä enemmän parametreista lähteessä /4/. Robotti sisältää PID parametrejä, joita tässä insinöörityössä ei käsitellä.

`static bool portOpen=false;`	Staattinen muuttuja true / false portOpen muuttuja.
`int robotPort=5021;`	robotPort kokonaisluku muuttuja bluethooth serverin portti numero 5021.
`std::string buff0[]={"192.168.1.10"};`	Merkkijono muuttuja jossa ip osoite bluethooth server.
`std::string * buff = buff0 ;`	Buff osoitin osoittaa buff0 sisältöön.
`std::string ipaddress = buff ->c_str();`	
`static SocketClient * m_socketClient[5];`	

`if (portOpen==false)`
`{`
`try{`
`m_socketClient[3] =`
 	
`new`
	Jos portOpen on sama kuin false mennään if lauseeseen. Try eli yritetään ottaa yhteys m_socketClient(3) käytössä ollut robotti. Luodaan uusi SocketClient jäsenet ip osoite ja

SocketClient(ipaddress,robotPort);
}
catch (...)
{
m_socketClient[3] = NULL;
 	
std::stringstream out;
out << "Port number: " << robotPort << ".";
 	
MessageBox(HWND_DESKTOP,out.str().c_str(),"Error: Unable to open port",MB_OK|MB_ICONEXCLAMATION);
}
	robotin portti.
	
	catch ei sisällä mitään toimintaa.
	
	asetetaan m_socketClient(3) robotti NULL tyhjäksi.
	
	Esitellään tulostus funktio out.
	
	Tulostetaan muuttujan robotPort avulla portin numero näytölle.
	
	Tulostetaan näytölle virhe ilmoitus, jos portin aukaisemisessa tapahtuu virhe.

portOpen=true;	asetetaan portOpen true todeksi.
std::stringstream out;	Esitellään tulostus funktio out.
out << "[$][+=";	Aloitetaan komentosarja dollari merkillä ja tulostetaan se out funktiolle.

	Syötetään robotille komento sarja ” – merkin jälkeen ja lopetetaan ” –merkin jälkeen.
int targetvl = (char)5;	Esitellään targetvl eli vasemman pyörän paikka char 5.
int targetvr = (char)10;	Esitellään targetvr eli oikean pyörän paikka char 10.
int vl = (int)targetvl*6.0;	Esitellään vl kokonaisluku muuttuja vl ja annetaan vasemman pyörän nopeus kertaa 6.
int vr = (int) targetvr*6.0;	Esitellään vr kokonaisluku muuttuja ja annetaan oikean pyörän nopeus kertaa 6.
out << vl;	Annetaan vasemman pyörän nopeus out funktiolle.
out << ",";	Lisätään komento sarjaan pilkku erottamaan vasemman ja oikean pyörän nopeudet toisitaan.
out <<vr;	Annetaan vr muuttujalla oikean pyörän nopeus.
out << "]";	Päätetään komentosarja] – merkillä.

out << "[+~][~]";	Lisätään ja vähennetään pyörien nopeuksia komentosarjassa.
std::string Result;	Esitellään Result merkkijono muuttuja.
Result = out.str();	Kopioidaan tulos out.str() funktiosta Result muuttujaan.
char *sz;	Esitellään merkkijono pointteri *sz.
sz = new char[Result.length() + 1];	Katsotaan Result muuttujan pituus.
strcpy(sz, Result.c_str());	Kopiodaan Result muuttujan sisältö sz muuttujaan.

Ikkuna akna009(300,820,1000,870);
sprintf(buffer, "lähetetty string: %s\0", sz);
akna009.Kopsaa(buffer);
akna009.Nayta();
	Tulostetaan Merkkijono, joka on lähetetty robotille.

delete [] sz;	Tuhotaan sz sisältö.
m_socketClient[3]->SendBytes(out.str());	Lähetetään bitteinä merkkijono robotti 3:lle. Palautetaan arvo.
return;	
}	Jos if lause ei onnistu tehdään seuraavaa. Esitellään out merkkijono muuttuja.
else	
{	
std::stringstream out;	

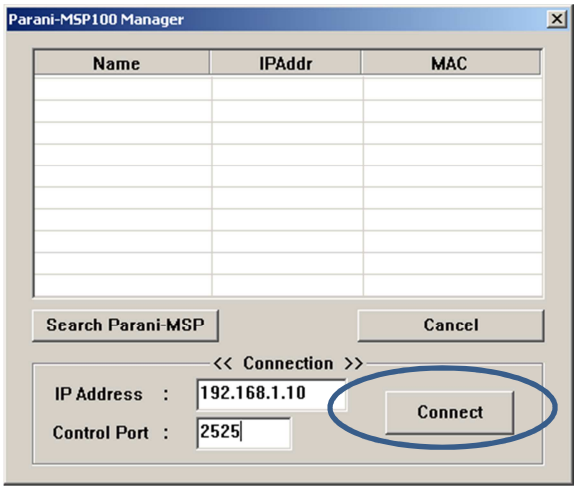
int targetvl = (char)5;
 	
int targetvr = (char)10;
 	
int vl = (int)targetvl*6.0;
 	
int vr = (int) targetvr*6.0;
 	
out << vl;
 	
out << ",";
 	
out << vr;
 	
out << "]";
 	
out << "[:][+~][~]";
std::string Result;
 	
Result = out.str();
 	
char *sz;
 	
sz = new char[Result.length() + 1];
	Esitellään vasemman pyörän paikka targetvl kokonaisluku muuttujalle char 5.
	
