Tallinna Ülikool
Digitehnoloogiate Instituut

UURIMUS KIRJUTUSVAHENDIT KASUTAVA PRINTERI LOOMISEST ARDUINO ABIL

Seminaritöö

Autor: Kardo Jõeleht
Juhendaja: Jaagup Kippar

Autor: ..... „ ..... „ 2016
Juhendaja: ..... „ ..... „ 2016
Instituudi direktor: ..... „ ..... „ 2016

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Käesolevas töös annan suvalist kirjutusvahendit kasutava printeri näitel ülevaate sellest, kuidas oleks teoreetiliselt ja ka praktiliselt võimalik luua seadet, mille põhiloogika on aluseks nii printeritele, 3D-printeritele, plotteritele kui ka APJ- ehk CNC-pinkidele. Üldistatult on niisugune tehnoloogia kasutusel igasugusel arvuti poolt juhitud operatsioonil, kus füüsilist seadet on vaja täpselt eri suundades liigutada.

Pealkirjas on sõna uurimus, kuna käesolev töö ei sisalda sajaprotsendilist infot, kuidas selline töötav seade luua ja seda igapäevaselt efektiivselt kasutada. Tegemist on pigem eksperimenteerimise ja katsetustega, prototüübi loomisega. Eesmärgiks on kogeda võimalikult palju projektiga seotud erinevaid probleeme ja neile lahendusi leida.

Kuna süsteemi kontrolleriks olen valinud Arduino Uno, annan esiteks väikese ülevaate kontrolleri võimalustest.

Teiseks tutvustan sarnaseid teiste valmistatud lahendusi. Ma ei kasutanud neid kui juhendeid, vaid panin mitme süsteemi peale kokku visiooni oma lahendusest.

Edasi tuleb arutelu materjalide ja koostamise teemal ning töö käigus minu poolt loodud seadme erinevate arendusetappide kirjeldus. Töö selles peatükis kirjeldan esiteks iga etappi ja teiseks toon välja tähtsamad asjad, mida protsessi käigus õppisin ja mida tuleks kindlasti tähele panna või tulevikus sarnast projekti tehes teisiti teha.

Töö vajalikkus on põhjendatud Arduino populaarsuse pideva kasvuga (Lisa 1.) ja suure isikliku huviga. Tahtsin kasutada võimalust kogeda huvitavat protsessi ja oma tööga nii endale kui teistele robootikahuvilistele midagi pakkuda.

Suunatud on kirjutis eelkõige kõigile neile, kes plaanivad ise mõne robootikalahenduse disainida ja ka ehituse ette võtta, aga ka neile, kes lihtsalt millegi huvitavaga tutvuda tahavad.

1. Arduino/Genuino Uno tutvustus

Tahaksin esiteks selgitada, et Arduino tootjad lõid hiljuti tütarkaubamärgi Genuino. Edaspidi hakatakse Euroopa, Aasia, Kanada, Lõuna-Ameerika ja Aafrika turgudel müüma Genuino tootemargi all olevaid tooteid, kuid sisuliselt on need samad seadmed. Arduino kaubamärk jääb identifitseerima USA's müüdavaid tooteid. Kuna aga Arduino kaubamärk on hetkel ikkagi tuttavam, kasutan töös Arduino nime.

Kasutan töö vältel nimesid Arduino Uno ja lihtsalt Arduino, kuid mõeldud on ikkagi Arduino Uno't. Kirjeldustes tekitab Uno lisamine liigset müra.

Arduino Uno on trükiplaat elektroonikaseadmetega, terviklik programmeeritav kontrollersüsteem, mille südameks on 32 KB mäluga [Atmel Corporation, 2015] ATmega328P mikrokontroller. Lisaks mikrokontrollerile on plaadil 14 sisend-/väljundpesa, 1 USB-pesa arvutiga ühendamiseks ja muud vajalikud toetavad komponendid. Protsessori kiiruseks on 16 MHz [Arduino.cc, 2016]. Kui Arduino Uno võimalustest väheks jääb, on Arduino'sid veel mitmeid erinevaid. Lisaks pakutakse palju erinevaid laiendusi ja lisamooduleid.

Tegemist on prototüüpimiseks mõeldud kontrolleriga, mis on odav ja mida on lihtne kasutada. Lisaks riistvarale tulevad kaasa ka Java'le sarnane programmeerimiskeel ja IDE, millega on mugav programmikoodi luua ja kood otse ka mikrokontrollerisse laadida.

Igasse pessa on võimalik mõni elektroonikakomponent ühendada ja programmikoodis iga pesa kohta määrata, kas tegemist on väljundi või sisendiga, mis annab võimaluse kas lugeda sissetulevat pinget või saata pinget välja vastavalt programmis tehtud otsustele.

Näiteks LED-lambi süütamine kulgeb Arduino Uno't kasutades järgmiselt. Esiteks tuleb ühendada LED'i pluss ~250 Ω takisti ühte otsa. Teine takisti ots tuleb ühendada näiteks Arduino pesasse 1 ja LED'i miinus pesasse GND. Järgmiseks määrata koodis pesa 1 väljundiks ja lülitada pesa 1 sisse ehk anda temale pinge. Programmi süsteemi laadides hakkab LED põlema.

Sarnaselt on võimalik ühendada kõikvõimalikke elektroonikaseadmeid, mis mahuvad Arduino Uno poolt teenindatavasse pinge- ja vooluvahemikku, milledeks on vastavalt 6-20V süsteemipinget ja 20mA voolu ühe pesa kohta [Arduino.cc, 2016]. Suuremate voolude korral, nagu ka minu projekti puhul, saab kasutada välist toiteallikat ja Arduino abil ainult loogikaga tegeleda.

Arduino'sse laetav kood on reeglina ülesehitatud kolmes osas, nagu toodud järgmises koodinäites.

```
int muutuja1 = 3;
int muutuja2 = 5;
void setup(){
}
void loop(){
}
```

Kõigepealt tulevad muutujad, mida programmi töö käigus kasutatakse. Seejärel säteteplokk `setup()` ja viimaseks igavene tsükkel `loop()`. `setup()` „käiakse” programmi käivitamisel ühe korra läbi. Selles osas määratakse tavaliselt sisend-/väljundpesad, aga tehakse ka algväärtustamised või näiteks *Serial*-ühenduse käivitamine, kui vaja. `loop()` plokk on, nagu nimigi ütleb, tsükkel, mida „käiakse” programmi töö käigus aina uuesti ja uuesti läbi. Töö lõppeb Arduino väljalülitamisel või uuestilaadimisel.

2. Teiste poolt loodud lahendused

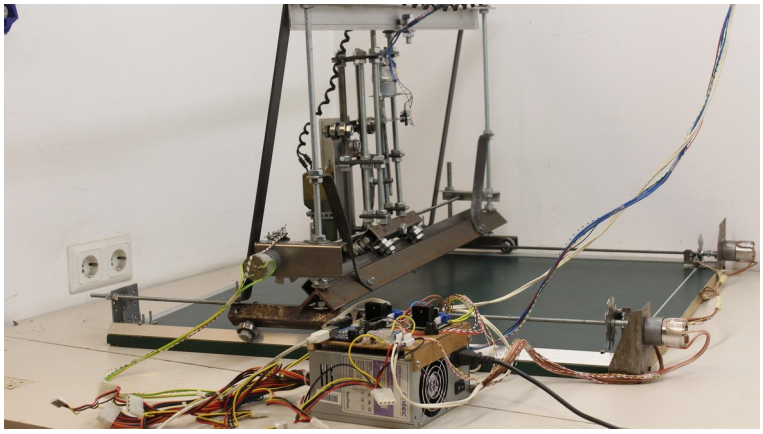
Enne praktilise poolega alustamist uurisin mitmeid allikaid. Tegu on YouTube keskkonnas olevate videotega, kuna sellistest lahendustest saab video abil parima ülevaate. Käesolevas peatükis kirjeldan mõned minu tööd enim mõjutanud teostused.

2.1. APJ-pink lihtsate tööriistade abil

Videos [HomoFaciens, 2015] näidatakse, kuidas lihtsate tööriistade abil luua Arduino Uno poolt kontrollitud APJ-pink, mis on näha illustratsioonil 2.1. Põhilisteks tähelepanupunktideks on platvormide siinide disain ja osutamine asjaolule, et konstruktsioon peab olema piisavalt tugev. Isegi kergemate asjade liigutamiseks peab konstruktsioon olema üllatavalt tugev, et saada täpseid tulemusi.

Platvormid toetavad kuullaagritele, mis omakorda sõidavad nurkraua peal. Kuullaagreid on kasutatud kui rattaid. Selline lahendus tagab platvormi samal kõrgusel püsimise ja viib kõikumise minimaalseks.

Liikumist juhitakse keermelattide abil, mis on kõige odavam lahendus, kuid aeglane, ja nagu videos ka selgub, üks vetrumise ja ebastabiilsuse allikaid.



