CODEN:LUTEDX/(TEIE-3125)/1-66(2023)

Industrial Electrical Engineering and Automation

Automatisk precisionstestning av skärmbaserade produkter

Axel Johnson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Automatisk Precisionstestning av Skärmbaserade Produkter

Industriell Elektroteknik och Automation

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Axel Johnsson

Handledare: Christin Lindholm
Examinator: Mats Lilja

13 juni 2023

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2023

Sammanfattning

Pekskärmar är en typ av inmatningsenhet som möjliggör en interaktion mellan en användare och en elektronisk enhet. Interaktionen sker genom att låta användaren röra med en eller flera fingrar eller en stylus-penna på skärmen. Pekskärmar används idag till en rad olika elektroniska produkter som till exempel mobiltelefoner, surfplattor, datorer och även bankomater. På grund av sitt breda användningsområde behöver pekskärmar ibland dimensioneras och byggas om till en specifik produkt. Detta gör att pekskärmen måste fungera tillsammans med en mjukvara av till exempel ett operativsystem utan problem. För att att säkerställa att processen mellan en pekskärm och en mjukvara fungerar, behövs det testas.

Det här examensarbetet handlar om att utveckla en automatiserad testmetod för att testa precisionen hos en pekskärm. Examensarbetet har utförts i samarbete med Axis Communications AB. Tester kan appliceras på många olika områden när det gäller pekskärmar. I detta examensarbetet kommer fokusen ligga på att testa precision genom att skapa ett testverktyg som kombinerar hårdvara med mjukvara. Examensarbetet som utförs på Axis Communications AB, där tidigare tester har skett förhand och manuellt. På grund av detta ligger det i företagets intresse att skapa ett automatiserat testverktyg för skärmbaserade produkter.

Testmetoden som byggts för att testa precision på en pekskärm är uppdelad i hårdvara och mjukvara. Hårdvaran består av en modifierad 3D-skrivare med en stylus-penna monterad på sig. Mjukvaran står för att styra 3D-skrivaren och automatisera rörelser som utför tester på en pekskärm. Då denna metod inte har byggts innan blev det ett nytt område att utforska. Mjukvaran är programmerad i Java och en 3D-skrivare av typen velleman vertex delta K8800 används.

Resultatet av examensarbetet visar att det är möjligt att skapa en automatiserat testmetod för pekskärmens precision med hjälp av Java. Testverktyget kan initieras med hjälp av en INI-fil vilket möjliggör testning av skärmar med olika storlekar. G-code, som genererades med hjälp av ett grafiskt användargränssnitt (GUI) eller en terminal, styr 3D-skrivaren. Både GUI och terminalen, som har samma syfte att skapa och läsa in testfiler infördes i examensarbetet och skickar sedan G-code kommandon till 3D-skrivaren. Även datainhämtning från en pekskärm möjliggjordes då en Raspberry pi med hjälp av SSH-anslutning till mjukvaran kan överföra koordinatdata från pekskärmen.

Nyckelord

Testautomation

3D-skrivare

precisionstestning

skärmtestning

Java

Prototyp

Axis Communications AB

Abstract

Touchscreens are a type of input device that facilitate interaction between a user and an electronic device. This interaction is enabled by the user touching the screen with one or more fingers or a stylus pen. Today, touchscreens are implemented in a wide range of electronic products for example mobile phones, tablets, computers, and even ATMs. Due to their wide range of applications, touchscreens sometimes need to be dimensioned and redesigned for a specific product. Consequently, the touchscreen must operate problemless with software, for example an operating system. To ensure the process between a touchscreen and software functions effectively, it needs to be tested.

This thesis focuses on developing an automated testing method to test the precision of a touchscreen. It has been conducted in collaboration with Axis Communications AB. Tests can be applied in various areas concerning touchscreens. In this thesis, the focus lies on testing precision by creating a testing tool that combines hardware and software. The thesis is carried out at Axis Communications AB, where previous tests have been performed manually. Therefore, the company has an interest in developing an automated testing tool for screen-based products.

The testing method built to measure the precision of a touchscreen is divided into hardware and software. The hardware consists of a modified 3D printer with a stylus pen mounted onto it. The software is responsible for controlling the 3D printer and automating movements that conduct tests on a touchscreen. Since this method has not been implemented before, it was a new area to explore. The software is programmed in Java and a Velleman Vertex Delta K8800 type 3D printer is used.

The results of the thesis indicate that it is feasible to create an automated testing method for touchscreen precision using Java. The testing tool can be initiated with an INI file, making it possible to test screens of varying sizes. G-code, generated through a Graphical User Interface (GUI) or a terminal, controls the 3D printer. Both the GUI and terminal, which share the same purpose of creating and reading test files, were implemented in the thesis and subsequently send G-code commands to the 3D printer. Additionally, data acquisition from a touchscreen was facilitated as a Raspberry Pi can transfer coordinate data from the touchscreen to the software via an SSH connection.

Keywords

Testautomation

3D-printer

precision test

Screen test

Java

Prototype

Axis Communications AB

Förord

Detta examensarbete gjordes av samarbete med teamet på New business intercom QA hos Axis Communications AB. Ett stort tack går till Bachar Khamis och Lars Gustafsson från Axis som handledde det tekniska arbetet med projektet och lärde mig otroligt mycket om kunskaper som jag inte alls förväntade mig lära mig av detta examensarbete.

Utan handledaren Christin Lindholm hade examensarbetet inte kunnat framföras. Därför går ett stort tack till Christin för en riktigt bra handledning och struktur på hela arbetet. Mycket bra tips och kunskap tilldelades under projektets gång som tillförde en viktig del av examensarbetet. Frågor kunde svaras på och svaren var alltid utförliga vilket hjälpte mig att förstå vad som skulle göras.

Till sist vill jag tacka Mats Lilja som examinator på detta examensarbetet. Utanför rollen som examinator stod han till hjälp med strukturen för arbetet och utan den extra handledningen från Mats hade examensarbetet inte kunnat genomföras.

Innehåll

1 Introduktion

I det här kaptiltet introduceras examensarbetet som görs i samarbete med Axis Communications AB. Syftet med examensarbetet kommer att beskrivas, målen redovisas och problemformuleringarna ställs som sedan under projektets gång ska besvaras.

1.1 Bakgrund

Under denna rubrik nämns det information om bakgrunden till detta examensarbete. Bakgrunden ska ge en initial bild på examensarbetet där syftet beskrivs och relaterade studier av projektet tas upp.

1.1.1 Axis Communications AB

Axis Communications AB är ett företag som tillhandahåller nätverkslösningar inom videoövervakning, accesskontroll, intercom och audiosystem. Företaget har sitt huvudkontor i Lund, Sverige.[1] Företaget grundades 1984 av Mikael Karlsson, Martin Gren och Keith Bloodworth och började med att skapa något de kallade för ThinServer Technology som kan liknas med det som idag kallas för Internet of Things (IoT). År 1996 lanserade dem därefter världens första nätverkskamera, och sedan dess har företaget lanserat en mängd olika säkerhetslösningar. Idag har Axis Communications AB ett brett utbud av olika IP-baserade produkter och ägs av det japanska företaget Canon Inc.[2]

1.1.2 Relaterade studier

Det har tidigare skrivits en del examensarbeten som behandlar just området testautomation. Det kan vara allt från automatiska funktionalitetstester för webapplikationer[3] till automatiska tester för fysisk access till kontrollsystem.[4] Det här examensarbetet behandlar skärmbaserade testautomationsmetoder. I det här fältet har det gjorts en liknande studie av Ragnar Wernersson som på uppdrag av Sony Mobile i Lund, Sverige undersökte huruvida det går att utveckla ett billigt automatiserat verktyg för att testa tidsfördröjning av sveprörelser på mobiltelefoner med pekskärm.[5] Det som är nytt med just det här examensarbetet är att det som undersöks är om det är möjligt att ta fram ett automatiserat verktyg som testar precision på en pekskärm under en längre tids användning.

1.2 Syfte

Syftet med examensarbetet är att ta fram en testmetod för automatiserad testning som simulerar användning av en pekskärm på skärmbaserade produkter. Den här testningen har som avsikt att pågå i en veckas tid med resultat som speglar ett års användning av produkten. Det som står i fokus för verktyget som tas fram är att testa precision för skärmen, där ett mål är att använda en modifierad 3D-skrivare av modell Velleman vertex delta K8800. 3D-skrivaren ska kombineras med en mjukvara som står för styrningen och datahantering. 3D-skrivaren med mjukvaran kan simulera en användare med hjälp av en ombyggnad på skrivarhuvudet där en stylus-penna monterats. Denna har som avsikt att agera som ett finger på skärmen. Vid sidan om hårdvaran ska också mjukvaran programmeras i Java som styr maskinen genom att via seriell uppkoppling skicka G-Code kommandon till 3D-skrivaren. Därefter ska ett program skapas som samlar in testdata från den testade skärmen för att öppna möjligheten att verifiera resultat av testningen. Syftet med att ta fram ett automatiskt verktyg för skärmbaserad testning grundar sig i att Axis Communications AB i dagsläget inte har något verktyg för att automatiskt kunna testa sina skärmbaserade produkter. Att ha ett sådant verktyg skulle underlätta för utveckling av nya produkter då det bidrar till att hitta eventuella fel tidigare i utvecklingsprocessen.

1.3 Mål

Under denna rubrik kommer nödvändiga mål och uppgifter beskrivas för att genomföra examensarbetet. Målen kan uppnås oberoende av varandra men skapar tillsammans en helhetsbild för hela examensarbetet.

1.3.1 Utveckla ett verktyg för automatisk testning av skärmar

Det som är det huvudsakliga målet med det här examensarbetet är att utveckla ett verktyg för att möjliggöra automatisk testning av hårdvara i form av en pekskärm. Testningen ska bestå av att rörelser kan utföras på en pekskärm automatiskt.

1.3.2 Simulera ett års användning av pekskärmen

Testmetoden ska under en veckas tid simulera ett års användning av skärmen. Ett års användning bestäms i det här fallet med en uppskattning Axis har gjort, att pekskärmen kommer användas 50000 gånger under ett års tid för det bestämda testfallet. Tidsintervallet har bestämts för att kunna testa en längre tids användning under en kortare period. Att simulera en långstidsanvändning under en kortare period möjliggör testning av produkter under utveckling och problem kan hittas i ett tidigt skede.

1.3.3 Tillämpa flexibilitet av testmetoden

Mjukvaran som utvecklas inom ramen för det här examensarbetet ska vara möjlig att applicera på flertal olika skärmbaserade produkter. Därför ska programmet som skapas ha som mål att tillåta ändringar av initiala värden. Samt ska det finnas möjlighet för hårdvaran att modifieras för att passa andra skärmbaserade produkter.

1.4 Problemformulering

Under examensarbetets gång önskas följande frågor besvaras:

1. Hur går det att automatisera en testmetod för att testa långtidsanvändning av en skärm?
2. Vilka parametrar bör tas i hänsyn vid testning av en pekskärm?
3. Hur kan data inhämtas i automatiserad testning?
4. Vilka faktorer kan påverka testresultatet?

