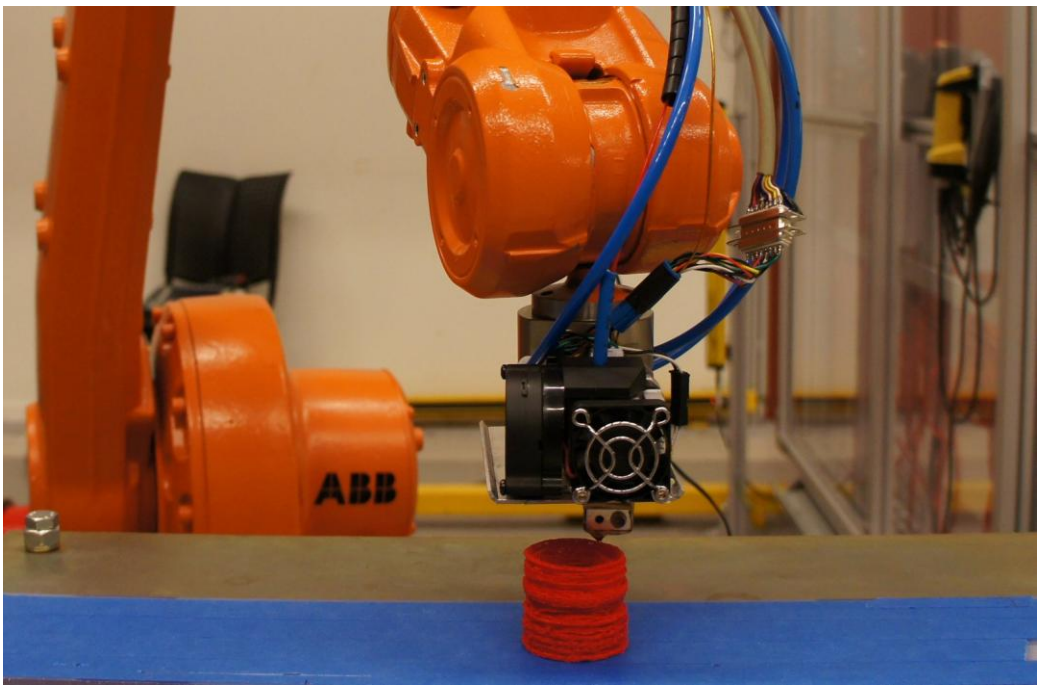
CHALMERS



Robot Prototyping

– en industrirobot som 3D-skrivare

Kandidatarbete inom Automation och Mekatronik

MARKUS HÄGERSTRAND
ROBIN LINDHOLM
TOMAS NILSSON

Institutionen för produkt- och produktionsutveckling
Avdelningen för produktionssystem
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2013

Sammanfattning

3D-skrivare och additiv tillverkning har vuxit i popularitet på senare tid. De används både av privatpersoner och av företag främst för att skapa enklare detaljer och prototyper. I konventionella 3D-skrivare sitter oftast skrivarhuvudet placerat på en kartesisk robot med tre frihetsgrader. Kandidatarbetet syftar till att analysera lämpligheten att istället ha skrivarhuvudet på en industrirobot med sex frihetsgrader. Dessa är större och dyrare men finns ofta redan på plats i verkstadsindustrin och har andra egenskaper än kartesiska robotar. Detta medför andra möjligheter och begränsningar. Lämpligheten att använda industrirobotar för additiv tillverkning analyserades genom att ett prototypsystem konstruerades och utvärderades. Systemet består av ett skrivarhuvud från 3D-skrivaren Makerbot Replicator 2 monterat på en industrirobot av typen ABB IRB1600-8/1.45. Det resulterande 3D-skrivarsystemet skrev ut olika typer av modeller och det visade sig att en lägre utskriftshastighet gav finare utskriftsföremål. Att använda en robot som 3D-skrivare ger stora möjligheter att förverkliga konstruktioner som tidigare inneburit komplikationer, men för att de ska ta över uppgifter som i dagsläget görs av andra maskiner krävs ytterligare utveckling. Potentiella användningsområden identifierades framförallt inom tillverkningsindustrin, där industrirobotar redan har en naturlig plats i produktionen, och det finns processer som potentiellt kan förbättras med hjälp av en 3D-skrivarfunktion.

Abstract

3D-printers and additive manufacturing have grown in popularity in recent years. The printers are used by both individual hobbyists and businesses, mainly to create simple details and prototypes. The most common printer consists of an extruder mounted on a Cartesian coordinate robot with three degrees of freedom. This thesis investigates the suitability of having the extruder mounted on an industrial robot with six degrees of freedom. In general, the robots are bigger and more expensive but often already exist in the manufacturing industry,and they have different characteristics than Cartesian coordinate robots. This results in other possibilities and limitations. The suitability of using industrial robots for additive manufacturing was analyzed by constructing a prototype and then evaluating it. The prototype system consists of an extruder from a Makerbot Replicator 2 3D-printer mounted on an industrial robot ABB IRB1600-8/1.45. The resulting 3D-printing system was used to print a few different models and it was found that a lower speed yielded a better result. However, further development is needed for such a system to take over tasks in the industry. Potential applications were identified to exist in an industry environment where industrial robots already are installed, and there are tasks that can be simplified or improved with 3D-printing.

Förord

Stor tack till handledare Per Nyqvist samt Göran Stigler och Hans Sjöberg för deras hjälp i PPU-labbet på Chalmers tekniska högskola.

Innehåll

1 Inledning **1**
1.1 Syfte 1
1.2 Avgränsningar 2
1.3 Notationer 2

2 Systembeskrivning **3**
2.1 Systembeskrivning 3D-skrivare 3
2.1.1 STL-formatet 3
2.1.2 Uppdelning av CAD-modeller i lager 3
2.1.3 Fused Deposition Modelling 4
2.1.4 G-code 4
2.1.5 Arduino 5
2.2 Systembeskrivning industrirobot 5
2.2.1 Industrirobot ABB 1600-8/1.45 5
2.2.2 RobotStudio - utvecklingsmiljö för robotprogrammering 5
2.2.3 RAPID - programmeringsspråk för ABB-robotar 6
2.2.4 Robotens konfiguration 6
2.2.5 Kalibrering av arbetsobjekt och verktyg 6

3 Metod **8**
3.1 CAD-modeller för utskrift 8
3.2 Uppdelning av CAD-modeller i lager 9
3.3 G-code för utskrift 9
3.4 RAPID-kod för styrning av robot 10
3.5 Skrivarhuvud och upphängning 10
3.6 Kommunikation mellan robot och styrning av skrivarhuvudet . 10

4 Resultat **12**
4.1 Beskrivning av systemet 12
4.1.1 Robotverktyg - skrivarhuvud 13
4.1.2 Användargränssnitt för skrivarhuvud 13
4.1.3 Plattform för utskrift 14
4.1.4 C# -kod och interface 15
4.1.5 RAPID-kod 16
4.1.6 Arduino-kod 17
4.2 Experiment 18

5 Diskussion **19**
5.1 Systemets utformning 19

5.2 Industrirobotens lämplighet som 3D-skrivare 21
5.3 Potentiella användningsområden 21
5.4 Vidareutveckling 23

A Instruktioner för användning **i**

B Kopplingsbeskrivning **iii**

C Komponenter och kretsschema **vi**

D RAPID-kod **viii**

E Arduino-kod **xii**

F C#-kod **xviii**

1 Inledning

3D-skrivare är ett mycket aktuellt ämne. Från att tidigare ha kostat över hundra tusen kronor har de nu fallit kraftigt i pris och med enbart lite teknisk kunskap kan man idag bygga en egen enklare 3D-skrivare för ett par tusen kronor. 3D-skrivare används i dagsläget främst till att snabbt tillverka enklare prototyper men även för att tillverka specialiserade delar, till exempel delar till motorer eller implantat inom sjukvården. Många spekulerar i att 3D-skrivare är en disruptiv teknik som, när den är mogen, har potential att ersätta en stor del av dagens tillverkning.

Chalmers prototyplabb har en 3D-skrivare, där modeller tillverkas i plast. Prototyperna blir bra, men begränsas storleksmässigt och inköpskostnaden för den typen av maskiner är mycket stor. De flesta 3D-skrivarna idag är baserade på FDM (Fused Deposition Modelling) och använder kartesiska robotar för mekanisk förflyttning av skrivarhuvud. Med kartesiska robotar sker förflyttning av skrivarhuvud linjärt i x-, y- och z-led. Fördelen med denna typ av robotar är att de är billiga och har en enkel konstruktion. I verkstadsindustrin används ofta industrirobotar med sex frihetsgrader vilket möjliggör mer avancerade förflyttningar men också en mycket högre inköpskostnad. En tanke var att man skulle kunna använda dessa befintliga robotar för additiv tillverkning genom att använda ett skrivarhuvud som arbetsverktyg på roboten. Dessa robotar har helt andra egenskaper än kartesiska robotar vilket medför nya möjligheter men också eventuellt andra begränsningar. Till exempel är de inte lika begränsade i den volym de kan röra sig i vilket medför att de skulle kunna skriva ut större föremål.

Kombinationen av FDM och industrirobotar kan vara av intresse för verkstadsindustrin med flera olika potentiella applikationer. Ett exempel där det kan finnas ett intresse från industrin är vid nitning av flygplansvingar, där det uppstår mellanrum mellan vinge och stöd som idag fylls i manuellt. Istället skulle en befintlig robot, som idag används för nitning, kunna fylla igen håligheten med hjälp av additiv tillverkning.

1.1 Syfte

Kandidatarbetet ska analysera om en industrirobot av typen ABB IRB1600-8/1.45 är lämplig att användas för additiv tillverkning. Detta genomförs genom att en prototyp på en helhetslösning utformas och utvärderas. Projektets syfte kan sammanfattas till att

- utvärdera möjligheten att använda en industrirobot som 3D-skrivare genom att bygga en prototyp på en helhetslösning,
- utvärdera i vilken grad industrirobotar är lämpliga att använda som 3D-skrivare utifrån erfarenheter från projektet samt
- diskutera vilka potentiella användningsområden tekniken skulle kunna ha.

1.2 Avgränsningar

Projektets tidsram på 16 veckor medförde att flera avgränsningar för projektet gjordes. De flesta 3D-skrivare har ofta en begränsad storlek på föremålen de kan skriva ut, typiskt $20 \times 20 \times 20$ cm. En industrirobot skulle kunna skriva ut mycket större föremål. Projektet avgränsades till att i första hand försöka skriva ut föremål i storleksordningen mindre än $20 \times 20 \times 20$ cm. Som delsyfte skulle även de potentiella möjligheterna att skriva ut föremål i storleken $1 \times 1 \times 1$ m utvärderas§. Flera antaganden och förenklingar har fått göras angående vissa aspekter hos systemet. Hastigheten på plastutmatningen varierades inte under körning och utskrifter gjordes med konstant hastighet.

1.3 Notationer

För instruktioner i programmeringsspråken C#, RAPID, C och G-code används notationen "{kod}", till exempel {MoveJ}. För längre kodavsnitt är texten kursiverad.

2 Systembeskrivning

Projektet består av två system som ska kopplas samman, en 3D-skrivare och en industrirobot. Här presenteras en generell beskrivning av de två systemen.

2.1 Systembeskrivning 3D-skrivare

Med 3D-skrivare avses ett helt system som tillhandahåller funktioner för att skriva ut ett fysiskt objekt utifrån en CAD-modell. Att skriva ut betyder i detta sammanhang att tillverka föremålet, benämningen kommer från likheten i hur en vanlig skrivare skriver på papper. Tekniken kallas på svenska friformsframställning (Wikipedia 2013a).

Andra benämningar är additiv tillverkning och rapid prototyping. Framställning sker genom att föremålet skrivs ut lager för lager (Wikipedia 2013b). Att föremålet skrivs ut lagervis kan underlätta vid konstruktion av vissa typer av modeller.

2.1.1 STL-formatet

Formatet som används för att representera 3D-modeller av solida föremål är Stereolithography (STL), även kallat Standard Tessellation Language. Formatet beskriver föremålet genom att definiera ett stort antal trianglar som tillsammans approximerar alla ytgeometrier (Wikipedia 2013c). Många moderna CAD-program har möjlighet att exportera CAD-modeller till STL-formatet.

2.1.2 Uppdelning av CAD-modeller i lager

Geometrin över modellen som ska skrivas ut delas sedan upp i flera lager, en process som på engelska kallas slicing. Varje lager innehåller information av vilka regioner av lagret som innehåller material. Utifrån dessa regioner beräknas en lämplig bana för skrivarhuvudets förflyttning under utskriften (Makerbot 2013a). Processen har fått namnet slicing efter det första steget i processen.