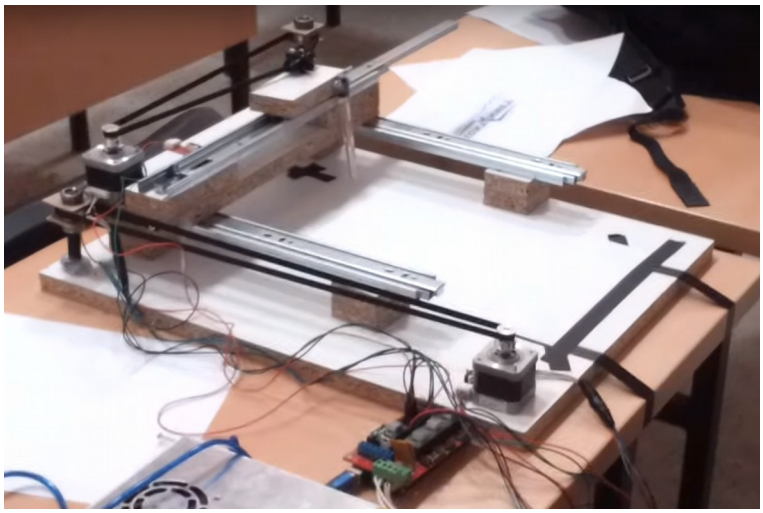
Illustratsioon 2.1: APJ-pink lihtsate tööriistade abil, http://www.homofaciens.de/technics-machines-cnc-v2_en_navion.htm, 25.02.2016

Lisaks eelnevale on siin kasutatud tavalisi mootoreid ja indekseeritud nende asukohta ketaste ja sensorite abil, kuid sellesse ma pikemalt ei süvenenud, kuna otsustasin juba varakult ära, et

kasutan oma süsteemis sammmootoreid, millede puhul on võimalus läbitud sammude arvu järgi teada saada, kus täpselt mootor asub, ilma seda väliste sensorite abil jälgimata.

2.2. Rihmadega Arduino plotter

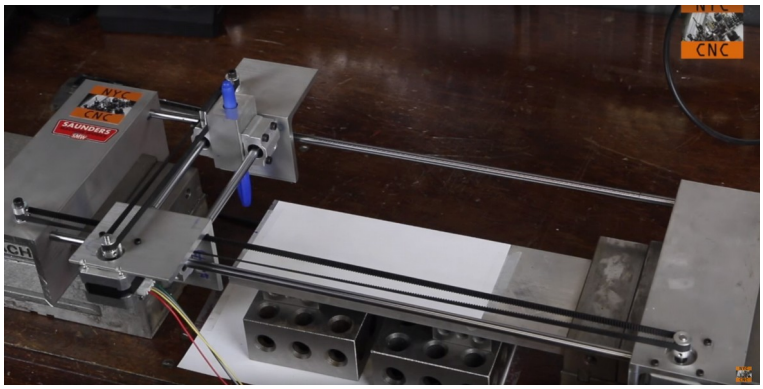
Videos [Veli Ahmet AKTEKE, 2014] ja illustratsioonil 2.2 on näha, kuidas platvormid on pandud sõitma sahtlite siinide peal. Märksõnaks siin on rihmade kasutamine liikumiseks, mis tagab kiirema liikumise ja nõuab mootoritelt väiksemat pöörete arvu. Pea liigutamiseks on kasutatud sammmootoreid.



Illustratsioon 2.2: Arduino plotter, https://www.youtube.com/watch?v=kkgxDEXfE_k, 25.02.2016

2.3. Arduino APJ-pink

Selles videos [NYC CNC, 2015] jääb asjast juba tööstuslikum mulje, nagu on näha ka illustratsioonil 2.3. Platvormid libisevad mööda ümarrauast siine ja nende liigutamiseks kasutatakse rihmasid. Viis, kuidas kogu süsteem on kokku monteeritud ja detailid lõigatud, tekitab kalli ja mitte enam nii koduse tunde. Üles ja alla liikumist ei ole antud videos tehtud, aga süsteemi täpsus ja kiirus on märkimisväärsed. Mootoritena kasutatakse sammmootoreid.



Illustratsioon 2.3: Arduino APJ-pink,

https://www.youtube.com/watch?v=WKmzlMvxC7I,

25.02.2016

3. Kirjutusvahendit kasutava printeri loomine

Selles peatükis kirjeldan etapphaaval, kuidas panin paika esialgse plaani, kui palju see töö käigus muutus, ja kuidas kõik omavahel kokku sobis või ei sobinud. Igas alapeatükis toon välja protsessi sammud, mida tulemuse saavutamiseks läbisin ja kirjeldan lahendusi, millede osas oma kogetu läbi targemaks sain. Samuti suunan tähelepanu asjadele, mida tuleks parema tulemuse saavutamiseks jälgida.

3.1. Esialgne plaan

Väga üldistatult oli mõte järgmine:

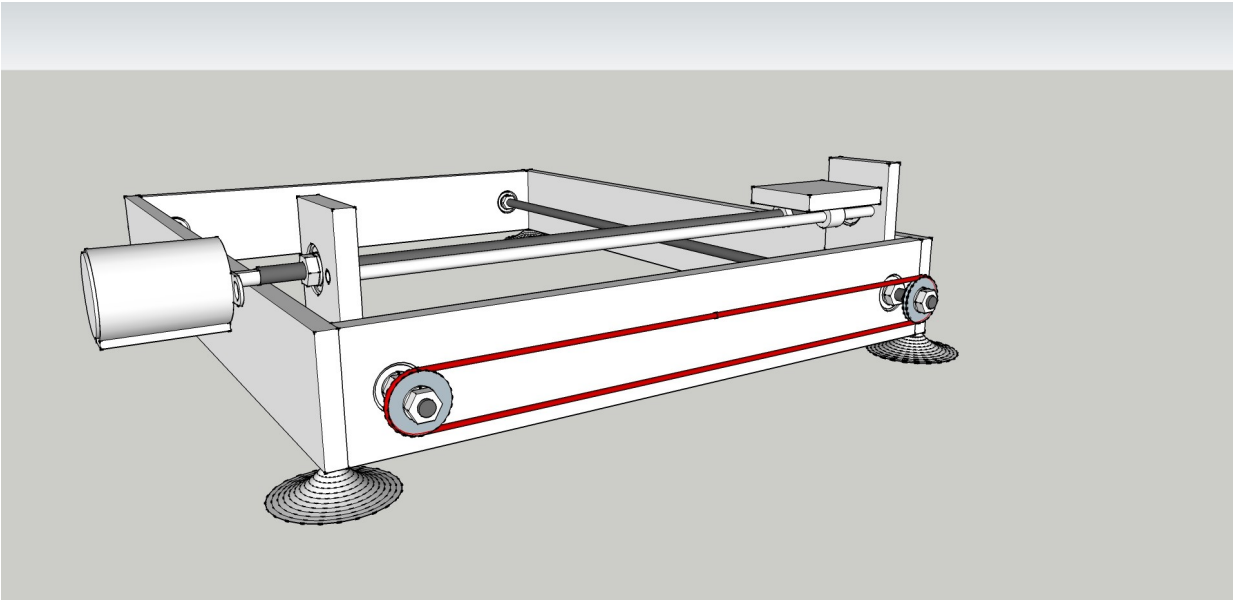
Kaks mootorit, kontroller, tarkvara ja mõned liikuvad osad. Asjad omavahel kokku ühendada, kinnitada kirjutajaks näiteks marker ja anda vastavalt ekraanil olevale svg (*Scalable Vector Graphics)* formaadis vektorgraafikakujutisele süsteemile liikumisjuhiseid. Tulemuseks oleks justkui käega kirjutatud kujutis ja alati ilma vigadeta ükskõik millisel pinnal.

Mitmed nädalad möödusid enne esimeste sammude tegemist planeerides ja infot otsides.

Esimene asi, mis oli kindel, oli vajadus mingisuguse raami järele. Kõigepealt planeerisin peas igasuguseid keerukaid lahendusi, aga tarkvaraarendusest olen õppinud, et süsteemid peavad koosnema võimalikult lihtsatest osadest, mistõttu sain aru, et tegelikult piisab tavalisest neljakandilisest ilma põhjata kastist. Neli külge, mida planeerisin lõigata alumiiniumlatist, pidid olema piisavalt pikad, et raam saaks ümbritseda A4 paberit ja platvorme.

Otsustasin varakult ära, et platvormide liikumiste kontrollimisel tahtsin kasutada keermelatte. Arvasin, et spetsiaalsete piisavalt võimekate rihmade hankimine oleks probleem. Koduste vahenditega rihma ja rihmaratast olin proovinud varem teha ja igasuguse vähegi suurema koormuse korral ei andnud need minu kogemuse põhjal tulemust. Tugevusest jääb tavaliselt puudu. Kuna selleks, et laia ristplatvormi liigutada, oli vaja kaks keermelatti sünkroonis liikuma panna, jätsin ühe rihma ikkagi disaini sisse, sest paremat lahendust ei leidnud. Kui ühel pool platvormi oleks olnud vedav keermelatt ja teisel pool ainult siin, oleks süsteem ilmselgelt liiga ebastabiilne jäänud. Nagu näha illustratsioonil 3.1, plaanisin panna kaks alumist, pikemat keermelatti, sõidutama ristplatvormi. Tumedad on joonisel keermelatid ja hele on ristplatvormil edasi-tagasi liikuva väikese platvormi stabilisaator. Alumiste keermelattide ühes otsas on

rihmarattad liikumiste sünkroniseerimiseks, kuna mõlema keermelati liikumiseks tahtsin kasutada ühte mootorit.



