YRKESHÖGSKOLAN
NOVIA

Implementering av en PID-regulator på mikrokontroller och styrning samt monitorering av regulatorn på en Androidenhet via Bluetooth®

Mikael Dahlbom

Examensarbete för ingenjörsexamen (YH)

El- och automationsteknik

Vasa 2023

EXAMENSARBETE

Författare: Mikael Dahlbom

Utbildning och ort: Automationsingenjör, YH, Yrkeshögskolan Novia i Vasa

Inriktning: Automationsteknik

Handledare: Joachim Böling

Titel: Implementering av en PID-regulator på mikrokontroller och styrning samt monitorering av regulatorn på en Android-enhet via Bluetooth®

Datum: 4.6.2023 Sidantal: 58 Bilagor: 2

Abstrakt

Detta examensarbete handlar om att implementera en PID-regulator på en mikrokontroller och att styra och monitorera regulatorn med en Androidenhet via Bluetooth. Syftet med arbetet var att undersöka om det är möjligt att implementera en PID-regulator på en mikrokontroller, hur bra den fungerar och om det går att styra och monitorera regulatorn med en Androidenhet via Bluetooth. Som mål för arbetet var att få en fungerande helhet, där en reglerad process kunde styras trådlöst från en Androidenhet. Androidenheten skulle också få resultat om hur processen fungerade.

Examensarbetet tar upp teori kring PID-regulatorer och styrteknik samt teori om Bluetoothkommunikation. Teorin handlar bland annat om reglerprinciper, regleregenskaper, inställning av regulatorer, Bluetoothtekniken, Bluetooth SIG och GATT-protokollet.

Tester blev utförda på en vätskenivåregleringsprocess för att undersöka att styrningen och resultatuppsamlingen fungerade korrekt. Regulatorn blev testad som P-regulator, P-regulator med inställd offset, PI-regulator och PID-regulator. Störningstest och börvärdesändringstest blev också utförda för att undersöka hur bra regulatorn reagerar på störningar.

Androidapplikationen som användes för att styra och monitorera regulatorn skapades i Android Studio. Applikationen hade funktionalitet som att ställa in parametrar för regulatorn, starta och stoppa processen, plotta bör-, ärvärde och styrsignal i en graf och skriva resultatet till en csv-fil. PID-regulatorn blev implementerad på mikrokontrollern Arduino Nano 33 iot och programmerades i Arduino IDE.

Resultatet blev en fungerande PID-regulator som kunde styra och monitorera en vätskenivåregleringsprocess.

Språk: svenska
Nyckelord: PID-regulator, mikrokontroller, process, Android, Bluetooth

OPINNÄYTETYÖ

Tekijä: Mikael Dahlbom

Koulutus ja paikkakunta: Automaatioinsinööri, AMK, Yrkeshögskolan Novia, Vaasa

Suuntautumisvaihtoehto: Automaatiotekniikka

Ohjaaja(t): Joachim Böling

Nimike: PID-regulaattorin ohjelmointi mikro-ohjaimeen ja regulaattorin säätö ja valvonta Android-laitteen ja Bluetooth®-tekniikan avulla

---

Päivämäärä: 4.6.2023 Sivumäärä: 58 Liitteet: 2

Tiivistelmä

Tämä opinnäytetyö käsittelee PID-regulaattorin ohjelmointia mikro-ohjaimeen ja regulaattorin säätäminen ja valvonta Android-laitteen ja Bluetooth-tekniikan avulla. Työn tarkoituksena oli selvittää, onko mahdollista ohjelmoida PID-regulaattoria mikro-ohjaimeen, miten hyvin se toimii ja onko mahdollista säätää ja valvoa regulaattoria Android-laitteen avulla. Tavoitteena oli saada hyvin toimiva kokonaisuus, jossa prosessia voi säätää langattomasti Android-laitteen avulla. Android-laite sai myös tietoa prosessin toimivuudesta, jokaisen prosessiajon jälkeen.

Opinnäytetyössä kerrotaan PID-regulaattorista, ohjaustekniikasta ja Bluetooth-kommunikaatiosta. Aiheet ovat mm. ohjausperiaatteet, ohjausominaisuudet, regulaattorin säätäminen, Bluetooth-tekniikka, Bluetooth SIG ja GATT-protokolla.

Regulaattoria testattiin nestetaso-ohjausprosessin avulla tutkiakseen, että regulaattorin säätö ja valvonta toimi oikealla tavalla. Regulaattoria testattiin P-regulaattorina, P-regulaattorina offsetarvolla, PI-regulaattorina ja PID-regulaattorina. Regulaattorille suoritettiin häiriötestejä ja asetusarvon muuttamistestejä tutkiakseen regulaattorin reagointikykyä.

Android-applikaatio, jolla regulaattoria säädettiin ja valvottiin, kehitettiin Android Studiossa. Applikaatiossa oli toiminallisuuksia kuten regulaattorin parametrien säätö, prosessin aloittaminen ja lopettaminen, asetus-, hetkellisarvon ja ohjaussignaalin piirtäminen kaavioon ja tuloksen kirjoittaminen csv-tiedostoon. PID-regulaattoria ohjelmointiin Arduino Nano 33 iot nimiseen mikro-ohjaimeen.

Tulokseksi saatiin toimiva PID-regulaattori, joka osasi säätää nestetaso-ohjausprosessia.

Kieli: Ruotsi
Avainsanat: PID-regulaattori, mikro-ohjain, prosessi, Android, Bluetooth

BACHELOR'S THESIS

Author: Mikael Dahlbom
Degree Programme: Bachelor's Degree in Engineering, Novia University of Applied Sciences, Vaasa
Specialisation: Automation Technology
Supervisor(s): Joachim Böling

Title: Implementation of a PID Controller on a Microprocessor and Controlling and Monitoring of the Controller via an Android Device with Bluetooth®

Date: June 4, 2023 Number of pages: 58 Appendices: 2

Abstract

This Bachelor's thesis is about the implementation of a PID controller on a microprocessor and the regulation and monitoring of the controller via an Android device with Bluetooth. The purpose of the thesis was to investigate if it is possible to implement a PID controller on a microprocessor, how well it will work, and if it is possible to control the controller via an Android device with Bluetooth. The aim of the thesis was to make a working totality where a process could be wirelessly controlled by an Android device. The Android device should also be able to receive results from the controlled process.

Theory about the PID controller, control engineering and Bluetooth communication is discussed in the thesis. The theory consists of control principles, control properties, tuning of PID controllers, Bluetooth technology, Bluetooth SIG, the GATT protocol, etc.

The PID controller was tested with a liquid level controlling process to determine if the controlling and monitoring of the PID controller worked correctly. The controller was tested as a P controller, a P controller with offset, a PI controller, and a PID controller. Disturbance and set point step tests were also done for the controller.

The Android application that controlled and monitored the controller was developed in Android Studio. The application had functionality like changing controller parameters, starting and stopping the process, plotting set point, process value and control signal in a graph, and writing the results to a CSV file.

The result of the thesis was a fully working PID controller that was able to control and monitor the level in a liquid level controlling process.

Language: Swedish
Key words: PID controller, microprocessor, process, Android, Bluetooth

Innehållsförteckning

1 Inledning

Detta examensarbete handlar om att implementera en PID-regulator på en mikrokontroller och att göra en Androidapplikation som sänder och mottar data från mikrokontrollern via Bluetoothkommunikation. Jag är själv uppdragsgivare för projektet och handledare för projektet är Joachim Böling, lektor vid Yrkeshögskolan Novia i Vasa. Mikrokontrollern kommer att vara en Arduino Nano 33 Iot med inbyggd Bluetoothmodul och det kommer troligtvis att behövas tilläggskomponenter för signalanpassning mellan mikrokontrollern och processen som ska styras. Androidenheten som kommer att användas för utvecklingen av PID-regulatorapplikationen är mobiltelefonen Samsung Galaxy A13. PID-regulatorn kommer att ställas in och testas med hjälp av en eller flera processer i Technobotnia. Examensarbetet kommer att ta upp en teoretisk del som berättar om PID-regulatorer och hur de ställs in och teori kring Bluetoothkommunikation. En praktisk del kommer att handla om Androidapplikationen, implementeringen av PID-regulatorn samt utförda test och inställningar för regulatorn.

1.1 **Syfte** och **mål**

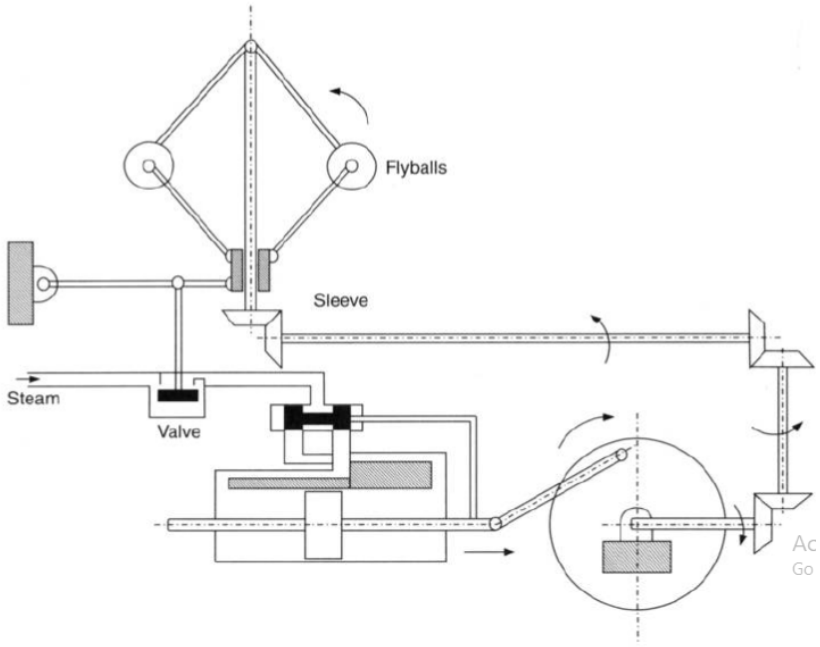
Syftet med examensarbetet var att undersöka om det är möjligt att implementera en PID-regulator på en mikrokontroller, hur bra den fungerar och huruvida det är möjligt att styra och monitorera den reglerade processen på en Androidenhet via Bluetooth. Arbetet kommer att ge en bra förståelse för hur mjukvara och hårdvara hänger ihop och lärdomar kring PID-regulatorer.

Målet med arbetet var att få en fungerande PID-regulator som kan styras och monitoreras trådlöst via en Androidenhet och som klarar av att styra olika processer. PID-regulatorn kommer också att få en auto tune-funktion som betyder att det borde vara möjligt att ställa in regulatorn automatiskt med endast ett knapptryck.

2 PID-regulatorn

Redan sedan stenåldern har människan varit bra på att kontrollera saker med hjälp av hens intelligens. Människan kontrollerade andra djur för att göra olika arbeten, till exempel häst och vagn för transport eller en oxe i jordbruket. Det tog dock väldigt lång tid

tills människan konstruerade det första automatiska kontrollsystemet, alltså där en process kan kontrolleras utan mänsklig interaktion. Det första välfungerande automatiska kontrollsystemet var tillverkat av James Watt år 1769 och användes för att reglera ångmotorns rotationshastighet. Systemet använde sig av runda vikter som påverkades av centrifugalkraften då de roterade med motorns hastighet. Då vikterna roterade snabbare trycktes de uppåt på grund av centrifugalkraften, det medförde att en ventil, som styrde flödet av ånga till maskinen, stryptes, vilket ledde till att motorns hastighet sänktes. Regulatorn såg ut som apparaturen i figur 1:



Figur 1: James Watts regulator för att styra ångmaskinens rotationshastighet. (Burns, 2001).

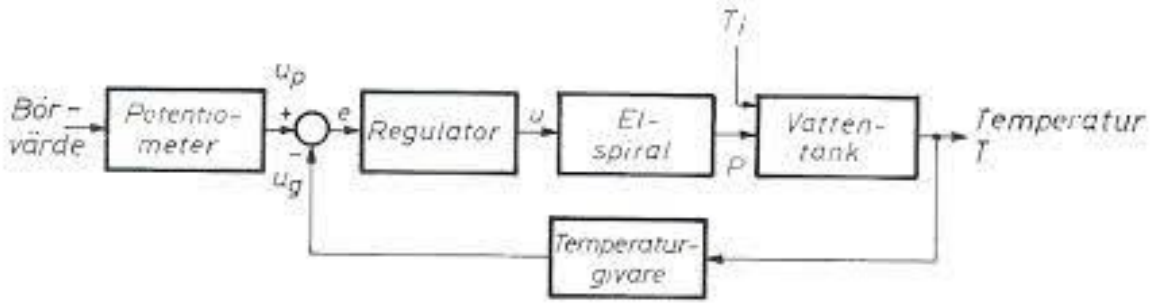
Watts regulator hade dock endast proportionella delen av dagens regulator, som betydde att regulatorns fel blev större ju större motorns last var. (Burns, 2001).

Deriverande delen användes första gången för att styra djupet för en Whitehead torped år 1868 och första PID-regulatorn med alla delar blev skapad av Elmer Sperry år 1911. Sperry hade dock inga matematiska uttryck för sin regulator, utan grundade regulatorn på intuition. Den första mannen som grundade matematiska lagarna för PID-regulatorn var rysk-amerikanska ingenjören Nicolas Minorsky år 1922. (PID controller: Origins, 2023).

2.1 Allmän teori om reglerteknik

Reglerteknik innefattar system som fungerar automatiskt, framför allt återkopplade automatiska system. Reglertekniken är också en teknik som används inom många olika områden så som kemiteknik, energiteknik och elkraftteknik. (Thomas, 2016). Den komponent som styr ett reglerat system brukar kallas för reglerenhet eller regulator och reglersystemet behövs främst för att det finns en variabel, som man vill att ska följa ett visst beteende. Reglersystemet kan till exempel sköta om att temperaturen i ett hus hålls konstant oberoende av vad utomhustemperaturen är. Reglerenheten i huset kan justera temperaturen med hjälp av ett eller flera värmeelement som finns i huset. Värmeelementen skulle i det här fallet kunna kallas för styrdon, eftersom de styr hur mycket värme som tillförs till huset. Utöver styrdonen behövs också något som håller koll på vad den aktuella temperaturen är i huset, annars vet ju inte reglerenheten hur mycket effekt den ska ge ut till styrdonen. Till det används en eller flera temperaturgivare som kan vara placerade utomhus och/eller inomhus. Genom att temperaturgivarna sänder information om temperaturen till reglerenheten, så fås ett så kallat återkopplat reglersystem. (Thomas, 2016).

Ett reglersystem brukar oftast åskådliggöras med blockscheman. Ett blockschema för ett reglersystem kan se ut som schemat i figur 2:



Figur 2: Blockschema över ett reglersystem. (Thomas, 2016).

Till vänster i figur 2 kan man se börvärdet som är det värde som reglersystemets storhet förväntas och önskas ta. Börvärdet ställs in med en potentiometer eller en börvärdesgivare som det också kallas. Runda ringen efter potentiometerblocket kallas för en differenspunkt. Här jämförs börvärdet med det aktuella värdet för processens storhet, som också brukar kallas ärvärde. Skillnaden mellan börvärdet och ärvärdet kallas för

reglerfel eller regleravvikelse och betecknas oftast med e. Nästa block är regulatorn som har som uppgift att räkna ut en lämplig styrsignal, u, utgående från reglerfelet. Om reglerfelet är positivt så ökar regulatorn styrsignalen medan om reglerfelet är negativt så minskar regulatorn styrsignalen. Till höger om regulatorn är styrdonet som i detta blockschema är en värmespiral. Styrdonet tar spänningssignalen från regulatorn och konverterar den till värme, och på så sätt justeras värmen för processen. Värmen i processen mäts konstant med en temperaturgivare som skickar informationen till regulatorn. T_i symboliserar en störning här, alltså en storhet som på ett oönskat vis påverkar processen.

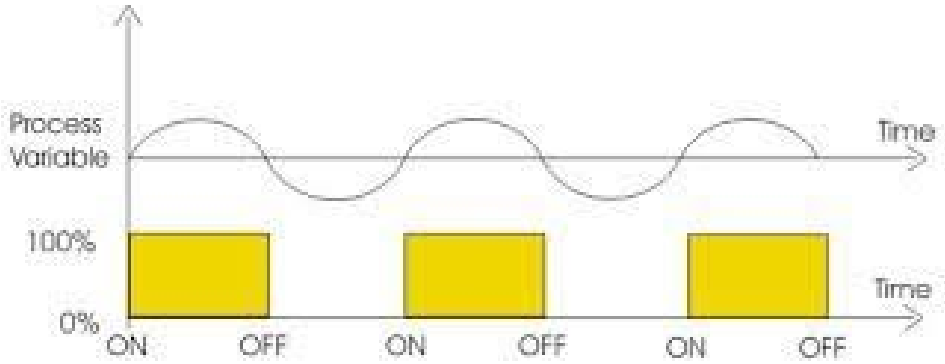
2.2 Processegenskaper

För inställning av regulatorer krävs att man har information om processtypen som regleras och dess egenskaper. Några egenskaper som en process kan ha är statisk förstärkning K, tidskonstant T och dödtid L. Den statiska förstärkningen berättar vad sambandet mellan utsignal och insignal för en process är. Alltså hur mycket kommer utsignalen att ändra ifall att insignalen till en process ändras. Tidskonstanten berättar hur snabbt utsignalen, för en process, kommer att uppnå sitt slutvärde. Tidskonstanten T definieras som den tid det tar för utsignalen av en process, att uppnå 63 % av sitt slutvärde. Dödtid är den tid det tar för utsignalen för en process, att över huvud taget börja ändra från sitt ursprungliga värde vid en ändring av insignalen. Dessa värden brukar användas för att ställa in regulatorer med hjälp av så kallade stegsvarsmetoder som kommer att tas upp i kapitel 2.4. Det finns också processer som saknar dessa egenskaper helt och hållet. Ett exempel är integrerande processer, som betyder att processens utsignal kommer att vara konstant med endast ett värde på insignalen. Om insignalen ändras från detta värde så kommer utsignalen att öka eller sjunka oupphörligen. Ett exempel på en sådan process är en tank, med inflöde och utflöde, där man vill hålla vätskan på en viss nivå. Ifall inflödet är större än utflödet så ökar nivån medan nivån sjunker ifall inflödet är mindre än utflödet. (Thomas, 2016).