2.1.3 Fused Deposition Modelling

Det vanligaste sättet att skriva ut föremål är genom att successivt lägga till material till föremålet, så kallad additiv tillverkning. De populäraste skrivarna idag för privatpersoner och mindre företag är baserade på additiv tillverkning och använder metoden Fused Deposition Modelling. Material i form av till exempel PLA (polylaktid) eller ABS (Akrylnitril-Butadien-Styren) upphettas till sin smältpunkt och extruderas genom ett strängsprutningsmunstycke, också kallat skrivarhuvud eller extruder. Skrivarhuvudet förs mekaniskt med hjälp av ett CAM-program (Computer Aided Manufacturing) till olika positioner där material lämnas (Wikipedia 2013e). När materialet svalnar och därmed stelnar, blir det en del av det solida föremålet.

Utmatningen av material sker med hjälp av en stegmotor som driver ett kugghjul. Stegmotorn matar in plasten i ett värmeelement där det upphettas till sin smältpunkt. Temperaturen på processen mäts med ett termoelement som är en givare som används för att mäta temperatur. Dessa givare är baserade på principen att då en metalltråd passerar genom en temperaturskillnad uppstår en potentialskillnad som går att mäta mellan de två ändpunkterna. Den vanligaste varianten är typ K som är lämplig för temperaturer inom intervallet -200 °C till +1350 °C (Wikipedia 2013i). Storleken på potentialskillnaden som uppstår är endast cirka 41 μV/°C vilket innebär att signalen normalt behöver förstärkas innan den kan mätas med en analog ingång på en mikrokontroller.

Utkriftsmaterialet PLA är en stärkelsebaserad, nedbrytningsbar termoplast som kan tillverkas av majsstärkelse eller rörsocker (Wikipedia 2013f). Plastens smältpunkt är 150-160 °C och plasten lämpar sig för formsprutning och extrudering (Wikipedia 2013g). Den har dålig tålighet för längre exponering mot vatten (Creative Tools 2013).

2.1.4 G-code

Den mekaniska rörelsen av skrivarhuvud och utmatning styrs generellt med numeriska instruktioner i programmeringsspråket G-code. En typisk instruktion vid 3D-utskrift är *G1 X169.050 Y169.630 F540.000 E0.06467*.

{G1} är instruktionen för kontrollerad rörelse, vilket innebär att rörelsen sker linjärt med en jämn hastighet (Wikipedia 2013h). {X169.050 Y169.630} beskriver att den nya målkoordinaten är x=169,05 mm samt y=160,630 mm. {F540} sätter hastigheten för rörelsen till 540 mm/min = 9 mm/s och

{E0.06467} sätter att under tiden som rörelsen sker ska stegmotorn, som styr utmatningen av plast, ta sig till sin position E=0,0647 mm. Z-koordinaten är i exemplet oförändrad vilket är typiskt för 3D-skrivare eftersom förflyttning i z-led generellt bara sker vid byte mellan de lager som skrivs ut.

2.1.5 Arduino

De flesta 3D-skrivarna som säljs till privatpersoner idag styrs av en mikrokontroller av typen Arduino. Arduino-systemet består av ett kretskort med en mikrokontroller och tillhörande elektronik samt en programvara för att skriva och ladda program till kortet (Wikipedia 2013d). Dessa är utvecklade för att vara billiga och enkla att använda men ändå kunna utföra många olika typer av uppgifter.

2.2 Systembeskrivning industrirobot

En industrirobot är en maskin som utför uppgifter inom industrin. De flesta industrirobotar från ABB utgörs av en ledad arm som kan röra sig kring sex axlar. På armen kan olika verktyg monteras. Eldrivna servomotorer och avancerade växellådor möjliggör en hög precision (ABB 2013a).

2.2.1 Industrirobot ABB 1600-8/1.45

På institutionen för produkt och produktionsutveckling vid Chalmers Tekniska högskola finns ett antal industrirobotar från ABB. För att undersöka kombinationen av industrirobot och 3D-skrivare fick projektet tillgång till en industrirobot av typen ABB IRB 1600-8/1.45 som är en mångsidig högprestandarobot med hög noggrannhet. Räckvidden är 1,45 m, den repeterbara noggrannheten 0,05 mm och laster upp till 8,5 kg kan hanteras (ABB 2013b).

2.2.2 RobotStudio - utvecklingsmiljö för robotprogrammering

RobotStudio är programvara för att offlineprogrammera ABB-robotar och simulera robotens arbete. En så kallad virtual controller (VC) i RobotStudio agerar likt styrskåpet till en verklig robot och ger användaren möjlighet att programmera och simulera program utan att en robot är inkopplad (ABB 2012). RobotStudio kan även arbeta mot riktiga styrskåp och ladda upp kompletta program för roboten.

2.2.3 RAPID - programmeringsspråk för ABB-robotar

RAPID är ett högnivåspråk utvecklat av ABB för styrning av ABB-robotar. Utöver att ha samma funktionalitet som i andra högnivåspråk har språket egna instruktioner speciellt designade för att styra robotar. Exempel på ett kommando är {MoveL} får roboten att röra arbetsverktyget i en linjär rörelse (ABB 2007). Syntax för {MoveL} är

MoveL p10, v1000, z0, tool0;

{MoveL} står för "move linear" och rör roboten i en rak linje, där {p10} är en fördefinierad så kallad {robtarget}. I en robtarget finns koordinaterna för den position som roboten ska röra sig till samt verktygets och robotaxlarnas orientering i nämnda position. {v1000} är hastigheten för rörelsen i mm/s och {z0} är ett zonedata-argument. {zonedata} anger hur nära roboten ska gå den programmerade positionen innan den utför nästa kommando. Att zonedata är {z0} betyder att när roboten är 0,3 mm ifrån den programmerade positionen börjar omorienteringen för nästa position, detta innebär alltså att roboten aldrig kommer att köra hela vägen fram till den programmerade positionen.

Det sista argumentet, i detta fall {tool0}, specificerar vilket verktyg som ska röra sig fram till positionen. I kombinationen av industrirobot och 3D-skrivare är det spetsen på munstycket som är verktyget.

I RAPID kan oönskade händelser hanteras med en felhanterare (ABB 2011b). I koden programmeras att när ett visst fel inträffar anropas felhanteraren och därifrån vidtas lämpliga åtgärder. Detta är viktigt för att undvika skador på roboten eller dess omgivning.

2.2.4 Robotens konfiguration

Med robotens konfiguration menas den uppsättning av axelinställningar roboten har i en viss position (ABB 2007). Med sex frihetsgrader har roboten teoretiskt sett ett stort antal möjliga konfigurationer i de flesta positioner, undantaget är extrempunkter på randen av robotens arbetsområde.

2.2.5 Kalibrering av arbetsobjekt och verktyg

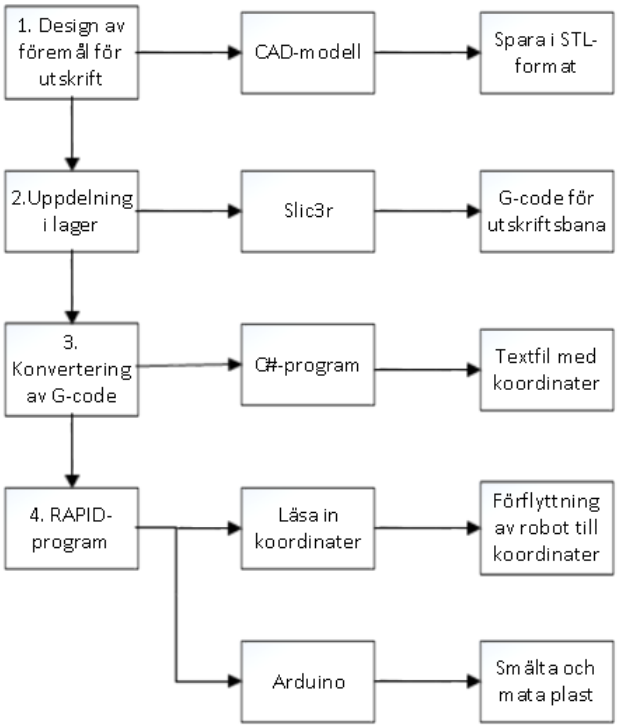
För att definiera ett koordinatsystem i det fysiska rummet, som roboten kan utgå från vid beräkning av rörelser, görs en kalibrering. Kalibrering-

en består av en rutin där robotens arbetsverktyg förflyttas till positioner i rummet. Utifrån dessa beräknas ett koordinatsystem för arbetsobjektet. En trepunktsmetod för att kalibrera arbetsobjektet utförs genom att först förflytta robotens kalibreringsverktyg till en position som ska ligga på x-axeln, därefter till ytterligare en punkt som ska ligga på x-axeln och till sist en punkt som ska ligga på y-axeln. Därmed är ett koordinatsystem definierat. För att kalibrera robotens arbetsverktyg (skrivarhuvudet) används en liknande metod där en spets placeras i rummet och arbetsverktygets TCP (tool center point) förs så nära spetsen. Detta utförs med 3 till 9 olika rotationer eller konfigurationer. Utifrån dessa beräknas sedan en relativ position och rotation för TCP.

Förflyttning av roboten under kalibreringen görs inte i RobotStudio utan av en FlexPendant. En FlexPendant är en liten fristående dator kopplad till roboten varifrån roboten kan styras för hand med knappar och reglage (ABB 2011a).

3 Metod

Projektet gick främst ut på att integrera flera delsystem. Här presenteras först en övergripande bild över alla delsystemen. Därefter beskrivs metoden som användes för att implementera en lösning av varje delsteg samt hur dessa integrerades. De funktioner som skulle ingå i 3D-skrivarsystemet identifierades och en hierarkisk uppgiftsanalys gjordes, se figur 1.



Figur 1: Arbetsgången i systemet. Användaren börjar med en design i form av en CAD-modell. Slic3r skapar G-code som blir till koordinater samt instruktioner för plastmatning i en textfil med hjälp av ett C#-program. Textfilen används av robotprogrammet vid körning.

3.1 CAD-modeller för utskrift

Systemet konstruerades för att kunna skriva ut modeller som en designer har gjort i ett CAD-program och sedan exporterat i det vanligaste formatet STL. Projektet utvecklades så att en designer ska kunna göra egna CAD-modeller inom vissa begränsningar. Exempel på en vanlig begränsning är att om föremålet har en del som är vinklad utåt kan denna vinkel vara

högst 45 °. Vid större vinkel får nya lager svårt att fästa på föregående lager och strukturen faller ner. Begränsningen går ofta att kringgå genom att rotera föremålet och skriva ut hela föremålet i en annan riktning vilket görs automatiskt av mjukvaran, men flera stora vinklar i olika riktningar begränsas. Problematiken kan lösas med en stödstruktur som sedan tas bort. Automatisk stödstruktur implementerades inte i projektet utan designern får ha med detta i CAD-modellen.

För att ha modeller att utvärdera systemet gjordes ett flertal CAD-modeller över lämpliga föremål att skriva ut för att testa och utvärdera systemet. CAD-programmet som användes var CATIA V5R19.

3.2 Uppdelning av CAD-modeller i lager

STL-filen måste sedan skivas i lager, en process som på engelska kallas slicing. Till det behövdes programvara. Att implementera en egen programvara som gör detta hade varit onödigt då sådan programvara redan finns. Programmet Slic3r valdes eftersom det har öppen källkod och dessutom kan generera komplett G-code med instruktioner för hur utskriften skulle utföras (Slic3r 2013).

3.3 G-code för utskrift

Den G-code som Slic3r producerade innehöll ett flertal instruktioner som till exempel hur skrivarhuvudet skulle värmas upp, skrivarhuvudets förflyttningar samt hur mycket plast som skulle matas ut. Koden tolkades och relevanta instruktioner identifierades. Instruktioner för förflyttning av skrivarhuvudet konverterades till robotens programmeringsspråk RAPID med hjälp av en rutin som programmerades i C#. Rutinen producerar en fil med koordinater och instruktioner för huruvida utmatning av plast ska ske under dessa.

Inställningen av temperatur implementerades med hjälp av en mikrokontroller. Värmeelementet startas manuellt, och styrs sedan av en mikrokontroller under körning. Eftersom denna styrning är gemensam för alla CAD-modeller behövdes den inte skapas till varje ny modell utan ändras manuellt vid eventuell förändring av hårdvara, till exempel vid byte till en plast med annan smälttemperatur.

3.4 RAPID-kod för styrning av robot

RAPID-koden läser in de koordinater som genererats av C#-programmet utifrån G-koden och låter roboten röra sig till dessa koordinater. Om det skall ske en utmatning av material under förflyttningen aktiveras en digital utsignal. Denna utsignal är förbunden med en digital ingång på den mikrokontroller som styr stegmotorn i skrivarhuvudet. Koden skapades och simulerades i RobotStudio.

För att ta reda på vilken konfiguration som roboten ska ha under körningen gjordes en kalibrering av robotens arbetsverktyg (skrivarhuvudet) och arbetsobjektet (bordet). Utifrån kalibreringen av robotens arbetsverktyg valdes en lämplig konfiguration som sedan användes i RAPID-koden.