1.5 Motivering av examensarbetet

Examensarbetet valdes med motivering att spegla utbildningen, högskoleingenjör inom elektroteknik med automation, där huvudämnet i detta examensarbete breddar kunskaper inom automation. Examensarbetet kommer att använda sig av någon form av hårdvara som ska simulera ett finger som interagerar med skärmen, mjukvara som styr hårdvaran och sedan även som ska behandla den data som samlas in i testningen. Det här examensarbetet kan gynna det omgivande samhället genom att de skärmbaserade produkter som produceras och kommer ut på marknaden är testade för att vara hållbara under en lång period, vilket bidrar till bland annat bättre hållbarhet för att dessa produkter kan komma att användas under en längre tid. Dessutom så gynnar det Axis Communications AB för att de kan få en automatiserad metod för att testa deras skärmbaserade produkter.

1.6 Avgränsningar

Att automatisera testning är ingen lätt uppgift i sig och det är ett väldigt brett område där det finns val att automatisera en hel process, vilket för detta arbete kan göra det alltför tidskrävande, eller bara vissa delar av en process och därför sätts följande avgränsningar på arbetet.

- Utveckla ett verktyg för endast den produkt som tillhandahålls för detta examensarbete.
- Endast ta fram en metod för att testa precision på pekskärmen.
- I mån av tid kommer metoder att tas fram för att kunna behandla fler parametrar i form av väderförhållanden som fukt eller temperatur.

2 Teknisk bakgrund

Kapitlet innehåller en beskrivning på vilken mjukvara, hårdvara och tekniska strategier som används i detta examensarbete. Kapitlet kommer ge en bättre teknisk förståelse för examensarbetet i sin helhet.

2.1 Mjukvara

Mjukvarorna kommer användas för att styra en 3D-skrivare, samla in data från den testade pekskärmen och behandla datan. Detta lägger en grund till projektet och möjliggör att hårdvara kan automatiseras.

2.1.1 Java

Java är ett programmeringspråk som är klassbaserat, objektorienterat och plattformsoberoende. Strukturen i Java grundar sig i att klasser skapar objekt som innehåller attribut och metoder. Metoder kan innehålla algoritmer som utför till exempel en matematisk beräkning men också returnering av attribut. Attribut är en variabel som är deklarerad inom en klass. Denna variabel lagrar information om objekt i klassen.

Programmeringsspråket är relaterat till C och C++. En skillnad mellan C++ och Java är att java har inyggd hjälp med att lokalisera syntaxfel i koden innan den har kompilerats. Syntax kan ses som en instruktion på hur språket ska skrivas det vill säga struktur och format. Kompilering är ett sätt att köra koden och driftsätta den. Vid kompilering översätts koden till bytekod. Java är en av världens största programmeringsspråk och används idag för att utveckla programvara för ett brett spektrum av applikationer. [6]

2.1.2 Java Swing

Java Swing är ett bibliotek som ingår i Java Fondation Classes (JFC). Det betyder att den ingår i Javas standardbibliotek och är en inbyggd funktion att skapa Användargränssnitt (GUI) med Java. Den ersatte den äldre funktionen för ett användargränssnitt Abstract Window Toolkit (AWT). Swing introducerades i Java 1.2 och är byggd ovanpå AWT helt i Java, det betyder att den är mer plattformsoberoende och anpassningbar än AWT.

I Java swing finns det JButton, JScrollPane, JTextfield, JLabel och många fler funktioner. JButton är en klass som skapar en knapp och med hjälp av en ActionListener kan en händelse vid knapptryck konfigureras. JScrollPane är en skrollbar panel, det vill säga att information kan presenteras och informationen är skrollbar likt en tidningsartikel. JTextfield skapar en textruta som en användare kan skriva i, med denna kan text hämtas i form av en String till Java programmet. JLabel bygger en textruta som visar text som en användare ej kan ändra utifrån användargränssnittet. [7]

2.1.3 G-Code

G-Code är ett standardiserat programmeringsspråk för CNC-maskiner och 3D-skrivare. Varje rad i koden representerar ett kommando som maskinen bearbetar. Koden består av alfanumeriska koder och symboler som är förståeliga för maskinerna. Dessa instruerar maskinen om vilken av de tre axlarna som ska röra sig och med vilken hastighet den ska arbeta med. [8]

2.1.4 Seriell kommunikation

Seriell kommunikation är ett kommunikationssätt för datorer och enheter. Det vanligaste sättet att kommunicera data med seriell kommunikation är via en USB-port. Vid seriell kommunikation överförs bit för bit med en hastighet på en bestämd baudrate. Seriell kommunikation möjliggör användning av till exempel mus och tangentbord men också externa enheter som kräver kommunikation med en dator. [9]

2.1.5 JSerialComm

JserialComm är ett Java-bibliotek som kan importeras i olika projekt. Biblioteket tillhandahåller metoder och klasser för att möjliggöra informationsöverföring mellan Javaprogram och hårdvara via seriell kommunikation.[10]

2.1.6 INI-fil

En INI-fil kan beskrivas som en konfigurationsfil för att lagra programinställningar och parametrar enligt ett textformat. Det möjliggör att programinställningar kan sättas utan att programmet körs och breddar programmets användningsområden. [11]

2.1.7 SSH

SSH står för Secure Shell och är ett nätverksprotocol som möjliggör en fjärranslutning datorer emellan via nätverk Fjärrstyrningen kan användas via ett grafiskt användargränssnitt men också genom kommandobaserat användargränssnitt. [12]

2.1.8 JSch

JSch är ett open-source bibliotek för Java. Detta bibliotek öppnar möjligheten att skapa och upprätthålla en SSH-anslutning till en fjärrdator. I biblioteket finns det flertal metoder för att ta ut data från en fjärrstyrd dator, men också för att skicka data. [13]

2.1.9 Python

Python är ett högnivåspråk likt Java, Det är objektorienterat men kan också skrivas som ett script. Ett python-script är en kod som enkelt kan kompileras utan att skapa en klass av koden. Det används mycket inom enkla matematiska beräkningar och enkla programvaror som har som mål att vara snabba och flexibla. [14]

2.1.10 Evdev

Evdev(Event Device) är en Linux input-drivrutin som ger ett standardiserat gränssnitt för att hantera enheter som till exempel mus och tagentbord. Evdev hanterar händelser från enheter som sedan skickas vidare till ett program som användaren använder via en application programming interface (API). Syftet med Evdev är att det underlättar operationer mellan olika typer av enheter och mjukvaror. Evdev kan enkelt användas med hjälp av Python [15]

2.2 Hårdvara

Hårdvaran kommer att användas för att testa en pekskräm. Detta görs med hjälp av en mjukvara som styr hårdvaran och hämtar data från testerna som hårdvaran utför.

2.2.1 3D-printer

3D skrivare har som huvudsyfte att skriva ut objekt i olika former av olika material men vanligast plast. Med ett munstycke som smälter en plasttråd kan materialet placeras så att den bygger ett objekt. För att munstycket ska förflytta sig används axlar med elmotorer som möjliggör att det kan förflyttas till vald koordinat.

För att kontrollera en 3D-skrivare används CAD (Computer-aided design), vilket är en programvara som beskriver en ritning på ett objekt som sedan används för att programmera 3D-skrivaren för att bygga upp objektet. I mer modern teknik har 3D-skrivare utvecklats för att kunna användas för fler syften än att bara bygga objekt av plast. Några av dessa användningsområden är hus, konstgjord mat, industriella delar av metall och bilar. Den egenskap som en 3D-skrivare har som kommer vara intressant för detta examensarbete är precision och möjlighet att styra skrivarhuvudet till önskad position. [16]

2.2.2 Raspberry Pi

Raspberry pi är en minidator. Operativsystemet är Rasperry pi OS, det är ett linux-baserat operativsystem. På en Raspberry pi är det möjligt att köra Python-program. Möjligthet finns att koppla in en extern skärm, ett tagentbord och en mus. Men det går även att låta minidatorn skapa en SSH-server som andra datorenheter kan ansluta till. [17]

2.2.3 Kapacitiv pekskärm

En kapacitiv pekskärm är en teknologi som kan känna av om en person trycker med sitt finger eller ett elektriskt ledande objekt på en skärm. Tekniken bygger på att skärmen känner av förändringar i elektriska fält. Pekskärmen är uppdelade i små celler i ett matrisnät. Varje cell har ett ursprungligt elektriskt fält, när ett elektriskt ledande objekt vidrör skärmen ger cellerna utslag då elektriska fältet kring cellerna har förändrats. På detta sätt kan position anges på pekskärmen vid ett tryck med ett finger eller ett elektriskt ledande objekt. [18]

2.2.4 Stylus penna

Stylus penna eller även benämnt en pekpenna finns i olika former. Det finns aktiva och passiva pekpennor. Aktiva pekpennor innehåller elektronik och skapar en elektronisk laddning som skärmen kan känna av. En passiv pekpenna har ett elektriskt ledande material i spetsen på pennan, detta gör att den kan ändra det magnetiska fält som finns på en kapacitiv pekskärm. En passiv pekpenna har som mål att simulera ett mänskligt finger pågrund av dess ledande egenskaper. [19]

2.3 Testautomation

Automatisk testning sker med hjälp av automation, vilket kan tillämpas på processer där en maskin eller en mjukvara utför en uppgift på exakt samma sätt varje gång. Vid testning av ett objekt utförs en rad olika tester för att pröva funktionalitet, hållbarhet, belastningskapacitet eller liknande. Det finns olika typer av testautomation och två exempel på områden där ofta testautomation tillämpas är tester på mjukvara och hårdvara.

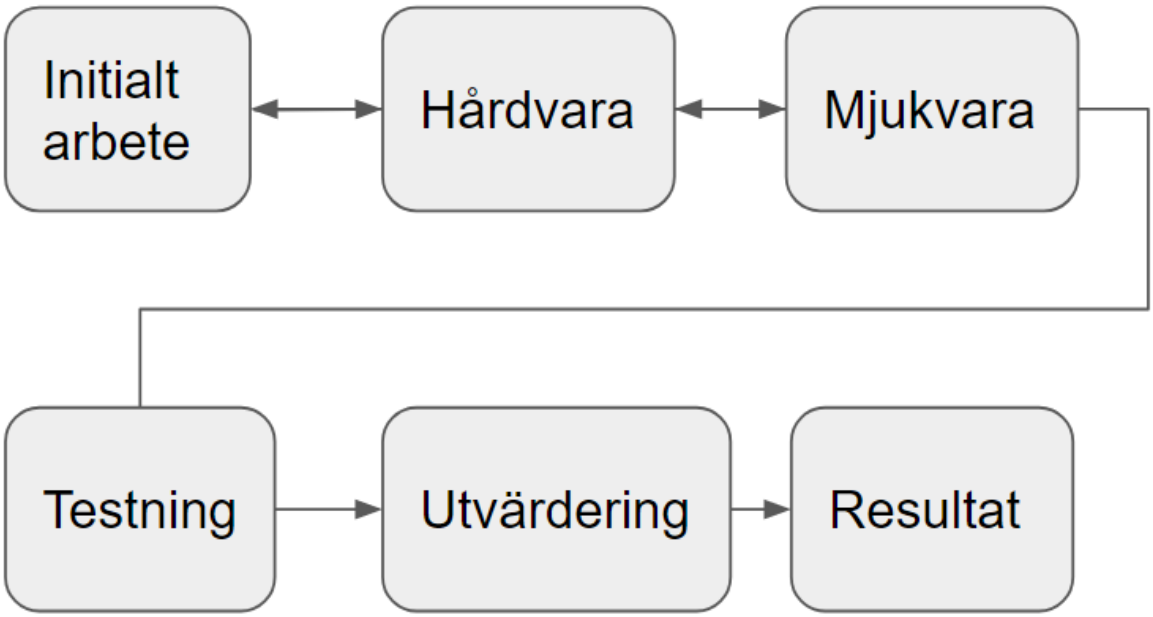
Beroende på vad som ska testas kan olika testtyper appliceras för deras ändamål men generellt sett så används automation för miljöer där precision eftertraktas och i många industrier kan automatiska processer ersätta människan och utföra arbetet exakt för varje objekt. En stor fördel när tester utförs med automation är att testerna blir mer exakta och att ett flertal tester kan utföras under en längre tid.[20]

3 Metod och Analys

Detta kapitel innehåller olika metoder som har använts för att utföra detta examensarbete. Det kommer motiveras varför olika metoder har valts och en beskrivning på hur dessa metoder har implementerats i arbetet. Under examensarbetet delades utförandet upp i 6 olika faser för att strukturera upp arbetet.