Illustratsioon 3.1: Esialgne plaan

Raami lühemate külgede sisse oleksid puuritud 18 mm läbimõõduga avad, millede sisse keemilise metalliga liimitud 18 mm välisläbimõõduga kinnised kuullaagrid. Kuullaagrites oleva ava läbimõõt oleks 6 mm. Täpselt sama, nagu kasutatud keermelattide välisläbimõõt. Keermelatid oleksid asetatud kuullaagritest läbi ja kinnitatud mõlemalt poolt kuullaagri külge mutri ja kontramutriga. Kontramutter ei laseks kinnitusmutril lahti tulla.

Illustratsioonil 3.1 on ka alumiste keermelattide sünkroniseerimiseks mõeldud rihma rihmarattad kinnitatud mutritega, mis oli esimene mõte, aga hiljem kasutasin hoopis 3D-printerites kasutatavaid GT2 20T rihmarattaid ja GT2 520 mm pikkust täpsusrihma.

Ristplatvormil liikuv väike platvorm liiguks samuti keermelati abil. Kui keermestatud detail kinnitada platvormi külge, keermestatud detailist panna läbi keermelatt, pöörata keermelatti ja takistada platvormil endal kaasa pöörlemast, hakkaks platvorm mööda keermelatti vastavalt selle pöörlemissuunale ühele või teisele poole liikuma.

Ristplatvormil endal ei oleks pöörlemise takistamiseks stabilisaatorlatti vaja, kuna ta oleks kahe keermestatud detaili abil kinnitatud keermelattide külge. Kui nüüd ühe keermelati otsa mootor paigutada ja kaks keermelatti omavahel rihmaga ühendada, hakkaks mootori pöörlemisel

lattidel olev platvorm vastavalt pöörlemissuunale ühes või teises suunas liikuma.

Platvormidel kasutatavateks keermestatud detailideks tahtsin kasutada mutreid, liimides need platvormide külge.

3.2. Põhikomponendid ja nende hankimine

Algatuseks proovisin kokku panna võimalikult täpse pildi sellest, milliseid detaile oleks kohe vaja, et saaks asjaga tegelema hakata. Kõige esimeseks uurisin välja, et mulle sobivad tõenäoliselt NEMA17 mootorid, mida kasutatakse ka populaarses avatud litsentsiga 3D-printeri projektis nimega RepRap [RepRap Team, 2015]. Alguses ma ei teadnud, missuguse suurusega või näitajatega mootor oleks võimeline minu seadmeid liigutama, aga kui olin otsustanud RepRap projektis kasutatava NEMA 17 kasuks, teadsin, et valitud mootorist on minu projekti jaoks kindlasti piisavalt. Lühend NEMA tähendab *National Electrical Manufacturers Association* [NEMA, 2016] ja 17 mootori standardiseeritud „otsaplaadi" või kinnitusklambri suurust [PBC Linear, -]. Kuigi sama nime alt võib leida pisut varieeruvaid omadusi, ei tekitanud see minu projekti puhul muret. Mootorid tellisin Ebay abil Hiinast, kahe mootori hinnaks kokku umbes 22€. Eestist ostes oleks hind võinud ka sadadesse eurodesse minna.

Kuna kui kaks detaili kokku suruda ja seejärel proovida ühte neist pöörlema panna, ei õnnestu see hästi, eraldub palju soojust ja materjalid kuluvad, teadsin ka, et kindlasti on vaja kuullaagreid. Kuullaagrite sisse plaanisin algusest peale mutritega fikseerida keermelatid. Kuullaagrid tellisin Ebay'd kasutades Inglismaalt. 10 laagri hinnaks umbes 5€.

Alguses rihma ja rihmarataste peale nii väga ei mõelnud, kuna keskendusin rohkem raami kokkupanemisele ja mõtlesin, et küll mingi lahenduse ikka leiab. Kui aga reaalne vajadus kätte hakkas jõudma, uurisin ja leidsin, et GT2 rihmad ja rihmarattad [Stock Drive Products / Sterling Instrument, 2016], mida kasutatakse samuti nagu minu valitud mootoreidki, erinevates täpsusprojektides, ka RepRap projektis, sobivad mulle väga hästi. Tellisin need Ebay abil Hiinast, 5 rihma maksid umbes 16€ ja 4 rihmaratast umbes 5€.

Siis muidugi Arduino Uno, mille ostsin Eestist 26€ eest. Hiljem sain teada, et vaja on ka mootorite kontrollereid ja valisin A4988 kuni 2 A voolutugevusega kontrolleri. Kaks niisugust oli vaja ja kokku läksid need maksma umbes 12€. Ostsin Eestist, Oomipoest. Valik langes sellele tootele põhiliselt seetõttu, et YouTube'i sirvides on näha, et NEMA 17 mootoritega on A4988 kontrollereid kasutatud ja need on ülesandeks sobivad.

Ülejäänud detailid, nagu 6 mm läbimõõduga keermelatt, M6 mutrid ja muud kinnitusvahendid, plekk, olid kas kodus olemas või sai need erinevatest Eesti ehituspoodidest ostetud.

Lisaks kõigele eelnevale oli vaja mingisugust mootorit ka kirjutusvahendi üles-alla liigutamiseks ja selleks tellisin samuti Ebay abil Hiinast 28BYJ-48 sammmootorid (3tk, kuna pakis oli 3) koos ULN2003 kontrolleritega, mis iga mootoriga kaasas olid. Need mootorid on oluliselt nõrgemad kui NEMA 17. Arvasin, et üles-alla liikumiseks on nad sobivad. 3 mootorit kokku maksid 7€. Võrdluseks on üks täpselt sama mootor koos kontrolleriga 17.02.2016 seisuga Oomipoes müügil 10€ eest.

3.3. Toiteplokk

Toiteplokk oli kindel plaan ise teha. Olin Internetist uurinud, et arvuti toiteplokist on lihtne 3 V, 5 V ja 12 V pakkuvat, olenevalt toiteploki võimsusest, üllatavalt suure suutlikkusega toiteplokki teha. Mina kasutasin 250 W võimsusega toiteplokki, mille maksimaalne väljundvool 12 V puhul on 9 A, minu projektis on vaja maksimaalselt 3 A. Väiksemate pingete puhul on maksimaalsed voolutugevused veel suuremad.

Toiteplokis on vaja järgmised juhtmed grupeerida – kollased, punased, oranžid ja mustad. Mustade juurde lisada üks ja ainus roheline ja ülejäänud võib ära lõigata. Must on miinus ehk maandus (*Ground, GND*), kollased +12 V, punased +5 V ja oranžid +3.3 V [Sitnalta, -]. Nimetatud juhtmeid võiks ka vähemaks lõigata, kuid kui soovida säilitada seadme maksimaalsed näitajad, peaks kõik mustad, kollased, punased ja oranžid juhtmed alles jätma.

Illustratsioon 3.2: Toiteplokk, väljundid

Et toiteplokk oleks ülejäänud süsteemist lahus ja viisakalt liidestatud, lisasin talle väljundid,

nagu näha illustratsioonil 3.2. Lisaks organiseeritusele ja heale väljanägemisele võimaldab see toiteplokki hiljem mugavalt ka mujal kasutada.

Toiteploki valmimisest on nii tööga kaasasoleval mälupulgal kui ka aadressil https://www.youtube.com/watch?v=XY-fmE_dNDQ (24.02.2016) video.

3.4. Mootorikatsetus

Kõigepealt tellisin NEMA 17 mootorid ära ja kui need pärast ühe kuu pikkust ootamist kohale jõudsid, tahtsin esimesed mootoritestid teha. Plaanisin esialgu ilma programmita katsetada, aga sammmootoreid on raske ilma mootorikontrollerita ja tarkvarata testida, seega pidin kohe ka Arduino ja mootoridraiveri hankima. Järgnevalt koodinäide, mida kasutasin mootori liikumise testimisel.

Sammmootori lõputu pöörlemine valitud suunas

```
int step = 7;
int dir = 4;
void setup() {
      pinMode(step, OUTPUT);
      pinMode(dir, OUTPUT);
}
void loop() {
      digitalWrite(dir, HIGH);
      digitalWrite(step, HIGH);
      delayMicroseconds(400);
      digitalWrite(step, LOW);
      delayMicroseconds(400);
}
```

Kui mootor, mootori juhtskeem A4988 ja Arduino on omavahel ühendatud [Polulu Corporation, 2016] saab koodi Arduinosse laadida ja käivitades hakkab mootor ühes või teises suunas „astuma”. Arduino väljund number 7 läheb A4988 STEP ühendusse ja väljund number 4 läheb A4988 DIR ühendusse. Arduino pesade numbreid võib vastavalt soovile ümber määrata. `dir` muutujat LOW'ks või HIGH'ks määrates saab mootori pöörlemissuunda muuta.