3.5 Skrivarhuvud och upphängning

Utmatning och smältning av material görs av ett skrivarhuvud monterat på roboten. Ett skrivarhuvud av typen Stepstruder MK8 från tillverkaren Makerbot införskaffades. Det är ett skrivarhuvud som används i 3D-skrivarna från deras Replicator-serie som finns till försäljning i handeln. Skrivarhuvudet skruvades fast på en metallplatta tillsammans med ett infäste till roboten.

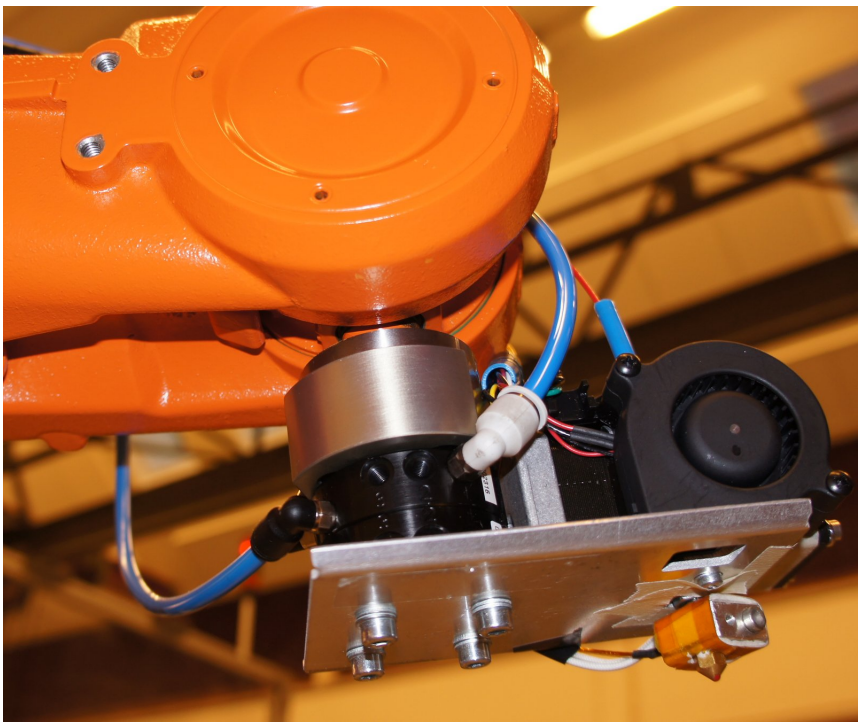
Utmatningen av plast i skrivarhuvudet sker med en stegmotor som drar fram plasttråden genom en uppvärmningsanordning där plasten smälter. Stegmotornhar 200 så kallade “steg”, ett steg innebär att mellan en halv och en millimeter plast matas. Materialet som användes var PLA men skrivarhuvudet klarar även att skriva ut med ABS. Temperaturen i skrivarhuvudet mättes av ett inbyggt termoelement av typ K. Signalen från denna förstärks med en termoelementsförstärkare MAX31855.

Den utmatade plasten kyls av en fläkt och en annan fläkt kyler stegmotorn. Från skrivarhuvudet löper kablage upp på robotarmen till en mikrokontroller. Kablarna skyddades för att klara av de rörelser som förekommer under en utskrift.

3.6 Kommunikation mellan robot och styrning av skrivarhuvudet

Styrningen av skrivarhuvudets funktioner sker med hjälp av en Arduino mikrokontroller placerad på roboten. Mikrokontrollern får en signal från

RAPID-programmet huruvida den ska låta stegmotorn köra framåt för att mata plast eller backa ett stycke, detta görs med hög, respektive låg signal. Backfunktionen minskar de trådar som kan uppstå av smält plast samt kompenserar för att plast fortsätter rinna ut en bit efter att stegmotorn stannar på grund av övertryck i munstycket. Signalen från en utgång på roboten är på 24V vid hög signal och under 1.5V vid låg. Spänningen omvandlas med hjälp av en spänningsdelare till 5V som är hög signal för Arduino. Endast en insignal används som kommunikation mellan robot och Arduino.



Figur 2: Skrivarhuvudet. Till höger syns en fläkt, värmeelementet inlindat i kaptontejp (värmeisolerande tejp) samt munstycket.

4 Resultat

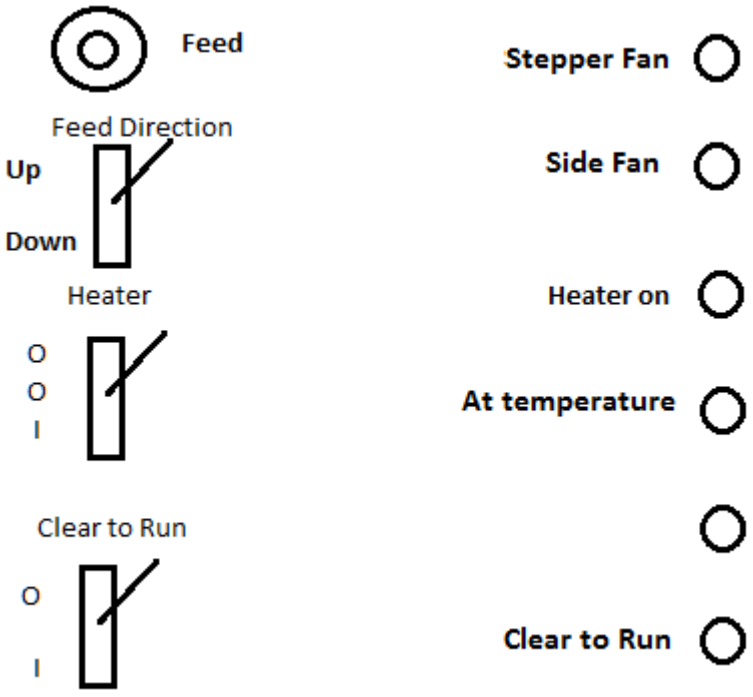
Här beskrivs systemet som har utvecklats i sin helhet. Därefter beskrivs hårdvaran som den består samt den mjukvara som används. Sist presenteras de resultat som erhållits från tester på systemet.

4.1 Beskrivning av systemet

Systemet som utvecklades är ett arbetsverktyg som monterat på en industrirobot. Detta verktyg kan sedan användas till att skriva ut föremål. Styrningen sker med hjälp av två mikrokontrollers varav den ena har digital insignal från robotens styrskåp. Med hjälp av ett RAPID-program styrs roboten utefter en förgenererad fil som RAPID-programmet läser av.

4.1.1 Robotverktyg - skrivarhuvud

Robotverktyget består av en bärande metallplatta där ett skrivarhuvud av modellen Stepstruder MK8, tillverkad av Makerbot, är placerad. På metallplattan är även en infästning för roboten monterad, se figur 2. Skrivarhuvudets munstycke är 0,4 mm. En Arduino mikrokontroller styr temperaturen på värmeelementet i skrivarhuvudet. Uppvärmningen kan stängas av och på med hjälp av en knapp inkopplad till mikrokontrollern. Mikrokontrollern är programmerad för att försöka hålla temperaturen runt börvärdet 220 °C. Utmatningen styrs av en annan Arduino. En fläkt kyler stegmotorn och en fläkt kyler den utskrivna plasten, båda fläktarna kräver 12 V. För kretsschema och lista på elektriska komponenter till robotverktyget se appendix C.

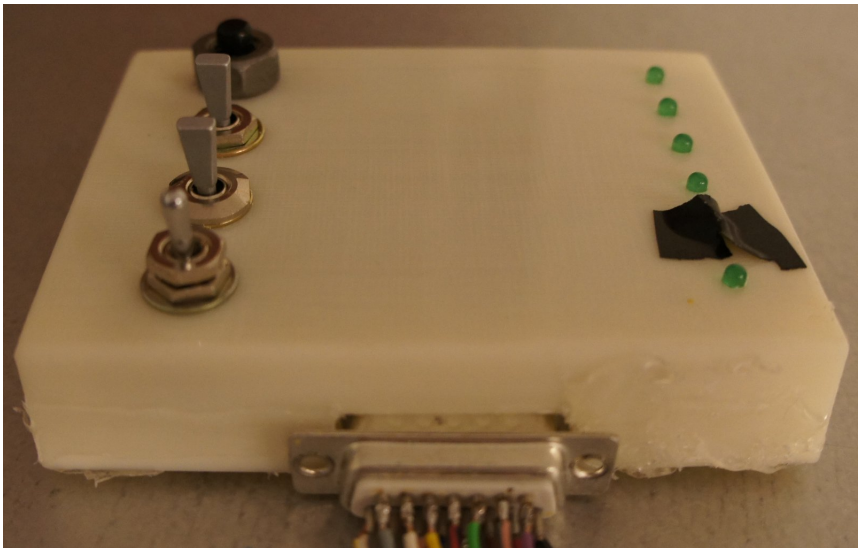


Figur 3: Schema över skrivarhuvudets kontrollpanel.

4.1.2 Användargränssnitt för skrivarhuvud

Vid utskrift styrs elektroniken på skrivarhuvudet från en kontrollpanel, se figur 4 och figur 3. Kontrollpanelen har ett reglage för att starta uppvärmningen av värmeelementet, ett reglage för att välja riktning för plastmatning-

en, en knapp för att mata plast samt ett reglage som kallas Clear-To-Run (CTR). CTR måste vara aktiverad för att automatisk matning via roboten skall vara tillåten, och för att manuell matning skall vara tillåten måste CTR vara låg. Detta för att förhindra att man råkar mata manuellt under automatiskkörning. Det finns fem LED-lampor som indikerar att värmeelementet är på, att värmeelementet har uppnått rätt temperatur, huruvida CTR är aktiverad samt att stegmotorfläkten och fläkten som är riktad mot den utmatade plasten är igång.



Figur 4: Skrivarhuvudets kontrollpanel. Knappar och reglage till vänster och LED-lampor till höger.

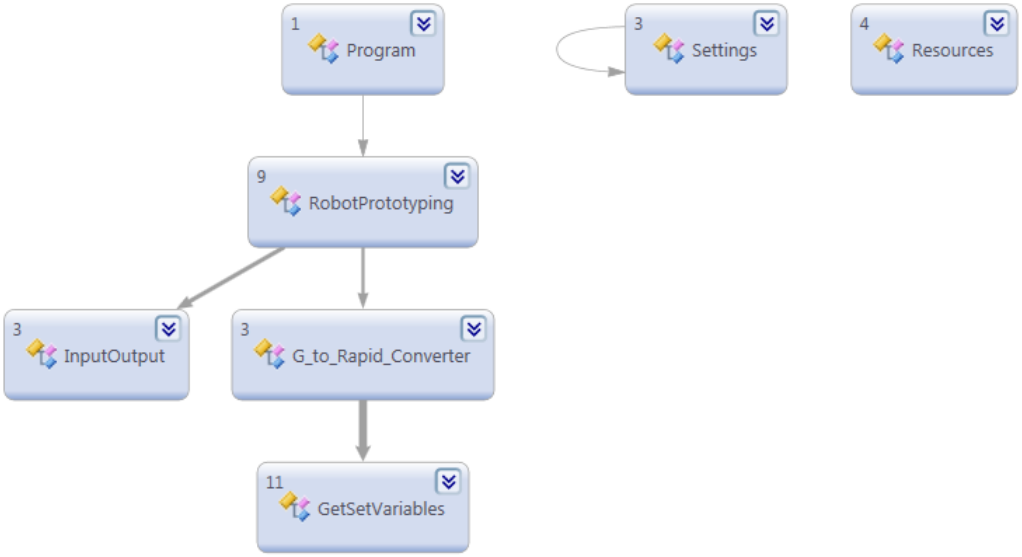
4.1.3 Plattform för utskrift

Systemet är byggt för att kunna skriva ut på en skiva av polykarbonat placerad på en fixtur. På skivan fästes Scotch 2090-tejp som PLA enkelt fäster på vid rumstemperatur. Tejpen placerades tätt, men inte överlappande då ytan i så fall skulle ha blivit ojämn. Någon millimeter mellanrum som innebär exponerad yta av polykarbonatskivan är inget problem. Vid kalibrering av arbetsobjektet (bordet) kalibreras xy-planet så nära tejpen som möjligt och x-axeln kan lämpligtvis kalibreras så nära kanten närmast roboten som möjligt. Z-axeln kalibreras rakt uppåt ifrån bordet. Observera att en z-axel kalibrerad nedåt skulle innebära att roboten försöker köra igenom bordet vilket skulle skada robot, arbetsverktyg och plattform, och detta är ej önskvärt. I koden för konvertering av koordinater adderades några millimeter i x- och

y-led för att utskriften inte skulle ske på kanten av bordet (som är origo i koordinatsystemet).