3.1 Faser

Examensarbetet blev uppdelat i 6 olika faser som visas i figur 3.1 där de tre första faserna är iterativa[21] och kan relatera tillbaka. Detta för att vid dessa faser har jag gått tillbaka och sökt efter ny information och data. Vid fasen av testning menas testning av skärmen, då valdes det att fortsätta processen sekvensiellt då behovet att gå tillbaka i processen inte ansågs vara nödvändig.



Figur 3.1: 6 olika faser där de tre första är iterativa och de tre sista är sekvensiella

3.2 Informationshämtning

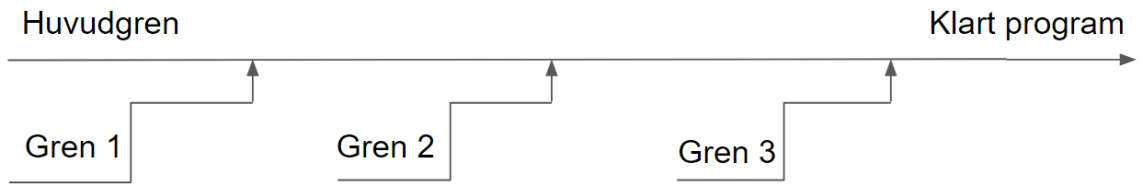
Inhämtning av information och kunskap låg till grund för att genomföra examensarbetet. Informationen dokumenterades i en anteckningsbok. Inhämtningen gjordes med hjälp av webbsidor, litteratur i form av artiklar och böcker, dokumentationer på datastrukturer och kontinuerliga möten med Axis. Mötena på Axis hade som syfte att skapa möjlighet för handledning av det praktiska arbetet men också för att tilldela information och kunskap som stod för en viktig del av examensarbetet. Alla möten skedde på Axis och informationen antecknades för att sedan användas som input i arbetet till rapporten av examensarbetet. Med informationen skapades även en bild på vad som behövde göras för att komma vidare i processen.

3.3 Kontinuerlig dokumentation

Vid utförande av praktiska moment så antecknades det data, nytillkomna buggar och vad som hade gjorts vid varje arbetsdag. Detta för att öppna möjligheten att gå tillbaka i processen och få en översikt över arbetet. Eftersom det blev svårt att angripa examensarbetet i sin helhet skapades delmål vid varje tillfälle. Dessa delmål strukturerade upp arbetet och förenklade processen att följa en röd tråd genom examensarbetet.

3.4 Funktionshantering

För att de tekniska handlederna enkelt skulle ha översikt över examensarbetet användes GIT[22], ett program för versionshantering där kod kan sparas emellanåt och laddas upp. Metoden innefattar att man delar upp varje arbetsprocess i olika grenar. Varje gren kan till exempel vara en ny utvecklingsfas av programmet. Sedan när en utvecklingsprocess är klar ska denna bli infogad(Eng: Merge) i master-grenen. Denna process beskrivs i figur 3.2. Före detta steg måste en infogningsförfrågan skickas till de tekniska handledarna. Denna förfrågan kommer sedan bli granskad och koden som den innehåller blir kommenterad. På grund av detta ges det feedback i varje steg av det praktiska momentet. Feedback blev en viktig del i arbetesproceduren då koden kunde följa en hög kvalité och antalet buggar minskades.



Figur 3.2: Förklarar processen vid mjukvaruutvecking av delade grenar.

3.5 Fas 1 - Initialt arbete

Vid första fasen strukturerades examensarbetet upp och ett initialt arbete utfördes. Här inhämtades information och kunskaper för att utföra arbetet hos Axis.

3.5.1 Kommunikation Axis Communications AB

Arbetet inleddes med en dialog tillsammans med Axis Communications AB. Efter ett initialt möte med Axis började examensarbetet planeras och blev indelat i mindre delar som tillsammans byggs ihop till ett fullständigt examensarbete. Frågeställningar utformades och avgränsningar sattes för att arbetet skulle passa in som ett examensarbete. Examensarbetet utfördes med hjälp av två tekniska handledare anställda på Axis, Bachar Khamis och Lars Gustafsson. Handledarna med sin expertis vägledde det tekniska arbetet genom att arrangera kontinuerliga möten för att diskutera framsteg, tilldela feedback och utvecklingsmöjligheter.

3.5.2 Fältstudie på relaterade testmetoder

I början av examensarbetet gjordes en mindre fältstudie av nuvarande testmetoder för pekskärmar. Fältstudien bestod till en början att söka upp information kring testmetoder i tidigare examensarbeten och hur andra företag testar pekskärmar. Denna information skapade sedan en bild av hur testmetoden kan se ut och bidrog med inspiration till det här examensarbetet.

3.6 Fas 2 - Hårdvara

Under denna fas beskrivs den hårdvara som valts ut och metoden för att tillämpa den till examensarbetet.

3.6.1 3D-skrivare

För att utföra rörelser på en skärm med precision fanns en idé med att använda en 3D-skrivare. Axis valde en 3D-skrivare av modell Velleman Vertex Delta K8800.[23] Denna skrivare är en delta-skrivare som med hjälp av 3 axlar kan positionera sig själv till en punkt med milimeterprecision. Första steget i processen var att modifiera 3D-skrivaren så att denna kan påverka en pekskärm. För att göra detta skruvades det ursprungliga skrivarhuvudet av och en insats som tillhandahållits av Axis monterades. På denna insats kan en stylus-penna skruvas i. Med den modifierade maskinen kan den nu användas för att simulera en användare som utför rörelse och gester på en pekskärm.

3.7 Fas 3 - Mjukvara

Här beskrivs hur utvecklingen av mjukvaran genomförts. Lösningar har även analyserats och det motiveras varför vissa lösningar blev bättre än andra.

3.7.1 Styrning av 3D-skrivare

3D-skrivaren av modell Velleman Vertex Delta K8800 styrs vanligtvis via ett CNC-gränssnitt.[8] I detta examensarbete ligger fokusen på att direktstyra maskinen genom G-Code kommandon. Detta möjliggör en noggran precision på vart pekpennan ska landa på skärmen. Att skriva G-Code på förhand för stora tester är inte optimalt då antalet rader av kommandon kan bli många. Därför beslutades det att använda sig av ett högnivåspråk för att enkelt skriva G-code kommandon. Högnivåspråket som i detta examensarbete har valts av exjobbaren till Java, implementerar G-code kommandon via ett flertal metoder. Av alla Javametoder skapar huvudmetoderna kommandon som speglar en sveprörelse och en tryckrörelse. Att kombinera dessa rörelser möjliggör en efterliknelse av en användare.

3.7.2 Grafiskt användargränssnitt (GUI)

Det skapades ett grafiskt användargränssnitt (GUI) för att enkelt navigera mellan olika Javametoder. Användargränssnittet designades och utformades med hjälp av det färdiga biblioteket Java Swing på grund av sin enkelhet. Gränssnittsfönstret består av två JPanels där första panelen innehåller olika alternativ för rörelser som 3D-skrivaren ska utföra. Alternativen är enkelt tryck till en koordinat eller en svep-rörelse från en koordinat till en annan. Den andra panelen har som uppgift att fungera som ett kontrollcenter där filer av G-code kan läsas in och sedan skickas till 3D-skrivaren. Det skapas även möjlighet att få en förhandsvisning av kommandona via en JScrollpane samt även kalibrering av 3D-skrivaren via en knapp.

Efter ett möte med de tekniska handledarna Lars och Bachar, bestämdes det tillsammans med exjobbaren att användargränssnittet skulle skapas för att uppfylla följande specifikation:

- Kunna ändra och verkställa arbetshastigheten för 3D-skrivare
- Sätta en arbetshöjd (Z), en min Z och max Z
- Tryckrörelse för en X och Y koordinat
- Sveprörelse för en X och Y till en annan X och Y
- Skriva G-code som kan sparas i en fil
- Ansluta och kalibrera 3D-skrivaren
- Kunna läsa in filer med befintliga tester som innehåller G-code kommandon
- Presentera G-code via en skrollbar panel
- Skicka G-code kommandon till en 3D-skrivare

Utöver tryckrörelse, svep-rörelse, filhantering, presentation av G-code och kalibrering behövdes resterande funktioner implementeras. För att implementera en arbetshöjd skrevs kod för att inhämta koordinater. Dessa koordinater sparades som ett attribut och bestämde arbetshöjden för 3D-skrivaren. Det skapades även möjlighet att låta programmet bestämma en arbetshastighet för 3D-skrivaren. Detta gjordes med att ta in en hastighet genom en JTextField i skapade GUI. Denna hastighet i form av en int omvandlades till G-code och blev skickad direkt till 3D-skrivaren.

Kod som visar exempel på algoritmen hur hastighet blev bestämd och omvandlad till G-code:

```
int speedInputFromUser = 150;
return "M220 S" + String.valueOf(speedInputFromUser) + "\n";
------------------------------------------------------------
Output: M220 S150
```

Det skapades även möjlighet för att öppna ett separat fönster för manuell styrning och byta till en Terminal-version av programmet.

3.7.3 Terminalstyrning

För att kunna styra 3D-skrivaren med textkommandon utan att använda ren G-code skapades en ny klass vid namn Terminal.java. Först byggdes det en JScrollPane och en svart bakgrund med vit text sattes. Denna ska simulera ett terminalfönster där loggar och information kommer att skrivas ut. Under denna sattes en JTextfield som låter användaren skriva in kommandon och text. Vid behandling av kommandon skapades en egen syntax.

Exempel på den syntax som skapades, här är arbetshöjden (Z) satt till 20-30. Output är G-code och input är kommando av skapad syntax:

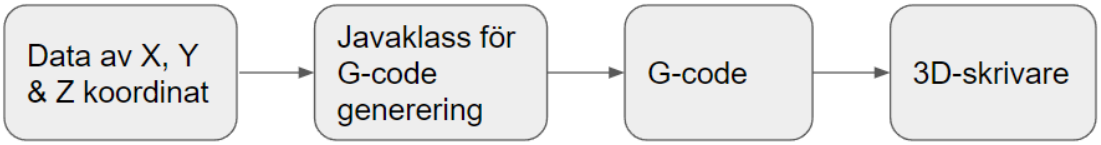
```
Input: tap 10,10 repeat 2
-------------------------
Output:
G1 X10 Y10
G1 Z20
G1 Z30
G1 X10 Y10
G1 Z20
G1 Z30
```

Denna syntax valdes för att kunna skriva större tester av G-code. För att översätta kommandorna till G-code skapades en klass CmdTerminalhandler.java. Denna klass har som huvudsyfte att översätta inkommande kommandon från terminalen till G-code. Men det skapades även andra funktioner som 'print gcode' vilket skriver ut och redovisar nuvarande G-code som är skriven på skroll panelen. Även en filinläsningsfunktion implementerades för att kunna starta redan befintliga tester.