	Esitellään oikean pyörän paikka targetvr kokonaisluku muuttujalle char 10.
	
	Annetaan oikean pyörän nopeus vr ja targetvr kertaa kuusi.
	
	Tulostetaan out funktiolle vasemman pyörän nopeus.
	
	Annetaan vasemman ja oikean pyörän erotus merkki pilkku.
	
	Tulostetaan oikean pyörän nopeus out funktiolle.
	
	Päätetään pyörien nopeuksien anto] – merkillä.
	
	Annetaan komentosarja kääntyä vasemmalle ja sitten oikealle.
	
	Esitellään Result merkkijono muuttuja.
	
	Result sisällöksi annetaan out.str() funktion sisältö.

	Esitellään merkkijono muuttuja osoitin *sz.
strcpy(sz, Result.c_str());	Katsotaan pituus Result muuttujasta.
Ikkuna akna010(300,850,1000,900);	Kopioidaan sz sisällöksi Result.c_str() funktion sisältö.
sprintf(buffer, "lähetetty string: %s\0", sz);	
akna010.Kopsaa(buffer);	Tulostetaan lähetetty
akna010.Nayta();	merkkijono näytölle.
delete [] sz;	Tuhotaan sz osoitin muuttuja.
m_socketClient[3]->SendBytes(out.str());	Lähetetään robotille bitteinä merkkijono out.str() funktiolla.
std::string testi = m_socketClient[3]->ReceiveLine();	Haetaan dataa robotti 3:lta ReceiveLine() funktiolla ja tallennetaan se testi merkkijono
sz = new char[testi.length() + 1];	muuttujaan.
strcpy(sz, testi.c_str());	Katsotaan testi merkkijonon pituus ja tallennetaan se sz muuttujaan.

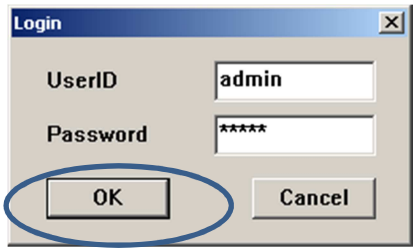
Ikkuna akna008(300,880,1000,930);
sprintf(buffer, "Testi %s\0", sz);
akna008.Kopsaa(buffer);
akna008.Nayta();
delete [] sz;
return;
	Tulostetaan sz muuttujan merkkijono tietokoneen näytölle sprintf funktiolla.
	Tuhotaan osoitin muuttuja sz kokonaan.
	palutetaan ohjelman arvo.

Tutkimme bluethoot palvelimen toimintaa ilman robottien kommunikointia ja robottien kommunikoinnin kanssa. Mitä oikastaa tapahtui ohjelmassa ja kuinka se otettiin käyttöön.

Ohjelman käynnistäminen: Käynistys valikko sieltä parani-msp100 josta parani-msp100 manager v1.0.2.