3.5. Raam

Raami ehitamisel tahtsin alguses kasutada 4 mm paksust ja 40 mm laiust alumiiniumlatti, aga kuna mul ei olnud keevitusvõimalust ja ilma keevitamiseta on metalle mõttetult keeruline korralikult fikseerida, kasutasin projektis hoopis 12 mm paksusega vineeri, millest lõikasin 40 mm laiused ribad ja ribadest omakorda sobivate pikkustega detailid. Kuna aga alguses hankisin ehitustuhinas ka alumiiniumlatid, kasutasin neid hiljem siiski.

Raami mõõtmed ei ole detailselt välja toodud, kuna need muutusid töö käigus nii palju, et algsest ei jäänud just palju viimasesse versiooni alles. Kindlalt saan aga öelda, et külglattide laius jäi algusest lõpuni 40 mm, kasutatud viineri paksus 12 mm ja väikesed risplatvormi (platvorm kahe alumise keermelati peal, selle peal sõidab teine väike platvorm) otsad ning nendel liikuv platvorm on 40 x 40 mm.

Ristplatvorm on lihtsalt sobivalt lai, et raami sisse mahtuda (mõlemast küljest umbes 5 mm eemal).

Lühema külje laiuseks olin alguses valinud 330 mm. Kui aga hiljem rihmad tellisin, pidin disaini rihma pikkuse järgi ümber tegema. Niisiis tegin lühemaid külgi ümber 540 mm pikkuse GT2 rihma jaoks sobivaks, mis alumisi keermelatte sünkroonis pöörlemas hakkas hoidma. Otsa enese oleks võinud sama pikaks jätta, aga kuna kirjutusvahend saab liikuda nagunii ainult keermelattide vahel, mitte nendest läbi tulla, oli mõistlik ka raami mõõtmeid vähendada.

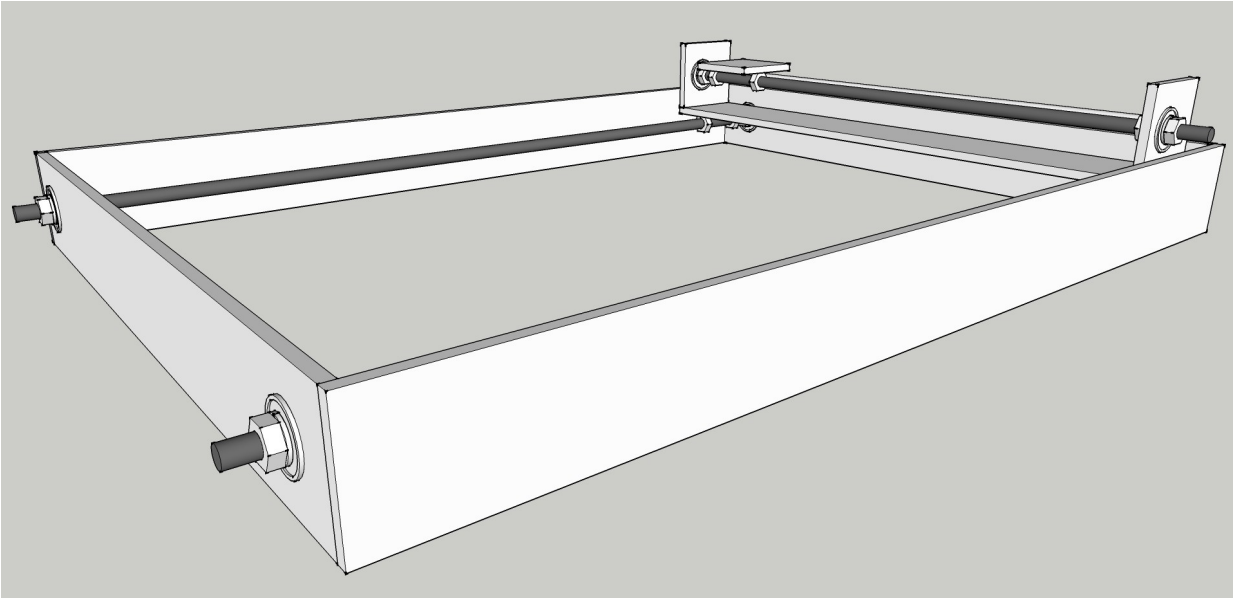
Kuna mul oli vaja lühemad otsad ümber teha ja alumiiniumlatid olid juba hangitud, kasutasin huvitavama väljanägemise huvides otsadena alumiiniumit. Kui üks detail on alumiiniumist ja teine vineerist, saab otsast läbi alumiiniumi vineeri sisse kruvid keerata ja ei teki kinnitusprobleemi.

Pikema külje pikkus oli alguses 400 mm, kuid ka see muutus töö käigus.

Kõik illustratsioonil 3.3 nähtavad avad on detailide suhtes ülevalt ja alt sama kaugel ehk tsentris.

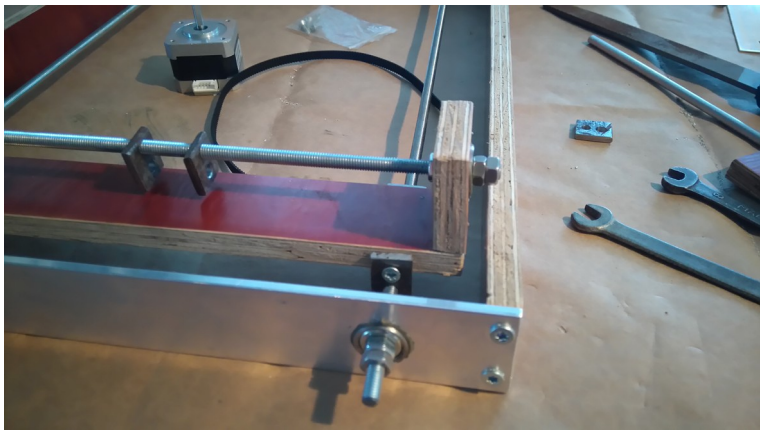
Raami mõõtmete kokkuvõtteks tahaksin veelkord lisada, et need ei ole väga olulised. Asjad, mida järgida, on joonistatava pinna suurus (kui suur pind jääb keermelattide vahele) ja et keermelatid oleksid lõigatud piisavalt pikad, et otsa õnnestuks mootor kinnitada (u. 30 mm varu kinnituse poolsest otsast on sobiv). Mootorite kinnitamiseks keermelattide otsa kasutasin

juhtmeisolatsiooni, mille elastsus vähendab pingeid detailidele, kuid on samas piisavalt jäik, et pöörlemist üle kanda. Juhtmeisolatsiooni kasutasin kui detaile ühendavat toru, mille sisse liimisin kuuma liimiga mootori võlli ja keermelati otsa.



Illustratsioon 3.3: Esialgne raam

Juuresoleval illustratsioonil 3.3 on näha, et platvormide keermestatud detailid on mutrid, nagu mainisin ka alapeatükis 3.1 Esialgne plaan. Tegelikult aga selgus katsetuste käigus, et mutreid selliselt platvormide külge liimida on tülikas ja nad ei jää piisavalt tugevasti kinni. Lahendusena lõikasin 3 mm paksusest süsinikterasest välja 12 mm x 15 mm detailid, milledele ühte lühemasse serva keermestasin M6 keerme ja teise lühemasse serva puurisin 5 mm augu detaili kruviga platvormi külge kinnitamiseks. Kirjeldatud detailid on näha illustratsioonil 3.4.

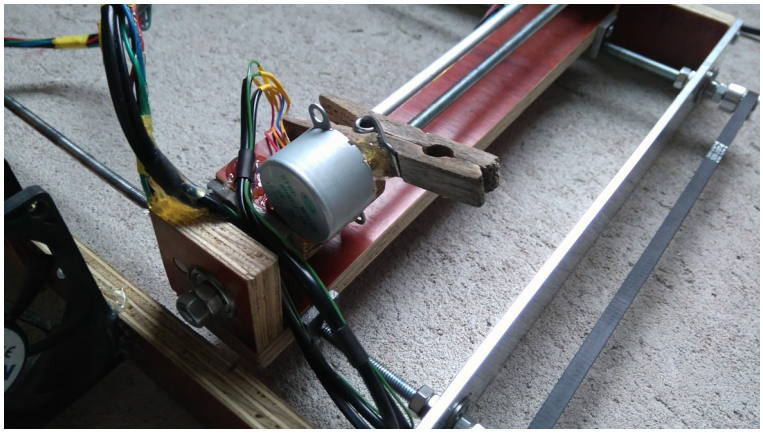


Illustratsioon 3.4: Keermestatud detailid

3.6. Kirjutusvahendi üles ja alla liigutamine

Üles ja alla liikumise süsteemi ma algusest peale väga detailselt ei plaaninud, kuna prioriteet oli saada ülejäänud süsteem tööle ja siis z-suunaga (üles-alla) edasi minna.

Ajutiselt testisin pesupulgaga. Kinnitasin pesupulga otse mootori külge, nagu näha illustratsioonil 3.5. Sellel lahendusel ilmnesid kaks põhilist puudust. Esiteks ei hoidnud pesupulk kirjutusvahendit piisavalt tugevasti kinni ja teiseks kirjutusvahendit üles ja alla liigutades võis joonistatavale pinnale väike joon tekkida, kuna kirjutusvahend keeras otse üles või alla liikumise asemel aluspinnalt üles ja sellele tagasi. Kuna aga illustratsioonil 3.5 näha olev mootor 28BYJ-48 on üsnagi aeglane, oli sellise lahenduse plussiks see, et mootor ei pidanud kirjutusvahendi üles või alla liigutamiseks isegi ühte täispööret tegema.