2.3 Reglerprinciper

Thomas (2016) definierar begreppet reglerprincip som sambandet mellan in- och utsignal för en regulator och hur det påverkar ett systems egenskaper. Den enklaste av

reglerprinciperna är tvålägesreglering eller till/från-reglering, vilket betyder att styrsignalen från regulatorn endast kan anta två olika värden. Styrsignalen kommer att anta något av dessa värden beroende på om felet, som regulatorn får in är positivt eller negativt. Tvålägesreglering är väldigt enkel att konstruera men tillika är den mindre noggrann än andra regleringsmetoder och det uppstår oftast svängningar i reglersystemet, vilket kan förorsaka mekaniskt slitage på styrdonet. Tvålägesreglering används till exempel av strykjärn och ugnar.



Figur 3: Graf över hur tvålägesreglering kan se ut. Den nedre signalen är styrsignal och den övre signalen är ärvärde. (Sigvardsson, 2019).

Det finns också trelägesregulatorer men de är väldigt ovanliga. En trelägesregulator kan konstrueras med två värmeelement i stället för ett. På det viset får man tre lägen, alltså inget värmeelement påslaget, ett värmeelement påslaget eller två värmeelement påslagna. Desto vanligare regulatorer är P-, PI- och PID-regulatorer, där P står för proportionell, I för integrerande och D för deriverande. Regulatorerna brukar vara konstruerade på så vis att P-, I- och D-delarna finns i samma regulator och användaren får välja vilka av delarna som ska vara aktiverade. Mera information om P-, I- och D-delarna tas upp i underkapitlen 2.3.1 till 2.3.3.

2.3.1 Proportionella delen

Den proportionella delen i en regulator ger en styrsignal som är proportionell mot reglerfelet. Matematiskt uttryckt skulle styrsignalen som en funktion av reglerfelet se ut så här:

$$u(t) = u_0 + K * e(t)$$

(1)

där u(t) är styrsignalens värde vid tidpunkten t, u_0 är styrsignalens normalvärde, K är regulatorns förstärkning och e(t) är reglerfelet vid tiden t.

Styrsignalen är i verkligheten alltid begränsad, alltså det finns ett maximi- och ett minimivärde som styrsignalen inte kan överskrida respektive underskrida. Det här gör att styrsignalen endast kommer att vara proportionell mot reglerfelet i ett visst intervall som kallas det proportionella bandet. Enligt Thomas (2016) är regulatorns förstärkning K, ett värde på hur mycket regulatorn ska ta i för att rätta reglerfelet. Ju större förstärkning desto snabbare reglering, men en stor förstärkning ger också sämre stabilitet, tvärtom gäller för en liten förstärkning. Ett mycket stort värde på förstärkningen ger ofta upphov till att systemet blir i svängning, liksom fallet med tvålägesregulatorn. Det dåliga med P-reglering är att det oftast blir ett kvarstående fel mellan ärvärde och börvärde. Det kan korrigeras med hjälp av integrering som tas upp i nästa kapitel. Ett annat fel med P-reglering är att det inte går att få en reglering som har både god stabilitet och är snabb. För att åtgärda det används derivering som kommer att tas upp i kapitel 2.3.3

2.3.2 Integrerande delen

Den integrerande delen i en regulator ger en styrsignal som är integralen av reglerfelet över tid. Matematiska uttrycket för det ser ut på följande sätt:

$$u(t) = \frac{1}{T_I} \int_0^t \mathrm{e(t)\ dt}$$

(2)

där u(t) är styrsignalens värde vid tiden t, T_I är integreringstiden och e(t) är felet vid tiden t. Integreringstiden T_I bestämmer snabbheten för integreringen. Om T_I är stort blir hastigheten för integreringen långsammare medan om T_I är litet så blir integreringen snabbare. Då man ställer in en regulator gäller det att hitta ett lämpligt värde för T_I,

eftersom ett stort T_I ger en långsam integrering men bra stabilitet och ett litet T_I-värde ger snabb integrering men dålig stabilitet. Integrerande delen brukar oftast användas tillsammans med proportionella delen för att få bort det kvarstående felet som uppstår vid endast P-reglering. En sådan regulator brukar kallas för PI-regulator och det är den mest använda reglerprincipen. (Thomas, 2016).

2.3.3 Deriverande delen

Den deriverande delen för en regulator ger en styrsignal som är derivatan av reglerfelet vid tiden t. Det matematiska uttrycket för D-delen ser ut så här:

$$u(t) = T_D \frac{de(t)}{dt} \quad (3)$$

Där u(t) är styrsignalens värde i tidpunkten t, T_D är deriveringstiden och e(t) är reglerfelet vid tiden t. Deriveringstiden T_D bestämmer hur stort bidrag deriveringen ger till styrsignalen. Ju större T_D desto mera påverkas styrsignalen av D-delen. Derivering används aldrig ensam utan är endast till för att komplettera P- och I-regulatorer. Derivering ger alltid en snabbare reglering och kan i vissa fall också öka stabiliteten, men den kan också sänka stabiliteten ifall processen har en stor dödtid eller om det finns högfrekventa mätstörningar. Då ska man inte använda D-reglering. Thomas (2016) säger att D-delen påverkar styrsignalen med samma tecken som P-delen ifall att felet ökar. Det betyder att kortvariga störningar elimineras och medelvärdet för reglerfelet minskar. Ifall felet minskar så ger D-delen ett bidrag till styrsignalen som är motsatt tecken jämfört med P-delen. Det här betyder att översvängar som uppstår på grund av processers tröghet blir mindre. D-delen används tillsammans med P-delen för att bilda en PD-regulator eller vanligare tillsammans med P- och I-delen för att bilda en PID-regulator. Formeln för en PID-regulator ser ut på följande sätt:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^t e(t)dt + T_D \frac{de(t)}{dt}\right] \quad (4)$$

2.4 Metoder för inställning av regulatorn

Det finns olika metoder för hur regulatorer ska ställas in. Det handlar alltså om metoder för att komma fram till lämpliga värden på förstärkningen K, integreringstiden T_I och deriveringstiden T_D. Metoderna går ut på att utföra ett stegsvarstest för en process, att utföra ett självsvängningstest för en process eller att utföra både stegsvar- och självsvängningstest för processen.

2.4.1 Stegsvarsmetoder

Stegsvarsmetoder går ut på att stegsvarsexperiment utförs på en oreglerad process för att få fram egenskaperna förstärkning K, tidskonstant T och dödtid L för processen. Stegsvarsexperimentet betyder att insignalen tar ett steg uppåt eller neråt och sedan undersöks hur utsignalen påverkas av steget i insignalen. De vanligaste stegsvarsmetoderna som Hägglund (2019) pratar om i sin bok Praktisk processreglering är Ziegler-Nichols-stegsvarsmetod, AMIGO-metoden och Lambdametoden.

2.4.1.1 Ziegler-Nichols-stegsvarsmetod

Ziegler-Nichols-stegsvarsmetod är framtagen av John G. Ziegler och Nathaniel B. Nichols, som arbetade för Taylor Instruments på 1940-talet, då de gjorde simuleringar med pneumatiska analogimaskiner. Ziegler-Nichols-metoden ger oftast en alltför aggressiv reglering eftersom Ziegler och Nichols ansåg att en dämpning på en fjärdedel ger en bra reglering men inom processindustrin vill man oftast ha mera dämpning. Ziegler-Nichols-stegsvarsmetod bör ej användas ifall processens dödtid är mycket kortare än tidskonstanten eftersom metoden då ger en alltför stor förstärkning och för kort integreringstid. (Hägglund, 2019).

Ziegler-Nichols-stegsvarsmetod ger följande värden för K, T_I och T_D:

Tabell 1: Parametrar för Ziegler-Nichols-stegsvarsmetod.

Regulator	K	T_I	T_D
P	$\frac{T}{K_p L}$		
PI	$\frac{0{,}9T}{K_p L}$	$3L$	
$PID_{parallell}$	$\frac{1{,}2T}{K_p L}$	$2L$	$0{,}5L$

(Hägglund, 2019).

2.4.1.2 AMIGO-stegsvarsmetoden

AMIGO-metoden är framtagen ca 60 år efter Ziegler och Nichols metod, alltså i början av 2000-talet. AMIGO står för Approximate M-constrained Integral Gain Optimization och den blev framtagen genom att utföra simuleringar med hjälp av datorer. AMIGO-metoden är dock inte endast experimentell utan den bygger också på optimering, vilket innebär att ytan av reglerfelet vid en stegstörning minimeras. AMIGO-metoden har samma dåliga egenskap som Ziegler-Nichols-stegsvarsmetod, alltså att den ej ska användas ifall dödtiden är mycket kortare än tidskonstanten. (Hägglund, 2019).

AMIGO-metodens parametrar räknas ut med hjälp av nedanstående tabell:

Tabell 2: Parametrar för AMIGO-stegsvarsmetoden.

Regulator	K	T_I	T_D
PI	$\frac{T}{K_p}\left(0{,}15 + 0{,}35\frac{T}{L} - \frac{T^2}{(L+T)^2}\right)$	$0{,}35L + \frac{13LT^2}{T^2 + 12LT + 7L^2}$	
$PID_{parallell}$	$\frac{1}{K_p}\left(0{,}2 + 0{,}45\frac{T}{L}\right)$	$\frac{0{,}4L + 0{,}8T}{L + 0{,}1T}L$	$\frac{0{,}5LT}{0{,}3L + T}$

(Hägglund, 2019).

2.4.1.3 Lambdametoden

Lambdametoden skapades på 1960-talet och har varit använd inom pappersindustrin. Förutom att lambdametoden använder sig av K, L och T, så finns det också en parameter λ som användaren själv kan ställa in. λ-parametern inverkar på regleringens robusthet och

snabbhet på så sätt att ett litet λ ger aggressivare reglering medan ett större λ ger mera robust reglering. λ-parametern bör väljas som en faktor av tidskonstanten förutom om dödtiden är lång, då väljs den som en faktor av dödtiden i stället. Det är vanligt att välja λ = T men Teknik AB, SSG menar att man borde välja den enligt följande:

$$\lambda = \begin{cases} T & "Aggressiv\ reglering" \\ 2T & "Säker\ reglering" \\ 3T & "Robust\ reglering" \\ 3L & "Process\ med\ lång\ dödtid" \end{cases}$$

Lambdametoden har fördelen att den inte ger för hög förstärkning och kort integreringstid ifall att dödtiden är mycket kortare än tidskonstanten. (Hägglund, 2019).

Lambdametoden ger följande parametervärden:

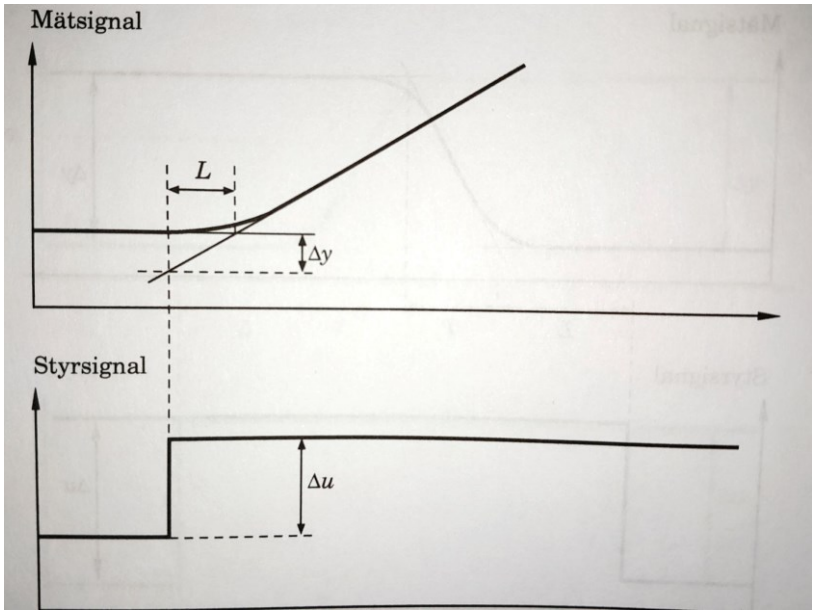
Tabell 3: Parametrar för Lambdametoden.

Regulator	K	T_I	T_D
PI	$\frac{T}{K_p(L+\lambda)}$	T	
$PID_{parallell}$	$\frac{\left(\frac{L}{2}+T\right)}{K_p\left(\frac{L}{2}+\lambda\right)}$	$T+\frac{L}{2}$	$\frac{TL}{L+2T}$

(Hägglund, 2019).

2.4.1.4 Stegsvarsmetoder för integrerande processer

Ziegler-Nichols- och AMIGO-stegsvarsmetoderna kan också användas på integrerande processer. Eftersom integrerande processer saknar tidskonstant och har en oändlig statisk förstärkning så används något som kallas hastighetsförstärkning i stället. Hastighetsförstärkningen betecknas K_v. Figur 4 visar hur dödtiden och hastighetsförstärkningen räknas ut för en integrerande process:



Figur 4: Stegsvar för en integrerande process. (Hägglund, 2019).

Hastighetsförstärkningen räknas ut på följande sätt:

$$K_v = \frac{\Delta y}{\Delta u L} \quad (5)$$

där y är ärvärdet, Δu är ändring i styrsignalen och L är dödtiden. Tabellerna för regulatorparametrarna för Ziegler-Nichols- och AMIGO-stegsvarsmetoderna konverterat för integrerande processer blir som följande:

Tabell 4: Parametrar för Ziegler-Nichols-stegsvarsmetod med hastighetsförstärkning.

Regulator	K	T_I	T_D
P	$\frac{1}{K_v L}$		
PI	$\frac{0{,}9}{K_v L}$	$3L$	
$PID_{parallell}$	$\frac{1{,}2}{K_v L}$	$2L$	$0{,}5L$

(Hägglund, 2019).

Tabell 5: Parametrar för AMIGO-stegsvarsmetoden med hastighetsförstärkning.

Regulator	K	T_I	T_D
PI	$\frac{0,35}{K_v L}$	$13,4L$	
$PID_{parallell}$	$\frac{0,45}{K_v L}$	$8L$	$0,5L$

(Hägglund, 2019).

2.4.2 Självsvängningsmetoder

Det finns en annan procedur för att ställa in regulatorer som baserar sig på att göra självsvängningsexperiment för en process. Denna metod tar oftast mera tid än stegsvarsexperiment och kan påfresta styrdonen ganska hårt, men processens parametrar kan bestämmas mera exakt. Metoderna går oftast ut på att öka förstärkningen för regulatorn tills processen hamnar i en så kallad självsvängning, vilket betyder att ärvärdet börjar harmoniskt oscillera kring börvärdet. Hägglund (2019) tar upp tre metoder som baserar sig på självsvängningstest, Ziegler-Nichols-självsvängningsmetod, relämetoden och AMIGO-självsvängningsmetoden.

2.4.2.1 Ziegler-Nichols-självsvängningsmetod

Ziegler och Nichols har också kommit upp med en självsvängningsmetod som görs på följande vis:

1. Sätt regulatorn i automatiskt läge med integral- och derivatadelarna bortkopplade.

2. Öka förstärkningen tills processen börjar självsvänga. Amplituderna för svängningen ska hållas konstant och inte avta eller öka.

3. Avläs förstärkningen då processen kommer i självsvängning. Detta kallas för den kritiska förstärkningen och betecknas K_c.

4. Mät periodtiden för självsvängningen. Detta kallas för den kritiska periodtiden och betecknas T_c.

Det finns vissa problem med metoden som till exempel ifall man väljer en större förstärkning än K_C, så kommer processen att bli instabil. Metoden lämpar sig inte heller för processer med lång dödtid. (Hägglund, 2019).

Ziegler-Nichols-självsvängningsmetod ger följande regulatorparametrar:

Tabell 6: Parametrar för Ziegler-Nichols-självsvängningsmetod.

Regulator	K	T_I	T_D
P	$0{,}5K_c$		
PI	$0{,}4K_c$	$0{,}8T_c$	
$PID_{parallell}$	$0{,}6K_c$	$0{,}5T_c$	$0{,}125T_c$

(Hägglund, 2019).