Klassen använder sig av Regex i strängar. Denna funktion plockar ut värden från inkommande strängar. Detta gör att programmet kan känna igen kommandon och omvandla värderna till G-code. G-code omvandlingen sker i den ursprungliga klassen 'GcodeGenerator.java'.

3.7.4 G-code generering

Vid G-code generering skapades en klass 'GcodeGenerator.java' som hanterar indata i form av koordinater. För att generera G-code används Strings i Java. Varje generering av G-code kommando är uppdelat i var sin Javametod där vid varje tillfälle den blir kallad på, retunerar en färdig sträng med ett kommando. Hela processen från data av koordinater till att 3D-skrivaren tar emot G-code förklaras i figur 3.3.



Figur 3.3: Förklarar processen för att generera G-code som sedan skickas till 3D-skrivaren

Klassen som hanterar G-code generering blev skapad så att den är nåbar från både GUI och terminalen. Detta gör att metoderna som skriver G-code ej behöver finnas i någon de klasserna. Eftersom att det endast är den här klassen som genererar och skapar G-code kommandon var det naturligt att implementera statiska attribut på nuvarande koordinater av 3D-skrivaren. Funktionen ger programmet information om var 3D-skrivaren är.

3.7.5 INI-fil

För att nå målet med att skapa möjlighet för flexibilitet av testmetoden, skapades en funktion för att kunna ta in initiala parametrar i Java. Detta för att varje skärmbaserad produkt kan ha olika dimensioner på bredd och höjd men också tjocklek. Att hårdkoda dessa parametrar omöjliggör en tillämpning av testmetoden på flera olika skärmbaserade produkter. För att läsa av INI-filen skapades en ny klass som hanterar INI-filen och plockar ut parametrar. Dessa parametrar ska sedan gå att nå från alla andra klasser i programmet. Hanteringen gjordes med hjälp av en HashMap. Detta gör att parameterns namn blir nyckeln och parametervärdet blir värdet till nyckeln.

3.7.6 Seriell kommunikation

För att kommunicera med 3D-skrivaren användes ett färdigt bibliotek vid namn JSerialComm. Biblioteket implementerades i ny javaklass som har målet att ansluta, samla data och skicka G-code till 3D-skrivaren. Det skapades en metod för att ansluta sig till 3D-skrivaren. Metoden hämtade nödvändiga parametrar från INI-filen, därefter kallades en metod från JSerialComm som anslöt programmet till 3D-skrivaren. Vid skickande av G-code skapades en metod som tar in en lista av G-code kommandon. Den gick igenom listan och skickade ett kommando i taget. Kommandona skickades genom att först omvandla strängen till bitar genom metoden getBytes(). Dessa G-code kommandon skickades utan att ta hänsyn till vilka kommando som inte hann läsas av, av 3D-skrivaren. Detta skapade problem då 3D-skrivaren tappade bort vissa G-code kommandon och rörelserna var inte proportionella mot kommandon, det vill säga att rörelserna inte följde den ordning som Kkmmandona skickades i. Därför blev det aktuellt att behandla G-code kommandon via en buffert.

När ett G-code kommando är skickat till 3D-skrivaren svarar denna med ett ACK i en form av sträng, det vill säga att när nästa kommando ska skickas måste denna vänta på ett ACK. Detta gjordes med hjälp av en FIFO-queue som är en lista med kö-struktur. Detta gör att vid varje ACK kan ett kommando plockas ut från första platsen i kön.

3.7.7 Manuell styrning

Vid manuell styrning skapades det Java-metoder som via ett fönster i Java-swing plockar ut koordinater vid ett musklick. För att plocka ut koordinater från musklick på ett Java swing fönster skapades en testkod som utvärderade lösningen. Testkoden gick ut på att först skapa ett fönster av typen JFrame. Fönsterstorleken sattes till 1080x720 pixlar. För att sedan låta fönstret känna av ett musklick användes metoderna 'addMouseListener' och 'mousePressed'. MousePressed har en MouseEvent som kontruktor vilket skrivs om till att skriva ut koordinaterna som musikonen klickar på i koordinatsystemet.

Koden visar ett exempel på hur koordinater kan plockas ut från musklick:

```
    public static void createWindow(){
       JFrame frame = new JFrame("Coordinate getter");
       frame.setSize(1080,720);
       frame.addMouseListener(new MouseAdapter() {
           @Override
           public void mousePressed(MouseEvent e) {
               System.out.println(e.getX()-(frame.getSize().getWidth()/2) +
                   " " + (e.getY()-(frame.getSize().getHeight()/2))*-1);
           }
       });
       frame.setVisible(true);
   }
```

Koden visar att det var viktigt att bestämma var origo skulle befinna sig i koordinatsystemet. Standardinställningar för origo i ett koordinatsystem skapat av Java swing var längst upp i vänstra hörnet. Detta samverkade inte med koordinatsystemet som 3D-skrivaren använder sig av då den har sitt origo i mitten på skrivarplattan (grunden där skärmen placerades).

Två kodexempel som visar skillnaden mellan origo-placering:

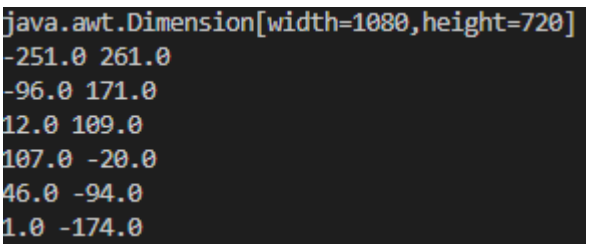
Koden nedan skriver ut koordinater där origo är längst upp till vänster

```
@Override
public void mousePressed(MouseEvent e) {
    System.out.println(e.getX() + " " + e.getY();
}
```

Koden nedan skriver ut koordinater där origo är i mitten på fönstret

```
@Override
public void mousePressed(MouseEvent e) {
    System.out.println(e.getX()-(frame.getSize().getWidth()/2) + " " +
        (e.getY()-(frame.getSize().getHeight()/2))*-1);
}
```

Samma metod för att ta ut koordinater användes sedan för att skriva en ny klass som blev ett eget fönster för manuell styrning. Det blev sedan klart att koordinaterna inte stämde överens med skrivarens koordinatformat på -60 till 60 X och -60 till 60 Y enligt figur 3.4. Det behövde åtgärdas genom att ändra fönsterstorleken så att koordinaterna stämmer överens med gränsvärderna för vad 3D-skrivaren kan hantera.



Figur 3.4: Utskriften av testkoden som plockar ut koordinater från ett fönster av Java Swing

När fönsterstorleken för koordinatsystemet skulle sättas började det med att anta koordinater som antal pixlar, det vill säga 120x120. För att fönstret skulle bli större multiplicerades denna dimension med 3. Storleken blev då 360x360.

Algoritmen för att räkna om koordinathämtningen syns i kodexemplet nedan. Algoritmen börjar med att formatera om origo så att den placeras i mitten. Detta görs med subraktion på koordinat-axlarna. Därefter delas värdet med tre då fönsterstorleken var dimensionerad som tre gånger större än verkliga koordinatera. Koden returnerar sedan de slutgiltiga X och Y koordinaterna.

```
/**
* Method to convert coordinates from window to 3D-printer coordinates
* @param x X coordinate from window
* @param y Y coordinate from window
* @return coordinate that fits 3D-printer format
*/
public String goToXY(double x, double y) {
    x = x - 180;
    x = x / 3;

    y = (y - 180) * -1;
    y = y / 3;

    return "X " + x + " Y " + y;
}
```

Dessa koordinater i programmet översattes sedan med hjälp av G-code generering enligt kodexemplet nedan:

```
x = 10;
y = 20;
String GcodeExample = "G1 X" + x + " Y" + y + "\n";
```

Koordinaterna skickas sedan direkt till 3D-skrivaren som då rör sig mot bestämd position.

Vid höjdstyrning skapades två knappar av JButton. Dessa knappar blev markerade som 'up' och 'down'. Genom ett Z (höjd) värde som var förbestämt i INI-filen, kunde knapparna öka eller sänka höjden på pekpennan med värdet på Z. Knapparna kallade på metoder som tar den nuvarande höjden och subtraherar eller adderar höjdskillnaden och sedan omvandlar detta till G-code. Detta G-code kommando skickas direkt till 3D-skrivaren.

Det skapades även en funktion som låter användaren skriva in en Z-koordinat och trycka på en knapp som styr 3D-skrivaren mot den höjden. Detta gjordes med att använda JTextField och en knapp av JButton.

För att inte den manuella kontrollen av 3D-skrivaren skulle påverka ett pågående test skapades även en knapp som stänger av och sätter på den manuella styrningen. Detta gjordes med hjälp av ett statiskt attribut av typen boolean. Attributet döptes till 'on', attributet sattes till true ifall manuella styrningen var på och tvärt om ifall den var avstängd.

Även klassen för seriella kommunikationen mellan 3D-skrivaren och Javaprogrammet behövde veta om den manuella styrningen var på. Anledningen var att vid varje nytt G-code kommand från manuella styrningen, gick 3D-skrivaren tillbaka till en slutposition som den gör när det inte finns fler G-code kommandon att skicka. Då infogades attributet i klassen och på grund av att den är deklarerad som statisk, blev det en enkel implementation utan att skapa ett nytt objekt av manuell styrning.

3.8 Fas 4 - Utveckling av ett testfall

Vid utveckling av ett testfall behövdes ett flertal parametrar tas i åtanke. Vid början av att skapa ett testfall måste initiala värden mätas av i form av mått på skärmen, lägsta höjd 3D-skrivaren får befinna sig i och vad som ska testas. För att försäkra sig om att 3D-skrivaren fungerar ihop med mjukvaran gjordes små initiala tester på en mobiltelefon med hjälp av manuell styrning. Detta för att se hur 3D-skrivaren kan röra sig och utföra gester på en vanligt pekskärm. Ett testfall skrevs för att skrolla flödet i en social media app. G-code skrevs genom att först mäta av en startpunkt för 3D-skrivaren. Denna punkt mättes genom att manuellt styra 3D-skrivaren till önskad koordinat. Koordinaten antecknades och via GUI användes funktionen för tryckrörelse. Tryckrörelsen öppnade då den önskade appen. Därefter skapades tre stycken sveprörelser som går från den nedre sidan av skärmen till den övre delen. Koordinaterna för dessa sveprörelse skapades även genom den manuella styrningen. Testningen skedde innan en buffert hade blivit implementerad.

3.9 Fas 5 - Utvärdering av testfall

För att gå vidare i processen var det nödvändigt att utvärdera resultatet av testfallet. Resultatet visade att flertal G-code kommandon hade tappats bort under den seriella uppkopplingen. Anledningen visade sig vara att 3D-skrivaren inte klarade av att få flertal G-code kommandon samtidigt. Detta åtgärdades med att skapa ett kö-system för varje kommando även kallat buffert.

3.10 Fas 6 - Resultat av testmetoden

För att utvärdera om testmetoden fungerar beskrivs det under denna rubrik hur data kunde hämtas och jämföras.

3.10.1 Inhämtning av data från pekskärm

Vid inhämtning av data från pekskrämen är det koordinater av position som registreras på pekskärmen som är intressant. För att läsa av detta användes ett färdigt bibliotek, evdev. Med detta bibliotek kan ett enkelt pythonscript skrivas för att hämta ut koordinater från pekskärmen. Eftersom att Evdev är linuxbaserat så användes en Raspberry pi som dator för pekskärmen. Med hjälp av en SSH anslutning från Raspberry pi till datorn där java programmet körs, kan data överföras och samlas in. För att skapa en SSH anslutning gjordes en klass som implementerar biblioteket JSch.