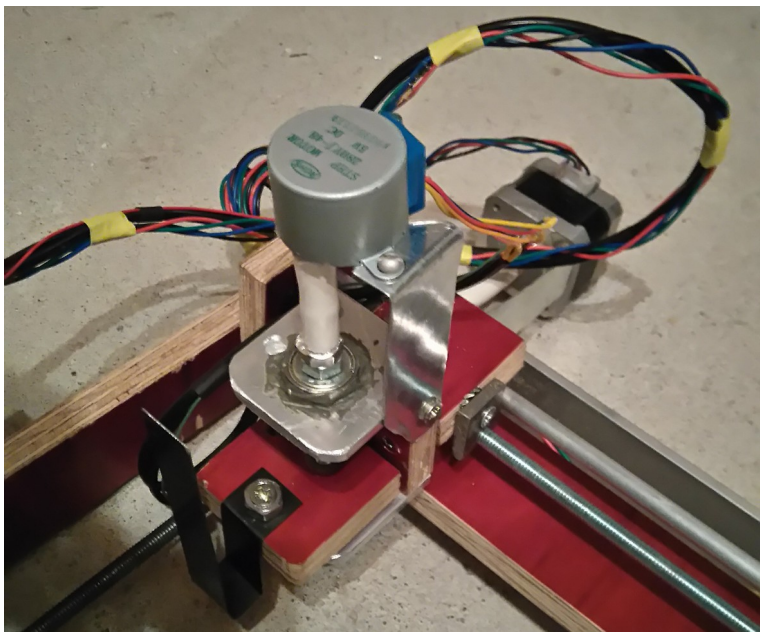


Illustratsioon 3.5: Pesupulk kirjutusvahendi hoidjana

Pesupulga haarde nõrkuse ja soovimatu joone tekkimise tõttu pidin uue lahenduse välja mõtlema ja otsustasin, et üles ning alla liikumist oleks samuti mõistlik keermelatiga lahendada. Ehitasin illustratsioonil 3.6 oleva süsteemi.

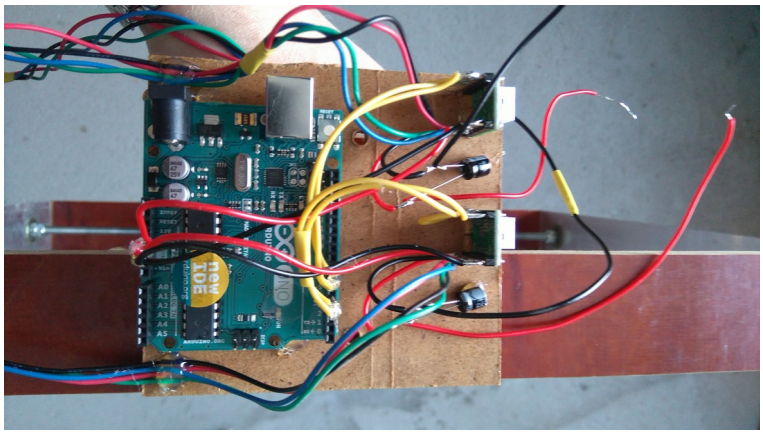
Illustratsioonil 3.6 on näha C-tähe kujuline detail. Keskel nähtav vineeritükk liigub keermelatti keerutades üles ja alla. Keermelati kõrval olev latt on stabiliseerimiseks, et platvorm ringi ei hakkaks käima, vaid z-suunal liiguks. Vineeritüki sisse on puuritud sobiva suurusega ava ja M6 mutter sisse liimitud, et keermestatud ava saada. Kogu detaili kinnitasin läbi pildil näha oleva tagumise vineeri ristplatvormi külge. Lõplikus detailis on keermelati ülemisse otsa kinnitatud mootor ja alla-üles liikuva vineerist platvormi eesmisse serva plekitükk, mis oma loomuliku paindlikkusega kirjutusvahendile ebatasasel pinnal survet avaldab. Kirjutusvahend on kinnitatud plekist klambri külge. Mootoriga ja klambriga detail on näha illustratsioonil 3.7.

Illustratsioon 3.6: Üles ja alla liikumine



Illustratsioon 3.7: Üles ja alla liikumine, täielik

3.7. Elektroonika ühendamine

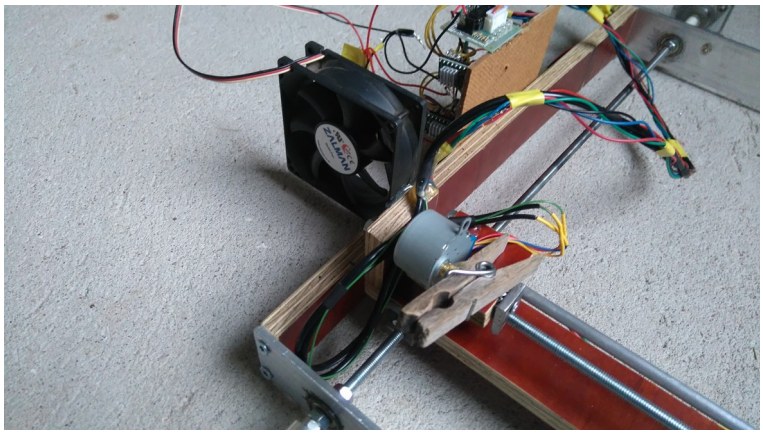


Illustratsioon 3.8: Plaadile liimitud elektroonikadetailid

Projektis vajalikud elektroonikakomponendid on 2 NEMA 17 sammmootorit, kaks mootoridraiverit A4988, 2 100 µF kondensaatorit, 1 28BYJ-48 sammmootor, 1 ULN2003 mootoridraiver, 12 V ja 5 V väljastav toiteplokk. Elektroonikaskeemi nägemiseks vaata Lisa 2. Skeemil toodud ventilaator on 80 mm läbimõõduga 12 V tarbiv arvuti ventilaator. A4988 moodulid läksid töö käigus ilma aktiivjahutuseta puudutusele liialt kuumaks ja seetõttu pidin

leidma lahenduse nende jahutamiseks. Kirjeldatud ventilaator sobis selleks suurepäraselt. Ventilaatori asetsemine elektroonika suhtes on näha illustratsioonil 3.9.

Et juhtloogika oleks paremini organiseeritud, kinnitasin Arduino, A4988 kontrollerid ja kondensaatorid kuuma liimi kasutades pappplaadi külge, nagu näha illustratsioonil 3.8.



Illustratsioon 3.9: Ventilaator

3.8. Tarkvara

Tarkvara ehitasin üles selliselt, et Arduino's on programm, mis ootab sisendit mootori liigutamiseks ühe sammu võrra. Määrasin ära erinevad karakterid, mida sisendina oodatakse ja kirjutasin valikulause vastava mootori „astumiseks” valitud suunda.

Lõplik Arduino kood on toodud lisas Lisa 3. Arduino koodi loomisel kasutasin tema keelejuhendit [Arudino.cc, 2016].

Koodi alguses on deklareeritud kasutatavad muutujad. `lowerDirectionPin` ja `upperDirectionPin` on vastavalt alumisi keermelatte ja ülemist keermelatti juhtivatele mootoritele. Sarnaselt on `lowerStepPin` ja `upperStepPin`.

Sammmootori liigutamiseks ühe sammu võrra on talle vaja anda pinge ja seejärel pinge maha võtta. Uue sammu tegemiseks anda uuesti pinge ja jälle maha võtta. `delayMicro` on see aeg mikrosekundites, kui kaua sellise vaheldumise vahel oodatakse.

Koodi `setup()` osas määratakse vajalikud algväärtustused ja edasi tuleb põhitsükkel.

Põhitsüklis kontrollib Arduino, kas talle on sümbol saadetud. Kui on, loeb ta selle sisse ja

järgmise valikulausega otsustab, millist mootorit millises suunas ühe sammu võrra edasi "astuda". Selleks määratakse suund ja kasutatakse vasatava mootori `stepPin`'ga `stepOnce()` funktsiooni. `lower()` ja `raise()` funktsioonid liigutavad, nagu funktsiooni nimedki ütlevad, kirjutuspead vastavalt alla või üles.

Teiseks komponendiks on kolmest klassist koosnev Java programm. Esimene klass Serial (Lisa 4.) loob läbi valitud pordi ühenduse Arduinosse ja loob teise klassina kirjeldatud objekti AutomaticSerialWriter (Lisa 5.). AutomaticSerialWriter'it kasutab Serial Arduino'ga suhtlemisel. Printer (Lisa 6.) klassis luuakse valitud pordi nimega (näiteks „COM3”) Serial objekt, mille kaudu saab liikumiskäsklusi printerisse saata.

Serial klassi abil luuakse soovitud pordiga ühendus, või kui see ei õnnestu, teatatakse sellest. Serial klassi `connect()` meetod võtab parameetrina pordi nime, näiteks "COM3".