4.1.4 C# -kod och interface

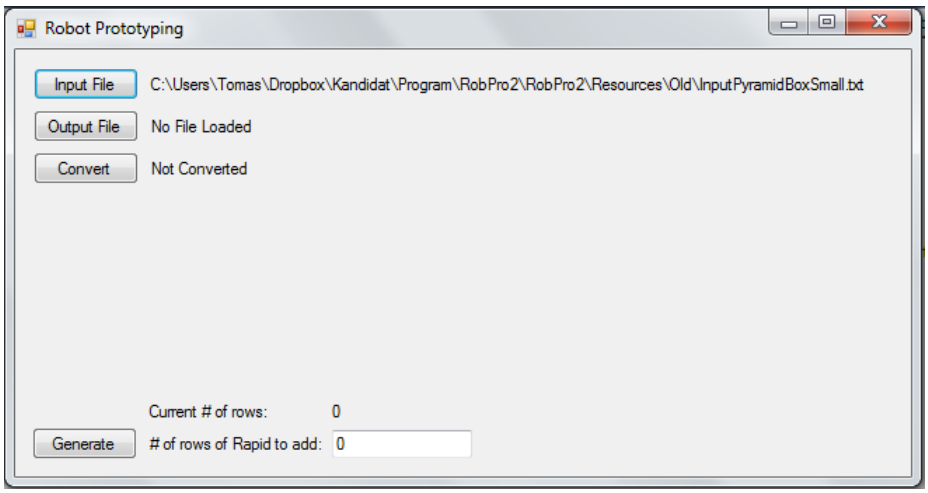
För fullständig C#-kod se appendix F och för figur över strukturen i C#-programmet se figur 5.



Figur 5: Kodstruktur i C#-programmet.

I C# programmerades en rutin som extraherar koordinaterna för utskriftsbanan, se figur 6. Programmet består av ett enkelt interface som tillåter användaren att välja en textfil med G-code och en utfil. Programmet skriver ut en textfil med koordinater samt instruktioner för matning av plast. En rad i textfilen ser ut som följande: “1;[10.5, 2.7, 5.0]”. Plastmatningsinstruktionen är “1” om skrivarhuvudet ska mata plast, annars “0”. Koordinaten anger sedan nästa position “[x,y,z]” i millimeter. Programmet har även möjlighet att translatera alla positioner i valfri riktning för att flytta utskriftsföremålet relativt utskriftsytan genom att göra modifikationer i koden. Tanken var att senare vidareutveckla programmet till att fungera på ett liknande vis, men med möjlighet att göra alla val direkt i programmet utan att behöva ändra i koden.

För att generera G-code används mjukvaran Slic3r, se figur 7. Inställningar som gjordes i programmet var “filament=1,75 mm”, “nozzle= 0,4 mm”



Figur 6: Användargränssnitt för C#- programmet.

och "layer height=0,4 mm". Inställningarna för "bed size" anpassades efter skrivytan.

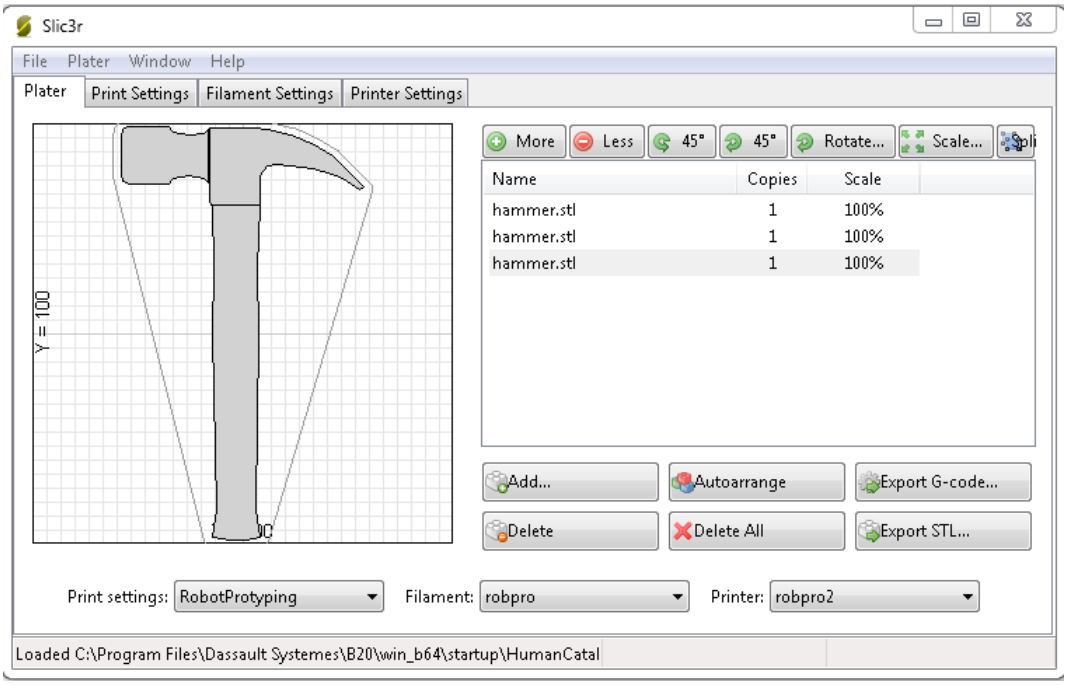
4.1.5 RAPID-kod

För fullständig RAPID-kod se appendix D.

Koden är strukturerad i rutiner som gör olika delar i utskriften. Den första rutinen {Open_file} upprättar kommunikationen med textfilen där alla koordinater finns. Därefter upprättas en övervakningsprocess i {TriggStopProc} som sätter den digitala utsignalen för matning av plast till 0 i händelse av att körningen av programmet avbryts. Sedan anropas {Test_plastic} som skriver ut en sträng med plast för att säkerställa att plasten är frammatad i skrivarhuvudet. Slutligen exekveras {Move_to_all} som förflyttar roboten till koordinaterna och skickar signaler om att mata plast under tiden.

I {Move_to_all} öppnar instruktionen {Open} kommunikationen med textfilen med koordinater och {ReadStr} läser in en rad i taget som en textsträng (ABB 2007). {StrToVal} tolkar och omvandlar strängen till vald datatyp, i detta fall en position.

{SetDO} ändrar värdet på en digital utsignal och används i kommunikationen med en mikrokontroller som styr skrivarhuvudets utmatning (ABB 2007). Om utsignalen är 1 betyder det att skrivarhuvudet ska mata plast, om den är 0 ska den dra tillbaka plasten ett kort stycke.



Figur 7: Användargränssnitt i Slic3r. Användaren laddar in sin CAD-modell i STL-format och Slic3r genererar G-code för utskriftsbanan.

Därefter ges roboten instruktionen {MoveL} så att den flyttar skrivarhuvudet till en annan position samtidigt som den matar eller inte matar plast. Programmet genomför inläsning av alla rader i textfilen tills det inte finns någon mer rad att läsa in. När slutet av filen nås, stängs kommunikationen med filen och programmets körning avslutas.

4.1.6 Arduino-kod

För fullständig Arduino-kod se appendix E.

Två stycken Arduino mikrokontroller programmerades, den ena styr matningen av plast och den andra styr smältningen av plast. Rutinen för att smälta plast sätts igång genom att användaren manuellt får slå om en knapp för att sätta igång upphettningen av skrivarhuvudets värmeelement. Så länge temperaturen är under 220 °C är värmeelementet igång. Uppvärmningen stoppas då temperaturen är över 220 °C. Samtidigt indikerar en LED-lampa när värmen är över 210 °C, så att användaren vet att plasten är tillräckligt varm för utskrift.

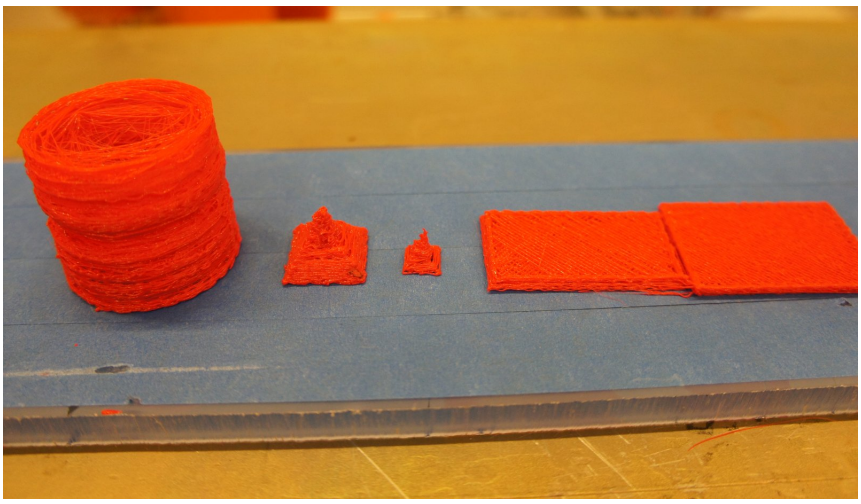
Matningen av plast sker med hjälp av en stegmotor som drar fram plasttråden genom skrivarhuvudet. Stegmotorn styrs av en mikrokontroller som i sin tur har en ingång från robotprogrammet. Om roboten skickar en signal om att mata plast, tillåts stegmotorn att stega framåt tills signalen nollställs. När signalen nollställs byter stegmotorn håll och stegar tillbaka några steg för att dra tillbaka lite plast.

4.2 Experiment

Ett antal föremål skrevs ut med 3D-skrivarsystemet. Se figur 8. Det första testet vi gjorde var de två sammansatta plattorna, det andra en liten pyramid, det tredje var samma pyramid men uppskalad till dubbla storleken, det sista testet var ett glas. Vid de fyra testen användes olika värden på robotens hastighet, stegmotorns hastighet och fyllningsgraden för materialet. I det första testet där vi skrev ut två plattor hade vi en fördröjning mellan varje steg på stegmotorn 36 ms, begränsad hastighet v50 på roboten och körning i 40% av hastigheten. Begränsad hastighet innebär att roboten inte kör i den programmerade hastigheten, utan med en reducerad hastighet. På de resterande tre testen hade vi en fördröjning på 19 ms, begränsad hastighet på v50 vid 80%. På det sista testet var skillnaden att fyllnadsgraden av plast var satt till 0.8 (80%) istället för 0.4 (40%). Ett koncentriskt fyllnadsmönster testades istället för ett rätlinjigt.

En lägre hastighet på utskriften medförde ett jämnare lager plast och ett bättre resultat. Att plasten hann stelna innan nästa lager applicerades var viktigt, annars höll föremålet inte formen. Vid konstruktion av mindre detaljer där det dröjer en kortare tid mellan lagren blev detta uppenbart och topparna på pyramiderna blev deformerade.

Glasets väggar skrevs föga oväntat ut på höjden. Väggarna var designade att bli raka men deras form blev oregelbunden med något som ser ut som spår i väggarna. När material skrevs ut i en bågformad bana var det svårare för plasten att fästa mot underlaget då plasten utsattes för en kraft i sidled, och en utskriven cirkel liknade mer en polygon. Plasten fick endast fäste vid vissa punkter och däremellan blev det en rak linje med plast. Detta beror på att föregående lager var lika bristfälliga och underlaget för plasten att fästa på ej var intakt.



Figur 8: De första föremålen som skrevs ut. Ett glas, två pyramider, och två sammanhängande plattor.

5 Diskussion

5.1 Systemets utformning

De alternativ vi hade gällande skrivarhuvuden var att antingen bygga ett eget, att köpa ett befintligt eller att köpa en hel 3D-skrivare och ta loss skrivarhuvudet. Då projektets omfattning var relativt stor valde vi att köpa en befintlig lösning så att kandidatarbetet kunde fokusera på att få systemet av delprocesser att fungera tillsammans. Detta medförde att vi fick ett robust och välbeprövat skrivarhuvud som egentligen var anpassat för en annan befintlig skrivare.

De vanligaste materialen vid 3D-printing är ABS och PLA. Båda fungerar utmärkt som material för projektets behov. ABS tål något högre temperaturer men behöver en uppvärmt byggplatta för att fästa. PLA behöver ingen uppvärmd byggplatta utan det räcker med en viss typ av tejp eller ett underlag av polykarbonat. Då det slutgiltiga målet var att kunna skriva ut stora föremål valdes ABS bort till förmån för PLA. En stor uppvärmd skrivyta är på flera sätt problematiskt, med bland annat stora värmeförluster till omgivningen. ABS avger vid upphettning en ånga som kan lukta något och irritera personer som är känsliga. PLA avger vid upphettning en ånga som generellt luktar och irriterar mindre än ABS. Då utskrifterna kommer att ske i en stor lokal över långa perioder är dessa egenskaper hos PLA en stor fördel. PLA

är dessutom mer miljövänligt då det tillverkas av biologiska material och är biologiskt nedbrytbart, till skillnad från ABS som tillverkas av olja.