2.4.2.2 Relämetoden

Relämetoden är en liknande metod som Ziegler-Nichols-självsvängningsmetod men den saknar vissa nackdelar som Ziegler-Nichols-metoden har. Relämetoden kommer att ge upphov till en styrsignal som ser likadan ut som tvälägesregulatorns, se figur 2 i kapitel 2.3. Detta kommer att sätta processen i en svängning som har en periodtid som är väldigt nära den kritiska periodtiden T_c och den kritiska förstärkningen K_c kan räknas ut då man vet amplituderna för in- och utsignalerna. Relämetoden går ut på följande:

1. Sätt regulatorn i manuellt eller automatiskt läge och vänta tills processen är i balans.

2. Justera styrsignalens begränsningar så att de är några procent ovan och under det aktuella värdet för styrsignalen.

3. Lägg regulatorn i automatiskt läge och öka förstärkningen till ett väldigt högt värde, till exempel 100.

4. Ärvärdet börjar nu att oscillera harmoniskt kring börvärdet och amplituden Δy på utsignalen kan avläsas och periodtiden T_c kan räknas ut.

Den kritiska förstärkningen K_c räknas ut med följande formel:

$$K_c \approx \frac{4\Delta u}{\pi \Delta y} \quad (6)$$

där Δu är amplituden för styrsignalen och Δy är amplituden för ärvärdet. K_c och T_c kan sedan användas för att räkna ut regulatorparametrarna med hjälp av formlerna som Ziegler och Nichols kom fram till. Se tabell 6. (Hägglund, 2019).

2.4.2.3 AMIGO-självsvängningsmetoden

AMIGO-självsvängningsmetoden använder också kritiska förstärkningen K_c och kritiska periodtiden T_c, som kan räknas ut genom att göra någondera tester från kapitel 2.4.2.1 eller kapitel 2.4.2.2. Dessutom så använder AMIGO-metoden också processens statiska förstärkning som lättast fås fram genom att göra ett stegsvarstest. AMIGO-självsvängningsmetoden kan ej användas för integrerande processer. (Hägglund, 2019).

AMIGO-metoden ger följande formler för att räkna ut regulatorparametrarna:

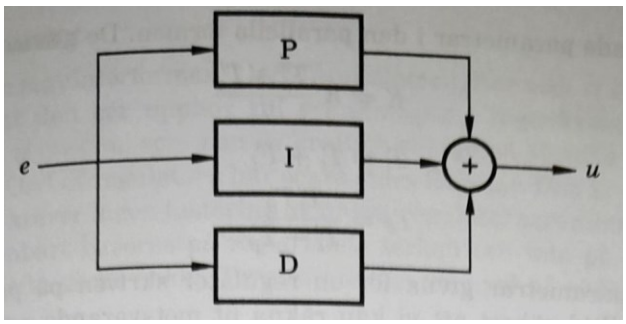
Tabell 7: Parametrar för AMIGO-självsvängningsmetoden.

Regulator	K	T_I	T_D
PI	$0{,}16K_c$	$\frac{K_p K_c T_c}{K_p K_c + 4{,}5}$	
$PID_{parallell}$	$\frac{\left(0{,}3\left(K_p K_c\right)^4 - 0{,}1\right) K_c}{\left(K_p K_C\right)^4}$	$\frac{0{,}6 K_p K_c T_c}{K_p K_c + 2}$	$\frac{0{,}15 T_c\left(K_p K_c - 1\right)}{K_p K_c - 0{,}95}$

(Hägglund, 2019).

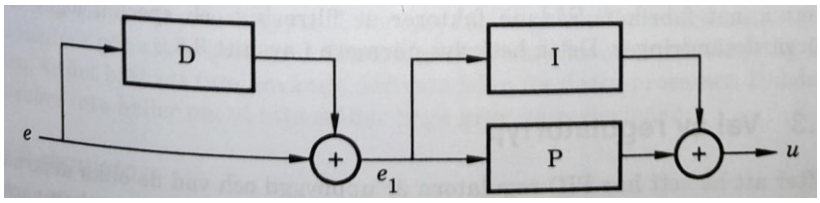
2.4.3 Seriell PID-regulator

PID-regulatorer kan vara seriella i stället för parallella, vilket ändrar alla ovannämnda uträkningsformler för PID-regulatorn. Den parallella PID-regulatorn kunde grafiskt beskrivas av figur 5:



Figur 5: Parallell PID-regulator. (Hägglund, 2019).

Den seriella PID-regulatorn kunde grafiskt se ut på följande sätt:



Figur 6: Seriell PID-regulator. (Hägglund, 2019).

För att konvertera formlerna för parallell PID till seriell kan följande formler användas:

$$K' = \frac{K}{2}\left(1 + \sqrt{1 - \frac{4T_D}{T_I}}\right) \quad (7)$$

$$T_I' = \frac{T_I}{2}\left(1 + \sqrt{1 - \frac{4T_D}{T_I}}\right) \quad (8)$$

$$T_D' = \frac{T_D}{2}\left(1 - \sqrt{1 - \frac{4T_D}{T_I}}\right) \quad (9)$$

där K', T_I' och T_D' är parametrarna för den seriella PID-regulatorn och K, T_I och T_D är parametrarna för den parallella PID-regulatorn. Det finns dock ett villkor för att konverteringen ska vara möjlig:

$$T_I \geq 4T_D \quad (10)$$

(Hägglund, 2019).

3 Bluetooth-kommunikation

Bluetooth® är en global standard som skapades på 1990-talet för att ersätta seriella datakablar. Bluetooth använder sig av trådlös kommunikation med hjälp av radiovågor. Användning av Bluetoothteknik är i dagens läge väldigt populärt eftersom Bluetooth är billigt, har låg effektförbrukning, lätt att använda, säkert, kompatibelt, med mera. Bluetooth används till exempel för att överföra filer mellan enheter, för att lyssna på musik, för utskrivning av dokument på skrivare eller för att prata i telefon med hands-free. I dagens läge installeras Bluetooth på alla mobiltelefoner, plattor och bärbara datorer som tillverkas. Bluetooth kan också definieras som ett WPAN (Wireless Personal Area Network) där PAN betyder att det är ett nätverk mellan personliga enheter som till exempel bärbara datorer och mobiltelefoner. (Gupta, 2013).

3.1 Historia

Tekniken som idag är känd som Bluetooth började utvecklas under 1990-talet av Telefonaktiebolaget LM Ericsson i Lund. Det var holländaren Jaap Haartsen och svensken Sven Mattison som fick uppdraget att utveckla specifikationen för Bluetooth och år 1997 hade de en fungerande lösning. År 1998 skapades Bluetooth SIG (Special Interest Group) av IBM och Ericsson med fem medlemsföretag: Ericsson, IBM, Intel, Nokia och Toshiba. Ett år senare lanserades första Bluetoothenheten på marknaden. Det var trådlösa hands-free-hörlurar och enheten vann Best of Show Technology Award-priset vid COMDEX. Idag finns det fem olika versioner av Bluetoothspecifikationen, 35 000 medlemmar i Bluetooth SIG och flera miljarder Bluetoothenheter tillverkas årligen. (Bluetooth, 2023).

3.2 Bluetooth Special Interest Group (SIG)

Bluetooth SIG är en privat förening som har som uppgift att publicera Bluetoothspecifikationer, administrera Bluetooth-kvalitetsprogram, skydda Bluetooth-varumärket och sprida Bluetoothtekniken. Företag har rätt att söka om medlemskap till Bluetooth SIG och medlemmar har rättigheter att använda Bluetoothspecifikationer, använda Bluetoothresurser och rätten att skapa produkter som använder sig av Bluetoothlicensen. Medlemskapet är gratis men det finns också ett avgiftsbelagt medlemskap, som ger snabbare tillgång till de nyaste Bluetoothspecifikationerna och rättigheten att medverka i utvecklingen av Bluetoothspecifikationer. Bluetooth SIG publicerar dessutom olika test för att testa Bluetoothprodukter och håller evenemang där medlemmar kan registrera och testa sina produkter. (Gupta, 2013).

3.3 Allmän teori om Bluetooth

Som redan blev nämnt i början av kapitel 3, är Bluetooth en standard för dataöverföring mellan enheter. Bluetooth har förutom datakanaler också kanaler för röstkommunikation, vilket betyder att ljud kan överföras från en enhet till ett par hörlurar. Bluetooth är ett så kallat ad hoc-nätverk, vilket betyder att Bluetooth inte är beroende av annan infrastruktur utan kan skapa förbindelser själv. Jämför till exempel med enheter som använder WiFi-nätverk som behöver routrar eller hotspots för att koppla enheter samman.

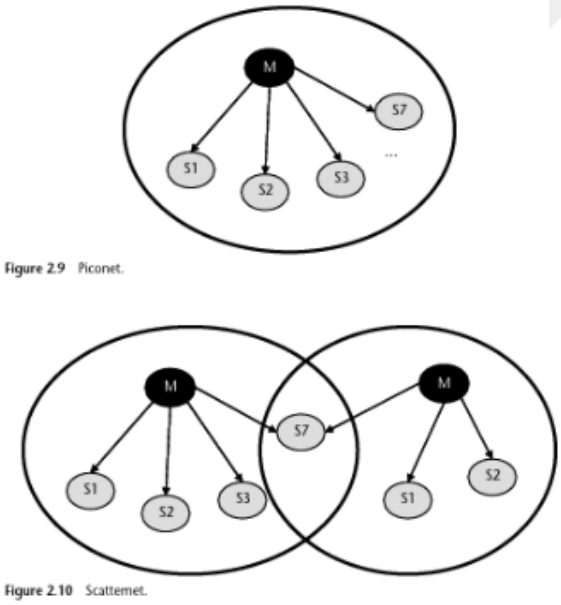
Bluetooth kan koppla enheter samman på ett avstånd som är max 100 meter men oftast används kortare sträckor. Bluetoothspecifikationen understöder olika effektnivåer för enheter, vilket gör att utvecklaren kan kombinera effektnivån med önskad räckvidd.

Dataöverföringshastigheten för Bluetooth 1.0 – 1.2 är max 721 kilobit per sekund (kbps) och detta kallas för Basic Rate (BR). För Bluetooth 2.0 – 2.1 är hastigheten max 2,1 Mbps och detta kallas för Enhanced Data Rate (EDR). Den snabbaste dataöverföringshastigheten 24 Mbps kom med Bluetooth version 3.0 och kallas för High Speed (HS). Efter Bluetooth 3.0 kom Bluetooth version 4.0 vilket medförde Bluetooth Low Energy (LE) som är ämnad för enheter som behöver ha en väldigt låg energiförbrukning, till exempel batteridrivna enheter. Bluetooth Low Energy-enheter brukar använda sig av lägre dataöverföringshastigheter och kortare maxräckvidd eftersom det drar ner energiförbrukningen. Bluetooth Low Energy kommer att introduceras närmare i nästa

kapitel. Den senaste Bluetoothspecifikationsversionen är Bluetooth 5.4 och den kom ut i februari 2023.

Bluetoothtekniken använder sig av det licensfria ISM-frekvensbandet på 2,4 GHz för kommunikation mellan enheter. Eftersom bandet är licensfritt så betyder det att också många andra tekniker använder bandet, vilket kan skapa interferens mellan Bluetoothenheter och enheter som använder andra tekniker. Mikrovågsugnar kan också skapa störningar för Bluetoothenheter. För att motverka interferensen så använder Bluetooth en teknik som heter frekvenshoppande spridningsspektrum (FHSS), vilket betyder att Bluetoothenheterna använder ett antal frekvenser nära 2,4 GHz och hoppar snabbt mellan dessa frekvenser. Frekvenserna är mellan 2400 – 2483,5 MHz.

Bluetooth använder sig av nätverkstopologierna piconet och scatternet. Bluetoothtekniken grupperar enheter som master-enheter och slav-enheter. En master-enhet kan vara uppkopplad till sju slav-enheter på samma gång. Detta bildar ett så kallat piconet. Ett scatternet är flera piconet som är kopplade till varandra och då kan alltså en enhet kommunicera med flera än sju enheter på samma gång. (Gupta, 2013).



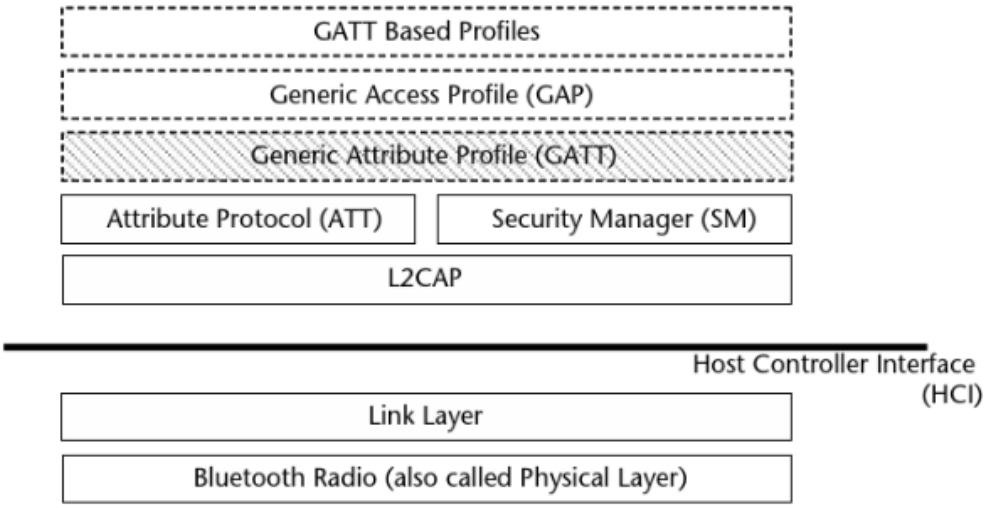
Figur 7: Piconet och scatternet. (Gupta, 2013).

3.4 Bluetooth Low Energy och Generic Attribute Profile (GATT)

Eftersom enheterna som användes för att göra examensarbetesprojektet använder sig av Bluetooth Low Energy (LE), så kommer dokumentet att ta upp mera information om

Bluetooth Low Energy. Bluetooth Low Energy kom med specifikationsversionen 4.0 och är ämnad för enheter som kräver låg energiförbrukning. Enheterna kan förväntas klara sig med små knappbatterier i flera månader och enheterna kan vara smarta armbandsur, stegmätare, pulsmätare, och liknande. LE-enheter har kort räckvidd och max dataöverföringshastighet på 305 kbps men brukar dock oftast använda mycket lägre dataöverföringshastighet. Mobiltelefoner och bärbara datorer brukar stöda både Bluetooth Classic (BR/EDR) och Bluetooth Low Energy (LE). (Gupta, 2013).

Protokollstacken för Bluetooth Low Energy-enheter ser ut som i figur 8:



Figur 8: Protokollstacken för Bluetooth Low Energy. (Gupta, 2013).

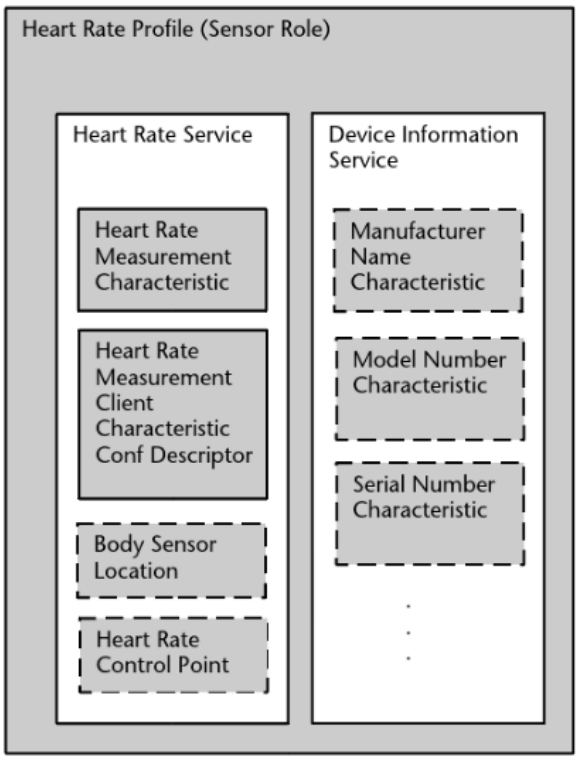
Lägst nere i figur 8 kan de lägsta lagren i protokollstacken ses, som alltså bestämmer fysiska egenskaperna för Bluetooth Low Energy. Det kan vara saker som vilken frekvens som används och moduleringen som används för att överföra informationen. Detta dokument kommer inte att ta upp desto mera information om de lägre lagren, i stället kommer Generic Attribute Profile-lagret (GATT) att presenteras noggrannare. Detta eftersom GATT-lagret spelade en stor roll för examensarbetet, då applikationerna som skapades handlar om just detta lager och hur man definierar olika objekt i lagret.