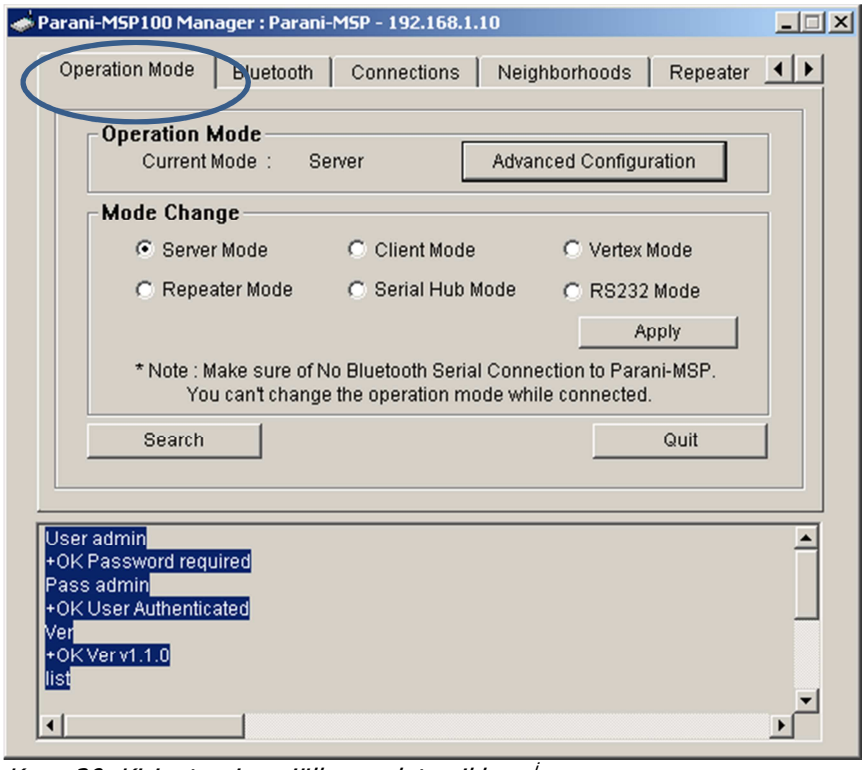


Kuva 18. Yhteyden avaus bluethooth serverille

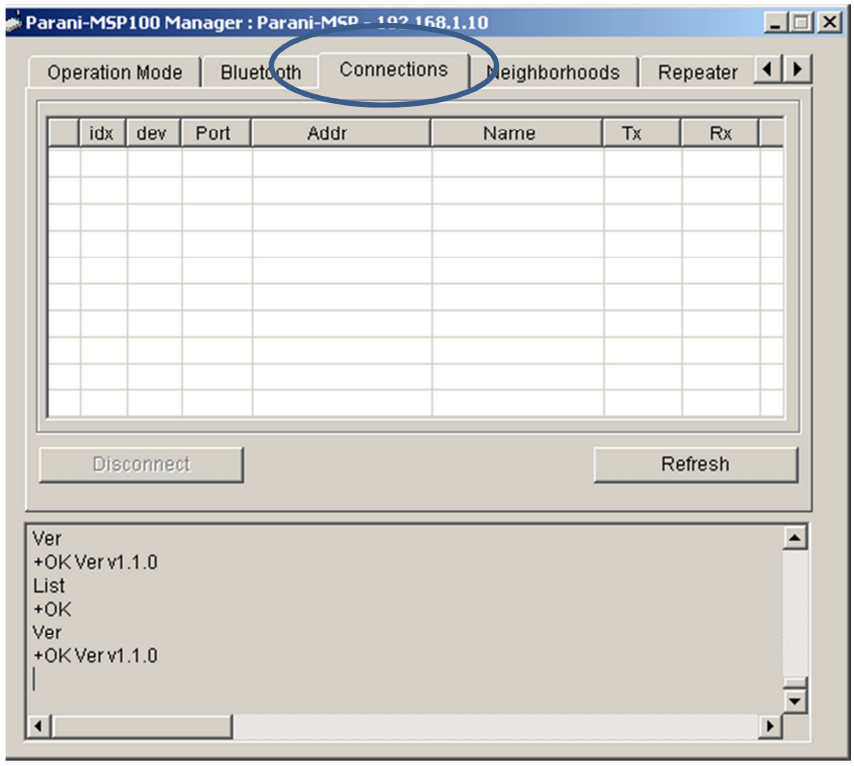


Kuva 19. Kirjautuminen bluethooth serverille

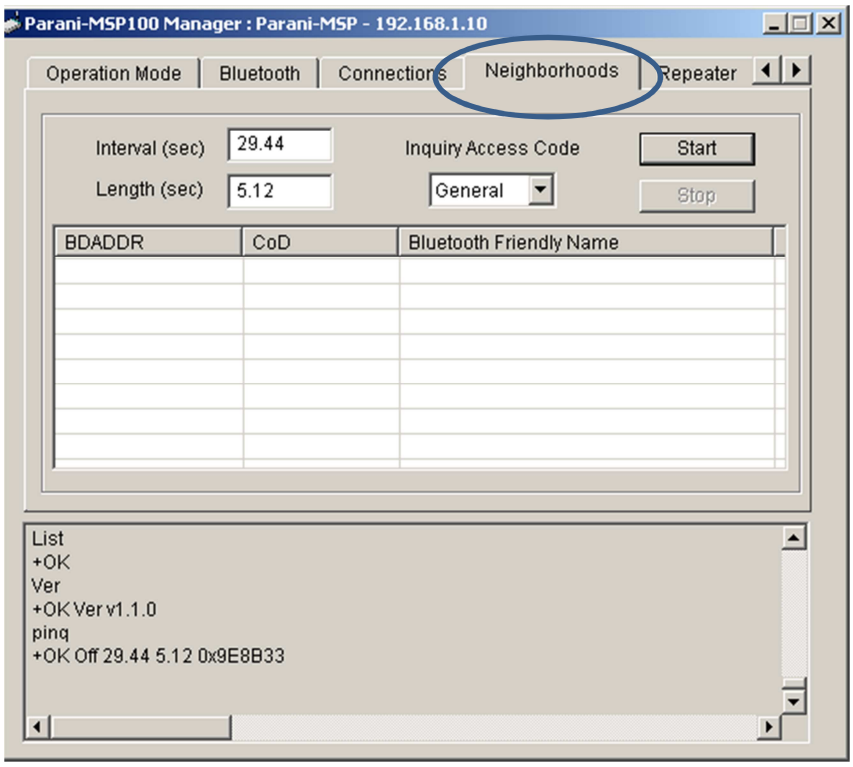
Kirjaudu sisään käyttäjä tunnus "admin" ja salasana admin. Paina "OK"