Då det inte fanns en uppenbar lösning från början uppstod det många hinder som var svåra att förutspå. Detta innebar att flera lösningar av tidiga delproblem behövde omarbetas. En del arbete visades sig vara överflödigt och användes inte. Ett exempel var att tid lades på PC SDK, som används för att styra en robot direkt via till exempel C#, vilket senare inte användes alls. Mycket testkod användes för att se hur senare problem skulle lösas innan slutgiltiga lösningar på de tidiga problemen kunde implementeras.

Många av delproblemen var komplexa och hade krävt en stor arbetsinsats att lösa från grunden. Men det visade sig att det ofta fanns färdiga lösningar som relativt enkelt kunde användas, om än några behövde specialanpassas något. Forumen för hobbyanvändare av 3D-skrivare präglas av en öppen kultur och användarna delar gärna med sig av sitt kunnande. Även mikrokontrollerföretaget Arduino delar med sig av sina lösningar och mycket av koden för att styra stegmotorer fanns väldokumenterad.

I projektet användes flera programmeringsspråk såsom RAPID, Arduino, G-code samt C#. Då tiden var begränsad förenklades vissa avancerade funktioner. Utmatningen av material och styrningen av roboten skulle kunna skötas av samma program, men det hade krävt en mer avancerad kod i RAPID som i sig inte är anpassat för att styra stegmotorer på det sätt som Arduino är. Robotstyrskåpet har möjlighet att kommunicera seriellt med Arduino men att utveckla en kommunikation för mer avancerade instruktioner låg utanför tidsramen för projektet. Lösningen som valdes innebär att robotens styrskåp inte har möjlighet att ha kontroll över matningen utan får förlita sig på att Arduinon sköter detta själv. Då Arduinon inte vet vilken typ av rörelse roboten gör försvann möjligheter såsom att styra temperaturen från roboten eller att mata ut med varierande hastighet. Vissa problem med att anpassa utmatningen, exempelvis att göra första lagret tjockare, kunde lösas genom att köra roboten långsammare under utskriften av detta lager.

För projektet räckte det med endast en robotkonfiguration, därför var det inte nödvändigt att ta fram en metod för att variera konfigurationen över volymen. Om roboten är tänkt att arbeta över en större volym kan konfigurationen skilja sig i olika positioner vilket innebär att roboten inte kan nå alla koordinater i samma konfiguration. En rutin för när roboten tvingas byta konfiguration skulle då behöva utvecklas.

Mot slutet av projektet uppstod en del problem med elektroniken. Uppvärmningen av skrivarhuvudet gick långsamt och det var svårt att nå rätt

temperatu, detta antogs bero till stor del av att upphängningen av aluminium verkade som en kylfläns och avledde värme. När plast matades ut under längre perioder föll temperaturen för snabbt för att processen skulle vara stabil. Detta kunde temporärt lösas genom att manuellt tillföra extra värme under processen. Felsökningen av elektronikfel medförde att flera av testerna inte kunde genomföras som planerat inom projektets tidsram. Experiment med utskrifter gick att genomföra men kvalitéten var för dålig för att skulle vara meningsfullt att analysera dem och det saknades tid att förfina processen. FDM är en känslig teknik där plaststrålen från munstycket kan böja sig vilket påverkar hur raka strängarna blir. Projektets prototyp testades inte på ett större föremål men räckvidden på roboten är tillräcklig för att kunna skriva ut objekt som är flera gånger större än de som Makerbot Replicator 2 producerar.

5.2 Industrirobotens lämplighet som 3D-skrivare

Industrirobotar har en stor massa vilket innebär ett större tröghetsmoment och större energiåtgång än i mindre 3D-skrivare. De har dock en hög repeterbarhet, och en bra kalibrering av arbetsobjekt och verktyg kan leda till fina resultat. Beräkningar av rörelser i 6 frihetsgrader kräver mer beräkningar men för roboten innebär detta i allmänhet inga problem. En stor utskrift kommer att bestå av många förflyttningar men detta kommer bara att föranleda i en längre textfil med koordinater.

Att skriva ut stora föremål med hög noggrannhet tar lång tid. En kommersiell 3D-skrivare av typen Makerbot Replicator har en utskriftshastighet på $24\text{cm}^3/\text{h}$ (Makerbot 2013b). Att avsätta en robot för ett ändåmål en lång tid i produktionen innebär stora kostnader. En snabbare teknik som eventuellt har lägre noggrannhet kan fortfarande vara intressant vid utskrift av stora föremål. Vidare studier på alternativa material och 3D-tillverkningstekniker behövs. Med FDM skulle industrirobotar fortfarande vara lämpliga för stora föremål men med liten volym då tillverkningstiden kan förkortas. Ett exempel är utskrift av stora fast ej solida detaljer.

5.3 Potentiella användningsområden

Ett syfte för projektet var att analysera potentiella användningsområden för tekniken. En möjlighet finns att använda tekniken för fyllnad av mellanrum som uppstår vid nitning av flygplansvingar. Då olika material har olika

egenskaper är det främst materialvalet som styr vilken tillverkningsteknik som används. I projektet användes den additiva tillverkningstekniken FDM och materialet PLA. PLA fäster inte på aluminium som är det vanligaste materialet vid tillverkning av flygplan. En tunn film polykarbonat ovanpå aluminiumplåten skulle få PLA att fästa även vid rumstemperatur.

Industrirobotar innebär en större inköpskostnad relativt kartesiska robotar, men då de har många användningsområden finns de ofta redan i de flesta fabriker. Ett robotverktyg för 3D-utskrifter skulle förvandla en industrirobot till en 3D-skrivare för relativt låg kostnad jämfört med att köpa ett helt 3D-skrivarsystem. För mindre projekt skulle en sådan lösning innebära en besparing, om en industrirobot redan innehavs.

Genom att ha en robot med generell förmåga och möjlighet att byta verktyg kan roboten användas till flera aktiviteter i samma process. Ett exempel kan vara att dra en elledning i en cementvägg under tiden den byggs upp. Det räcker då att roboten byter mellan verktyg eller att flera robotar arbetar samtidigt.

Att industriroboten har fler frihetsgrader än kartesiska robotar öppnar upp för att ha varierande orientering på arbetsverktyget. Då skulle det gå att skriva på ojämna ytor med arbetsverktyget fortfarande vinkelrätt mot ytan. Med ett utskriftsmaterial som fäster direkt skulle det gå att skriva ut på lutande ytor.

En industrirobot behöver inte nödvändigtvis hålla i skrivarhuvudet utan kan istället hålla i föremålet eller plattformen som föremålet befinner sig på. Sedan flyttas föremålet relativt ett fast skrivarhuvud som skriver ut material på plattformen. Det fasta skrivarhuvudet kan även bytas mot ett skrivarhuvud monterat på en annan robot. På så vis kan man under utskrift rotera föremålet 90°och därmed skriva ut rakt i föremålets sidled. Med FDM är en sådan utskrift i dagsläget inte möjligt utan stödmaterial. Stödmaterialet innebär andra begränsningar, till exempel måste materialet vara möjligt att komma åt efter utskrift i syfte att avlägsna det. Därför är det idag omöjligt att skriva ut vissa former av ihåligheter såsom en ihålig kub. Om det vore möjligt hade tekniken lett till önskvärda egenskaper såsom minskad materialåtgång, viktbesparing, ökad yta för värmeavledning, minskad värmeledningsförmåga genom materialet, samt möjligheter till nya mekaniska strukturer.

Tekniken behöver inte enbart begränsas till industrirobotar. Det finns flera andra typer av robotar med ledade armar som kan användas för additiv tillverkning. En ledad arm har flera attraktiva egenskaper, bland annat en stor arbetsvolym och en generell arbetsförmåga. Det byggs idag hus med hjälp

av stora kartesiska robotar som bygger upp cementväggar. En stor kartesisk robot körs till byggarbetsplatsen och bygger sedan ovanifrån. Här skulle det kunna vara intressant att kunna byta ut arbetsverktyget på grävskopan mot ett arbetsverktyg som gör om grävskopan till en 3D-skrivare som skriver ut i cement. Detta innebär att ingen extra byggplattform måste köras till byggarbetsplatsen, och att den möjliga arbetsytan blir större. Ofta behövs grävmaskinen redan för bygget och finns redan på plats.

Ifyllnader av håligheter kan även vara intressant i andra sammanhang. Defekter uppstår ofta vid normal användning på produkter vars estetiska utseende är viktig för konsumenten. En 3D-skrivare skulle då kunna fylla igen dessa. Ett exempel är att fylla igen defekter på bilar. Om en bil kör in i ett rum likt en automatisk biltvätt, kan bilen bli avläst av en skanner. Därefter skulle en ledad arm kunna föras runt bilen och fylla i eller jämna ut alla defekter som har uppstått i plåt och plastdetaljer. Därefter kan armen byta arbetsverktyg och applicera färg på materialet.

5.4 Vidareutveckling

RAPID-koden är anpassad för att flytta roboten i den hastighet som är lämplig beroende på stegmotorn som matar ut plast ur skrivarhuvudet. På så vis behöver inte stegmotorn ändra hastighet. En mer avancerad kommunikation, där stegmotorns hastighet sätts av roboten, lämnas till framtida vidareutveckling. Vilken vinkel på underlaget som det går att skriva ut olika material vid är intressant för vidare studier.

Det munstycke som används i skrivarhuvudet är 0,4 mm i diameter. Att diametern är så liten innebär att föremål med väldigt jämna och fina ytor kan skrivas ut men också att utskrifterna tar en lång tid. För ändamålet att skriva ut stora föremål med lägre detaljnivå är ett skrivarhuvud med större munstycke att föredra. För att detta ska fungera behövs ett större munstycke, mindre förändringar i mjukvaran och en förnyad kalibrering av robotverktyget.

Vid utskrift av större föremål med dimensioner som skiljer sig mycket från de kalibrerade kan roboten behöva byta konfiguration. Genom att analysera var i robotens koordinatsystem den behöver byta konfiguration skulle en konfiguration för varje koordinat kunna erhållas och roboten skulle kunna byta konfiguration under utskriften. Eventuellt skulle detta medföra att roboten tvingas göra vissa komplicerade rörelser långsammare.

Analys av noggrannhet och prestanda bör analyseras.

Referenser

ABB (2007) *Technical reference manual RAPID Instructions, Functions and Data types.* RobotWare 5.0 Document ID: 3HAC16581-1

ABB (2011a) *Operating manual IRC5 with FlexPendant.* Document ID: 3HAC16590-1

ABB (2011b) *Technical reference manual RAPID overview.* RobotWare 5.14 Document-ID: 3HAC16580-1

ABB (2012) *Operating Manual RobotStudio 5.14.* Document ID: 3HAC032104-001

ABB (2013a) *Industriroboten.* `http://www.abb.se/cawp/seabb361/1a2fa56f98d23270c1257251003027f7.aspx` (2013-05-16)

ABB (2013b) *IRB1600 Data Sheet* Document ID:PR10282EN_R7

Creative Tools (2013) *Jämförelse av plaster för FFF 3D-skrivare.* `http://www.creativetools.se/material-jamforelse-se` (2013-05-16)

Makerbot (2013a) *3D printing concepts.* `http://www.makerbot.com/support/guides/printing/#slicing` (2013-05-16)

Makerbot (2013b) *Makerbot Replicator* `http://store.makerbot.com/replicator.html` (2013-05-22)

Rpworld (2013) *Stereolithography Apparatus.* `http://rpworld.net/cms/index.php/additive-manufacturing/rp-rapid-prototyping/sla-stereo-lithography-apparatus.html` (2013-05-21)

Slic3r (2013) *About Slic3r.* `http://slic3r.org/about` (2013-05-16)

Wikipedia (2013a) *Friformsframställning.* `http://sv.wikipedia.org/wiki/Friformsframst%C3%A4llning` (2013-05-16)

Wikipedia (2013b) *3D printing.* `http://en.wikipedia.org/wiki/3D_printing` (2013-05-16)

Wikipedia (2013c) *STL (file format).* `http://en.wikipedia.org/wiki/STL_(file_format)` (2013-05-16)

Wikipedia (2013d) *Arduino.* `http://en.wikipedia.org/wiki/Arduino` (2013-05-16)

Wikipedia (2013e) *Fused deposition modeling.* `http://en.wikipedia.org/wiki/Fused_deposition_modeling` (2013-05-17)

Wikipedia (2013f) *Polylaktid.* `http://sv.wikipedia.org/wiki/Polylaktid` (2013-05-16)

Wikipedia (2013g) *Polylactid acid.* `http://en.wikipedia.org/wiki/Polylactic_acid` (2013-05-16)

Wikipedia (2013h) *G-code.* `http://en.wikipedia.org/wiki/G-code` (2013-05-16)

Wikipedia (2013i) *Thermocouple.* `http://en.wikipedia.org/wiki/Thermocouple#K` (2013-05-19)