GATT-lagret är ett lager som tar attribut från ATT-protokollagret och strukturerar dessa attribut hierarkiskt genom att gruppera attributen med hjälp av profiler, tjänster (service) och kanaler (characteristic). Läs mera om ATT-protokollagret i Guptas bok Inside Bluetooth Low Energy (Gupta, 2013). GATT-lagret definierar också kanalernas egenskaper

och rättigheter, alltså om de kan läsas, skrivas, avisera eller indikera. GATT-lagret krävs för Bluetooth Low Energy-enheter men kan också implementeras på Bluetooth Classic (BR/EDR). En Bluetooth Low Energy-enhet kan vara endera en klient eller en server och det är alltid servern som skapar tjänsterna och kanalerna för GATT-lagret. Profilerna, tjänsterna och kanalerna som tillhör GATT-lagret presenteras närmare i nedanstående underkapitel. (Gupta, 2013).

3.4.1 Profil (Profile)

En profil i GATT-lagret är ett block som innehåller tjänster och kanaler som behövs för funktionaliteten som profilen ger till enheten och applikationen. En profil kan till exempel vara en profil för en pulsmätare med tjänster och kanaler som behövs för pulsmätare. En grafisk presentation av detta ses i figur 9:

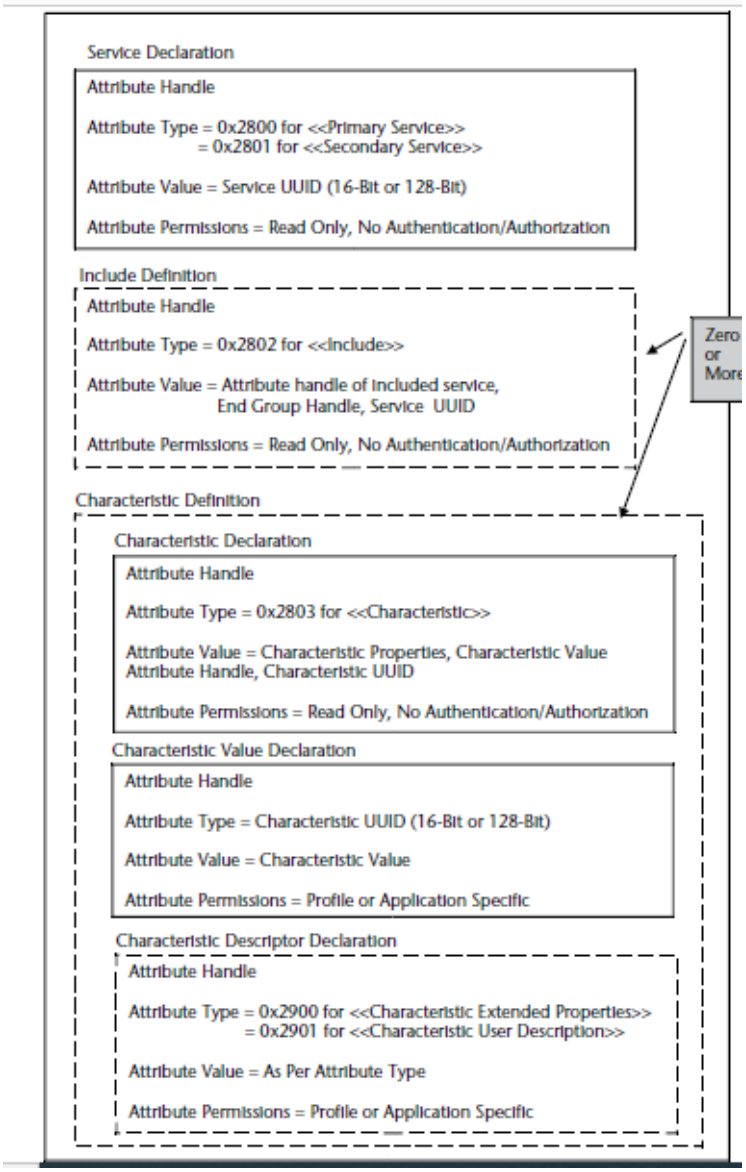


Figur 9: GATT-profil för en pulsmätare. (Gupta, 2013).

Den här profilen innehåller två tjänster och tjänsterna innehåller fyra respektive tre kanaler. Kanalerna som är i rutor med streckade konturer är villkorliga och allt annat är obligatoriskt.

3.4.2 Tjänst (Service)

En tjänst är en datastruktur i GATT-lagret som kan innehålla en eller flera kanaler och tjänsten berättar om vad profilen har för funktionalitet. Tjänsten kan vara endera primär eller sekundär tjänst, där den primära tjänsten berättar om en primär funktionalitet för profilen och den sekundära är en extra tjänst för profilen som inte är lika viktig. Tjänsten måste alltid definieras med en tjänstdeklaration som har tre obligatoriska attribut. Attributen är attributtyp, som är definierad som en universell unik identifierare (UUID), som är ett 16- eller 128-bitars nummer. Attributtypen berättar om det är en primär eller sekundär tjänst för profilen. Det andra obligatoriska attributet är attributvärde som också är en UUID och det tredje attributet är attributrättigheter för tjänsten som är endast läsbar (read-only), ingen autentisering (no authenication) och inget tillstånd (no authorization). Andra definitioner som tjänsten kan ha är inkluderingsdefinition som används för att referera till andra tjänster och kanaldefinitioner för kanaler som tjänsten innehåller. Se figur 10 för hur en tjänstdefinition kan se ut:



Figur 10: Tjänstdefinition som innehåller en kanal med kanaldefinition. (Gupta, 2013).

3.4.3 Kanal (Characteristic)

Liksom tjänsten så ska också kanalerna som hör till tjänsten ha kanaldefinitioner. De obligatoriska definitionerna är kanaldeklaration och kanalvärdesdeklaration. Kanaldeklarationen har samma attribut som tjänstdeklarationen, alltså attributtyp som är kanalens UUID, attributvärde och attributrättigheter som är samma som för tjänsten. Attributvärdet innehåller tre olika saker som är kanalegenskaper, alltså om värdet kan bli skrivet, läst, aviserat eller indikerat, värdets handtag (handle) som innehåller själva värdet och en UUID som berättar om vilken typs värde kanalen innehåller. Kanalvärdesdeklarationen består av attributtyp med samma UUID som kanaldeklarationen, attributvärde som är kanalens värde och attributrättigheter som här

sätts av ett högre lager eller implementeras av skaparen. Dessutom kan kanalen ha en tredje villkorlig deklaration som är en beskrivningsdeklaration (descriptor). Igen är attributen samma alltså typ, värde och rättigheter. Attributtypen är en UUID för beskrivningen, attributvärdet är beskrivningsvärdet och rättigheterna sätts av ett högre lager eller implementeras av skaparen. En beskrivning kan vara en sträng som berättar vad det finns för värde i kanalen, till exempel ”puls” ifall det är värdet för en pulsmätning. (Gupta, 2013).

4 Praktisk del

Examensarbetesprojektet innehåller två olika praktiska delar. En PID-regulatorapplikation för Androidenheter där man kan ställa in, styra och övervaka regulatorn och processen som styrs och en applikation på en Arduino-mikrokontroller som fungerar som regulator. Dessutom skapades ett litet kretskort där Arduino-mikrokontrollern och hårdvaran för signalanpassningen är. Androidenheten med applikationen är kopplad till Arduinon via Bluetoothkommunikation. Dessa två delar blev alltså en helhet för att styra en process. Dokumentet presenterar Android- och Arduino-applikationerna i underkapitlen 4.1 respektive 4.2 och information om det tillverkade kretskortet och signalanpassningen tas upp i kapitel 4.3.

Tester blev också utförda för att undersöka hur bra PID-regulatorn fungerade. Processen som användes var en nivåregleringsprocess som finns i Technobotnia. De utförda testerna, testresultat och en diskussion kring resultatet kommer att tas upp i kapitel 4.4.

4.1 Androidapplikationen

Som redan tidigare nämndes konstruerades en PID-regleringsapplikation för Androidenheter. Applikationen kan ställa in, styra och övervaka en reglerad process genom att skicka och motta data från regulatorn som är implementerad på en Arduino. Applikationen blev skapad i en integrerad utvecklingsmiljö (IDE) som heter Android Studio och som Androidenhet för applikationsutvecklingen användes mobiltelefonen Samsung Galaxy A13 med Androidversion 13. Mobiltelefonen har en ganska låg prestanda vilket är positivt då applikationen testas eftersom det är meningen att applikationen ska fungera på möjligast många enheter. Om applikationen blev testad på en High end-enhet så fås ingen bekräftelse om hur applikationen fungerar på en enhet med lägre prestanda.

4.1.1 Kort om Android Studio

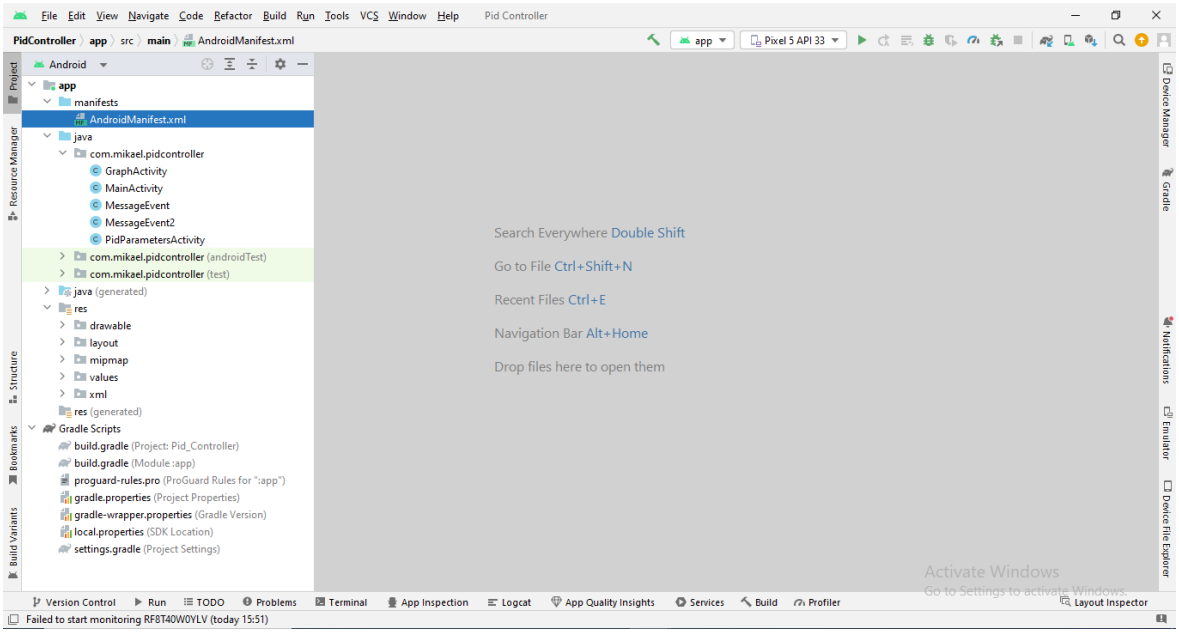
Android Studio är en integrerad utvecklingsmiljö som är skapad av Google LLC och ämnad för att skapa applikationer för Androidenheter. Android Studio släpptes första gången 8 december 2014 och utvecklas fortfarande. I Android Studio kan applikationsprojekt programmeras i endera Java eller Kotlin. Denna applikation är utvecklad i Java eftersom

den är baserad på andra projekt som också är skrivna i Java. Läs mera om de andra projekten i kapitel 4.1.5.

Applikationen blev skapad med hjälp av nyaste versionen av Android Studio som heter Electric Eel. Android Studio kan laddas ned gratis från deras hemsida:

https://developer.android.com/studio

Då ett projekt öppnas i Android Studio kommer det att se ut som figur 11:



Figur 11: Öppnat projekt i Android Studio.

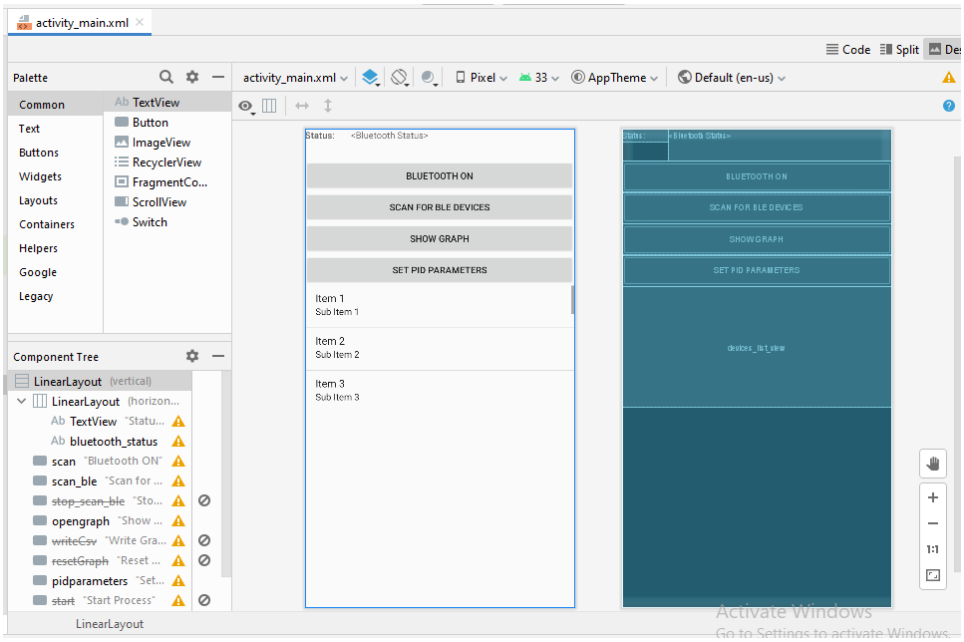
Till vänster i figur 11 ses projektstrukturen för det öppnade Androidprojektet, där alla filer som tillhör projektet finns. Dokumentet kommer kort att ta upp de viktigaste som man behöver veta om dessa filer för att komma i gång med ett Androidprojekt.

Högst uppe ser vi AndroidManifest.xml. Det är en fil som krävs i varje Androidapplikation och namnet för filen måste vara exakt AndroidManifest.xml, filnamnet får alltså inte ändras. AndroidManifest.xml-filen skickar information om applikationen till Android byggverktyg (buildtools), Android operationssystemet och Google Play. AndroidManifest.xml-filen innehåller olika definitioner, bland annat rättigheter som applikationen kräver, tema för applikationen, alltså hur applikationen ser ut och alla komponenter för applikationen till exempel applikationens aktiviteter. Se Android

Manifest.xml i bilaga 1 för ett exempel på hur en AndroidManifest.xml-fil kan se ut. (App Manifest Overview, 2023).

Under katalogen java finns projektets javaklasser och aktiviteter. I dessa finns kod som exekveras till exempel då applikationen öppnas eller användaren trycker på olika knappar på skärmen.

Under katalogen res finns olika resurser, alltså hjälpfiler där olika variabler kan definieras, som till exempel färger, strängar och vyer. Den kanske viktigaste resursen som används i så gott som alla Androidapplikationer är layout. Under layout-katalogen finns layout.xml-filer som innehåller de komponenter som visas på skärmen. Vyerna kan göras endera genom att skapa komponenterna med hjälp av xml-kod eller genom att grafiskt placera komponenterna i Android Studios designverktyg. Se figur 12 för ett exempel på hur en layoutfil kan se ut:



Figur 12: Layoutfil öppnad i Android Studios designverktyg.

En sak som ännu kan nämnas är byggverktyget Gradle som Android Studio använder sig av. I Gradle Scripts-filerna som kan ses nere till vänster i figur 11 kan man sätta in olika inställningar för sitt projekt, som till exempel sdk-version, alltså vilken version av Android som kan köra applikationen eller beroendeinställningar (dependencies). I examensarbetesprojektet sattes 'org.greenrobot:eventbus:3.0.0' och 'com.jjoe64:graphview:4.2.2' till under beroendeinställningar i build.gradle, för att

applikationen ska kunna använda sig av en eventbuskomponent och en grafkomponent. Varje gång som Gradlefilerna på något sätt ändras, så måste projektet synkroniseras igen. Men Android Studio kommer att meddela om detta genom en inforuta där det står ”Sync Now”. (Configure your build, 2023)

Uppe i mitten i figur 11 finns Android Studios Navigation Bar. Här finns några verktyg som till nästa presenteras i korthet. Längst till vänster ses en grön hammare. Genom att trycka på den kompileras projektet. Efter den finns en ruta där det står Pixel 5 API 33. Det är namnet på enheten som applikationen kommer att laddas upp till ifall att kör-knappen trycks in. Enheten kan vara en verklig enhet som till exempel en mobiltelefon som är kopplad via en USB-kabel till en dator med Android Studio, eller så kan det vara en emulator som finns inbyggd i Android Studio. Kör-knappen är till höger om rutan. Med den knappen laddas applikationen upp till en kopplad enhet. Till höger om kör-knappen finns det ännu en viktig knapp som användes väldigt flitigt vid utvecklingen av projektet. Det är knappen som ser ut som en insekt med en liten pil i högra nedre kanten. Den här knappen heter Attach debugger to Android process och den ansluter alltså Android Studios debugger, felsökningsprogram, till en öppnad process. Den används på så vis att applikationen först öppnas på Androidenheten och sedan ansluts debuggern till applikationen då knappen trycks. Det här gör att brytpunkter, breakpoints, kan sättas i koden för att felsöka den. (Developers Android Guides, 2023).