AutomaticSerialWriter viib suhtlust läbi. Ta peab olema *Runnable* ja `run()` meetodi sees on võimalik väljundit saata, aga printeriga interaktsiooniks kasutasin hoopis enda kirjutatud meetodit `send()`, mis võtab parameetrina sümboli ja saadab selle printerisse.

Printer klassis algväärtustatakse kõigepealt mitmed muutujad. `height` ja `width` on maksimaalne lubatud sammude arv (0,0) punktist ühes ja teises suunas. Selles koodis neid hetkel kasutatud ei olegi, aga plaan oli enne liikumist kontrollida, kas mootor on lubatud piirides.

`x` ja `y` peavad meeles praegust kirjutuspea asukohta.

`initSerial()` meetodis luuakse ühendus Arduino'ga.

Järgmiseks tulevad kaks põhimeetodit `moveX()` ja `moveY`, mis võtavad parameetritena soovitud `x` või `y` ja `basis`-nimelise muutuja. Need meetodid tegelevad liikumistega.

`basis` on pikem liikumine jagatud lühemaga. Algkoordinaatidelt soovitud koordinaatidele liikudes „astutakse” tsüklis mõlema mootoriga üksteise järel kõigepealt `basis` jagu. Liikumiste vahe ja lühema liikumise pikkuse kaudu arvutatakse intervall. Iga tsükli läbimisega loendatakse intervallini jõudmist. Kui jõuti, liigub pikemat liikumist tegev mootor veel ühe sammu edasi ja intervalli loendur läheb uuesti nulli. Selliselt toimides jõuavad mõlemad liikumised ühel hetkel lõpppunkti.

Meetod `moveTo()`, mis võtab argumentidena soovitud `x` ja `y`, koondab enda sees põhilise loogika. `basis`'e järgi tekib kolm juhtu - kas liigutakse ainult ühtpidi, mõlema mootoriga võrdselt või ühega rohkem kui teisega.

Järgnevad kasutatud meetodid erinevate arvutuste tegemiseks.

Java'ga jadaühenduse (*serial*) loomiseks kasutasin vabavaralist java teeki RXTX [RXTX developers, 2011]. Kasutamiseks tuleb *Downloads* sektsioonist hankida viimane *Binary* pakk. Pärast lahtipakkimist on näha enamlevinud operatsioonisüsteemide nimedega kaustad. Vastavast kaustast tuleb rxtxParallel.dll ja rxtxSerial.dll failid oma Java projektikausta kopeerida. Seejärel saab läbi Java jadaühendust kasutada. Lisa 4. ja Lisa 5. olev kood on suures osas RXTX projekti kodulehelt leitud koodinäidete abil tehtud.

Väikeseid ebastabiilsusi esines, aga üldiselt toimis RXTX hästi.

3.9. Tulemused

Tarkvara pidi esialgse plaani kohaselt hakkama svg-failist vektorgraafika elementide kirjeldusi lugema ja neid siis sammhaaval printerisse saatma. Selgus aga, et juba sirgjoone arvutamine on üpriski keerukas tegevus ja seega katabki Printer praegu ainult joone loogika ja omab ühte meetodit, millega väike prooviruut joonistada.

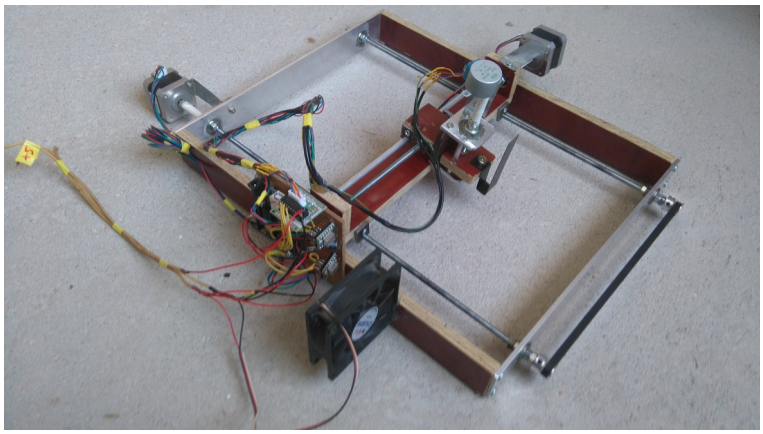
Prooviruudu joonistamisest on ka video aadressil https://www.youtube.com/watch?v=F_f1JetuKm0 (24.02.2016).

Kahjuks ei õnnestunud mul kokkuvõttes luua päris nii efektiivset asja, nagu tahtsin. Raam ei olnud lõpetuseks nii jäik, et kirjutuspead viisakalt paigal hoida ja sellega ringi sõita. Nurgas sai asjad tehtud, aga keskele poole liikudes muutus ebatäpsus juba liiga suureks ja otsustasin, et olen prototüübiga rahul ja ei hakka teda lõpuni viimistlema. Seetõttu jäi ka kood pigem eksperimendi tasemele kui sai valmislahenduseks.

Arvasin, et kasutatud väikeste detailide liigutamiseks ei ole suurt ja tugevat konstruktsiooni vaja, kuid kui niisugust projekti uuesti teha, oleks platvormidel kindlasti tugesid ja siine vaja, sest 6 mm läbimõõduga keermelatid vetruvad juba 300 mm pikkuse juures piisavalt, et saavutada mitteaktsepteeritav ebatäpsus. Ainuüksi keermelattidest, mis tükke liigutavad, ei ole ka nende raskuse kandmiseks piisavalt.

Suurim probleem oligi see, et kirjutuspea vetrus raami keskel olles oma raskuse tõttu rohkem kui 10 mm üles ja alla, mistõttu ei olnud võimalik (mõistliku ajaga) kirjutuspead pinnalt üles ja alla liigutada. Tööga kaasas oleval mälupulgal on ka video platvormi vetrumisest.

Kui seadet mingi aja vältel edasi viimistleda, oleks tõenäoliselt võimalik läbipaindumised minimeerida ja lisaks katsetustele ka tõsisemaid pilte joonistada, aga kõik see ei mahtunud töö aja sisse. Valminud seade on näha illustratsioonil 3.10.



Illustratsioon 3.10: Valminud seade

Pidasin projekti vältel ka blogi. Nii tihti, nagu alguses plaanisin, sinna ei kirjutanud, aga mitmed asjad on seal detailsemalt kirjeldatud. Lisaks blogile valmisid töö käigus ka paar videot ja oluliselt rohkem pilte, kui töösse otsustasin panna. Blogi on kättesaadav aadressil: http://kardo.xyz/arduino-printer (24.02.2016).

Tööga on kaasas ka mälupulk, millel on videod, pildid ja programmikood.

3.10. Seadme kasutusvõimalused

Ühe võimalusena, kui sarnane süsteem näiteks tahvli külge ehitada, oleks võimalik markeriga (või ka kriidiga) tahvlile samm-sammult diagramme või erinevaid jooniseid joonistada.

Nagu sissejuhatuses tähelepanu juhtisin, kasutavad projektis kasutatud loogikat igasugused seadmed, mida arvuti abil liigutatakse. Sellest tulenevalt oleks, kui kogu süsteem oluliselt tugevamalt üles ehitada, võimalik luua puuriv, lõikav APJ-pink. Seda, et see võimalik on, kinnitavad ka peatükis 2 toodud teiste poolt loodud lahendused.

Eelmist punkti edasi arendades võiks seadmele korraliku z-suuna lisamisega saada ka 3D-

printeri.

Lihtsalt sellisena, nagu minu loodud seade on, saaks igasugustele pindadele markerit kasutades justkui käsitsi tehtud joonistusi luua. Näiteks pilt *laptop'i* kaane peale või muud niisugust.

Kokkuvõte

Sain kogu süsteemi peaaegu niimoodi kokku nagu planeerisin. Detailid sobisid omavahel üllatavalt hästi. Koodiosa poolest sujus kõik samuti ligikaudu selliselt, nagu ette olin kujutanud.

Nagu alapeatükis 3.9 Tulemused kirjutasin, ei osutunud raam piisavalt jäigaks, et üle terve pinna joonistusi luua. Keskel läks kirjutuspea raskuse ja keermelattide läbivetrumise tõttu üles ja alla liikumine juba nii ebaühtlaseks, et ei olnud võimalik kirjutuspead pinnalt mõistliku ajaga üles tõsta ja alla lasta.

Oleksin muidugi tahtnud, et süsteem oleks ideaalselt toimima hakanud, et sisestan pildifaili ja tulemus tuleb, aga ka juba prooviruudu joonistamine tekitas hea saavutuse tunde.

Arvestades, et tegemist on eelkõige uurimuse ja testimisega, õnnestus projekt täielikult.

Kindlasti andis töö mulle hea ülevaate robootika ja tarkvara kombineerimisest ja kui sarnast asja uuesti alustada, on juba erinevaid mõtteid, kuidas midagi efektiivsemalt teha. Kasutaksin liikumisel kiiruse suurendamiseks rihmasid ja paneksin platvormid millegi peal sõitma.

Suures osas aga jäävad niisugused tööd alati katsetamise tasemele. Nagu programmikoodigagi, ei suuda kohe alguses kõigi asjade peale mõelda.