3.10.2 Automatiskt test

En metod skrevs för att automatiskt styra 3D-skrivaren samtidigt som metoden tog in data och validerade varje koordinat från pekskärmen. Testet utformades så att en slumpmässig koordinat valdes ut inom pekskärmens dimensioner. Koordinaten skickades sedan till 3D-skrivaren och ett koordinat-ID skapades som sparades i metoden. När 3D-skrivaren har utfört en rörelse på skärmen skapades en algoritm som hämtar inkommande koordinat från pekskärmen och kopplar ihop denna koordinat med koordinat-ID och 3D-skrivarens koordinat. Kopplingen konstruerades med Java Point[24] och HashMap.

Metoden byggdes i en while-loop och den forsätter testa tills en redan befintlig koordinat skickas till 3D-skrivaren. Då jämförs inkommande koordinaterna från pekskärmen med den redan befintliga koordinaten för samma 3D-skrivar-koordinat. Med denna algoritm kan variationer i pekskärmens precision detekteras.

3.10.3 Godkänt testfall

Den automatiska testmetoden skapades för att detektera precisionsskillnader. För att validera ett godkänt testfall behövdes ett godkänt område där precisionen får variera mellan. Variationen eller även kallat offset blev satt till värdet 200 av exjobbaren som en godkänd offset. Godkänt offset kunde även ställas in i INI-filen ifall metoden ska testa på skärmar med olika känslighet.

Om varje test rör sig inom den godkända ramen för variation i precision under alla 50 000 testfall kan hela det automatiska testet godkännas.

3.11 Källkritik

Källorna som används i detta examensarbete är antingen hämtade från Google Scholar[25], förstahandskällor från tillverkarna själva eller tekniska artiklar från olika företag. Google Scholar är en sökmotor som är en funktion av vanliga Google. Anledningen varför Google Scholar bedöms vara trovärdig är på grund av dess specialisering på akademiskt innehåll. Många artiklar som finns hos Google Scholar har även genomgott en 'Peer review' där forskningen har granskats och utvärderats av andra experter inom området.

När det gäller källor från tillverkare av produkter eller mjukvaror som används i examensarbetet bedöms informationen vara trovärdig. Detta är på grund av att företagen eller skaparen har ett intresse att berätta funktionalitet och specifikationer kring deras produkter eller mjukvaror. Intresset kan grunda sig i marknadsföring men också visa kompetens på marknaden.

Vid källor av tekniska artiklar från olika företag kan informationen vara bra sammanfattad av en samling av källor. Detta gör att läsaren får en övergripande uppfattning av ämnet.

Källor nedan som kommer från Google Schoolar:

[6] [8] [9] [11] [14] [21]

Källor nedan som kommer från tillverkare:

[1] [2] [3] [4] [5] [7] [10] [12] [13] [15] [22] [23] [24] [25]

Källor nedan som kommer från tekniska artiklar:

[16] [17] [18] [19] [20]

4 Resultat

Under denna rubrik redovisas resultat som det här examensarbetet har tillfört. Resultatet är uppdelat i underrubriker nedan som i helhet bildar ett fullständigt resultat för examensarbetet.

4.1 Examensarbetets resultat i helhet

Examensarbetet gav ett resultat som bevisar en möjlighet att automatisera ett testverktyg för precisionstestning. Resultatet visar också att hårdvaran i form av 3D-skrivare, stylus-penna och Raspberry pi kunde implementeras med hjälp av mjukvara.

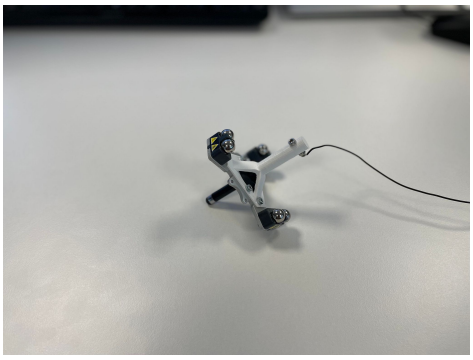
En mjukvara som styr en 3D-skrivare till att utföra konsekventa rörelser på en pekskärm har implementerats. Funktionerna i mjukvaran kan skapa skärmtester, starta skärmtester, samtidigt skicka data och kommunicera med en 3D-skrivare. 3D-skrivaren som blivit modifierad med en pekpenna kunde agera som ett finger och med rörelse, utföra tester på en pekskärm. Pekskärmen via en Raspberry pi med SSH-anslutning till en dator, skickade inkommen koordinatdata. Datan som blivit ingångsdata från skärmen kunde sedan jämföras med skickade koordinater till 3D-skrivaren. Jämförelsen mellan koordinater från 3D-skrivaren och koordinater från pekskärmen kunde sedan ge ett resultat om testfallet.

4.2 Hårdvara

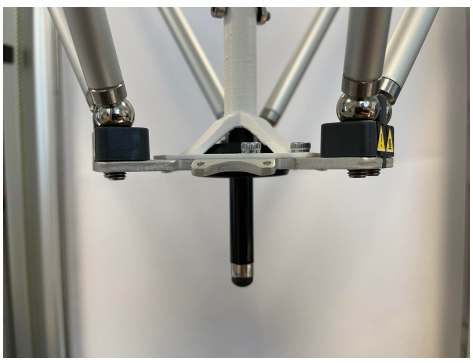
Under den här rubriken presenteras det färdiga resultatet av hårdvaran och dess roll i examensarbetet. Hårdvaran består av en 3D-skrivare, stylus-penna och Raspberry pi. 3D-skrivaren och stylus-pennan är direkt kopplade medan Raspberry pi behandlar och skickar ut data från en pekskärm.

4.2.1 3D-skrivare och stylus-penna

Resultatet av ombyggnationen av 3D-skrivaren med en stylus-penna visade att modellen kunde agera som ett finger. Den passiva stylus-pennan som var kopplad till en jordad punkt på 3D-skrivaren, skapade en skillnad av det elektriska fältet på den kapacitiva pekskärmen. Denna skillnad var nog för pekskärmen att känna av en händelse. Figur 4.1 och 4.2 visar resultatet av ombyggnationen av printerhuvudet.



Figur 4.1: Modifierat printer-huvud med en passiv pekpenna



Figur 4.2: Styluspenna monterad på 3D-skrivaren

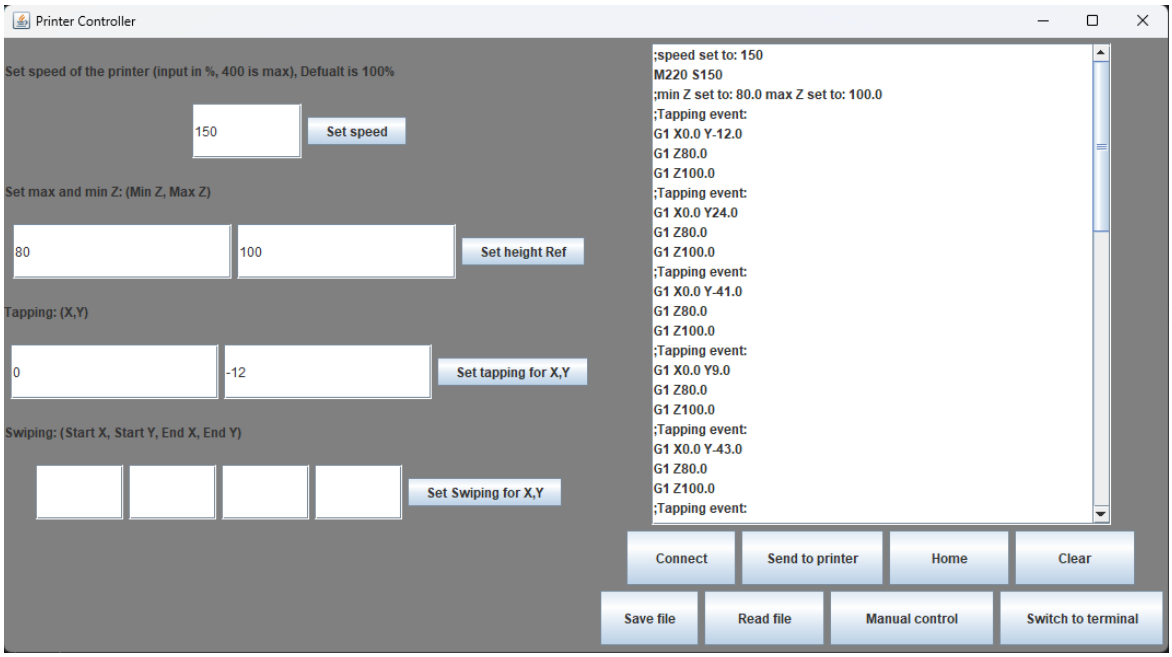
3D-skrivaren uppvisade en bra precision och instruerade rörelser följdes utan problem på grund av en buffert i mjukvaran som blev implementerad. Denna buffert blev en kö för olika G-code kommandon. Nästkommande G-code kommando blev endast skickat till 3D-skrivaren om 3D-skivaren svarade med ett ACK på det föregående G-code kommandot.

4.3 Mjukvara

Vid den här sektionen redovisas resultatet av den utvecklade mjukvaran som mestadels har programmerats i Java. Resultatet visar kopplingarna mellan olika funktioner som samverkar och skapar ett brett program för att utföra olika typer av uppgifter.

4.3.1 Grafiskt användargränssnitt (GUI)

Det skapade grafiska användargränssnittet även kallat för GUI som visas i figur 4.3 blev en central del för programmet att nå alla funktioner. Denna del baserades på Java Swing och kunde programmeras så att värden kunde överföras till andra klasser där algoritmer behandlade datan.



Figur 4.3: Överblick på GUI-versionen av programmet

Kodexempel visar grunderna kring hur data skickas till en klass och sedan åter hämtas i form av G-code.