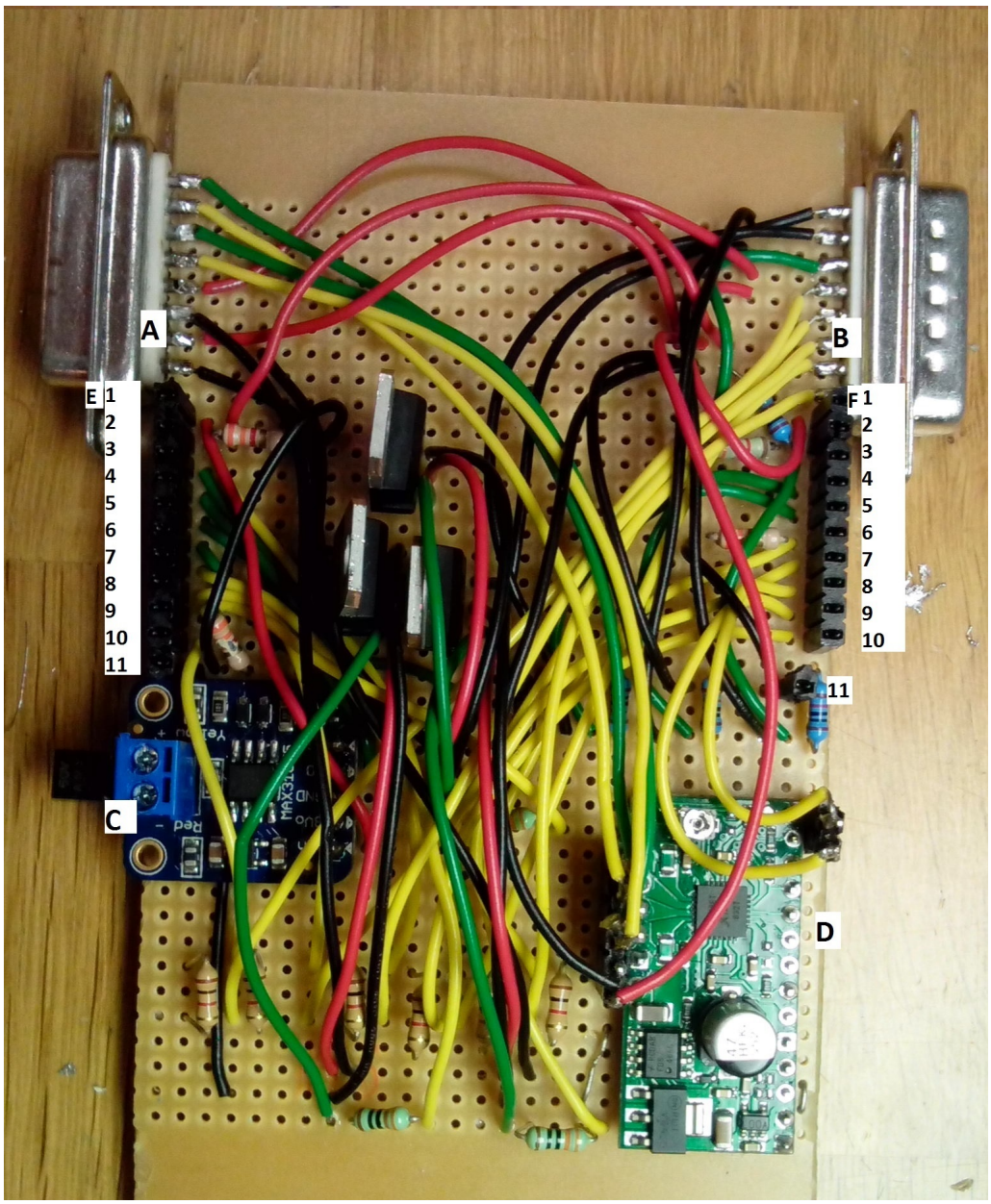
A Instruktioner för användning

1. Designa den CAD-ritning som ska skrivas ut.
2. Spara CAD-ritning STL-format.
3. Starta Slic3r och kör Configuration Wizard.
4. Ange i inställningar:Makerbot, 1.75mm filament och 0,4 mm munstycke.
5. Under Print Settings - Layers and Perimeters ange layer height = 0,4 mm.
6. Öppna STL-filen för CAD-ritningen i Slic3r.
7. Ändra eventuellt inställningarna för att ändra utskriftsbanan.
8. Klicka på Export G-code.
9. Konvertera filen med G-code till .txt-format.
10. Skapa en utfil (.txt) som du vill skriva till.
11. Om det behövs kan utskriftspositionen translateras i C#-koden.
12. Starta programmet (antingen genom exekveringsfilen eller via Visual Studio prjektet)
13. Klicka på “Input File” och välj textfilen med G-code.
14. Klicka på “Output File” och välj textfilen du vill skriva till.
15. Klicka på “Convert” och den valda utfilen innehåller nu koordinaterna för utskrift
16. Kalibrera work object och ändra bord3dprinter i MainModule.
17. Ändra koordinater i Test_plastic så att en sträng med plast skrivs ut på en tom yta.
18. Kontrollera konfigurationen inom utskriftsvolymen och ändra eventuellt i MainModule.
19. Kontrollera rätt sökväg till filen med koordinater.
20. På roboten: skapa ett nytt program, ladda upp MainModule till programmet.
21. Håll robotverktyget positionerat mot robotens verktyg.

22. På FlexPendant sätt DO10_1 till hög, verktyget sitter nu fast.
23. På FlexPendant sätt DO10_1 till låg igen, verktyget sitter fortfarande fast. För att senare ta loss verktyget, använd DO10_2.
24. På FlexPendant välj verktyget nozzle3dprinter som tool.
25. Utför en kalibrering av utskriftsytan.
26. Stega inkrementellt runt z=1mm i sidled och se vid vilken höjd som munstycket river upp tejp. Addera denna höjd till alla koordinater i C#-koden. River munstycket upp tejp vid z=0,9 så addera 0,9 mm så att första lagret (som är 0,4 mm i diameter) börjar vid 1,3mm.
27. Jogga roboten till en lämplig plats där filamentet har möjlighet att matas in i skrivarhuvudet.
28. Slå på uppvärmningen av skrivarhuvudets värmeelement. Vänta tills uppvärmningen temperaturen har nått 220 °C.
29. Tryck ner knappen för manuell matning och för samtidigt in filament i skrivarhuvudet tills stegmotorn får tag i filamentet.
30. Kör manuellt tills tillräcklig mängd plast har gått igenom skrivarhuvudet för att gamla rester ska vara helt borta. Avbryt matningen.
31. På FlexPendant, välj Debug, Set PP to main, och för att slutligen starta programmet.

B Kopplingsbeskrivning

Se figur 9.



Figur 9: Kommunikationshub med kretskort.

A: D-Subkontakt till skrivarhuvudpaketet

1. Stegmotorfläkt, negativ pol
2. Stegmotorfläkt, positiv pol
3. Sidofläkt, negativ pol
4. Sidofläkt, positiv pol
5. Stegmotor D
6. Stegmotor B
7. Stegmotor A
8. Stegmotor C
9. Värmeelement, negativ pol
10. Värmeelement, positiv pol

B: D-subkontakt till kontrollpanelen och industrirobot

1. LED, stegmotorfläkt
2. LED, Clear-To-Run
3. LED, värmeelement
4. LED, temperatur
5. LED, sidofläkt
6. Input ABB
7. Reglage , Clear-To-Run
8. Reglage , värmeelement
9. Spänningskälla, 12V, negativ pol
10. Spänningskälla, 12V, positiv pol
11. Jord ABB
12. 5V, Arduino 1 (stegmotorstyrning)
13. 5V, Arduino 2 (temperaturstyrning)
14. Reglage, mata plast
15. Reglage, matningriktning

C: Adafruit - Thermocouple Amplifier MAX31855

D: Pololu - A4988 Stepper Motor Driver Carrier with Voltage Regulators

E: Kopplingar till Arduino 2 (Temperaturstyrning) [#: där siffran är ingången på Arduino]

1. 5V
2. 2: Utgång, sidofläktkontroll
3. 3: Adafruit, CLK
4. 4: Adafruit, CS
5. 5: Adafruit, DO
6. 6: Ingång, reglage, värmeelement
7. 10: Utgång, LED värmeelement på/av
8. 11: Utgång, LED temperatur uppnådd
9. 12: Utgång, värmeelement på/av
10. 13: Utgång, LED sidofläkt på/av
11. Jord

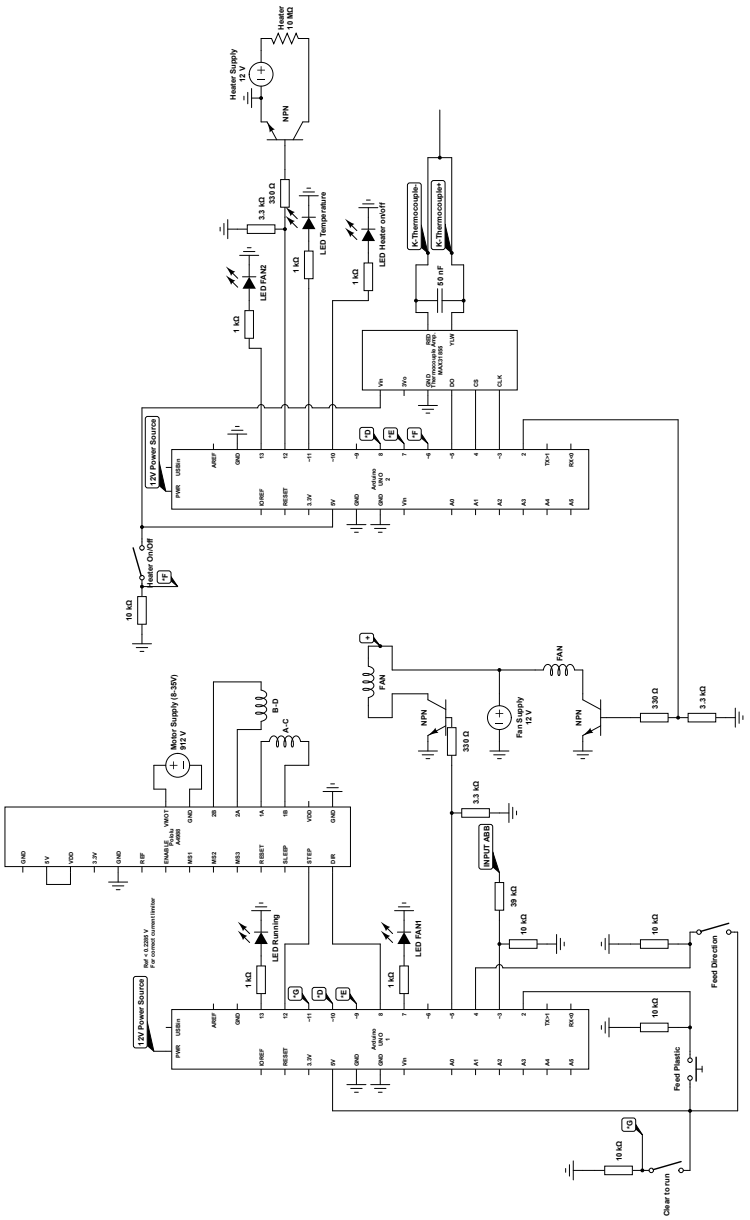
F: Kopplingar till Arduino 1 (Stegmotorstyrning) [#: där siffran är ingången på Arduino]

1. 5V
2. 3: Ingång, från ABB
3. 2: Ingång, reglage för matning
4. 4: Ingång, reglage för matningsriktning
5. 11: Ingång, reglage för Clear-To-Run
6. 5: Utgång, stegmotorfläktkontroll
7. 7: Utgång, LED stegmotorfläkt på/av
8. 8: Utgång, matningsriktning
9. 12: Utgång, stega
10. 13: LED, Clear-To-Run
11. 11: Jord

C Komponenter och kretsschema

Följande elektronisk hårdvara ingår i lösningen.

- 2 st. Arduino Uno
- 1 st. Pololu A4988
- 1 st. Thermocouple Amplifier MAX31855
- 3 st. NPN transistorer
- 1 st. NO (normally open) knapp
- 3 st. reglageknappar
- 17 st. resistanser
  - 1 st 39 kΩ
  - 5 st. 10 kΩ
  - 3 st. 3.3 kΩ
  - 5 st. 1 kΩ
  - 3 st. 330 Ω
- 5 st. spänningskällor (3 st. 9 V & 2 st. 12 V)



Figur 10: Kretsschema för elektroniken i 3D-skrivaren.