4.1.2 Att ansluta en Androidenhet till en dator med Android Studio

Att ansluta en Androidenhet till en dator som kör Android Studio är inte så enkelt som det låter. För det första måste enheten sättas i utvecklarläge. Det görs lite olika beroende av enhet och vilken version av Android som körs. Samsung Galaxy A13 med Androidversion 13 som användes i applikationsutvecklingen sattes i utvecklarläge genom att gå till Inställningar > Om telefonen > Programvaruinformation > Tryck Kompileringsnr 7 gånger och sedan Utvecklaralternativ > aktivera USB-felsökning.

Efter att enheten är i utvecklarläge så måste Google USB-drivrutiner ännu installeras på datorn. Det gäller endast för datorer med Windows-operativsystem. Datorer med Mac OS X eller Linux kan skippa det steget. Lättaste sättet att installera drivrutinerna är att gå i Android Studio till Tools > SDK Manager > SDK Tools > kryssa i Google USB Driver > Apply > OK > Finish. Efter det här borde det gå att ansluta enheten till Android Studio via en

USB-kabel och då namnet på enheten syns i rutan, som nämndes i förra stycket, så har Android Studio identifierat enheten och sedan är det bara att ladda upp projektet till enheten.

4.1.3 Aktiviteter

Varje Androidapplikation är uppbyggd av så kallade aktiviteter. En aktivitet är egentligen bara en java- eller kotlinklass som ärver en av Androids standard aktivitetsklasser som till exempel AppCompatActivity. Den aktivitet som öppnas då en applikation startas brukar kallas för huvudaktivitet och det är den enda aktiviteten som alltid krävs. Från huvudaktiviteten kan sedan andra aktiviteter startas. Huvudaktiviteten ska definieras på ett speciellt sätt i AndroidManifest.xml-filen. Förutom att den ska ha en <activity>-tagg med namnet på aktiviteten, så behöver den också ha exported=true, vilket gör den synlig för andra applikationer och så behöver den också ha en <intent-filter>-tagg som ska innehålla taggarna <action android:name="android.intent.action.MAIN" /> och <category android:name="android.intent.category.LAUNCHER" />. Aktiviteter som inte är huvudaktiviteter behöver endast aktivitetstaggen med namnet för aktiviteten. Ett exempel på hur aktiviteter definieras i AndroidManifest.xml-filen:

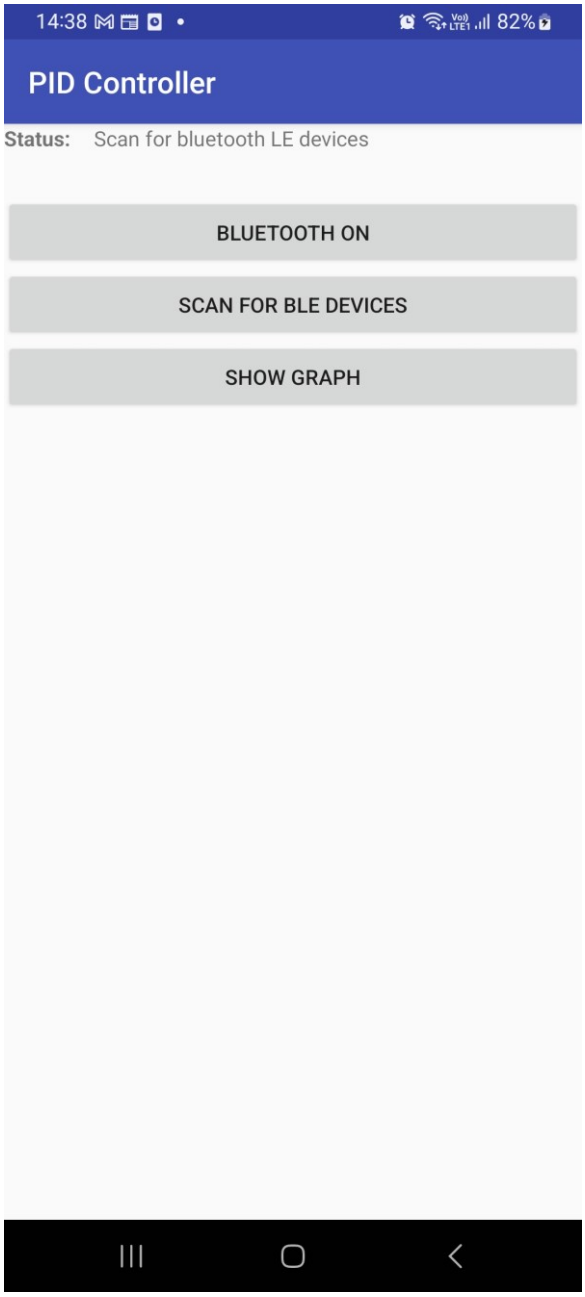
Kodexempel 1: Aktivitetsdefinitionerna i AndroidManifest.xml-filen för PID-regulatorapplikationen.

```
<activity android:name="com.mikael.pidcontroller.MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name="com.mikael.pidcontroller.GraphActivity"
    android:exported="true"
    android:screenOrientation="landscape"></activity>
<activity android:name="com.mikael.pidcontroller.PidParametersActivity">
</activity>
<activity android:name="com.mikael.pidcontroller.AboutApp"></activity>
```

PID-regulatorapplikationen har tre olika aktiviteter och tre vanliga Javaklasser. Till nästa tas aktiviteternas utseende och funktion upp.

4.1.3.1 Huvudaktivitet

Huvudaktiviteten för applikationen heter MainActivity och det är den aktivitet som öppnas då användaren startar applikationen. Huvudaktiviteten ser ut på följande sätt då applikationen startas upp:



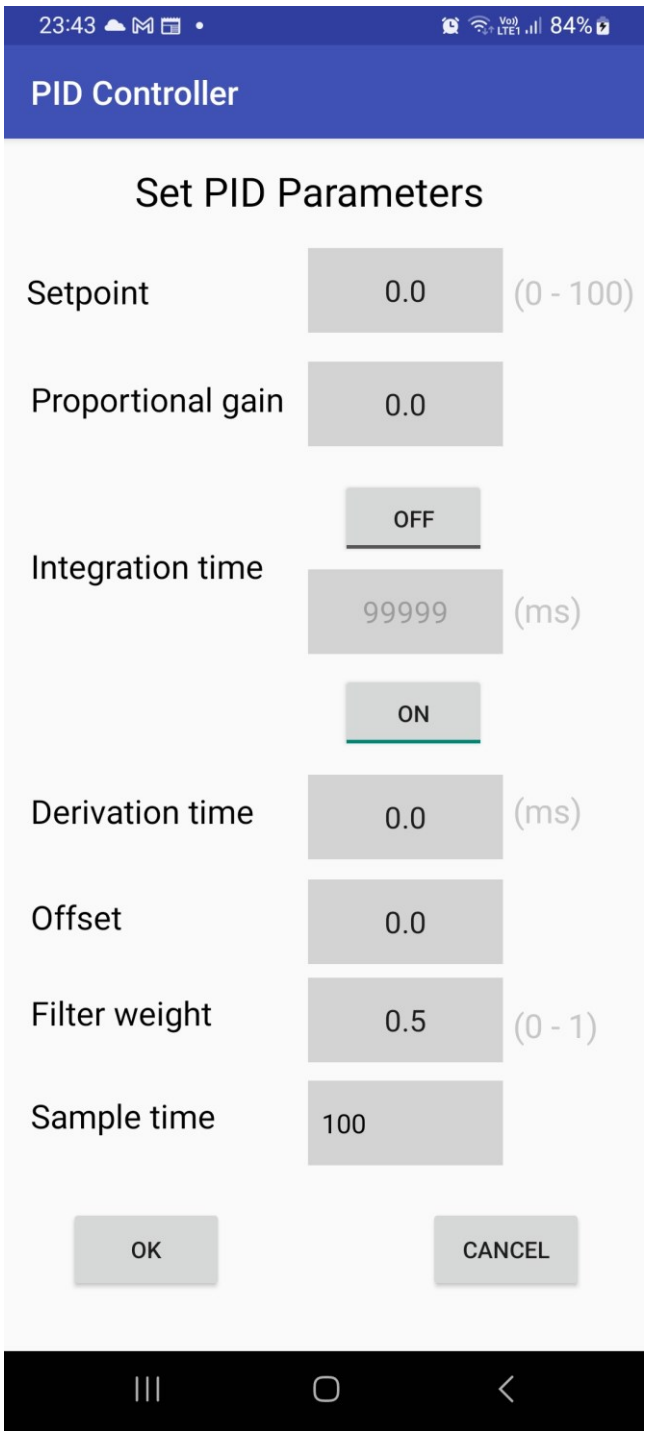
Figur 13: Huvudaktiviteten då applikationen startas.

Överst i figur 13 ses namnet på applikationen som är PID Controller, efter det så finns en statustext som är ämnad för att ge information till användaren för att göra det lättare att använda applikationen. Under statustexten hittas en BLUETOOTH ON-knapp. Med den knappen kan Bluetooth aktiveras på enheten ifall den inte redan är aktiverad. Att aktivera Bluetooth från en applikation kräver alltid tillåtelse från användaren, så om användaren trycker på den knappen så fås en förfrågan om att applikationen skulle vilja slå på Bluetooth, som då måste tillåtas. Efter den kommer en till knapp som heter SCAN FOR BLE DEVICES. Om användaren trycker på den knappen så börjar enheten söka efter andra närliggande enheter som använder Bluetooth Low Energy, tillika som knappen byter text till STOP SCAN FOR BLE DEVICES. Efter att applikationen hittar andra enheter kommer de att sättas till i en lista som börjar efter SHOW GRAPH-knappen.

Om användaren trycker på knappen STOP SCAN FOR BLE DEVICES så kommer sökandet att avslutas och de hittade enheterna kommer att visas i listan. Ifall användaren inte stoppar sökandet så kommer det att stoppas automatiskt efter tio sekunder, men det rekommenderas att stoppa sökandet så snabbt som den rätta enheten hittas, eftersom Bluetoothsökning är väldigt energikrävande. Användaren kan sedan trycka på den enhet i listan som hen vill koppla applikationen till. Efter att applikationen är kopplad till enheten kommer en annan knapp, SET PID PARAMETERS, att dyka upp. Den knappen öppnar en aktivitet som heter PidParametersActivity och den introduceras i kapitel 4.1.3.2. Under sökningsknappen finns en knapp som heter SHOW GRAPH. Den knappen öppnar en ny aktivitet som heter GraphActivity och den presenteras i kapitel 4.1.3.3.

4.1.3.2 PID-parameteraktivitet

PID-parameteraktiviteten är en aktivitet där användaren kan sätta in önskade värden för sin PID-regulator. PID-parameteraktiviteten startas genom att trycka på SET PID PARAMETERS i huvudaktiviteten. Aktiviteten heter PidParametersActivity och ser ut på följande sätt:



Figur 14: PID-parameteraktivitet för att ställa in PID-parametrar.

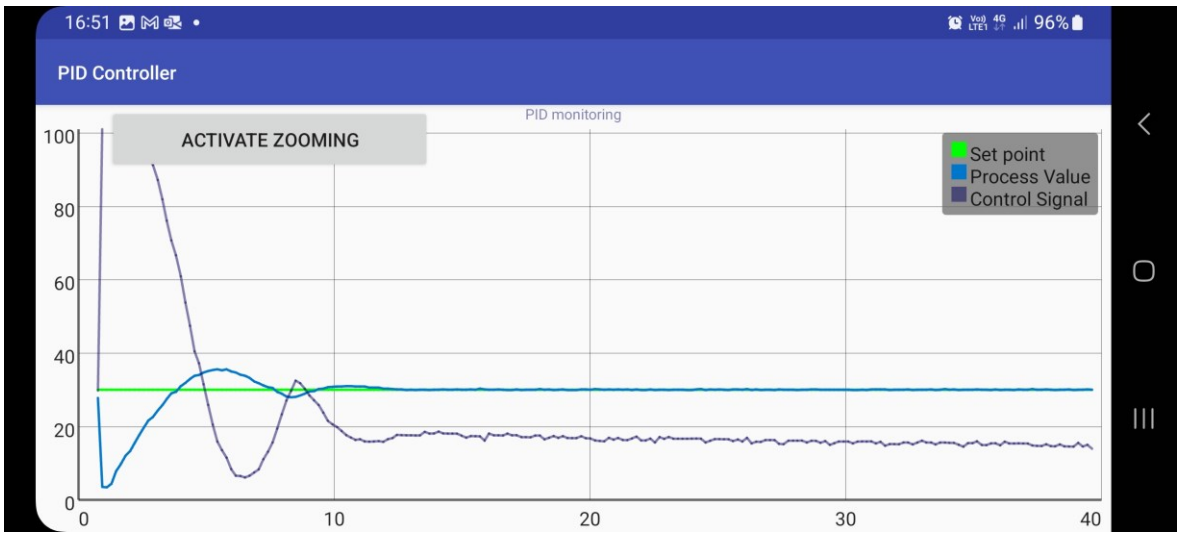
Högst uppe i PID-parameteraktiviteten i figur 14 finns en instruktion till användaren, Set PID Parameters, så att hen vet att här ska man sätta in parametrar för PID-regulatorn. Första parametern som kan sättas in är börvärdet (set point) och då användaren trycker på textrutan brevid Setpoint-texten, så kommer ett tangentbord med siffror att öppnas. Med hjälp av tangentbordet kan användaren sätta in ett önskat börvärde som ska vara ett reellt tal i procent mellan 0 och 100. Nästa textruta fungerar på exakt samma sätt. Här

sätter användaren in den proportionella förstärkningen för regulatorn. Till nästa finns en På/Av-knapp som styr om regulatorn ska ha integrering aktiverad eller inte. Genom att användaren trycker på den knappen kommer integreringen att aktiveras och användaren kan sätta in önskad integrationstid i textrutan under knappen. Deriveringsparametern fungerar på samma sätt förutom att när man aktiverar deriveringen så kommer en filterruta fram där man kan ställa in en filtreringsvikt som anger hur mycket ärvärdessignalen filtreras. Integrations- och deriveringstiderna ska sättas in i millisekunder.

Efter det finns en textruta för offset ifall att användaren vill ha en offset på kontrollsignalen. Sedan i rutan under offseten eller filtreringsvikten kan användaren ställa in samplingstid för regulatorn. Samplingstiden finns i en rullgardinsmeny där användaren har möjligheten att välja mellan värdena 100, 200, 300, 400 eller 500 millisekunder. Om allt är inställt på önskat sätt kan användaren trycka på OK-knappen, som stänger PID-parameteraktiviteten och skickar värdena till huvudaktiviteten som igen öppnas. Ifall att användaren vill tillbaka till huvudaktiviteten utan att ställa in eller ändra parametrar så kan hen trycka på CANCEL-knappen eller bakåtknappen.

4.1.3.3 Grafaktivitet

Grafaktiviteten heter GraphActivity och den öppnas genom att trycka på SHOW GRAPH-knappen i huvudaktiviteten. Grafaktiviteten är den aktivitet som plottar börvärde, ärvärde och kontrollsignal i en graf. Processen kan alltså övervakas och resultat kan fås via grafaktiviteten. Om användaren har satt i gång processen så kommer bör-, ärvärde och kontrollsignal att plottas i grafen i realtid och sedan när användaren har stoppat processen, så kan hen ännu undersöka de plottade värdena i en statisk graf. Grafen öppnas alltid i liggande format och kan till exempel se ut på följande sätt:



Figur 15: Applikationsgraf för en reglerad process.

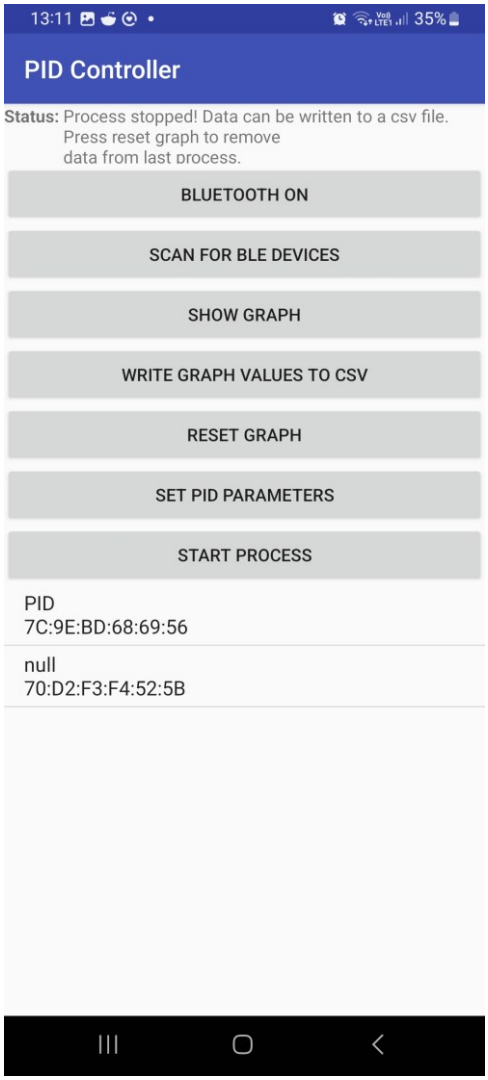
I grafaktiviteten finns förutom grafkomponenten också en knapp där användaren kan byta läge mellan zoomläge och kursorläge. Standardläget är kursorläge och det betyder att en kursor kommer att dyka upp ifall användaren trycker på ett ställe på grafen. Bredvid kursorn kommer en liten textruta med information om vad bör-, ärvärde och kontrollsignalen har för värde i den punkten som kursorn är placerad på. Ifall att användaren byter till zoomläge, så kan användaren zooma in eller ut tidsaxeln med hjälp av två finger och kniptekniken. X-axeln är i tid och sekunder medan y-axeln är en procent av maxvärde för börvärdet.