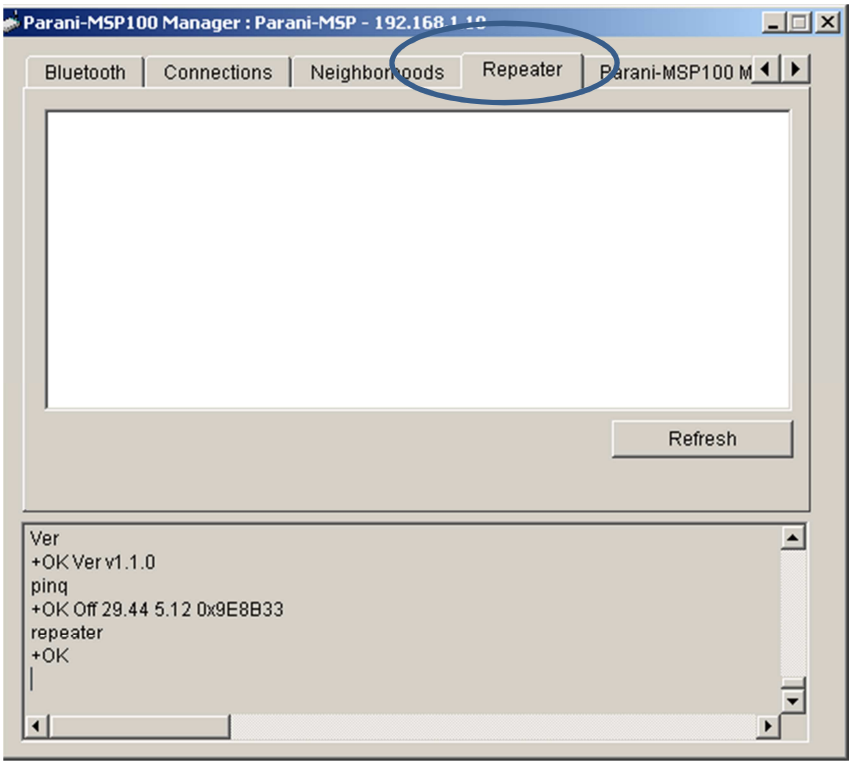
Kuva 20. Kirjautumisen jälkeen oletus ikkuna