D RAPID-kod

```
MODULE MainModule
   CONST confdata targetconfig := [1, -1, 0, 0]; !robot configuration
   VAR iodev file; !coordinate file
   VAR string str;
   VAR string savestr; !!the recent digital output
   VAR bool ok; !checking the StrToVal conversion
   VAR pos position; !modifiable position for robtarget
   VAR robtarget temprobtarget :=[[100, 100,  100],[0.0122162,0.98626,
          -0.164722,0.00290463],targetconfig,
          [9E9,9E9,9E9,9E9,9E9,9E9]]; !modifiable robtarget
   VAR num button_number; !flexpendant buttons
   CONST errnum unknown_str := 10; !unknown string in coordinate file
   CONST errnum illegal_z := 20; !error for z<0
   PERS wobjdata table3dprinter := [FALSE,TRUE,"",[[-486.55,886.809,613.113],
         [0.999911,-0.00904704,0.000671354,0.00982808]],
         [[0.000357628,0.00166893,-0.00107288],[1,8.25185E-06,-1.87711E-06,
          -1.51781E-06]]]; !calibrated workobject
   PERS tooldata nozzle3dprinter := [TRUE,[[-24.4901,-66.269,86.1957],[1,0,0,0]],
            [0.5,[5,5,5],[1,0,0,0],0,0,0]]; !calibrated 3D-printer tool
   PERS restartdata restart;

   PROC main()
      Open_file;
      TriggStopProc restart\DO1:=DO11_4, DO11_4;
                          !when robot stops, stop printing plastic
      Test_plastic;
      Move_and_print;
   ENDPROC

   !Prints a plastic string before printing an object to
                  clear the extruder of other material
   PROC Test_plastic()
      position := [370, 86.3, 200];
      temprobtarget.trans := position;
      MoveJ temprobtarget,v100,z10,nozzle3dprinter\WObj:=table3dprinter;
      position := [370, 86.3, 1.3];
      temprobtarget.trans := position;
      MoveL temprobtarget,v100,fine,nozzle3dprinter\WObj:=table3dprinter;
      SetDO DO11_4, 1;
      WaitTime 0.1;
      position := [258, 86.3, 1.3];
      temprobtarget.trans := position;
      MoveL temprobtarget,v100,fine,nozzle3dprinter\WObj:=table3dprinter;
      SetDO DO11_4, 0;
      WaitTime 0.1;
      position := [100, 10, 200];
```

```
   temprobtarget.trans := position;
   MoveL temprobtarget,v100,fine,nozzle3dprinter\WObj:=table3dprinter;
ENDPROC

!opens communication with coordinate file in chosen file path
PROC Open_file()
   Open "/filepath/" \File:= "coordinatefile.txt",
                  file \Read;
   ERROR
      IF ERRNO = ERR_FILEOPEN THEN
         TPWrite "Device Access Error nr 41676";
         TPReadFK button_number, "What to do?",  stEmpty, stEmpty,
                           stEmpty,"TRYNEXT", "RETRY";
         IF button_number = 4 THEN
            TRYNEXT;
         ELSEIF button_number = 5 THEN
             RETRY;
         ENDIF
      ENDIF
   ENDPROC
```

```
!reads coordinates, sets output to Arduino for printing plastic and moves robot
   PROC Move_and_print()
      savestr :="0"; !the recent digital output
      WHILE (true) DO
         str := ReadStr(file \Delim:=";"); !reads a string to the delimiter ";"
                        which is a 1 or 0
         IF str="EOF" THEN !Finished if program has reached end of file.
                       Move robot for easy access to the printer tool.
            TPWrite "End of file.";
            SetDO DO11_4, 0;
            WaitTime 0.5;
            position := [0, 0, 300];
            temprobtarget.trans := position;
            MoveL temprobtarget,v100,fine,nozzle3dprinter\WObj:=table3dprinter;
            Close file; !closes the communication with the coordinate file
            RETURN; !break while loop
         ELSEIF str="1" THEN ! print plastic
            IF savestr="0" THEN
               SetDO DO11_4, 1;
               WaitTime 0.1;
            ENDIF
         ELSEIF str ="0" THEN !stop printing plastic
            IF savestr="1" THEN
               SetDO DO11_4, 0;
               WaitTime 0.1;
            ENDIF
         ELSE !if other unknown string, go to error handler
            TPWrite str;
            RAISE unknown_str;
         ENDIF
         str := ReadStr(file\Delim:="]"); !reads the position string to "]"
         str := str + "]"; !add a "]" to string
         ok := StrToVal(str,position); !converts string to position
         IF ok=FALSE THEN ! if not a position string, go to error handler
            TPWrite str;
            RAISE unknown_str;
         ENDIF
         IF position.z < 0 THEN !to prevent moving the extruder tool into the table
            RAISE illegal_z;
         ENDIF
         str := ReadStr(file); !reads the rest of the line, should contain nothing
         temprobtarget.trans := position; !set new position
         MoveL temprobtarget,v100,fine,nozzle3dprinter\WObj:=table3dprinter;
                              !move to position
      ENDWHILE
      ERROR !error handler
         IF ERRNO = unknown_str THEN !if unknown string in coordinate file
            StopMove;
            SetDO DO11_4, 0;
```

```
            WaitTime 0.1;
            TPErase;
            TPWRite "Unknown String.";
            TPReadFK button_number, "What to do?",  stEmpty, stEmpty, stEmpty
                                    ,"TRYNEXT", "RETRY";
            IF button_number = 4 THEN
               TRYNEXT;
            ELSEIF button_number = 5 THEN
               RETRY;
            ENDIF
         ENDIF

         IF ERRNO = illegal_z THEN !if z < 0
            StopMove;
            SetDO DO11_4, 0;
             WaitTime 0.1;
            TPErase;
            TPWRite "Position z less than 0.";
            TPReadFK button_number, "What to do?",  stEmpty, stEmpty, stEmpty
                                          ,"TRYNEXT", "RETRY";
            IF button_number = 4 THEN
               TRYNEXT;
            ELSEIF button_number = 5 THEN
               RETRY;
            ENDIF
         ENDIF
   ENDPROC
ENDMODULE
```

E Arduino-kod

Arduino 1: Stepper Control

```
// Part of the controllsystem for a ABB-mounted 3D-printing head
// Made for the Bachelor's thesis on Chalmers Tekniska Högskola
//
// Made by: Tomas Nilson
//          Robin Lindholm
//          Markus Hägerstrand
//
// Date: 2013-05-22

// Constants declaring pin numbers
const int feedPin = 2; // Feeds the plastic
const int inputABBPin = 3; // Input from ABB
const int feedDirectionPin = 4; // Direction
const int fanCtrlPin = 5; // Fan on/off

const int LEDFan1Pin = 7; // LED for one of the fans
const int dirCtrlPin = 8; // Direction control
const int comInputPin = 9; // Connected to the second Arduino (7) // Not used
const int comOutputPin = 10; // Connected to the second Arduino (8) // Not used
const int clrToRunPin = 11; // Clear to run switch
const int stepperPin = 12; // Do one step.
const int LEDRunPin = 13; // LED to indicate if it's running  & onboard ledPin

// Constants
const int delayTimeStep = 9; // Standard delay for a step
const int steps = 200;        // Number of steps
          (1.8 deg per steps results in 360/1.8 = 200 steps per revolution) [#]
const int backFeedSteps = 5; // Nmber

// Variables
int feedButtonState = 0; // Variable to save the state of the feedbutton
int inputABB = 0; // Variable to save the input from ABB
int clrToRun = 0; // Save the switchstate of the Clear-to-Run switch
int comInput = 0; // Input from the other Arduino that is supposed to communicate
          when the temperature is in range
int doOnce = 0; // Do the backstep/forwardstep once at the switch of inputABB signal

// Initial setup
void setup()
{
  // Define pins as INPUTS
  pinMode(feedPin, INPUT);       // Pin2
  pinMode(inputABBPin, INPUT);       // Pin3
  pinMode(feedDirectionPin, INPUT);       // Pin4
```

```
  pinMode(comInputPin, INPUT);       // Pin9
  pinMode(clrToRunPin, INPUT);       // Pin11

  // Define pins as OUTPUTS
  pinMode(fanCtrlPin, OUTPUT);       // Pin5
  digitalWrite(fanCtrlPin, HIGH); // Turn the fan on
  pinMode(LEDFan1Pin, OUTPUT);       // Pin7
  pinMode(dirCtrlPin, OUTPUT);       // Pin8
  pinMode(comOutputPin, OUTPUT);       // Pin10
  pinMode(stepperPin, OUTPUT);       // Pin12
  pinMode(LEDRunPin, OUTPUT);       // Pin13
}

// Loop this, forever!
void loop()
{
  feedButtonState = digitalRead(feedPin); // Is the feed button pressed?
  inputABB = digitalRead(inputABBPin); // Input from ABBrobot
  comInput = digitalRead(comInputPin); // Reads the communicationport
          from the other Arduino
  comInput = HIGH;  // Ignore communication between Arduinos
  clrToRun = digitalRead(clrToRunPin); // Reads if the clear to run button is switched
  digitalWrite(LEDFan1Pin, digitalRead(fanCtrlPin)); // Light the LED to indicate
          whether the stepperfan is on or not
  digitalWrite(LEDRunPin, clrToRun); // Light the LED to indicate what state the
          clear-to-run switch is in

  // Just allow manual feed if NOT clear to run and the heater is warm
  if((feedButtonState == HIGH) && (clrToRun == LOW) && (comInput == HIGH))
  {
    digitalWrite(dirCtrlPin, digitalRead(feedDirectionPin)); // Write the direction
    doOneStep(); // Step once
  }

  // Just allow automatic feed if Clear-To-Run
  if((inputABB == HIGH) && (clrToRun == HIGH) && (comInput == HIGH))
  {
    digitalWrite(dirCtrlPin, HIGH); // Write the direction
    if(doOnce == 0) // Just do it once when signal is switched
    {
      doStepFast(backFeedSteps); // Step 'backFeedSteps' number of steps
      doOnce = 1; // Prepare for another signal switch
    }
    doOneStep();
  }

  // If input is LOW, back the feed up 'backFeedSteps' number of steps
  if(inputABB == LOW)
  {
```

```
    if(doOnce == 1)
    {
      digitalWrite(dirCtrlPin, LOW); // Write the direction
      doStepFast(backFeedSteps);
      doOnce = 0;
    }
  }
}

// Do one step
void doOneStep()
{
  digitalWrite(stepperPin, HIGH);
  delay(1);
  digitalWrite(stepperPin, LOW);
  delay(1);
  delay(delayTimeStep);
}

// Do one step, but with no delay
void doOneStepFast()
{
  digitalWrite(stepperPin, HIGH);
  delay(1);
  digitalWrite(stepperPin, LOW);
  delay(1);
}

// Do a number of steps
void doStep(int numOfSteps)
{
  int i;
  for(i=0; i< numOfSteps; i++)
  {
    doOneStep();
  }
}

// Do a number of steps, but fast
void doStepFast(int numOfSteps)
{
  int i;
  for(i=0; i< numOfSteps; i++)
  {
    doOneStepFast();
  }
}
```

Arduino 2: Heater Control

```
#include "Adafruit_MAX31855.h"

// Part of the controllsystem for a ABB-mounted 3D-printing head
// Made for the Bachelor's thesis on Chalmers Tekniska Högskola
//
// Made by: Tomas Nilsson
//          Robin Lindholm
//          Markus Hägerstrand
//
// Date: 2013-05-22

// Constants declaring pin numbers
const int fanCtrlPin = 2; // Fan on/off
const int thermoCLKPin = 3; // Adafruit Thermocouple Amplifier
const int thermoCSPin = 4; // Adafruit Thermocouple Amplifier
const int thermoDOPin = 5; // Adafruit Thermocouple Amplifier
const int heaterSwitchPin = 6; // Switch for the heater
const int comOutputPin = 7; // Connected to the second Arduino (9) // Not used
const int comInputPin = 8; // Connected to the second Arduino (10) // Not used

const int LEDHeaterPin = 10; // LED to indicate if the heater is on
const int LEDTempPin = 11; // LED to indicate if the correct temperature is reached
const int heaterCtrlPin = 12; // Turn the heater on/off
const int LEDFan2Pin = 13; // LED to indicate if the fan is on

// Variables
int targetTemp = 235; // Target temperature to reach before running
int targetTempVariance = 10; // Variance allowed
int heaterSwitchState = 0; // The state of the heater switch

Adafruit_MAX31855 thermocouple(thermoCLKPin, thermoCSPin, thermoDOPin);
          // Class for reading temperature via Adafruit Thermocouple Amplifier

// Loop this forever!
void setup()
{
  // Begin a serial communication via USB
  Serial.begin(9600);
  Serial.println("MAX31855 test");

  // Define pins as INPUTS
  pinMode(heaterSwitchPin, INPUT); // Pin6
  pinMode(comInputPin, INPUT); // Pin8

  // Define pins as OUTPUTS
  pinMode(fanCtrlPin, OUTPUT); // Pin2
  pinMode(comOutputPin, OUTPUT); // Pin7
```

```
  pinMode(LEDHeaterPin, OUTPUT); // Pin10
  pinMode(LEDTempPin, OUTPUT); // Pin11
  pinMode(heaterCtrlPin, OUTPUT); // Pin12
  pinMode(LEDFan2Pin, OUTPUT); // Pin13

  // wait for MAX Adafruit chip to stabilize
  delay(500);
}

void loop()
{
  double c = thermocouple.readCelsius(); // Read the thermocouple
  heaterSwitchState = digitalRead(heaterSwitchPin); // Read the heaterswitch state
  digitalWrite(LEDHeaterPin, heaterSwitchState); // Turn the LED on/off
          depending on the switch
  digitalWrite(LEDFan2Pin, digitalRead(fanCtrlPin)); // Turn the LED on/off
          depending on if the fan is on or not

  // If heaterswitch is on and the temperature too low, heat, otherwise do not
  if((c < targetTemp) && (heaterSwitchState == HIGH))
  {
    digitalWrite(heaterCtrlPin, HIGH);
  }
  else
  {
    digitalWrite(heaterCtrlPin, LOW);
  }

  // Turn the LED and the fan on if we are at the correct temperature,
          within a certain variance
  // Communicate to the other Arduino that the temperature is within limits.
  if((c < (targetTemp + targetTempVariance)) && (c > (targetTemp - (targetTempVariance/2))))
  {
    digitalWrite(LEDTempPin, HIGH);
    digitalWrite(fanCtrlPin, HIGH);
    digitalWrite(comOutputPin, HIGH);
  }
  else
  {
    digitalWrite(LEDTempPin, LOW);
    digitalWrite(fanCtrlPin, LOW);
    digitalWrite(comOutputPin, LOW);
  }

  // basic readout test, just print the internal temperature of the
          MAX adafruit chip and the current temp
   Serial.print("Internal Temp = ");
   Serial.println(thermocouple.readInternal());
   if (isnan(c)) {
```

```
      Serial.println("Something wrong with thermocouple!");
    } else {
      Serial.print("C = ");
      Serial.println(c);
    }
  delay(1000);
}
```