Kod som tar X och Y värden från GUI och skickar vidare dessa värden:

```
tapB.addActionListener(e -> {
            // Get X and Y coordinates from GUI
            int X = Integer.parseInt(xTappingInput.getText());
            int Y = Integer.parseInt(yTapppingInput.getText());
            //Send to G-code Generator
            gGen.getTap(X, Y);
        });
```

Kod där data presenteras från en lista med G-code kommandon:

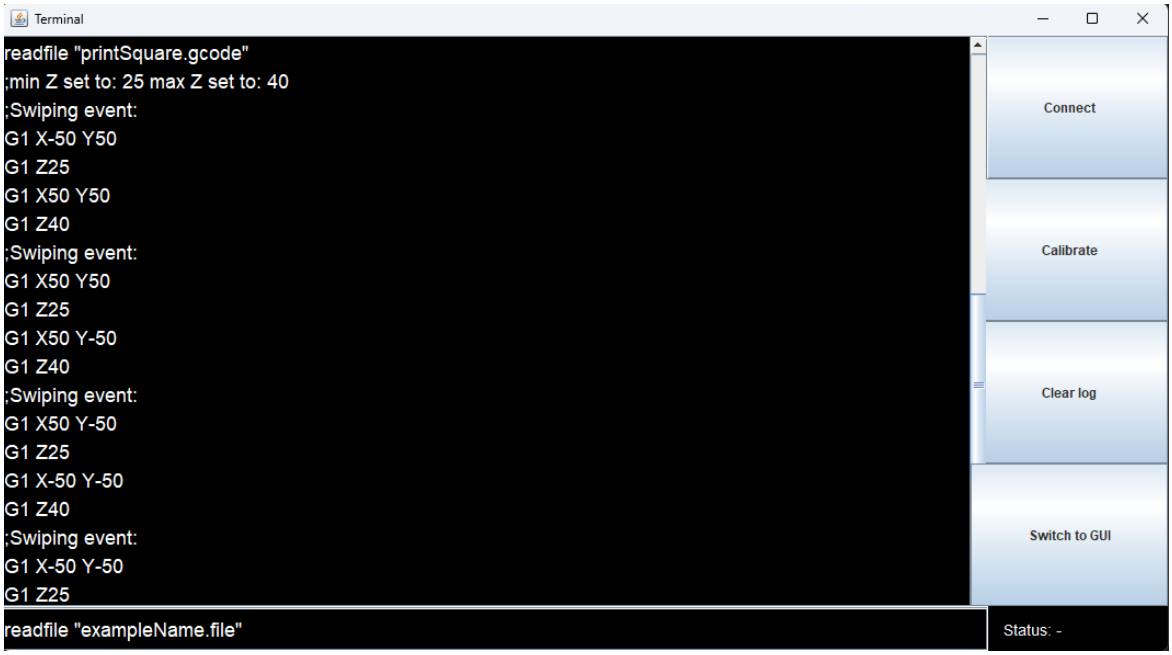
```
//Create panel of a JPanel object
JPanel panel = new JPanel();
//Get list of current G-code
JList<String> list = new JList<String>(gGen.getList());
//Create a ScrollPane and add to panel
JScrollPane scrollPane = new JScrollPane(list);
panel.add(scrollPane);
```

Med hjälp av GUI kan funktioner enligt lista tillämpas:

- Ändra och verkställ arbetshastighet för 3D-skrivare
- Sätt en arbetshöjd (Z), en min Z och max Z
- Tryckrörelse för en X och Y koordinat
- Sveprörelse för en X och Y till en annan X och Y
- Skriven G-code kan sparas i en fil
- Anslut och kalibrera 3D-skrivaren
- Läsa in filer med befintliga tester som innehåller G-code kommandon
- Presentera G-code via en skrollbar panel
- Rensa befintlig G-code i panelen
- Skicka G-code kommandon till en 3D-skrivare
- Öppna fönster för manuell kontroll
- Öppna Terminal-versionen av programvaran

4.3.2 Terminal

Terminalen som visas i figur 4.4 skapades med Java Swing. Med denna version kan ett kommando skrivas för att nå olika funktioner i programmet. Med hjälp av den egenskapade syntaxen och Java regex kan data plockas från användaren och sättas in i funktioner som skickar G-code till skrivaren. Resultatet av detta gjorde att en användare snabbt kunde skriva G-code utan att behöva klicka sig fram i ett GUI. Terminalen blev också skapad med fyra stycken knappar i form av en meny. Denna meny förenklade processen att använda terminalen då funktioner för varje knapp blev lättare att nå.



Figur 4.4: Överblick på Terminal-versionen av programmet

Eftersom att Terminalen var skapad med egen syntax gjordes en egen klass som behandlar alla kommandon. Vid behandlingen av kommando så måste varje inkommen text jämföras med ett korrekt format av ett kommando. Passar denna in går den vidare till en metod som utför vald funktion.

Koden nedan visar ett kommando på en tryckrörelse som går vidare till en metod:

```
/**
 * Handle a command from Terminal calls the rightfull method
 * @param textIn The command from Terminal
 */
public void handleText(String textIn) {
    if (!tap(textIn)) {
        Terminal.logListModel.addElement("'" + textIn + "' is not a
            command!" );
        Terminal.logListModel.addElement("type: \"help\" for commandlist
            ");
    } else {
        System.out.println("input: " + textIn);
    }
}
/**
 * Method that checks if command fits format for tap-motion
 * @param in command from terminal
 * @return True if command fits format and add G-code command to buffer
 * @return False of command not fit format of tap-motion
 */
public boolean tap(String in) {
    //Format of "tap X,Y repeat N" X & Y are coordinates, N are number
        of repeats
    String regex = "tap\\s(-?\\d*),(-?\\d*)\\s*(repeat\\s(\\d*))?";
    Matcher matcher = Pattern.compile(regex).matcher(in);
    if (matcher.matches()) {
        double x = Double.parseDouble(matcher.group(1));
        double y = Double.parseDouble(matcher.group(2));
        int repeat = Integer.parseInt(matcher.group(4));
        for (int i = 0; i < repeat; i++) {
            serial.addToBuffer(gGen.getTap(x, y));
        }
        return true;
    } else {
        return false;
    }
}
```

Terminalversionen av programmet blev mer flexibel att arbeta med än GUI-versionen. Detta på grund av att den var lättare att modifiera. Om en ny funktion av programmet skulle skapas blev det endast ett nytt kommando som behövs implementeras.

Med hjälp av terminalen kan funktioner enligt lista tillämpas:

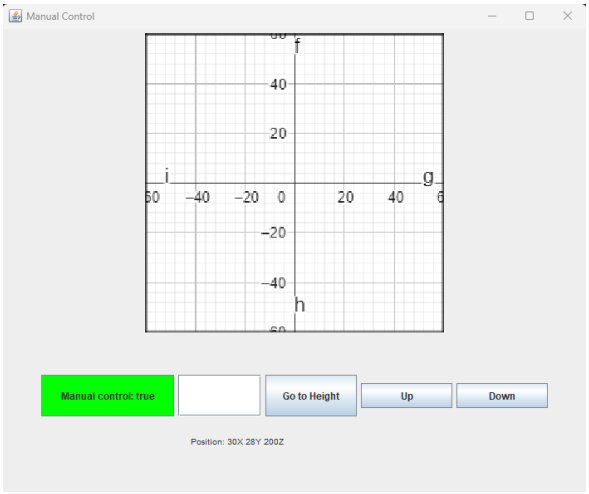
- Ändra och verkställ arbetshastighet för 3D-skrivare
- Sätt en arbetshöjd (Z), en min Z och max Z
- Tryckrörelse för en X och Y koordinat
- Sveprörelse för en X och Y till en annan X och Y
- Skriven G-code kan sparas i en fil
- Anslut och kalibrera 3D-skrivaren
- Läsa in filer med befintliga tester som innehåller G-code kommandon
- Presentera G-code i terminal-fönstret
- Rensa befintlig G-code i terminal-fönstret
- Skicka G-code kommandon till en 3D-skrivare
- Öppna fönster för manuell kontroll
- Öppna GUI-versionen av programvaran
- Gå till en X och Y koordinat
- Lista alla befintliga kommandon
- Tilldela felmeddelanden i terminal-fönstret

4.3.3 Manuell styrning

När knappen för manuell styrning eller kommandot i terminalen anropas, öppnas ett eget fönster. Detta fönster blev den manuella styrningen som resulterade i att sökning av koordinater förenklades. Sökning av koordinater möjliggjorde att rätt koordinater hamnade i de tester som skapades.

Ett två-dimisionellt koordinatsystem som visas i figur 4.5 blev tilldelat i en panel som kan hämta koordinater från musklick. Koordinaterna redovisas i fönstret i en JLabel. Samtidigt som en användare trycker i koordinatsystemet rör sig 3D-skrivaren till koordinaten. Detta gör att användaren kan gå till önskad position på skärmen och sedan läsa av koordinaten för berörd punkt.

För att ha möjlighet att ändra höjd på 3D-skrivaren skapades två knappar och ett skrivfält. Knapparna rör skrivaren upp eller ner i en förbestämd höjdskillnad som kan bestämmas med hjälp av en INI fil. I skrivfältet kan en höjd av en siffra skrivas i, med knapptryck på 'Go to Height' flyttar skrivaren höjd till användarens önskan.



Figur 4.5: Överblick på manuell styrning av 3D-skrivaren

Koden nedan visar hur koordinater kan plockas ut från knapptryck på en panel och sedan skickas vidare till andra metoder som behandlar dessa koordinater.

```
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        position.setText("Position: " + gGen.getXY());
        serial.addToBuffer(gGen.goToXY(e.getX(), e.getY()));
    }
});
```

4.3.4 G-Code generering

Vid G-code generingen har det skapats en egen klass där det finns metoder som tar in data i form av koordinater och retunerar färdiga strängar av G-code kommandon. Implementationen av klassen gav ett resultat av att testfiler kunde skapas med hjälp GUI eller terminalen.

I klassen sparades det attribut som nuvarande koordinater av 3D-skrivaren. Detta gjorde att objektet av klassen alltid visste om vart spetsen på stylus-pennan befann sig. Vid kallelse av en metod som ändrar dessa koordinater, ändrades också dessa attribut.

Exempel på G-code generering:

```
/**
* Conctructor of G-code generator class
*/
public GcodeGenerator() {
    listModel = new DefaultListModel<String>();
    ini = new InitValues();
    maxZ = ini.getMaxZ();
    minZ = ini.getMinZ();
    penDown = "G1 Z" + String.valueOf(minZ) + "\n";
    penUp = "G1 Z" + String.valueOf(maxZ) + "\n";
    currentHeight = ini.getStartHeight();
}

/**
* Creates a G-code commando for tap-motion
* @param X The X-coordinate
* @param Y The Y-Coordinate
* @return G-code commando of tap-motion
*/
public String getTap(double X, double Y) {

    String gPos = "G1 X" + String.valueOf(X) + " Y" + String.valueOf(Y)
        + "\n";
    //"listModel" is temporary storage of G-code commando
    //before it goes to the buffer
    listModel.addElement("\n" + ";Tapping event: " + "\n");
    listModel.addElement(gPos);
    listModel.addElement(penDown);
    listModel.addElement(penUp);
    return gPos + penDown + penUp;
}
```

4.3.5 Buffert av G-code kommando

Bufferten blev skapad för att säkerställa att varje kommando skickas till 3D-skrivaren och att varje kommando har blivit exekverad av maskinen. Bufferten programmerades genom att skapa en kö för alla G-code kommandon. Detta skapade en säkerhet av att kommandorna blir skickade i ordning. När ett kommando ska skickas körs det först en koll om 3D-skrivaren är redo för ett nytt kommand. Detta görs genom att vid varje kommando skickat, svarar 3D-skrivaren med ett ACK. Bara när ett ACK har fåtts så kan ett nytt kommando skickas.

Kod som kontrollerar att 3D-skrivaren har skickat ett ACK

```
/**
 * Check if the printer is ready to take another command.
 * @return True if it is ready, False if it is not.
 */
public boolean isReady() {
    boolean isOk = false;
    String[] read = read();
    ArrayList<String> sList = new ArrayList<String>();

    for (String s : read) {
        if (!s.equals("") || !s.contains("")) {
            sList.add(s);
        }
    }
    for (String s : sList) {
        if (s.contains("ok") || s.startsWith("ok")
        || sList.size() == 0) {
            isOk = true;
        } else {
            isOk = false;
        }
    }
    sList.forEach(s -> System.out.println("Ready: " + s.equals("ok")));

    return isOk;
}
```

När G-code ska skickas till 3D-skriveren används en while-loop. Denna loop skickar ett G-code kommand i taget och kallar på isReady() enligt kodstycket nedan.

```
while (!buffer.isEmpty()) {
    boolean ready = isReady();
    if (ready) {
        command = buffer.poll();
        comPort.writeBytes(command.getBytes(), command.length());
        Terminal.logListModel.addElement("Sent command: " + command);
        System.out.println("Sent command: " + command);
        ready = isReady();
    }
}
```

4.3.6 Koordinathämtning

Vid koordinathämtning användes SSH-anslutning till en raspberry pi. Vid skapande av Javaklassen som tog hand om SSH-anslutningen resulterade detta i tre metoder som stod till grund för processen.