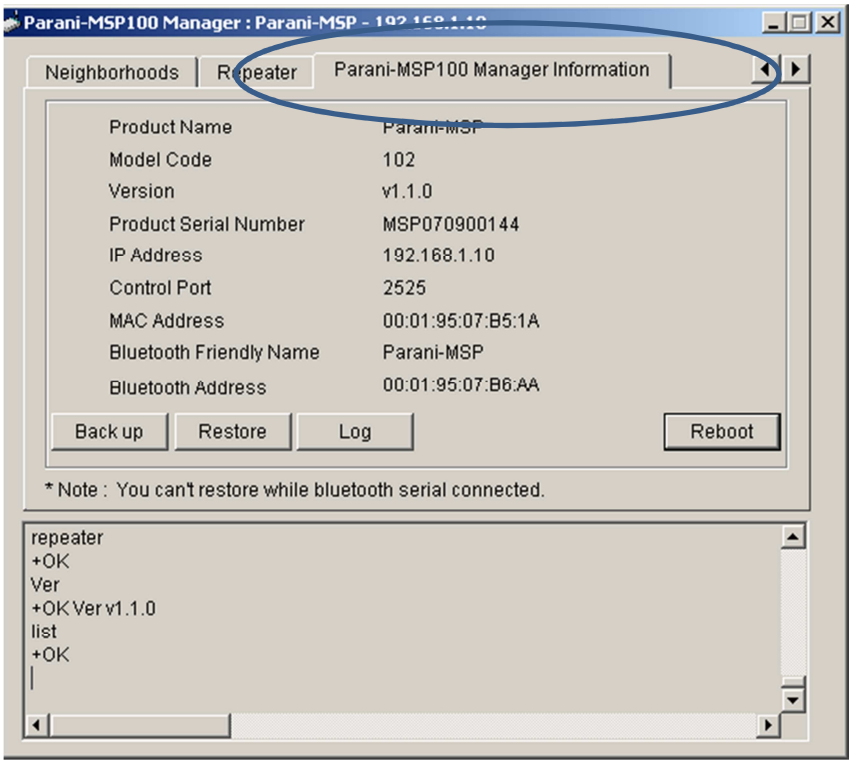
Kuva 21. Yhteys välilehti



Kuva 22. Neighborhoods välilehti

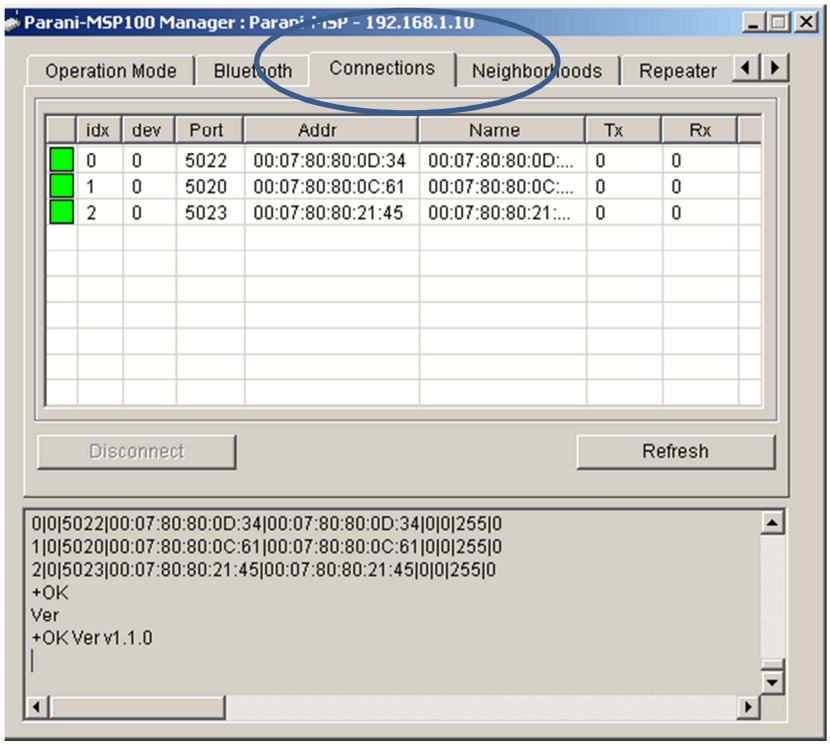


Kuva 23. Repeater välilehti

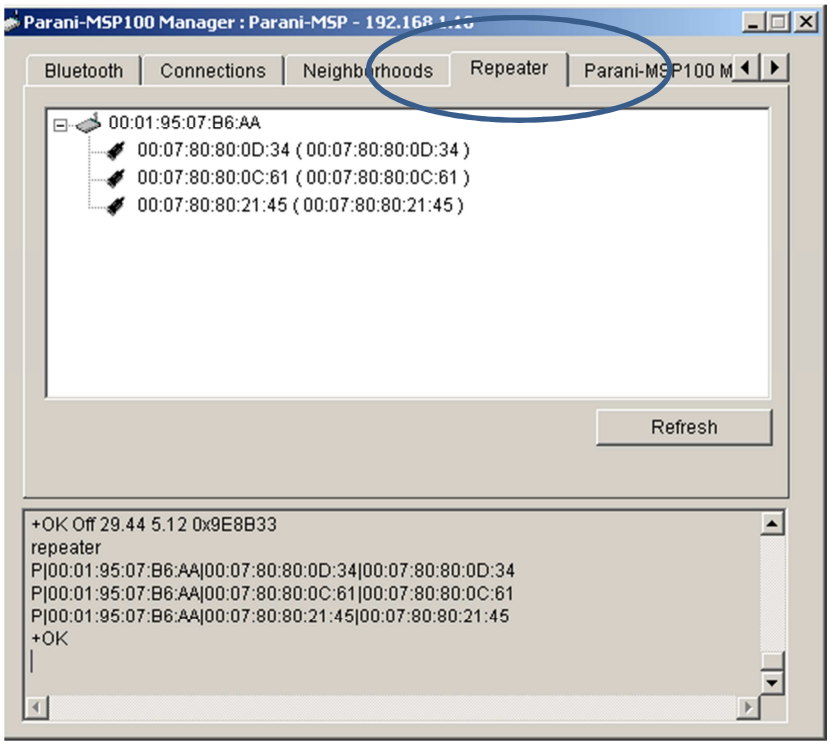


Kuva 24. Bluethoot serverin info sivu

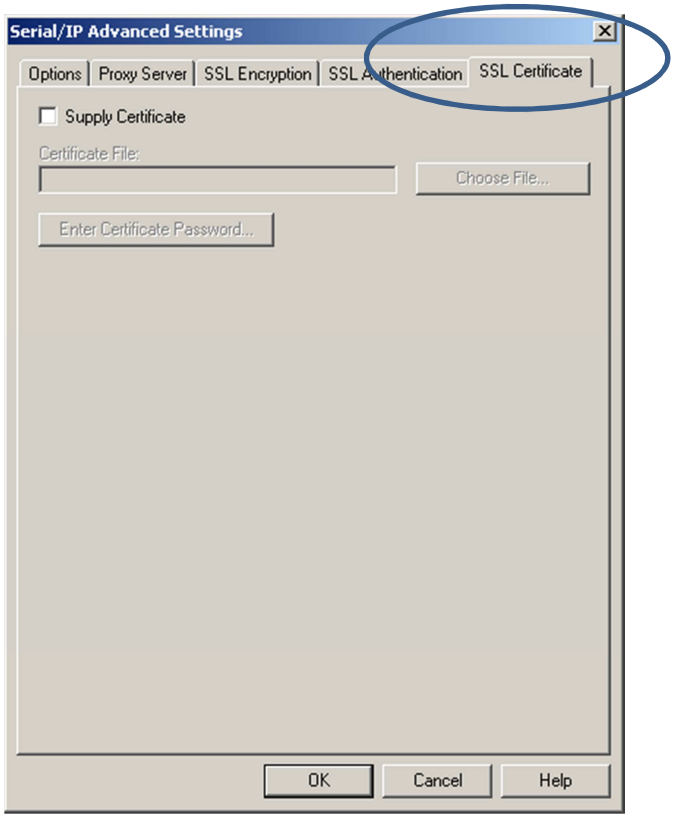
Seuraavaksi robotit käynnistettiin (Sininen joukkue).