4.1.3.4 Huvudaktivitetens andra lägen

Huvudaktiviteten kan befinna sig i olika lägen beroende på om användaren just har startat applikationen, om PID-parametrarna är inställda eller om processen är stoppad. Ifall användaren precis har startat applikationen så kommer huvudaktiviteten att vara i det läge som kan ses i figur 13 under kapitel 4.1.3.1 medan om användaren nyss har satt in PID-parametrar så kommer huvudaktiviteten att få till en knapp, START PROCESS under SET PID PARAMETERS-knappen. Med den knappen kan användaren skicka de insatta parametrarna till regulatorn, så att processen startas. Efter att processen blir startad kommer en annan knapp STOP PROCESS att dyka upp under startknappen. Ifall att användaren sedan trycker på stoppknappen, så kommer regulatorn att stoppa processen och två nya knappar dyker upp i huvudaktiviteten, WRITE GRAPH VALUES TO CSV och RESET GRAPH.

Med WRITE GRAPH VALUES TO CSV kan de värden som nu finns i grafen skrivas till en csv-fil som sedan kan undersökas närmare i till exempel Excel. Csv-filen kommer att hamna i Intern lagring\Documents\PID_data\graphData.csv på enheten som skrev csv-filen. Observera också att csv-filen blir ersatt av en ny fil då användaren igen trycker på WRITE GRAPH VALUES TO CSV.

RESET GRAPH-knappen kommer att radera allt som finns i grafen och på så sätt kan man börja nästa test från noll. Då användaren trycker på resetknappen kommer en varningsruta att dyka upp som frågar om användaren är säker på att hen vill radera alla värden från grafen. Användaren raderar allt genom att trycka OK eller kan ångra sig och trycka AVBRYT. Då en körning pågår kan användaren också skicka in nya värden till regulatorn genom att öppna PID-parameteraktiviteten, ställa in de nya värdena och trycka på startknappen igen. Till exempel kan en stegändring av börvärdet göras på det viset.



Figur 15: Huvudaktivitetens utseende efter en stoppad process.

4.1.4 Nedladdning av applikationen och applikationsrättigheter

Applikationen kan laddas ned från GitHub genom att följa den här länken. Applikationen kan endera laddas ned som vanligt, jämför Google Play, genom att trycka på app-release.apk på GitHub-sidan eller genom att ladda ned hela projektet, importera det i Android Studio och ladda upp applikationen till en enhet därifrån. Vissa tillåtelser kan också behöva godkännas då applikationen laddas ned från GitHub och installeras men enheten borde säga till om det.

Efter att applikationen är nedladdad och installerad på önskad enhet kan applikationen öppnas på normalt sätt. Som sagt så kommer huvudaktiviteten att öppnas upp då applikationen startas. Applikationen kommer att behöva rättigheter för åtkomst till enhetens exakta plats och rättigheten att hitta, ansluta till och fastställa relativ position för andra enheter i närheten, som är en rättighet för Bluetooth. Bluetoothrättigheten kommer att begäras då användaren trycker på BLUETOOTH ON-knappen om Bluetooth inte är aktiverad. Ifall att Bluetooth redan var aktiverad före applikationen startades kommer rättigheten inte att begäras i detta skede och en inforuta kommer att säga att Bluetooth redan är aktiverad. Ifall att användaren nekar tillståndet för Bluetoothrättigheten kommer en inforuta att säga att applikationen kräver rättigheten för att aktivera Bluetooth och rättigheten kommer att begäras igen då användaren trycker på BLUETOOTH ON igen. Ifall att användaren igen nekar rättigheten, så kommer den inte att begäras flera gånger utan Android antar att det betyder att användaren inte vill ge tillstånd till rättigheten för denna applikation. Om användaren ändå vill använda applikationen måste hen gå till applikationsinställningar och manuellt tillåta rättigheten eller återinstallera applikationen. Rättigheterna för applikationer hittas på olika ställen beroende på enhet och Androidversionen för enheten. För Samsung Galaxy A13 med Androidversion 13 hittades rättigheterna under Inställningar > Appar > PID Controller > Behörigheter.

Rättigheten för exakt plats kommer att begäras då användaren trycker på SCAN FOR BLE DEVICES och samma regler gäller också för den rättigheten. Ifall att Bluetooth redan var aktiverad då applikationen startades, så kommer Bluetoothrättigheten att begäras direkt efter rättigheten för plats. Efter att applikationen har fått rättigheterna kan man använda den enligt instruktionerna i kapitlen 4.1.3.1 till 4.1.3.4.

4.1.5 Projekt som applikationen är baserad på

Applikationen är baserad på ett antal andra projekt som hittades på GitHub. Kod och struktur har använts direkt från vissa projekt medan andra projekt endast användes för att få information och lärdomar över hur något specifikt ska programmeras i Android. Till nästa kommer vissa av de andra projekten att introduceras i korthet. Alla projekt som applikationen baserades på är skrivna i Java och länkar till dessa projekt kan hittas i källförteckningen i slutet av detta dokument. Dessa projekt är gjorda av kunniga kodare och kan ge god information om hur Androidapplikationer kan skapas. Projekten kan också laddas ned eller klonas från GitHub och undersökas i Android Studio.

4.1.5.1 Android Simple Bluetooth Example

Android Simple Bluetooth Example är ett Androidprojekt som är gjort av Justin Bauer och som egentligen är menad för enheter som använder sig av Bluetooth Classic. PID-applikationen använder utseendet och strukturen från Bauers projekt och metoden bluetoothOn(), som används för att aktivera Bluetooth i PID-applikationen, är lånad från detta projekt. (Bauer, 2016).

4.1.5.2 Android BLE Connect Example

Android BLE Connect Example är ett Androidprojekt som är gjort av Joel Wasserman och handlar om att koppla ihop enheter som använder sig av Bluetooth Low Energy. PID-applikationen använder stora delar av Wassermans projekt för att hantera Bluetoothkopplingen mellan applikationen och mikrokontrollern, och Wassermans projekt har bra information om hur man ska tänka när man programmerar Bluetooth Low Energy i Android. Största delarna av BluetoothGattCallback-objektet med metoder är kopierade från Wassermans projekt. Se bilaga 1. (Wasserman, 2016).

4.1.5.3 Android GraphView

Android GaphView är en grafkomponent för Androidapplikationer som är gjord av Jonas Gehring. Gehrings projekt är tillsatt i Gradle beroendeinställningar enligt instruktionerna från Gehrings projektsida i Github (Gehring, 2013). Man ska alltså sätta med raden ”implementation 'com.jjoe64:graphview:4.2.2'” i gradle.build-filens dependencies-block i Android Studio för att kunna använda GraphView som en layoutkomponent. Med Android GraphView kan olika grafer skapas, till exempel linjegrafer eller stapeldiagram, och det

finns också många olika funktionaliteter som att zooma i grafen, sätta in en kursor eller en teckenförklaring. (Gehring, 2013) .

4.1.5.4 Android Greenrobot EventBus

Det sista projektet som presenteras är Android Greenrobot EventBus som är skapad av Markus Junginger. Den ska sättas till på samma sätt som Android GraphView genom att sätta till ”implementation 'org.greenrobot:eventbus:3.0.0'” i gradle.build-filens dependencies-block. Eventbussen är en hjälpkomponent som kan sköta om kommunikation mellan olika komponenter i ett Androidprojekt. Det går att implementera en eller flera sändare, publisher, som kan posta händelser, events, till Eventbussen och så kan implementerade mottagare, subscriber, motta händelsen, som sändaren postade, från Eventbussen. (Junginger, 2021).

Med hjälp av Eventbussen skickar PID-applikationen data mellan de olika aktiviteterna. Som exempel mottar huvudaktiviteten kontrollsignalen och ärvärdet från regulatorn via Bluetooth och postar värdena till grafaktiviteten via Eventbussen. Se Jungingers GitHub-sida eller bilaga 1 i detta dokument för att förstå hur sändare, mottagare och postandet av händelser till Eventbussen kan implementeras.

4.2 Implementering av PID-regulatorn på mikrokontrollern

PID-regulatorn för examensarbetesprojektet är som sagt implementerad på en mikrokontroller. Mikrokontrollern som användes var en Arduino Nano 33 Iot med en inbyggd Bluetoothmodul. Kontrollern är alltså menad för att användas i ”Sakernas internet”-baserade applikationer men kan också användas för Bluetoothapplikationer. Nano 33 Iot har processorn M0 32-bit SAMD21 tillverkad av ARM® Cortex® och använder sig av en 48 MHz klockfrekvens. Processorn har 256 kB SRAM primärminne och 1 MB flash sekundärminne. Kontrollerns radiochip är en Nina W102 ublox-modul och kan användas som Bluetooth- eller WiFi-radio, dock inte båda tillika. Nina-modulen har eget inbyggt minne som är 448 kB ROM, 520 kB SRAM och 2 MB flash. Kontrollern har dessutom en inbyggd gyroskop och accelerometer som baserar sig på ett LSM6DS3-chip. Det finns 14 digitala in-/utgångar och av dessa kan 5 användas med pulsbreddsmodulering, 8 analoga utgångar och man kan skapa externa avbrott på alla in-/utgångar. Spänningsnivån på in-/utgångarna är 3,3 V men det finns också en utgång som matar ut en konstant spänning

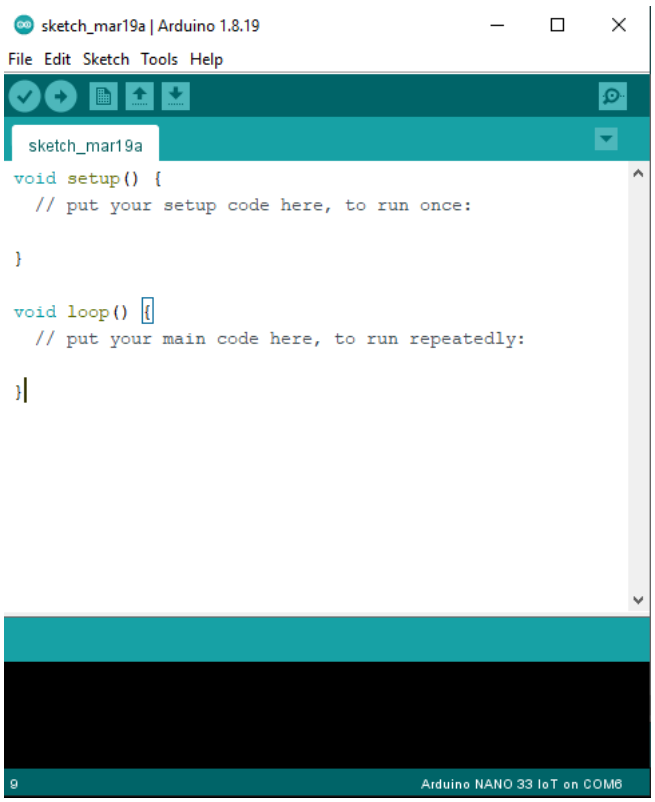
på 5 V. Kontrollern kan strömförses med en 5 – 18 V spänning till en av pinnarna, Vin, eller så kan den också ta sin ström från en kopplad microUSB-kabel. Via USB-kabeln kan man också ladda upp ny kod till kontrollern. Mikrokontrollern är väldigt liten till storleken, endast 18 mm x 45 mm och har en väldigt låg effektförbrukning på 7 mA per I/O-pinne. I nästa kapitel introduceras Arduinos integrerade utvecklingsmiljö (IDE) som användes för att programmera PID-regulatorn. (Nano 33 Iot, 2023).

4.2.1 Kort om Arduino IDE

Arduino IDE är en integrerad utvecklingsmiljö för att skriva Arduinokod och ladda upp den till en Arduinomikrokontroller. Arduinokod baserar sig på C- och C++-kod med vissa extra inbyggda funktioner, som till exempel digitalWrite() som sätter en utgång på Arduinokortet hög eller låg. Arduino IDE kan laddas ned gratis på Arduinos hemsida:

https://www.arduino.cc/en/software

Då Arduino IDE är nerladdat och en applikation är öppnad så kommer det att se ut som i figur 16:



Figur 16: Tom sketch öppnad i Arduino IDE.

Det är alltså en texteditor med lite extrafunktionalitet. När Arduino IDE öppnas så kommer det redan att finnas två block med kod som krävs för alla Arduinoprojekt. Det första kodblocket är en setupkod som kommer att köras endast en gång då koden laddas upp till mikrokontrollern. Det andra blocket är en loop som körs om och om igen. I setupkoden kan man till exempel skriva en utgång hög som önskas vara hög i början. Det svarta fältet som ses under koden är ett meddelandefält. Hit kommer olika meddelanden, som till exempel felmeddelanden eller att koden har blivit uppladdad till mikrokontrollern. Vid en uppladdning kommer meddelandefältet också att berätta hur mycket av mikrokontrollerns minne som används av koden som laddas upp.

Lägre ned till höger kan Arduinokortet som används ses och vilken serieport som det är kopplat till. Ovanför koden finns en knapp med ett bocktecken och ifall användaren trycker på den knappen, så kommer IDE:n att kompilera koden och kolla om det finns fel. Ifall koden är felfri så fås meddelandet Done compiling och minnesanvändningen syns i meddelandefältet, medan om det finns fel så fås ett meddelande om detta och vad som felet kan vara. Knappen till höger om bocktecknet är en knapp som kompilerar och laddar upp koden till en uppkopplad mikrokontroller.

Ovanför dessa knappar finns menyknapparna File, Edit, Sketch, Tools och Help. File-knappen har ganska samma funktionalitet som andra applikationer, alltså spara, öppna, och så vidare. Utöver det finns det en ganska trevlig funktionalitet Examples, som öppnar olika färdiga kodexempel. Edit-knappen har också helt normal funktionalitet. Under Sketch-knappen hittas funktionalitet som har något med sketchen att göra, alltså koden som finns i det vita fältet. Här hittas bland annat kompilering och uppladdning av koden eller inkludering av bibliotek.

Genom att öppna Tools-menyn kan koden till exempel formateras så att den får en snyggare struktur. Arduinokortet samt porten som den är kopplad till sätts också till under Tools-menyn och dessutom finns här en mycket användbar funktionalitet som kallas för Serial Monitor. Med Serial Monitor kan meddelanden skickas från koden till en ruta som öppnas om användaren trycker på Serial Monitor. Monitorn måste startas i setupkoden i sketchen genom att skriva Serial.begin(BAUD_RATE);, där BAUD_RATE är hastigheten som monitorn ska ha. Det är ganska normalt att välja en baud på 9600 bitar per sekund. Sedan kan ett meddelande skrivas till monitorn med hjälp av kommandot

Serial.print(”Meddelande”); eller Serial.println(”Meddelande”); om man vill byta rad efter meddelandet. Under Tools-knappen finns också Manage Libraries…, där man kan importera nya bibliotek till projektet. Help-menyn har normal funktionalitet. (Overview of Arduino IDE 1, 2023).

Arduino har också en editor för webbläsare men detta dokument tar inte upp någon information om den. Intresserade läsare kan läsa mera om Arduino WebEditor här:

https://docs.arduino.cc/arduino-cloud/getting-started/getting-started-web-editor

4.2.1.1 Installering av SAMD21 core på Arduino IDE

Som sagt så användes en Arduino Nano 33 Iot-mikrokontroller som PID-regulator. Mikokontrollrar med SAMD21 core-processorn kan ej direkt hittas i Arduino IDE, utan några extra steg måste tas för att hitta dem. Det krävs att SAMD 21 core installeras i IDE:n som görs på följande sätt: Gå till Tools > Board > Board Manager. Skriv sedan samd i sökfältet som har öppnats. Ett fält med rubriken Arduino SAMD-Boards borde nu visas och om muspekaren förs till detta fält så borde en installera-knapp dyka upp. Tryck på den! Efter att installationen är klar borde det gå att välja Arduino Nano 33 Iot under Tools > Board > Arduino SAMD Boards > Arduino Nano 33 Iot. Instruktionerna hittas också på Arduinos hemsida. (Installing the SAMD21 core for MKR boards, 2023).

4.2.2 Arduino PID-projektet

För att få Bluetooth Low Energy-funktionalitet till Arduinoprojektet så inkluderades biblioteket ArduinoBLE.h och en tjänst med tio kanaler blev skapad. Se kapitel 3.4 för att veta vad en tjänst och en kanal är. Åtta av kanalerna är av typen float, och de innehåller alltså flyttal medan två kanaler är av typen int och innehåller heltal. Fem kanaler har läs- och skrivrättigheter och de resterande har läs-, skriv- och aviseringsrättigheter. De kanaler som har aviseringsrättigheter har också beskrivningsdefinitioner eftersom detta krävdes av Androidapplikationen. 128-bitars universella unika identifierare skapades för alla objekt genom att generera dem från en sida på internetet. Sidan med UUID-generatorn kan hittas här.