För att ansluta till en SSH-host som i detta fall var raspberry pi behövs 'user', 'password', 'host' och 'port'. User är användarnamn, password är lösenord till anslutningen, host är IP-adressen till Raspberry pi och port är vilken port kommunikationen ska ske över. Koden nedan visar resultatet av en anslutning via SSH med Java.

```
/**
 * Establish SSH-connection & enable datareading from SSH
 */
public void connect() {
    try {
        JSch jsch = new JSch();
        session = jsch.getSession(user, HOST, PORT);
        session.setPassword(password);
        session.setConfig("StrictHostKeyChecking", "no");
        session.connect();
        System.out.println("Connected!");
        channel = (ChannelShell) session.openChannel("shell");
        inputStream = channel.getInputStream();
        outputStream = channel.getOutputStream();
        channel.connect();
        dataThread = new Thread(() -> {
            getData(System.out::println);
        });
        dataThread.start();
    } catch (JSchException | IOException e) {
        e.printStackTrace();
    }
}
```

Koden nedan visar resultat av Java som tillåter en användare att skicka kommandon i form av String till en SSH-host

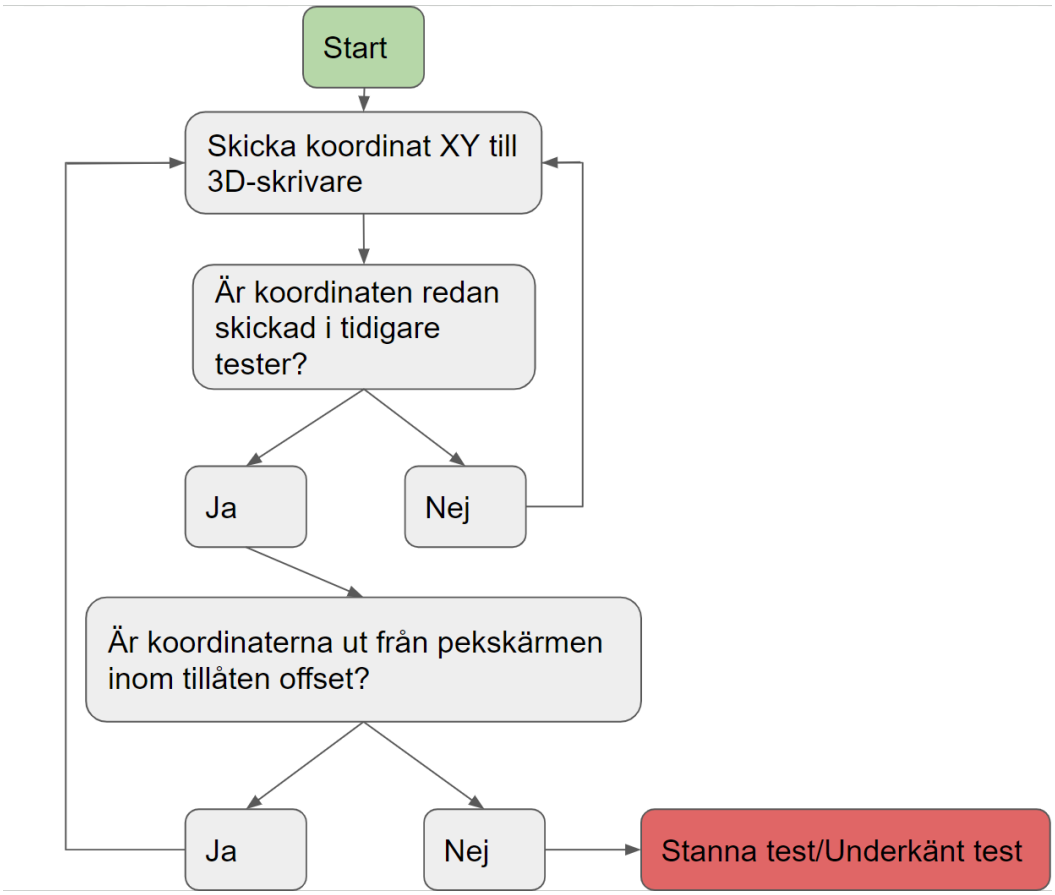
```
/**
 * Send commandline to SSH-host
 * @param command - The command sent to SSH-host
 */
public void sendCommand(String command) throws IOException {
    outputStream.write((command + "\n").getBytes());
    outputStream.flush();
}
```