Kuva 25. Robottien yhteys välilehti

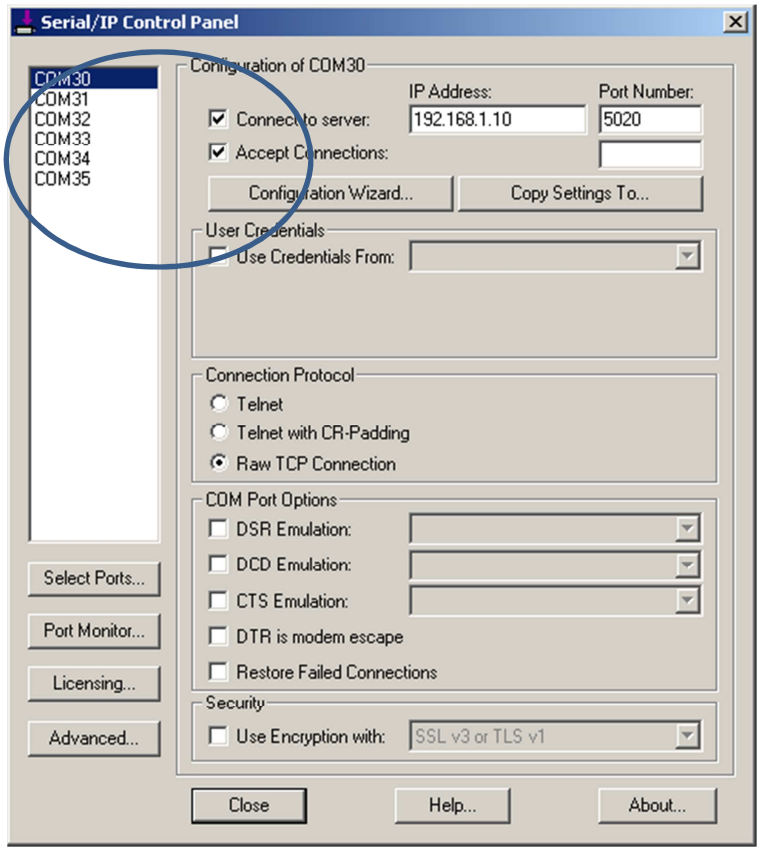


Kuva 26. MAC - osoitteiden näkymä repeater



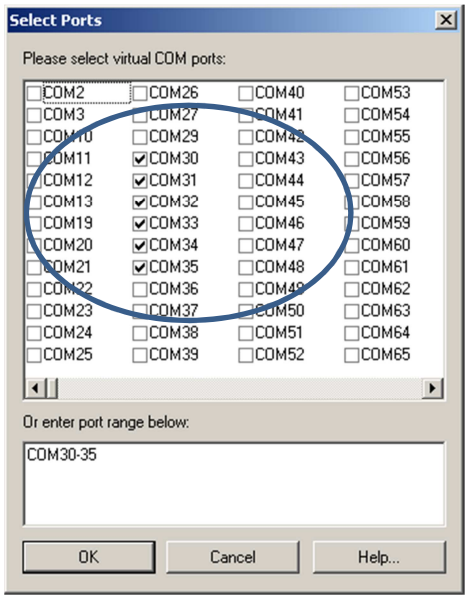
Kuva 27. Serial / ip asetukset

Serial – IP kansio, josta saadaan Control Panel ohjelma käyntiin.