F C#-kod

Robotprototyping.cs

```
// Part of the controllsystem for a ABB-mounted 3D-printing head
// Made for the Bachelor's thesis on Chalmers Tekniska Högskola
//
// Made by: Tomas Nilsson
//          Robin Lindholm
//          Markus Hägerstrand
//
// Date: 2013-05-22

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace RobPro2
{
    public partial class RobotPrototyping : Form
    {
        public RobotPrototyping()
        {
            InitializeComponent();
        }

        // Filenames for input/output files
        protected String inputFileName, outputFileName;

        /// <summary>
        /// Button clicked
        /// Method to specify the file to read G-Code from.
        /// It has to be in the ".txt" format.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_LoadFile_Click(object sender, EventArgs e)
        {
            String inputGCodeFilePath = InputOutput.LoadFilePath();
            if (inputGCodeFilePath != null && Path.GetExtension(inputGCodeFilePath)
                    == ".txt")
            {
```

```
            lbl_InputFileLoaded.Text = inputGCodeFilePath;
            inputFileName = inputGCodeFilePath;
        }
        else
        {
            MessageBox.Show("Error loading the G-Code, have you
                made sure it is in .txt format?");
        }
    }

    /// <summary>
    /// What to do when the form closes.
    /// Check for a close confirmation
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void RobotPrototyping_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (MessageBox.Show("Are you sure you want to close?",
                "Confirm Close", MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            e.Cancel = false;
        }
        else
        {
            e.Cancel = true;
        }
    }

    /// <summary>
    /// Closes the application
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void RobotPrototyping_FormClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }

    /// <summary>
    /// Button clicked
    /// Method to specify what file to write to.
    /// It has to be in the .txt format.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void btn_OutputFile_Click(object sender, EventArgs e)
    {
        String outputGCodeFilePath = InputOutput.LoadFilePath();
```

```
    if (outputGCodeFilePath != null &&
             Path.GetExtension(outputGCodeFilePath) == ".txt")
    {
        lbl_OutputFileLoaded.Text = outputGCodeFilePath;
        outputFileName = outputGCodeFilePath;
    }
    else
    {
        MessageBox.Show("Error loading the outputfile, have you
             made sure it is in .txt format?");
    }
}

/// <summary>
/// Button clicked
/// Method that calls the convert function.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_ConvertCode_Click(object sender, EventArgs e)
{
    if (inputFileName != null && outputFileName != null)
    {
        double[] XYZ = { double.Parse(txtB_X.Text), double.Parse(txtB_Y.Text),
            double.Parse(txtB_Z.Text) };
        G_to_Rapid_Converter.GCodeToRapidConverter(inputFileName,
            outputFileName, XYZ);
        lbl_FileConverted.Text = "File Converted";
    }
    else
    {
        MessageBox.Show("Error: There is something wrong with
            the input/output files, have you specified them?");
    }
}

/// <summary>
/// Button clicked
/// Method that calls a class that generates a dummy code with coordinates
///
/// Not used in anything but testing
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_GenerateCode_Click(object sender, EventArgs e)
{
    if(outputFileName == null)
    {
        outputFileName = "C:\\Users\\Tomas\\Dropbox
```

```
                \\Kandidat\\Program\\RobPro2\\RobPro2\\Resources\\OutputFile";
            }
            G_to_Rapid_Converter.GenerateRapid(outputFileName,
                     int.Parse(txtB_NumOfLines.Text));
            lbl_NumOfRows.Text = InputOutput.NumberOfLines(outputFileName).ToString();
        }

    }
}
```

G_to_Rapid_Converter.cs

```
// Part of the controllsystem for a ABB-mounted 3D-printing head
// Made for the Bachelor's thesis on Chalmers Tekniska Högskola
//
// Made by: Tomas Nilsson
//          Robin Lindholm
//          Markus Hägerstrand
//
// Date: 2013-05-22
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Globalization;
namespace RobPro2
{
    class G_to_Rapid_Converter
    {
        /// <summary>
        /// Converts GCode (from a .txt file) to pure coordinates
           and high/low signal depending on the stepstruder should feed or not
        /// </summary>
        /// <param name="inputFileName">Input filename</param>
        /// <param name="outputFileName">Output filename</param>
        public static void GCodeToRapidConverter(string inputFileName,
          string outputFileName, double[] XYZ)
        {
            // Load output file
            StreamWriter sw = new FileInfo(outputFileName).AppendText();
            // Load input file
            StreamReader sr = new StreamReader(inputFileName);
            GetSetVariables variablesXYZEF = new GetSetVariables();
            Dictionary<string, double> result = new Dictionary<string, double>();
            int numLines = 0;
            // Read every line
            while (sr.Peek() >= 0)
            {
                numLines++;
                bool posChanged = false, feedForward = false;
                // double posOffsetX = 100, posOffsetY = 50, posOffsetZ = 0.9;
          // Change the position of the print
                double posOffsetX = XYZ[0], posOffsetY = XYZ[1], posOffsetZ = XYZ[2];
                double varEOld = 0;
                string input = sr.ReadLine();
                // Find and remove commented code (;)
                int index = input.IndexOf(";");
                if (index >= 0)
```

```
                input = input.Substring(0, index);
            // Find G1 (coordinate information) and extract information
                 and save it to X,Y,Z
            if (input != "" && input.Substring(0,2) == "G1")
            {
                input = input.Trim();
                string[] pairs = input.Split(' ');

                // Go through the string pairs and find key values, if a pos.
                           has changed, write a new one.
                foreach (string pair in pairs)
                {
                    result.Add(pair.Substring(0, 1), double.Parse(pair.Substring(1),
                           CultureInfo.InvariantCulture));
                }
                if (result.ContainsKey("X"))
                {
                    variablesXYZEF.varX = result["X"];
                    posChanged = true;
                }
                if (result.ContainsKey("Y"))
                {
                    variablesXYZEF.varY = result["Y"];
                    posChanged = true;
                }
                if (result.ContainsKey("Z"))
                {
                    variablesXYZEF.varZ = result["Z"];
                    posChanged = true;
                }
                if (result.ContainsKey("E"))
                {
                    variablesXYZEF.varE = result["E"];
                    if (variablesXYZEF.varE > varEOld)
                        feedForward = true;
                    else
                        feedForward = false;
                    varEOld = variablesXYZEF.varE;
                }
                if (result.ContainsKey("F"))
                {
                    variablesXYZEF.varF = result["F"];
                }
                result.Clear();
                CultureInfo customCulture = (System.Globalization.CultureInfo)
                          System.Threading.Thread.CurrentThread.
CurrentCulture.Clone();
                customCulture.NumberFormat.NumberDecimalSeparator = ".";
                System.Threading.Thread.CurrentThread.CurrentCulture =
```

```
  customCulture;

            // Just update the lines if a position has changed, and depending
  if the feedvalue is high or low
            if(posChanged && feedForward)
                sw.WriteLine("1;[" + (variablesXYZEF.varX + posOffsetX) +
                       "," + (variablesXYZEF.varY + posOffsetY) + "," +
                                (variablesXYZEF.varZ + posOffsetZ) + "]");
            if (posChanged && !feedForward)
                sw.WriteLine("0;[" + (variablesXYZEF.varX + posOffsetX) +
                      "," + (variablesXYZEF.varY + posOffsetY) + "," +
                                (variablesXYZEF.varZ + posOffsetZ) + "]");
            posChanged = false;
        }
    }

    sw.Flush();
    sw.Close();
    sr.Close();
}

/// <summary>
/// Generates Rapid Code targets and movecommands in the
            shape of a rectangle in a textfile.
/// Starting coordinate is 0,0,0 (X, Y, Z)
///
/// Only used for testing, when for example you don't have a
             pregenerated G-code
/// </summary>
/// <param name="outputFileName">The file to write too</param>
/// <param name="numOfLines">Number of lines to generate</param>
public static void GenerateRapid(string outputFileName, int numOfLines)
{
    StreamWriter sw = new FileInfo(outputFileName).AppendText();
    const int length = 100, width = 100, speed = 100;
    double X = 0, Y = 0, Z = 0;
    for (int i = 0; i < numOfLines; i++)
    {
        sw.WriteLine("[" + X + "," + Y + "," + Z + "]");
        if (Y == length)
            Y = 0;
        else
            Y = length;
        if (i % 2 != 0 && i != 0)
            X = X + 1;
        if (X == width)
        {
            Z = Z + 1;
            X = 0;
```

```
                }
            }

            //for (int i = 0; i < numOfLines; i++)
            //{
            //    sw.WriteLine("\tMoveL target_" + i + ",v"
                    + speed +",fine,Pen_TCP\\WObj:=Bord;");

            //}

            sw.Flush();
            sw.Close();
        }
    }
}
```

GeSetVariables.cs

```
// Part of the controllsystem for a ABB-mounted 3D-printing head
// Made for the Bachelor's thesis on Chalmers Tekniska Högskola
//
// Made by: Tomas Nilsson
//          Robin Lindholm
//          Markus Hägerstrand
//
// Date: 2013-05-22

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RobPro2
{
    /// <summary>
    /// Gets and sets values needed for a print.
    /// </summary>
    class GetSetVariables
    {
        double _x, _y, _z, _e, _f;
        /// <summary>
        /// Get/Set value
        /// X coordinate
        /// </summary>
        public double varX
        {
            get
            { return this._x; }
            set
            { this._x = value; }
        }

        /// <summary>
        /// Get/Set value
        /// Y coordinate
        /// </summary>
        public double varY
        {
            get
            { return this._y; }
            set
            { this._y = value; }
        }

        /// <summary>
```

```
        /// Get/Set value
        /// Z coordinate
        /// </summary>
        public double varZ
        {
            get
            { return this._z; }
            set
            { this._z = value; }
        }

        /// <summary>
        /// Get/Set value
        /// E variable coordinate
        /// </summary>
        public double varE
        {
            get
            { return this._e; }
            set
            { this._e = value; }
        }

        /// <summary>
        /// Get/Set value
        /// F variable
        /// </summary>
        public double varF
        {
            get
            { return this._f; }
            set
            { this._f = value; }
        }
    }
}
```

InputOutput.cs

```
// Part of the controllsystem for a ABB-mounted 3D-printing head
// Made for the Bachelor's thesis on Chalmers Tekniska Högskola
//
// Made by: Tomas Nilsson
//          Robin Lindholm
//          Markus Hägerstrand
//
// Date: 2013-05-22

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace RobPro2
{
    /// <summary>
    /// Handles input/output information
    /// </summary>
    class InputOutput
    {
        /// <summary>
        /// Starts a FileDialog and gets the filepath to the selected file.
        /// </summary>
        /// <returns>String containing the filepath.</returns>
        public static String LoadFilePath()
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Multiselect = false;

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                return openFileDialog1.FileName;
            }
            else
            {
                return null;
            }
        }
        public static int NumberOfLines(string filePath)
        {
            return File.ReadLines(filePath).Count();
        }
    }
}
```