Koden nedan visar inhämtning av data från SSH-anslutningen.

```
/**
 * Read data from SSH-host
 */
public void getData(Consumer<String> dataHandler) {
    BufferedReader reader = new BufferedReader(new
        InputStreamReader(inputStream));
    try {
        String line;
        while ((line = reader.readLine()) != null) {
            dataHandler.accept(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

4.3.7 Testvalidering

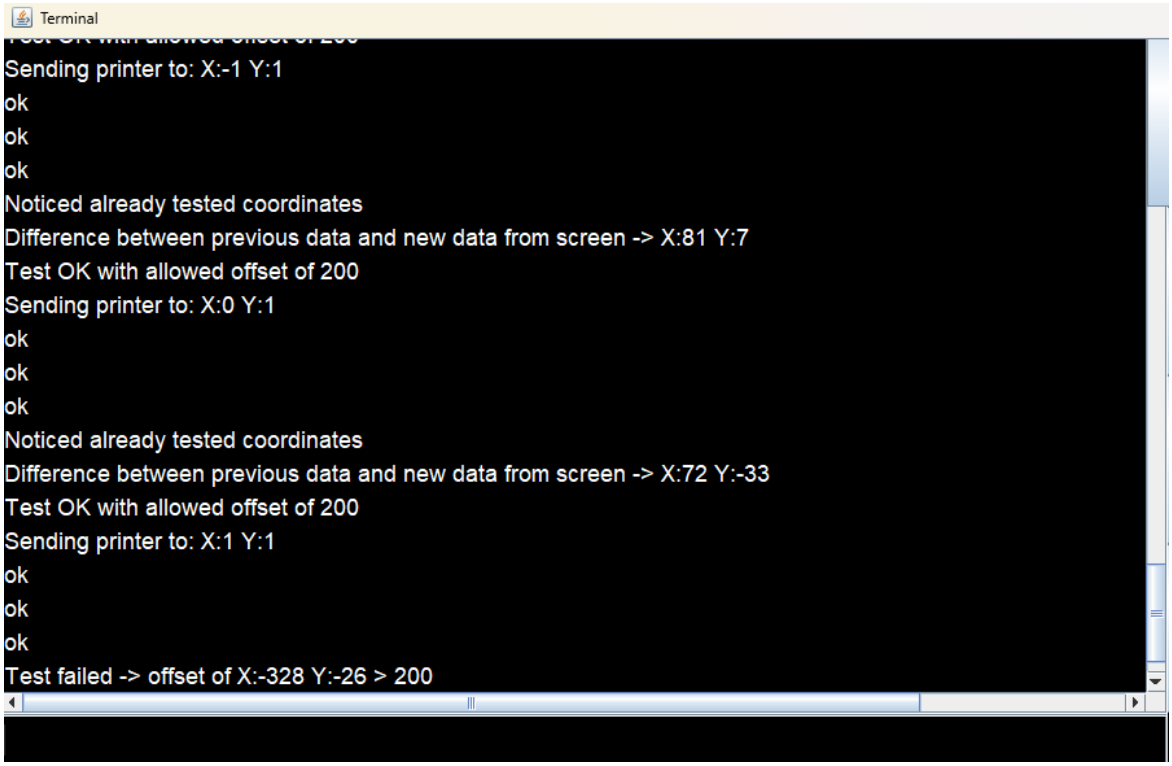
Vid testvalidering användes algoritmen som beskrivs i figur 4.6. Algoritmen blev implementerad i en egen metod som kan startas i Terminalen med ett kommando. När metoden är startad utförs tester tills metoden hittar en avvikelse där precisionskillnaden överstiger godkänd offset. Om den hittar en sådan avvikelse är testet underkänt.



Figur 4.6: Flödesschema av automatiskt test

Resultatet av metoden visar att den kan hitta en avvikelse. I testfallet som visas i figur 4.7 flyttades skärmen från 3D-skrivaren för att simulera att pekskärmen skickar ut fel position i form av koordinater. Testet stannades och utskriften visar hur mycket positionen varierade för samma tryckt koordinat av 3D-skrivaren.

Utskriften visar även de ACK som 3D-skrivaren skickar tillbaka efter varje G-code kommando. Dessa ACK kan kännas igen som 'ok' i Terminalen.



Figur 4.7: Utskrift från ett underkänt test

5 Slutsats

I den här sektionen tas en sammanfattning upp av de viktigaste resultaten av detta examensarbete. Det kommer även redovisas svar från problemformuleringarna för examensarbetet i sektion 2.4.

Syftet av examensarbetet som förklaras i sektion 2.2, var att konstruera en automatiskt testmetod för en pekskärm. Denna testmetod var teoretiskt tänkt att testa precision på en pekskärm med hjälp av en 3D-skrivare och en styrande mjukvara. Tillsammans var målet att vara helt autonom. Resultaten speglar syftet då en prototyp med en 3D-skrivare och en styrande mjukvara har skapats. Mjukvaran kan styra 3D-skrivaren genom att skicka G-code kommandon. Mjukvaran kan skriva nya tester men även läsa in befintliga tester. Programmet kan även användas genom ett GUI eller en terminal. För att hitta koordinater inför skrivande av tester så erhåller även mjukvaran möjlighet till att manuellt styra 3D-skrivaren till vald position på pekskärmen. För att validera tester och undersöka beteendet hos pekskärmen används en Raspberry pi som skickar data till Javaprogrammet via SSH. Syftet med examensarbetet har uppfyllts på grund av de erhållna resultaten.

Målen med examensarbetet beskrivs i sektion 2.3 där tre delmål nämns. Utifrån beskrivningarna kan huvudmålet 'Utveckla ett verktyg för automatisk testning av skärmar' bedömmas som uppfyllt med hänvisning till examensarbetes angivna resultat. Resultatet visade att en automatiserad testmetod har utvecklats med både mjukvara och hårdvara för att utföra rörelser och gester på en pekskärm.

Vid delmålet om att simulera ett års användning av pekskärmen, visade testmetoden en god möjlighet till att nå målet. Dock på grund av att hela målet inte fick plats inom ramen för examensarbetet kunde ej 50 000 tester genomföras. Därav blev ej hela målet uppfyllt.

Det slutgiltiga delmålet om att tillämpa flexibilitet av testmetoden visade sig bli uppfyllt. På grund av en implementation av programmet med en INI fil kunde parametrar ändras för att passa den skärm som skulle bli testad. För att hårdvaran ska vara flexibel behövdes hela ytan på 3D-skrivarens ritningsplan vara tillgänglig. Detta kunde uppfyllas till en gräns på 180 mm x 180 mm.

5.1 Svar på frågeställningar

Vid denna rubrik redovisas svar på frågeställningarna utifrån erhållet resultat av examensarbetet.

5.1.1 Problem 1 - Hur går det att automatisera en testmetod för att testa långtidsanvändning av en skärm?

Resultatet påvisar att skapad mjukvara samspelar bra med hårdavaran. Tillsammans skapar de en testmetod som kan appliceras på en pekskärm. Pågrund av att mjukvaran kan skriva tester och läsa in befintliga tester öppnar det möjligheten att starta stora tester som kan pågå upp till en vecka. Hårdvaran styrs av mjukvaran och eftersom att 3D-skrivaren är en maskin som redan är konstruerad för att arbeta under lång tid uppstod inte större begränsningar.

5.1.2 Problem 2 - Vilka parametrar bör tas i hänsyn vid testning av en pekskärm?

Ett flertal parametrar blev uppmärksammade under examensarbetets gång. En parameter som bör tas i hänsyn är höjdreferensen(max höjd och minsta höjd) till den skärm som ska testas. Då 3D-skrivaren ensam inte vet när den har tryckt på pekskärmen. Det är också viktigt för att inte 3D-skrivaren ska skada skärmen. Vid tester genom examensarbetets gång blev det klart att höjdreferensen måste sättas som ett initialt värde i en INI-fil. Detta skapar en enkelhet att applicera testmetoden på en annan skärm med en annan höjd.

5.1.3 Problem 3 - Hur kan data inhämtas i automatiserad testning?

Under examensarbetets gång märktes det att data kan inhämtas och dokumenteras på olika sätt och från olika källor. Vid data från 3D-skrivaren, dvs ACK-meddelandena, hämtades dessa från seriell koppling mellan datorn och 3D-skrivaren. Vid hämtning av data från pekskärmen gjordes detta med hjälp av en Raspberry pi som var direktkopplad med en javaklass via SSH anslutning. Datan hämtades i form av String som sedan kunde behandlas.

5.1.4 Problem 4 - Vilka faktorer kan påverka testresultaten?

Vid initiala tester av hårdvaran uppstod ett problem med stylus-pennan. Då den till en början var isolerad, kände inte pekskärmen av stylus-pennan. Detta åtgärdedes med hjälp av att koppla stylus-pennan till en jordadpunkt på 3D-skrivaren. På detta sett skapades tillräckligt med skillnad av elektriska fält på pekskärmen när styluspennan vidrörde skärmen.

5.2 Reflektion över konfidentiell information

Axis Communications AB är ett bolag ägnad åt säkerhet. När det gäller produkter som ska tillföra säkerhet till kunder, är det viktigt att produkternas tillverkningsprocess och källkod behålls hemligt för utomstående parter. När detta hålls konfidentiell skapar det en ytligare brandvägg för obehöriga. Som många andra företag gäller även konfidentiallitet för nyutvecklade men inte färdiga produkter. Detta för att inte läckor ska uppstå och att orätta företag tar patent för en produkt.

Examensarbetet har som uppgift att utveckla en testmetod som testar pekskärmar. Vissa pekskärmar som testas kan fortfarande vara under utvecklingsfasen, därför behövs testdata och resultat att hållas hemligt. Detta för att inte skada företaget eller oroa marknaden ifall fel hittas. Men också för att inte dela testresultat med konkurrenter som utför samma sorts testning.

Eftersom att de flesta på Axis har skrivit på ett NDA (Non-disclosure agreement), kan konfidentiell information inom Axis delas. På detta sätt kan diskussioner och utveckling fortfarande ske även fast informationen är hemlig.

Oansvarigt hanterande av konfidentiell information kan skapa läckor och skada företaget. Även de privatpersoner och utomstående företag som använder sig av produkter från Axis kan skadas om hemlig information om en produkt läcks.

5.3 Framtida utvecklingsmöjligheter

Utförandet av examensarbetet väckte många tankar om utvecklingsmöjligheter. Projeketet i sin helhet går att utarbeta och utveckla på många plan. Delmålet om att ta med parametrar som regn och fukt kan vara viktigt att räkna som en påverkande faktor när det gäller precisionen på en skärm. Att skapa en testmetod som tar med sådana parametrar kan bli en uppgift för framtida utvecklingar.

På grund av att stylus-pennan är monterad utan dämpning på skärmen går det ej att visa att pekpennan trycker för hårt eller för löst på skärmen. Med en dämpningsfjäder mellan hållaren och stylus-pennan skulle detta kunna bidra till att tryckkraft inte överstiger det maximala som pekskärmen tolererar. Detta kan vara en framtida åtgärd och en utvecklingspotential för examensarbetets hårdvara.

För att säkerställa att användningen av GUI och terminalen ska ske utan buggar är en utvecklingsmöjlighet att skapa en mer centrerad struktur på mjukvaran. Att ha en datastruktur där klasser ärver statiska attribut och att programmet minimeras. Detta gör att programmet kan startas sekvensiellt och i rätt ordning. Även att mer felsäkringar implementeras. Om initiala värden är felinsatta kan 3D-skrivaren skada skärmen. För att inte detta ska hända kan en trycksensor kopplas på stylus-pennan som känner av tryckkraften. Om denna överstiger ett visst värde måste då 3D-skrivaren stanna. En annan utvecklingspotential av prototypen är att sammankoppla övervakning av långtidstestningen med en kamerasensor. Detta gör att om skärmen skulle flytta på sig kan kameran detektera detta.

Vid avgränsningarna nämns det att det endast ska utvecklas en metod för att testa precision på en pekskärm. Med tanke på resultatet kan denna metod också appliceras på apptestning. Då testerna utför rörelser och gester som liknar en användare finns det möjlighet att utveckla testmetoden för app-utveckling. Det som behöver utvecklas isåfall blir hur mjukvaran bestämmer ett godkänt eller ett underkänt testfall. Vid minnestestning, det vill säga minnesläckor kan denna metod appliceras utan en vidare utveckling förutsatt att valideringsmjukvaran är programmerad i mobiltelefonen. Detta är pågrund av 3D-skrivaren redan kan utföra det antal rörelser som en användare bestämmer själv.

6 Terminologi

Här beskrivs begrepp som nämns i rapporten med korta förklaringar.

- String - En variabeltyp i java som innehåller text, sifftor och tecken
- int - En variabeltyp i java som innehåller reella heltal
- Stylus-penna - en pekpenna
- Metod - En funktion som kan utföra algoritmer eller returnera variabeltyper i Java
- Retunera - skicka en variabeltyp eller objekt från en metod i Java
- Javabibliotek - Färdigimplementerad Java kod som kan användas av andra klasser
- Regex - reguljärt uttryck. Består av en sträng som följer särskilda syntaxregler och används mycket i många texteditorer och sökning av specifika ord.

Litteratur

[1] Axis Communications. *About Axis — Axis Communications*. We Are Axis, 2023. URL: https://www.axis.com/about-axis (hämtad 2023-03-21).

[2] Axis Communications. *Moments that made us — Axis Communications*. History, 2023. URL: https://www.axis.com/about-axis/history (hämtad 2023-03-21).

[3] Johan Wulf och Erik Fahlström Svensson. *Automated Functional Tests for a Web Application*. 2022.

[4] Jonas Klauber. *Automatic Timing Test of Physical Access Controll Systems*. 2014.

[5] Ragnar Wernersson. *Robot Control and Computer Vision for Automated Test System on Touch Display Products*. 2015.

[6] James Gosling och Addison-Wesley. *The Java Language Specification*. Addison -Wesley, Cop, 2014.

[7] Javapoint. *Java JScrollPane - javatpoint*. www.javatpoint.com, 2011. URL: https://www.javatpoint.com/java-jscrollpane (hämtad 2023-05-04).

[8] Kamran Latif, Anbia Adam, Yusri Yusof och Abdul Kadir. "A review of G code, STEP, STEP-NC, and open architecture control technologies based embedded CNC systems". I: *The International Journal of Advanced Manufacturing Technology* 114 (juni 2021), s. 2549–2566. DOI: 10.1007/s00170-021-06741-z. (Hämtad 2023-05-03).

[9] Daniel B. Fasnacht, Adrian M. Whatley och Giacomo Indiveri. *A Serial Communication Infrastructure for multi-chip Address Event Systems*. IEEE Xplore, maj 2008. DOI: 10.1109/ISCAS.2008.4541501. URL: https://www.researchgate.net/publication/224317789_A_serial_communication_infrastructure_for_multi-chip_address_event_systems (hämtad 2023-05-03).

[10] . *JSerialComm*. GitHub, maj 2023. URL: https://github.com/Fazecast/jSerialComm (hämtad 2023-05-03).

[11] Ben Tyers. "INI Files". I: *Apress eBooks* (jan. 2016), s. 155–160. DOI: 10.1007/978-1-4842-2373-4_18. (Hämtad 2023-05-03).

[12] Tatu Ylonen. *SSH Secure Shell home page, maintained by SSH protocol inventor Tatu Ylonen. SSH clients, servers, tutorials, how-tos.* www.ssh.com, 2023. URL: https://www.ssh.com/academy/ssh (hämtad 2023-05-10).

[13] JCraft. *JSch - Java Secure Channel*. www.jcraft.com, 2016. URL: http://www.jcraft.com/jsch/ (hämtad 2023-05-10).

[14] Simon Yuill och Harry Halpin. *Python*. 2006. URL: http://www.ibiblio.org/hhalpin/homepage/notes/pythonhandout.pdf (hämtad 2023-05-01).

[15] Vojtech Pavlik. *1. Introduction — The Linux Kernel documentation*. docs.kernel.org, 1999. URL: https://docs.kernel.org/input/input.html (hämtad 2023-05-03).

[16] Sam Daley. *What is 3D Printing? How do 3D Printers Work? What can be 3D Printed? — Built In*. builtin.com, juli 2022. URL: https://builtin.com/3d-printing (hämtad 2023-05-10).

[17] opensource.com. *What Is a Raspberry Pi?* Opensource.com, 2012. URL: https://opensource.com/resources/raspberry-pi (hämtad 2023-05-10).

[18] Andrew Zola. *What is capacitive touch screen? - Definition from WhatIs.com*. WhatIs.com, 2022. URL: https://www.techtarget.com/whatis/definition/capacitive-touch-screen (hämtad 2023-05-10).

[19] Todd Davis. *Explained! How Does Capacitive Stylus Work?* essentialpicks.com, maj 2021. URL: https://essentialpicks.com/how-does-capacitive-stylus-work/ (hämtad 2023-05-12).

[20] Hellen Issak. *Vad är automatisering?* Ramboll, 2023. URL: https://se.ramboll.com/press/artiklar/vad-ar-automatisering (hämtad 2023-03-21).

[21] Craig Larman. *Agile and Iterative Development: a Manager's Guide*. Addison-Wesley Professional, 2004. URL: https://books.google.se/books/about/Agile_and_Iterative_Development.html?id=76rnV5Exs50C&redir_esc=y (hämtad 2023-05-10).

[22] Git. *Git*. Git-scm.com, 2019. URL: https://git-scm.com/ (hämtad 2023-05-10).

[23] Velleman. *DATASHEET VERTEX DELTA 3D PRINTER (K8800)*. velleman-projects.com, 2018. URL: https://www.velleman.eu/downloads/0/infosheets/datasheet_k8800_en.pdf (hämtad 2023-05-03).

[24] Oracle. *Point (Java Platform SE 8)*. docs.oracle.com. URL: https://docs.oracle.com/javase/8/docs/api/java/awt/Point.html (hämtad 2023-05-19).

[25] Google Scholar. *About Google Scholar*. Google.com, 2019. URL: https://scholar.google.com/intl/en/scholar/about.html (hämtad 2023-05-10).