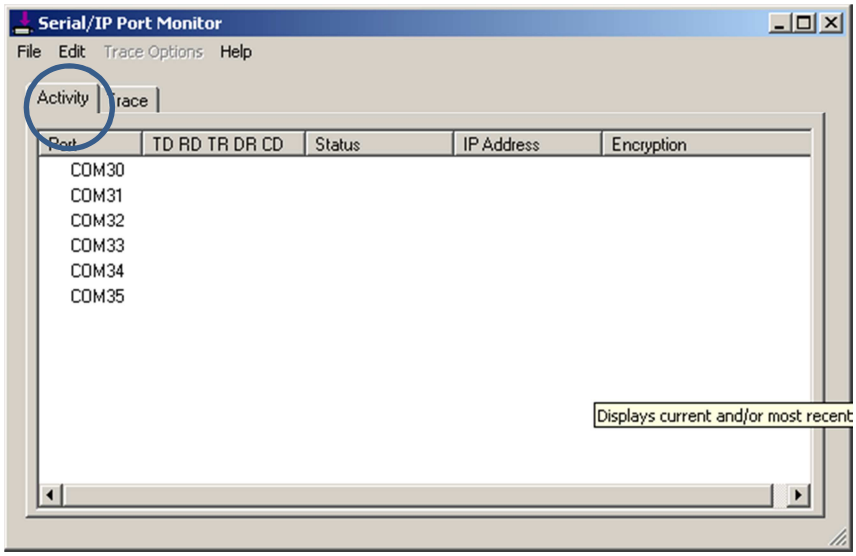


Kuva 28. Serial - ip control panel

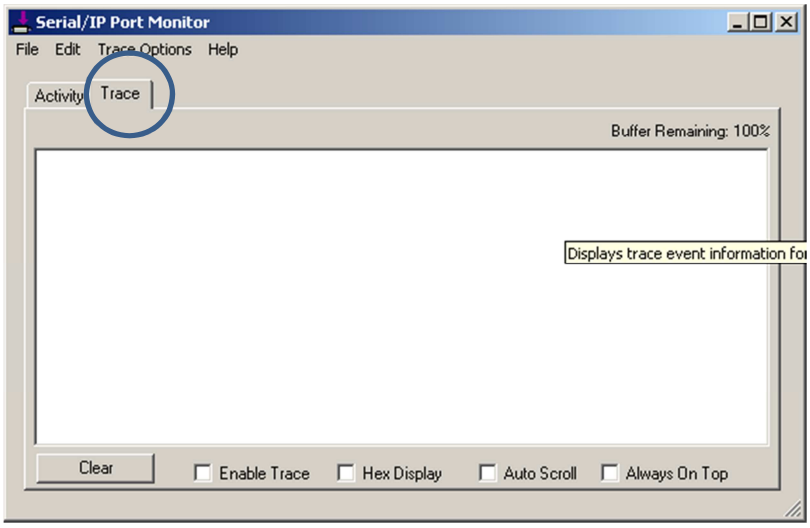
Sarjamuotoinen yhteys COM31 portti vastaava socket numero 5021.
Sarjamuotoinen yhteys COM32 portti vastaava socket numero 5022.
Sarjamuotoinen yhteys COM33 portti vastaava socket numero 5023.
Sarjamuotoinen yhteys COM34 portti vastaava socket numero 6020.
Sarjamuotoinen yhteys COM35 portti vastaava socket numero 6021.