Huvitav on projekti puhul see, et saab kogemusi paljudest eri valdkondadest. Isegi detailide koguse poolest nii väikese seadme puhul on suhteliselt palju planeerimist, erinevate komponentide hankimist, plaanide joonistamist, ehitamist ja koodi kirjutamist. Kui kõik lõpuks kasvõi ligilähedaselt algsele mõttele kokku sobitub, on väga hea tunne.

Ka annab projekti läbimine hea ülevaate sellest, kui palju on isegi väikese sammude arvuga protsessis testimist, et jõuda stabiilse soovitud tulemuseni.

Kõigile, kes soovivad midagi sarnast ette võtta, soovitan varuda palju kannatust ja järjekindlust.

Kasutatud kirjandus

Arduino.cc. (2016) . Language Reference. Loetud kuupäeval 20.02.2016 aadressil https://www.arduino.cc/en/Reference/HomePage

Arduino.cc. (2016) . Arduino Uno Overview. Loetud kuupäeval 14.02.2016 aadressil https://www.arduino.cc/en/Main/ArduinoBoardUno

Atmel Corporation. (2015) . ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTESIN-SYSTEM PROGRAMMABLE FLASH DATASHEET. Loetud kuupäeval 14.02.2016 aadressil http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

HomoFaciens. (2015) . Homemade CNC machine with materials from a DIY, built with simple tools. Loetud kuupäeval 10.11.2015 aadressil https://www.youtube.com/watch?v=mJ-TZvFpY58

NEMA. (2016) . NEMA - The Association of Electrical Equipment and Medical Imaging Manufacturers. Loetud kuupäeval 17.02.2016 aadressil http://www.nema.org

NYC CNC. (2015) . DIY Cheap Arduino CNC Machine - Machine is Complete AND Accurate!. Loetud kuupäeval 12.11.2015 aadressil https://www.youtube.com/watch?v=WKmzlMvxC7I

PBC Linear. (-) . Stepper Motor NEMA 17. Loetud kuupäeval 17.02.2016 aadressil http://www.pbclinear.com/Download/DataSheet/Stepper-Motor-Support-Document.pdf

Polulu Corporation. (2016) . A4988 Stepper Motor Driver Carrier. Loetud kuupäeval 19.02.2016 aadressil https://www.pololu.com/product/1182

RepRap Team. (2015) . Motor FAQ. Loetud kuupäeval 17.02.2016 aadressil http://reprap.org/wiki/Motor_FAQ

RXTX developers. (2011) . Rxtx. Loetud kuupäeval 20.02.2016 aadressil

http://rxtx.qbang.org/wiki/index.php/Main_Page

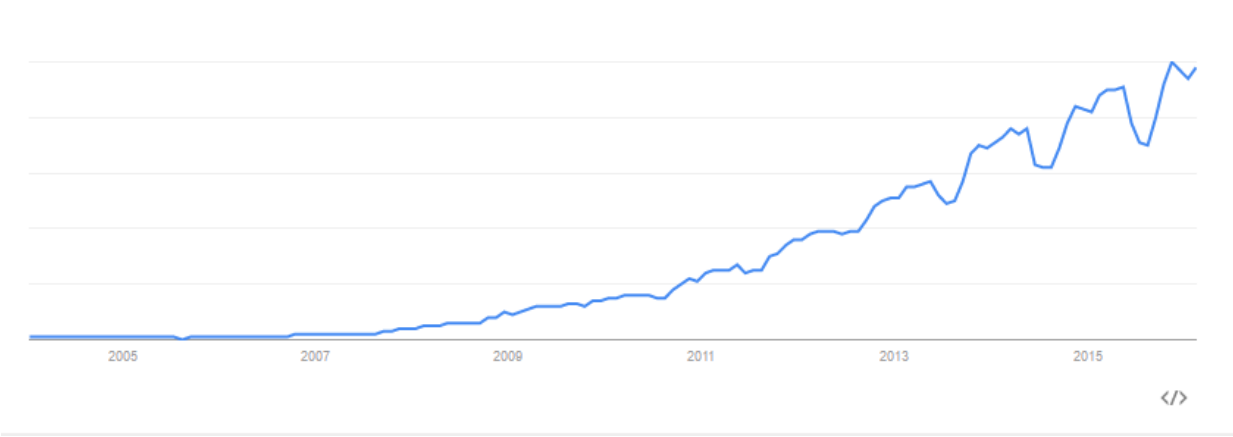
Sitnalta. (-) . Convert an ATX Power Supply Into a Regular DC Power Supply!. Loetud kuupäeval 17.02.2016 aadressil http://www.instructables.com/id/Convert-an-ATX-Power-Supply-Into-a-Regular-DC-Powe/

Stock Drive Products / Sterling Instrument. (2016) . GT®2 Timing Belts. Loetud kuupäeval 17.02.2016 aadressil http://www.sdp-si.com/products/belts/gt2.htm

Veli Ahmet AKTEKE. (2014) . Arduino Plotter. Loetud kuupäeval 10.11.2015 aadressil https://www.youtube.com/watch?v=kkgxDEXfE_k

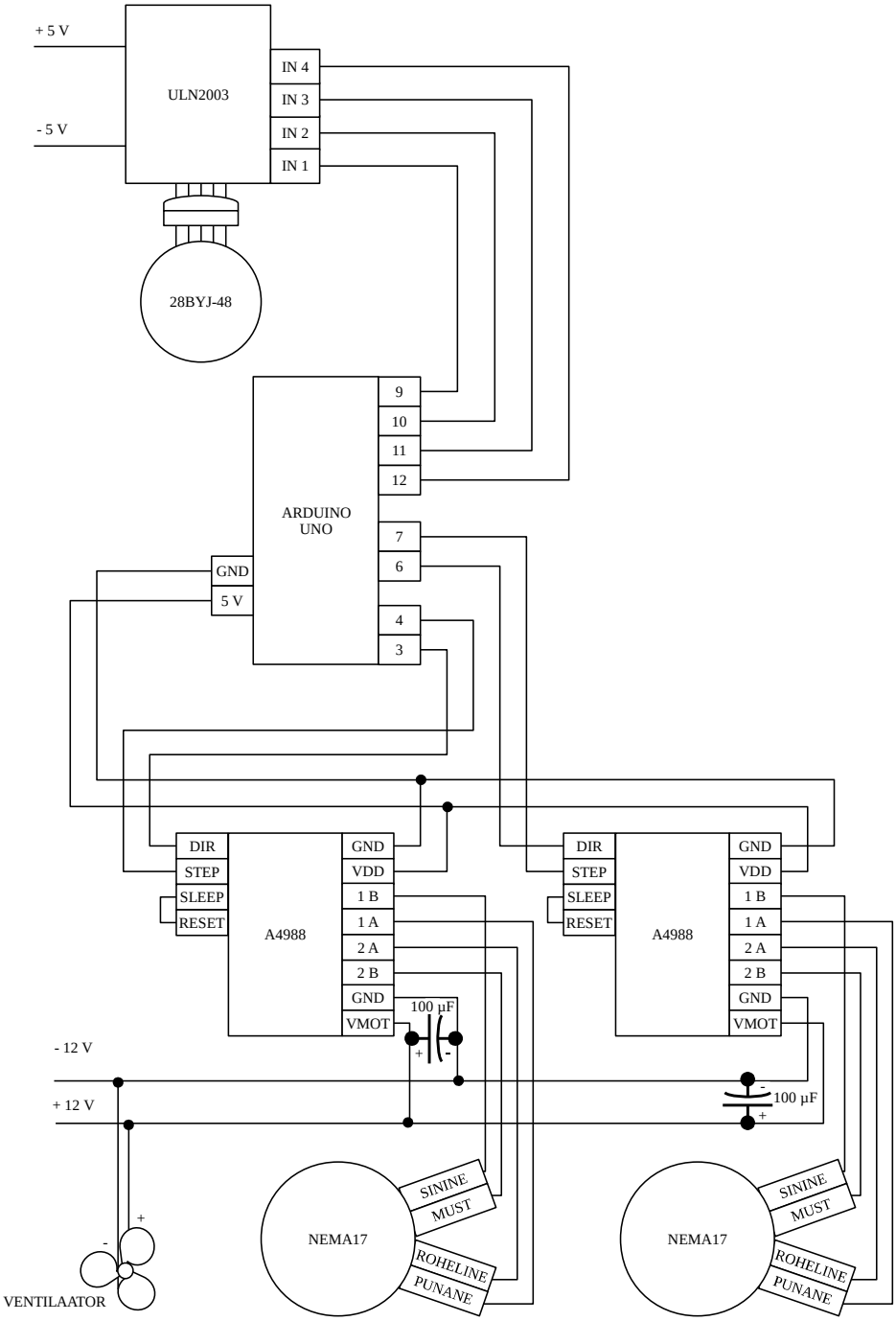
LISAD

Lisa 1. Google Trends otsing „Arduino”