Några inställningar sattes till i setupkoden för att starta Bluetoothmodulen, göra den synlig och för att skapa tjänsten. För att starta Bluetoothmodulen användes kommandot

BLE.begin(), för att sätta namnet som andra Bluetoothenheter ser, så användes BLE.setLocalName(”name”), sedan skulle tjänsten göras synlig med kommandot BLE.setAdvertisedService(serviceName), sedan sattes tjänsten till i Bluetoothapplikationen med hjälp av BLE.addService(serviceName) och till sist gjordes enheten synlig för andra enheter med BLE.advertise(). Beskrivningarna skulle också sättas till i kanalerna med kommandot characteristicName.addDescriptor(descriptorName) och sedan skulle kanalerna sättas till i tjänsten med serviceName.addCharacteristic(characteristicName).

I loopkoden sattes BLEDevice central = BLE.central() till, som är Bluetoothenheten som Arduinon är kopplad till, som vid utvecklingen av applikationen var en Samsung Galaxy A13. För att kolla om enheterna var kopplade kunde man göra en if-sats och kontrollera värdet för central på följande sätt: if (central). Central är sann om enheterna är kopplade och falsk annars. Koden LED användes som underlag för att programmera Arduinoapplikationen. Koden hittas under File > Examples > ArduinoBLE > Peripheral > LED och det krävs att ArduinoBLE-biblitoteket har blivit importerat för att koden ska finnas. Mera information och funktioner om ArduinoBLE hittas genom att följa länken i källförteckningen. (ArduinoBLE, 2023).

4.2.2.1 Läsandet av Bluetoothdata

Läsandet av en kanals värde i koden var enkel och gjordes med kommandot characteristicName.readValue(buffer, valueSize). Det här gjorde att kanalens aktuella värde sparades i variabeln buffer som var av datatypen bytearray. ValueSize valdes till fyra för floatkanalerna och två för intkanalerna eftersom Arduinos float är 32-bitar eller fyra bytes och int är 16-bitar eller två bytes i storlek. Efter läsningen sparades fyra eller två siffror i variabeln som måste konverteras till flyttal eller heltal. Mera om det i kapitel 4.2.2.3. Mera info om att läsa Bluetoothdata i Arduino hittas här:

https://www.arduino.cc/reference/en/libraries/arduinoble/blecharacteristic.readvalue/

4.2.2.2 Skrivandet av Bluetoothdata

Det var ännu enklare att skriva ett värde till en kanal i Arduino och det gjordes med kommandot characteristicName.writeValue(value), där characteristicName är kanalen som önskas skrivas och value är värdet som skrivs till kanalen. Detta kommando returnerar ett boolskt värde som är sant ifall skrivningen lyckades. Arduinoapplikationen

implementerades på så vis att den kollar det boolska värdet och skriver igen ifall det inte lyckades på första gången, eftersom Androidenheten ibland är långsam på att ta emot data, vilket gör att skrivningen misslyckas. Mera info om att skriva Bluetoothdata i Arduino hittas här:

https://www.arduino.cc/reference/en/libraries/arduinoble/blecharacteristic.writevalue/

4.2.2.3 Konvertering av Bluetoothdata till flyttal och heltal

För att konvertera en bytearray till ett flyttal i Arduino kan man göra på följande sätt:

```
float value = *(float *) &buffer;
```

, där buffer är bytearray-variabeln och value är en floatvariabel som talet sparas i. Det finns så klart också andra sätt att göra det på men detta var ett väldigt enkelt sätt. Det fanns ännu en sak som måste göras och det var att ändra ordningen på elementen i bytearrayen. Detta behövdes eftersom Android sparar flyttal i little-endian-format medan Arduino-mikrokontrollern vill ha dem i big-endian-format. Ordningen ändrades genom att spara talen i tvärtom ordningsföljd i en annan bytearray-variabel.

Konverteringen från bytearray till heltal gjordes på ett annat sätt men det var lika lätt. Det gjordes så att det första elementet skiftades 8 steg till vänster och sedan adderades det andra elementet till det skiftade värdet. Detta görs så här:

```
int value = (buffer[0] << 8) + buffer[1];
```

, där buffer[0] och buffer[1] är talen i bytearrayen.

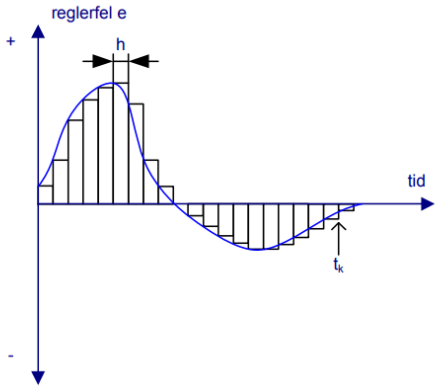
4.2.2.4 Process för att räkna ut kontrollsignalen

För att räkna ut kontrollsignalen användes PID-funktionerna som dokumentet tog upp i kapitel 2.3.1 – 2.3.3. Funktionerna behöver dock först ändras till en diskret form eftersom mikrokontrollern arbetar med diskreta värden. Styrsignalen för P-delen blir som den är, alltså

$$u(t) = u_0 + K * e(t) \quad (11)$$

där u(t) är styrsignalen vid tiden t, u_0 är styrsignalens normalvärde som kan sättas in som en offset i Androidapplikationen, K är förstärkningen och e(t) är reglerfelet vid tiden t.

Styrsignalen för I-delen måste approximeras eftersom det inte går att analytiskt integrera en diskret funktion. Ett bra sätt att approximera integralen är att summera ihop rektangelareor av reglerfelet som en funktion av tid och multiplicera summan med samplingstiden.



Figur 17: Approximering av integralen av reglerfelet.

I figuren ovan är den blåa linjen reglerfelsfunktionen e(t) och h är samplingstiden. Styrsignalen för I-delen blir då:

$$u(t) = \frac{K}{T_I} \sum_{t=0}^{t} e(t) * h \qquad (12)$$

där u(t) är styrsignalens värde vid tiden t, e(t) är felets värde vid tiden t och h är samplingstiden. Summan betyder att man tar en summa av alla reglerfel från det första felet, e(0) till det aktuella felet, e(t), alltså e(0) + e(h) + e(2*h) + … + e(t). (Wahlfrid, 2007).

Styrsignalen för D-delen måste också approximeras eftersom det inte går att derivera en diskret funktion. Derivatan approximeras lättast genom att ta differensen mellan felets aktuella värde och felets värde i föregående samplet och dela det med samplingstiden. (Wahlfrid, 2007).

Styrsignalen för D-delen blir då:

$$u(t) = K\,T_D\,\frac{e(t) - e(t-h)}{h}$$

(13)

där u(t) är styrsignalens värde vid tiden t, K är förstärkningen, T_D är deriveringstiden, e(t) är felet vid tiden t och h är samplingstiden.

För en PID-regulator ger det här styrsignalen på diskret form:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\sum_{t=0}^{t} e(t) * h + T_D\,\frac{e(t) - e(t-h)}{h}\right]$$

(14)

där u(t) är styrsignalens värde vid tiden t, K är förstärkningen, T_I är integreringstiden, T_D är deriveringstiden, h är samplingstiden och e(t) är reglerfelet vid tiden t.

Regulatorn implementerades som en parallell regulator, alltså så att P-, I- och D-delarna räknas ut parallellt och adderas sedan ihop för att bilda styrsignalen. Regulatorn implementerades också med en villkorlig integrering, vilket betyder att I-delen endast räknas ut då styrsignalen är innanför sina gränser.

4.2.2.5 Implementerade mjukvarufilter

Regulatorn har två filter implementerade i Arduinokoden för att få mindre svängningar i signalerna och på det viset en bättre reglering. Det ena filtret är ett glidande medelvärdesfilter och det är implementerat på regulatorns alla delar och det andra filtret använder sig av enklaste formen av exponentiell utjämning och är implementerat endast på derivatadelen. Filtrering hjälper till att få en stabilare reglering men tillika så gör det regleringen långsammare. De båda implementerade filtren filtrerar insignalen till PID-regulatorn, alltså processvärdet.

Glidande medelvärdesfiltret räknar ut ett medelvärde på alla värden fram till det aktuella värdet. Som sagt är filtret på processvärdet och det är gjort genom att läsa processvärdet konstant, addera läsningarna till en summa och vid uträkningen av styrsignalen dela

summan av processvärdena med antalet läsningar, för att få ett medelvärde av alla processvärdena fram till det aktuella värdet. Det kan se ut på följande sätt:

Kodexempel 2: Det glidande medelvärdesfiltret implementerat i Arduino.

```
float summa = 0, processvärde = 0;
int räknare = 0, samplingstid = 100;
void loop( ) {
   summa += analogRead(A0);
   räknare++;
   if(millis( ) % samplingstid == 0) {
      processvärde = summa/räknare;
      summa = 0;
      räknare = 0;
   }
}
```

analogRead(A0) är en Arduinofunktion som läser värdet för en spänningssignal på ingång A0 och millis( ) är en annan Arduinofunktion som räknar förfluten tid i millisekunder sedan Arduinoapplikationen startade.

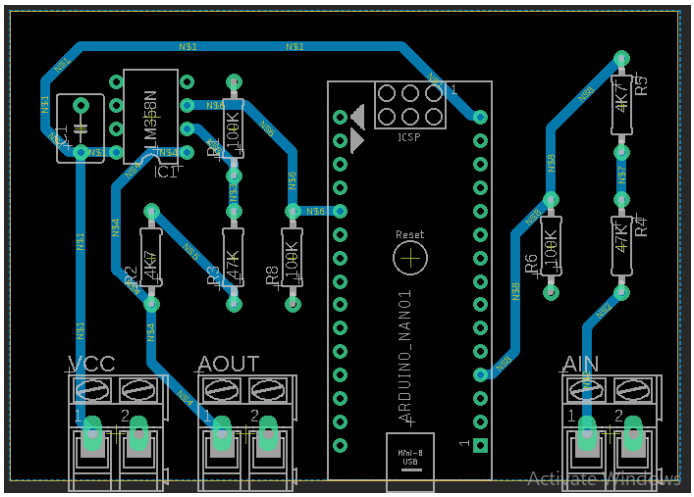
Det andra filtret är som sagt ett filter som använder sig av exponentiell utjämning och det filtret har en filtreringsvikt som användaren kan ställa in enligt tycke. D-delen använder båda filtren på så vis att processvärdet först blir medelvärdesfiltrerat och sedan sätts exponentialutjämningen på det filtrerade värdet. Ett matematiskt uttryck för den exponentiella utjämningen ser ut på följande sätt:

$$s_0 = x_0$$
$$s_t = \alpha x_t + (1-\alpha)s_{t-1}, \qquad 0 \leq \alpha \leq 1, \qquad t > 0 \qquad \textbf{(15)}$$

där s_0 är det filtrerade värdet vid tiden t = 0, x_0 är det icke-filtrerade värdet vid tiden t = 0, s_t är det filtrerade värdet vid tiden t, α är filtreringsvikten och x_t är det icke-filtrerade värdet vid tiden t. Ju mindre filtervikten α är desto mera filtrering. Se bilaga 2 för att veta hur filtret är byggt i Arduinokoden. (Exponential smoothing, 2023).

4.3 Hårdvara för signalanpassning

Signalanpassningen för PID-regulatorn blev mycket simpel. De komponenter som används är sju motstånd, två kondensatorer, en operationsförstärkare och tre tvåpoliga kontakter. Dessa komponenter är tillsammans med Arduinon satta på ett kretskort som blev fräst i Technobotnia. Kretskortslayouten blev ritad i verktyget EAGLE och ser ut som figuren nedan:



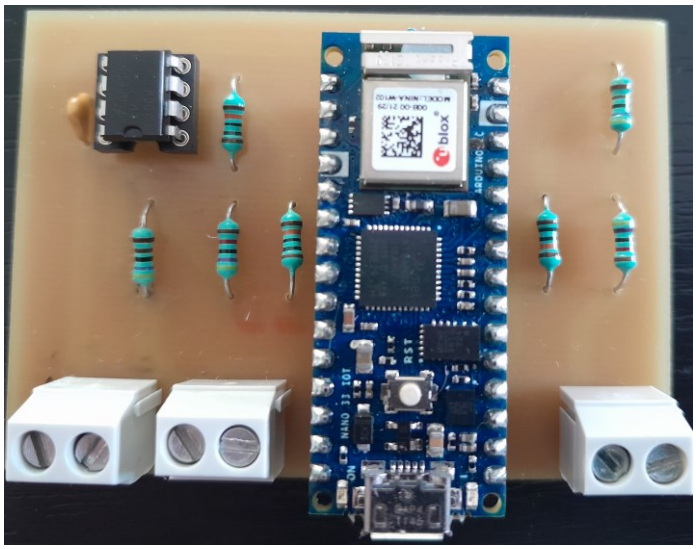
Figur 18: Kretskortsdesign skapad med verktyget EAGLE.

På nedre kanten kan kontakterna ses och börjandes från vänster finns kontakt för matningsspänning, VCC, kontakt för utsignal från PID-regulatorn, AOUT, som styrdonet kopplas till, Arduinons mikro-USB-kontakt och kontakt för insignal till PID-regulatorn, AIN, som ärvärdessignalen kopplas till. Alla kontakter har dessutom en pol för jord men endast en av kontakterna behöver kopplas till jord. Till vänster bredvid Arduinon kan operationsförstärkaren ses. Den är kopplad som en icke-inverterande operationsförstärkare och har som uppgift att förstärka 3,3 V utsignalen från mikrokontrollern till en 5 V signal. Den förstärker också strömmen som kan matas till styrsignalen, vilket är bra eftersom Arduinon bara klarar av att ge ut ca 20 mA. Bredvid utsignalen från mikrokontrollern finns också ett så kallat pull-down-motstånd som drar ner utsignalen från mikrokontrollern till jord då pinnen inte skrivs till. Detta behövs eftersom signalen annars börjar flyta över 0 V, vilket skulle ge en oönskad signal från operationsförstärkaren. Till höger ses en enkel spänningsdelare som har som uppgift att sänka 5 V processvärdessignalen till en 3,3 V signal.

I ett senare skede av projektets gång blev ännu en kondensator fastsatt vid AOUT för att filtrera pwm-signalen som mikrokontrollern skickar ut. Den fungerar som ett lågpassfilter tillsammans med motstånden som redan fanns på kortet. Kondensatorn valdes till ett lämpligt värde så att gränsfrekvensen blev några gånger mindre än pwm-signalens frekvens och större än frekvensen för styrsignalen. Pwm-signalfrekvensen var 15 kHz och styrsignalens frekvens var 10 Hz eller mindre eftersom den som snabbast behöver ändra värde med 100 millisekunders intervall. Ett passligt värde på kondensatorn var 10 nF vilket gav en gränsfrekvens på cirka 306 Hz. Gränsfrekvensen räknades ut på följande sätt:

$$f_{gräns} = \frac{1}{2\pi RC} = \frac{1}{2\pi * 52k\Omega * 10nF} = 306{,}7\ Hz \quad (16)$$

Kretskortet blev i verkligheten att se ut som i figur 19:



Figur 19: Tillverkat kretskort med Arduinomikrokontrollern och annan hårdvara.

4.4 Testandet av PID-regulatorn

PID-regulatorn blev testad med hjälp av en nivåregleringsprocess som finns i Technobotnia. Processen är en integrerande process, så testerna är gjorda med självsvängningsexperiment och Ziegler-Nichols-metoden blev använd för att få fram parametrar.

4.4.1 Beskrivning av processen som ska regleras

Processen som ska regleras är en nivåregleringsprocess i Technobotnia som är byggd av en tank eller ett rör med ett vatteninflöde och -utflöde. Vattnet tillförs i tanken med hjälp av en pump som styrs av ett styrsystem som regulatorn kopplades till. Styrsystemet har en spänningsnivå på 0 – 5 V, vilket alltså betyder att vätskenivån ges som en signal mellan 0 V och 5 V och likaså tar systemet in en signal, som styr pumpen, som är mellan 0 V och 5 V. Processen är som sagt en integrerande process, vilket betyder att det bara finns ett värde på styrsignalen som kommer att hålla nivån konstant. Med börvärdet på 30 % var detta värde cirka 24 % eller 1,2 V.