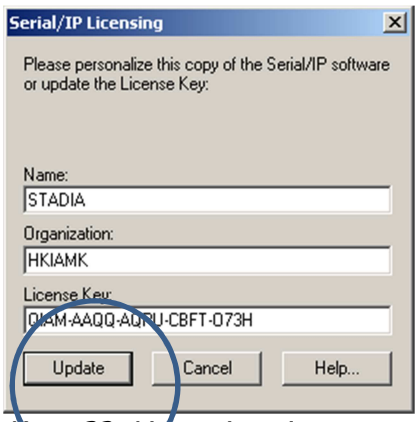
Kuva 29. Porttien valinta ikkuna



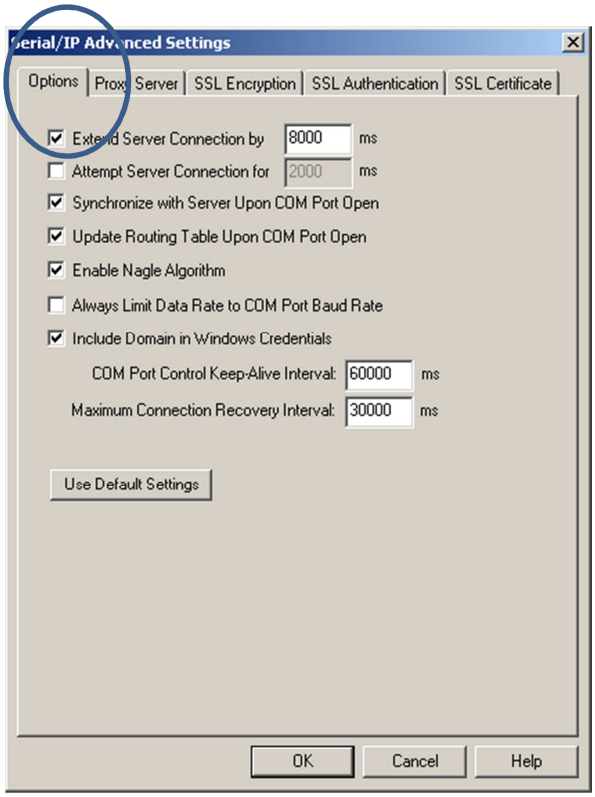
Kuva 30. Porttien monitorointi



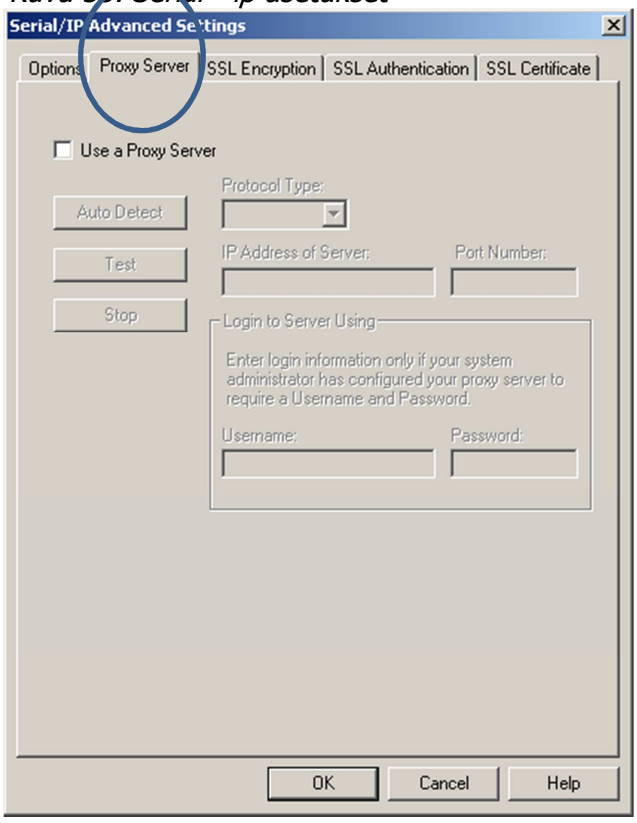
Kuva 31. Porttien monitorointi jäljitys ikkuna



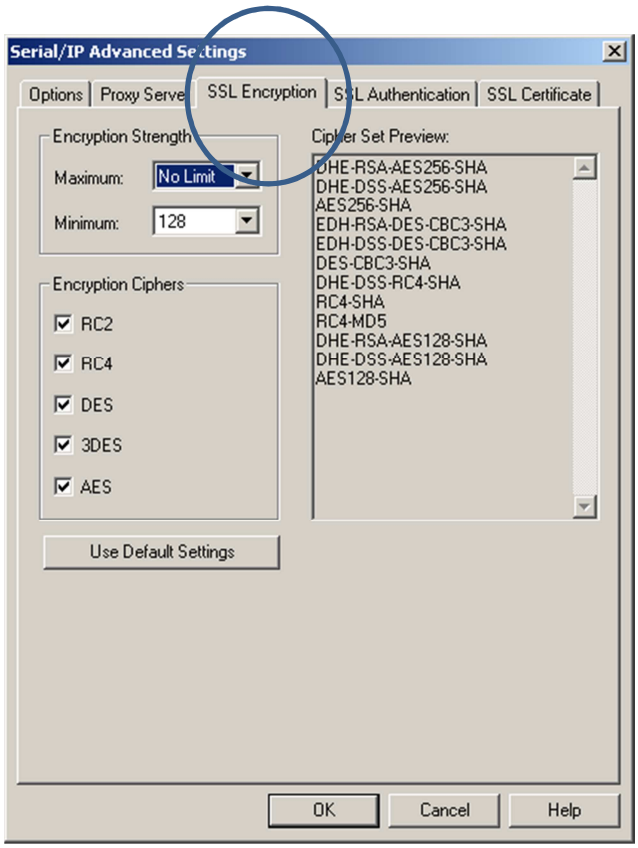
Kuva 32. Lisenssi sopimus



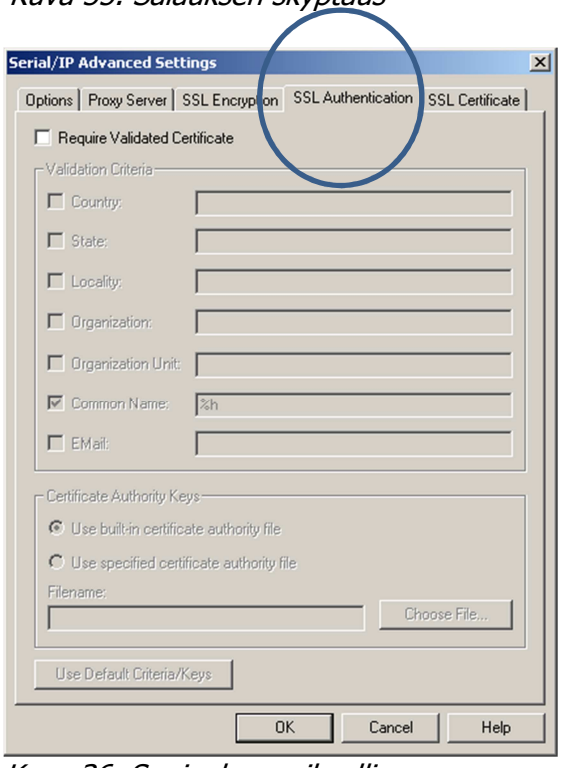
Kuva 33. Serial - ip asetukset



Kuva 34. Serial - ip proxy server asetukset



Kuva 35. Salauksen skyptaus



Kuva 36. Suojauksen oikeellisuus