Otsingutulemus kuupäeval 19.02.2016

Lisa 2. Elektroonikaskeem



Lisa 3. Arduino kood

```
const int lowerDirectionPin = 3;
const int lowerStepPin = 4;
const int upperDirectionPin = 6;
const int upperStepPin = 7;
const int delayMicro = 450;

int in1 = 9;
int in2 = 10;
int in3 = 11;
int in4 = 12;
int delayMillis = 2;
int cycles = 3000; // Kirjutusvahend üles ja alla, mitu sammu

void setup() {
  pinMode(upperDirectionPin, OUTPUT);
  pinMode(upperStepPin, OUTPUT);
  pinMode(lowerDirectionPin, OUTPUT);
  pinMode(lowerStepPin, OUTPUT);
  digitalWrite(upperDirectionPin, HIGH); // LOW mootori
              suunas, HIGH eemale
  digitalWrite(lowerDirectionPin, HIGH); // LOW mootori
              suunas, HIGH eemale

  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
```

```
  digitalWrite(in4, LOW);

  Serial.begin(9600);
}

void loop() {
  if(Serial.available() > 0){
    char serialInput = Serial.read();
    if(serialInput == 's'){
      digitalWrite(lowerDirectionPin, HIGH);
      stepOnce(lowerStepPin);
    } else if(serialInput == 'w'){
      digitalWrite(lowerDirectionPin, LOW);
      stepOnce(lowerStepPin);
    } else if(serialInput == 'd'){
      digitalWrite(upperDirectionPin, LOW);
      stepOnce(upperStepPin);
    } else if(serialInput == 'a'){
      digitalWrite(upperDirectionPin, HIGH);
      stepOnce(upperStepPin);
    } else if(serialInput == 'l'){
      lower();
    } else if(serialInput == 'r'){
      raise();
    }
  }
}

void stepOnce(int stepPin){
  digitalWrite(stepPin, HIGH);
  delayMicroseconds(delayMicro);
  digitalWrite(stepPin, LOW);
  delayMicroseconds(delayMicro);
```

```
}

void raise(){
  for(int i=0; i<cycles; i++){
    digitalWrite(in4, HIGH);
    delay(delayMillis);
    digitalWrite(in4, LOW);
    digitalWrite(in3, HIGH);
    delay(delayMillis);
    digitalWrite(in3, LOW);
    digitalWrite(in2, HIGH);
    delay(delayMillis);
    digitalWrite(in2, LOW);
    digitalWrite(in1, HIGH);
    delay(delayMillis);
    digitalWrite(in1, LOW);
  }
}

void lower(){
  for(int i=0; i<cycles; i++){
    digitalWrite(in1, HIGH);
    delay(delayMillis);
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    delay(delayMillis);
    digitalWrite(in2, LOW);
    digitalWrite(in3, HIGH);
    delay(delayMillis);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
    delay(delayMillis);
    digitalWrite(in4, LOW);
```

```
    }
}
```

Lisa 4. Java Serial kood

```
package printer;

import java.io.OutputStream;

import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

public class Serial2 {

  AutomaticSerialWriter serialWriter;

  public Serial2(){
    super();
  }

  void connect(String portName) throws Exception{
    // Pordi nime järgi handler
    CommPortIdentifier portIdentifier =
     CommPortIdentifier.getPortIdentifier(portName);

    // Kontroll, kas port on juba kasutusel
    if(portIdentifier.isCurrentlyOwned()){
      System.out.println("Error: Port " + portName + " on juba
                                kasutusel!");
    } else {

      // Kui pole, vaatame edasi
      CommPort commPort =
     portIdentifier.open(this.getClass().getName(), 2000);
```

```
        if(commPort instanceof SerialPort){
          SerialPort serialPort = (SerialPort) commPort;
          serialPort.setSerialPortParams(9600,
            SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);

          OutputStream out = serialPort.getOutputStream();
          out.flush();

          serialWriter = new AutomaticSerialWriter(out);
          new Thread(serialWriter).start();

        } else {
          System.out.println("Error: Port " + portName + " ei
                                ole jadaport!");
        }

      }
    }

    public AutomaticSerialWriter getSerialWriter(){
      return serialWriter;
    }
}
```

Lisa 5. Java AutomaticSerialWriter kood

```
package printer;

import java.io.IOException;
import java.io.OutputStream;

public class AutomaticSerialWriter implements Runnable {
  OutputStream out;

  public AutomaticSerialWriter(OutputStream out){
    this.out = out;
  }

  // Run sees pole midagi. Runnable peab olema, et kasutada,
     aga
  // käske saadan send meetodiga.
  public void run(){}

  public void send(char c){
    try {
      out.write(c);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Lisa 6. Java Printer kood

```
package printer;

/*
 * Printer 0.1
 * Alguses peab olema 0, 0 asendis ja pea üleval.
 */

public class Printer {
  Serial2 serial;
  AutomaticSerialWriter serialWriter;

  int height = 56305;
  int width = 38060;
  int x = 0;
  int y = 0;
  boolean penUp = true;
  int penWaitMillis = 25000; // Ooteaeg pärast alla/üles käsu
     andmist

  public Printer(){
    initSerial();
  }

  public void initSerial(){
    System.out.println("Alustamine...");
    try{
      Serial2 serial = new Serial2();
      serial.connect("COM3");
      serialWriter = serial.getSerialWriter();
    } catch(Exception e){
```

```
    e.printStackTrace();
  }
}

public void moveX(int x, int basis){
  for(int i=0; i<basis; i++){
    if(this.x > x){
      serialWriter.send('a');
      this.x--;
    } else if(this.x < x){
      serialWriter.send('d');
      this.x++;
    }
    try {
      Thread.sleep(2);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}

public void moveY(int y, int basis){
  for(int i=0; i<basis; i++){
    if(this.y > y){
      serialWriter.send('s');
      this.y--;
    } else if(this.y < y){
      serialWriter.send('w');
      this.y++;
    }
    try {
      Thread.sleep(2);
    } catch (InterruptedException e) {
```

```
      e.printStackTrace();
    }
  }
}

public void moveTo(int x, int y){
  System.out.println("Alustan liikumist");

  int xLength = calculateMovementLength(this.x, x);
  int yLength = calculateMovementLength(this.y, y);
  char longer = longerMovement(xLength, yLength);
  int difference = calculateLengthDifference(xLength,
                       yLength);
  int basis = calculateMovementBasis(xLength, yLength);
  int waitCycles = calculateWaitCycles(xLength, yLength,
                       difference);

  if(basis == 0){
    System.out.println("Üks ei liigu üldse");
    if(longer == 'x'){
      while(this.x != x){
        moveX(x, 1);
      }
    } else if(longer == 'y'){
      while(this.y != y){
        moveY(y, 1);
      }
    }
    System.out.println("Üks ei liigu lõpp");
  } else if(basis >= 1){
    int waitCycleCounter = 0;
    System.out.println("Intervalliga");
    if(longer == 'x'){
```

```
      while(this.x != x || this.y != y){
        if(waitCycleCounter == waitCycles){
          moveX(x, basis+1);
          moveY(y, basis);
          waitCycleCounter = 0;
        } else {
          moveX(x, basis);
          moveY(y, basis);
          waitCycleCounter++;
        }
      }
    } else if(longer == 'y'){
      while(this.x != x || this.y != y){
        if(waitCycleCounter == waitCycles){
          moveX(x, basis);
          moveY(y, basis+1);
          waitCycleCounter = 0;
        } else {
          moveX(x, basis);
          moveY(y, basis);
          waitCycleCounter++;
        }
      }
    }
    System.out.println("Intervalliga lõpp");
  }
  System.out.println("Liikumine lõppenud");
}

public int calculateWaitCycles(int xLength, int yLength, int
                                    difference){
  if(xLength < yLength){
    return Math.round(xLength/difference);
```

```
  } else if(xLength > yLength){
    return Math.round(yLength/difference);
  } else if(xLength == yLength){
    return 1;
  } else {
    return 0;
  }
}

public int calculateLengthDifference(int xLength, int
                                     yLength){
  return Math.abs(xLength-yLength);
}

public int calculateMovementLength(int start, int stop){
  return Math.abs(stop-start);
}

public char longerMovement(int xLength, int yLength){
  if(xLength > yLength){
    return 'x';
  } else if(yLength > xLength){
    return 'y';
  }
  return 'e'; // e nagu equal
}

public int calculateMovementBasis(int xLength, int yLength){
  int basis = 0;
  float xLengthFloat = xLength;
  float yLengthFloat = yLength;
  if(xLength == 0 || yLength == 0){
    basis = 0;
```

```
  } else if(xLength > yLength){
    basis = Math.round(xLengthFloat / yLengthFloat);
  } else if(yLength > xLength){
    basis = Math.round(yLengthFloat / xLengthFloat);
  } else if(xLength == yLength){
    basis = 1;
  }
  return basis;
}

public void togglePen(){
  System.out.println("Liigutan pliiatsit");
  if(penUp){
    System.out.println("Üleval");
    serialWriter.send('l');
    penUp = false;
  } else {
    System.out.println("All");
    serialWriter.send('r');
    penUp = true;
  }
  try {
    Thread.sleep(penWaitMillis);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}

public void drawTestRectangle(Printer p){
  p.togglePen();
  p.moveTo(0, 5000);
  p.moveTo(5000, 5000);
  p.moveTo(5000, 0);
```

```
    p.moveTo(0, 0);
    p.togglePen();
  }

  public static void main(String[] args) {
    Printer p = new Printer();
    p.drawTestRectangle(p);
  }

}
```