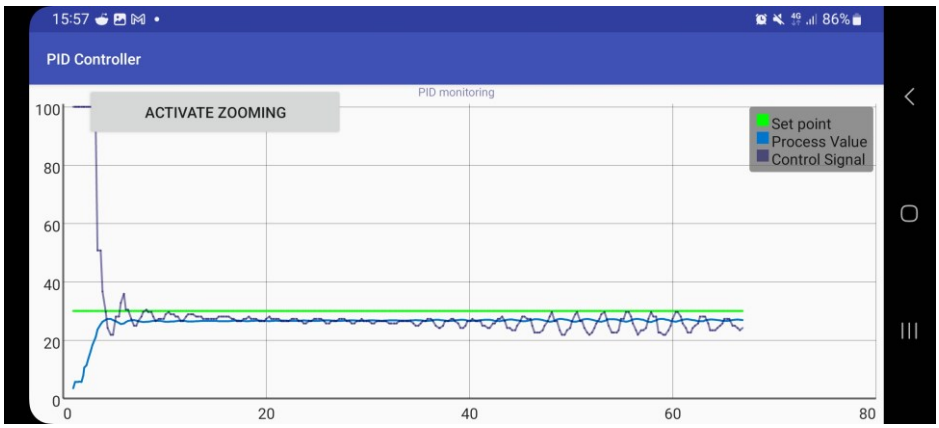
4.4.2 Manuell inställning av regulatorn

För att komma fram till värden för att ställa in regulatorn användes Ziegler-Nichols-självsvängningsmetod. Processen kom i självsvängning med börvärde 30 %, förstärkning K = 8,0 med en samplingstid på 100 ms för regulatorn. Den kritiska periodtiden blev 1,44 sekunder och den kritiska förstärkningen var alltså 8,0. Ziegler-Nichols-metoden gav dessa parametrar:

Tabell 8: Uträknade parametrar enligt Ziegler-Nichols-självsvängningsmetoden.

Regulator	K	T_I	T_D
P	4		
PI	3,6	1200 ms	
PID	4,8	720 ms	180 ms

I figur 20 ses en skärmdump av applikationen då Ziegler-Nichols-självsvängningstestet blev utfört:



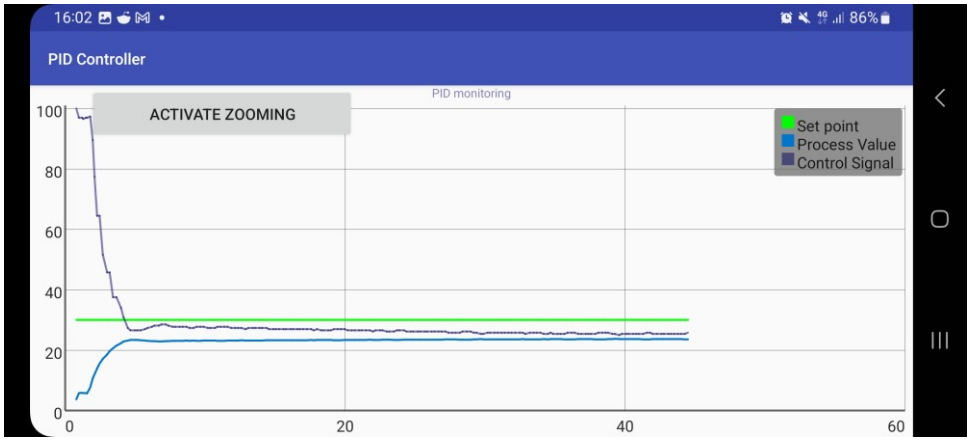
Figur 20: Skärmdump av självsvängningstestet.

Det tog en tid tills processen började självsvänga men efter att svängningen börjar så är den ganska konstant. Processen är egentligen en dubbeltankprocess, där den andra tanken fylldes långsammare och det krävdes att nivån steg till börvärdet i båda tankarna före processen började självsvänga.

4.4.2.1 Resultat från manuella inställningen

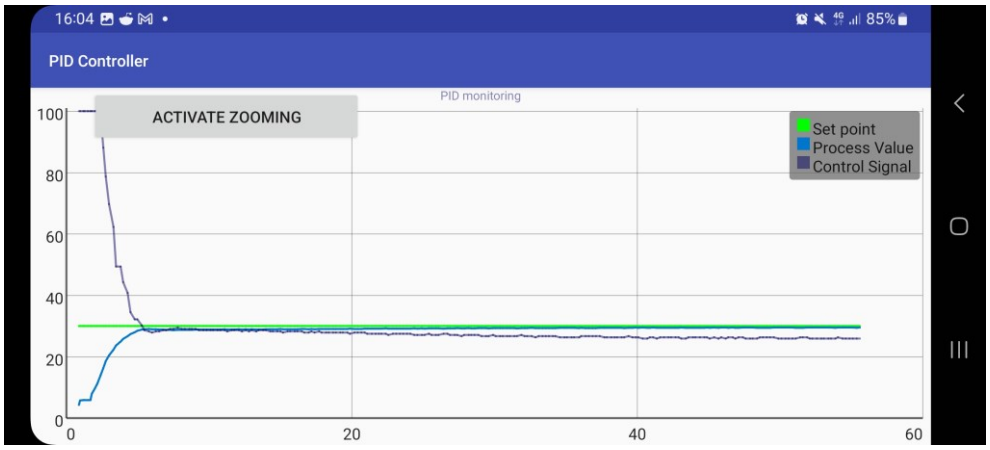
I detta kapitel kommer resultaten, som Ziegler-Nichols metoden gav, att presenteras. Regulatorn blev testad som P-regulator, P-regulator med offset, PI-regulator och PID-regulator och regulatorn ställdes alltså in med parametervärdena från tabell 8.

Regulatorn blev först inställd som en P-regulator och resultatet för den kan ses i figur 21. Det kvarstående felet, som oftast blir då man endast använder P-reglering, kan också ses.



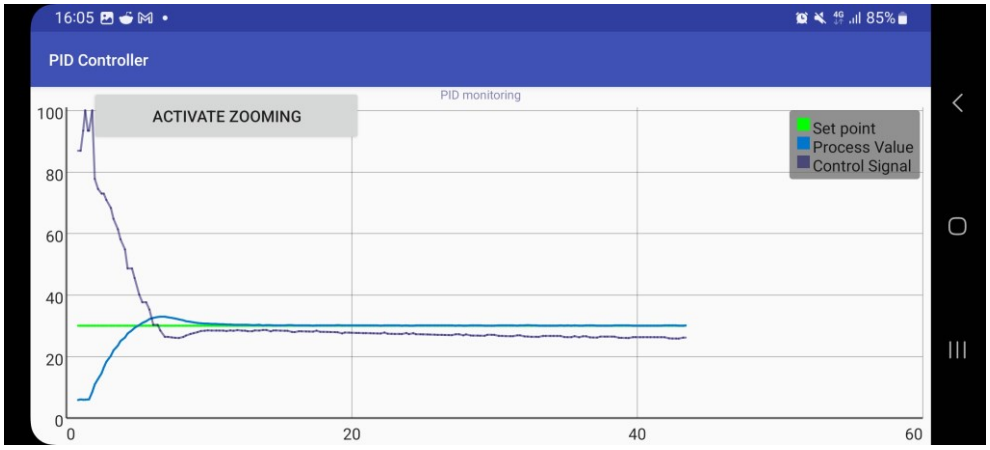
Figur 21: Inställd P-regulator.

Ett test med P-reglering och offset blev också utfört. För att veta vad offseten borde vara tas börvärde minus ärvärde i figur 21, så fås värdet för offseten. Offseten var 6,0 % här. Det gav ett fint resultat, vilket kan ses i figur 22:



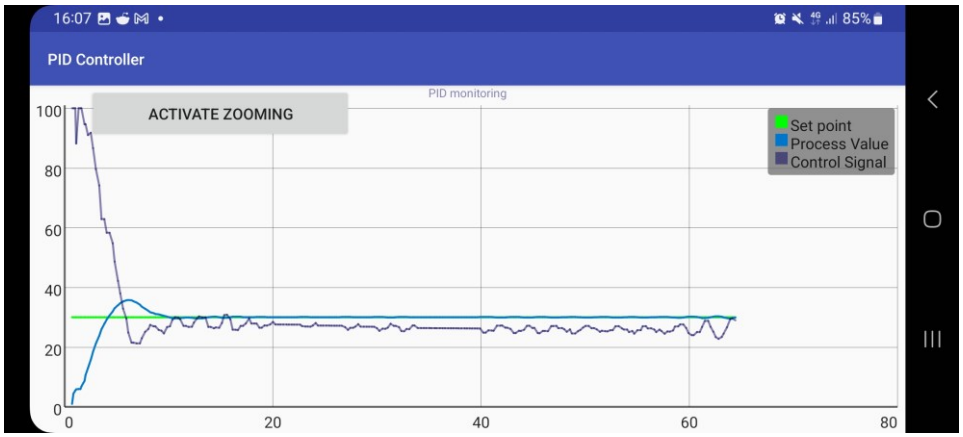
Figur 22: P-regulator med offset inställd.

För PI-regulatorn tar det lite längre tid för att komma upp till börvärdet men det kvarstående felet har försvunnit. Processen får också en översväng här som visar att Ziegler-Nichols-metoden är aggressiv, vilket kom upp i teoridelen. Översvängen kunde troligtvis fås bort med mera integrering eller mindre proportionell förstärkning. Resultatet för PI-regulatorn kan ses i figur 23:



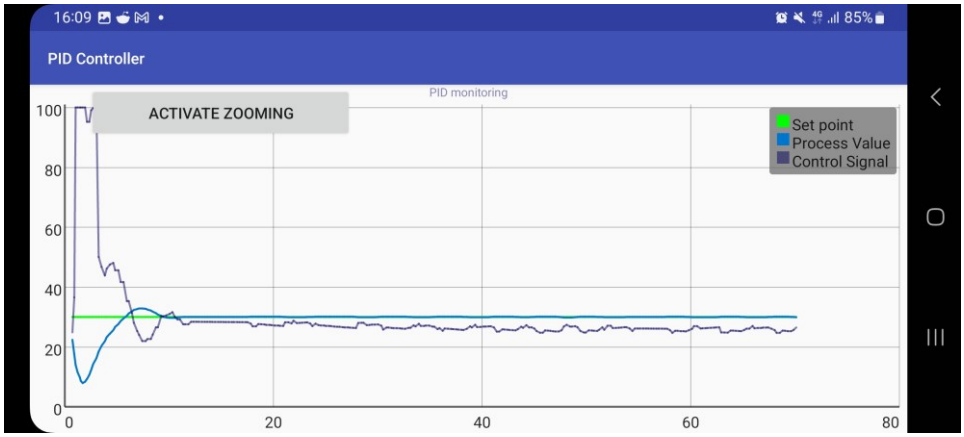
Figur 23: Inställd PI-regulator.

PID-regulatorn var snabbare att nå börvärdet men styrsignalen är tyvärr ganska ostabil på grund av D-delen. Filtreringsvikten är inställd till 0,7 i figur 24, så det är ganska lite filtrering.



Figur 24: Inställd PID-regulator.

Ett test med mera filtrering utfördes också. Filtreringsvikten i figur 25 är 0,3:



Figur 25: PID-regulator med mera filtrering.

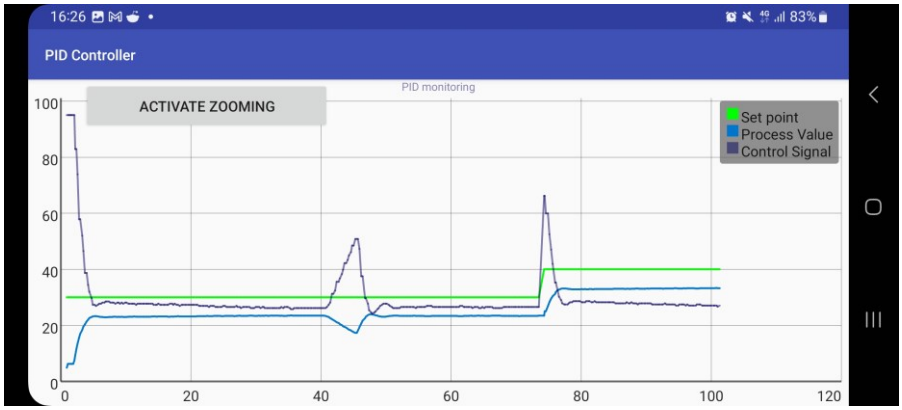
Med mera filtrering blev regleringen mycket bättre.

En viktig egenskap för regulatorer är hur bra de reagerar på störningar och hur snabbt de reglerar processens storhet till ett nytt värde vid en börvärdesändring. Som det nämndes i teoridelen av detta dokument så är en störning en storhet som på ett oönskat sätt påverkar reglerprocessen. En störning kunde skapas för nivåregleringsprocessen genom att strypa slangen som tillförde vatten till processen. En börvärdesändring gick att göra

direkt från Androidapplikationen genom att skriva in ett nytt börvärde till en redan startad process.

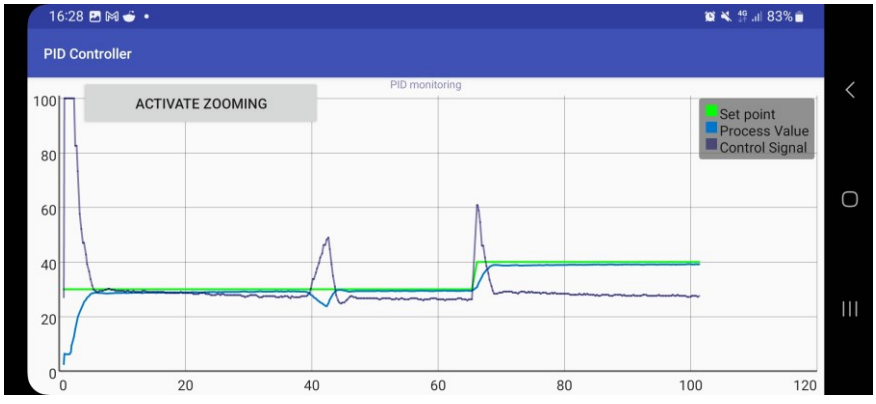
Resultatet från störningstesterna kan ses i figurerna 26 till 29 och både störnings- och börvärdesändringstesterna blev gjorda under samma körning. Resultaten som regulatorn gav var ganska förväntade.

I figur 26 kan P-regulatorns störningstest ses. P-regulatorn är ganska snabb att korrigera efter störningen men har det kvarstående felet.



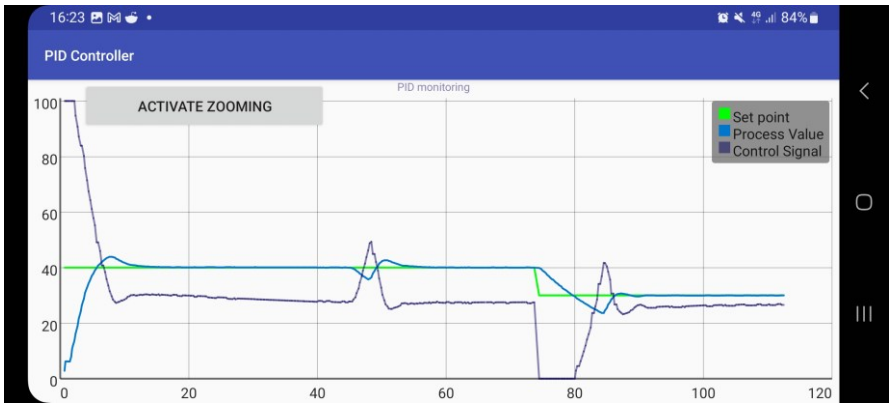
Figur 26: Störningstest och börvärdesändring för P-regulatorn.

Störningstestet för P-regulatorn med inställd offset gav ett riktigt bra resultat men offsetvärdet behövde ju fås fram. Offseten förväntas också att ändras ifall att börvärdet ändras mycket, vilket betyder att det troligtvis skulle bli ett kvarstående fel ifall att börvärdesändringen var större. Resultatet för P-regulatorn med inställd offset kan ses i figur 27:



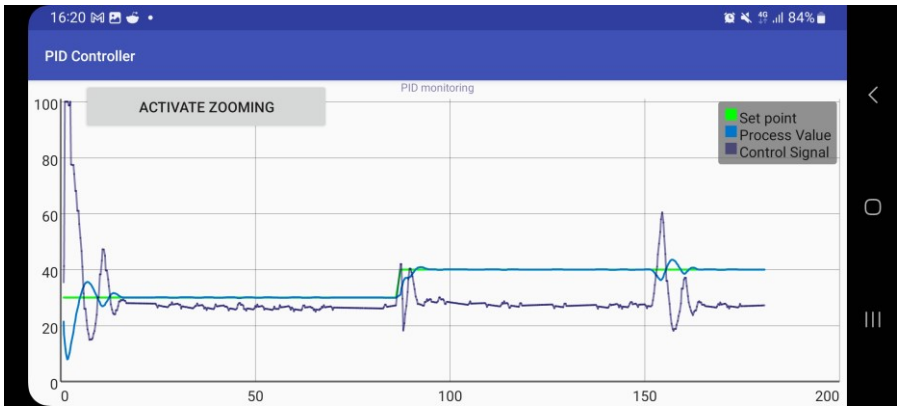
Figur 27: Störningstest och börvärdesändring för P-regulator med offset.

PI-regulatorn är lite långsammare än P-regulatorn men justerar ärvärdet tillbaka till börvärdet efter en störning eller börvärdesändring. PI-regulatorn är också väldigt stabil. Resultatet från störningstestet för PI-regulatorn kan ses i figur 28. Här är börvärdesändringen negativ som ska fungera lika bra som en positiv börvärdesändring.



Figur 28: Störningstest och börvärdesändring för PI-regulator.

I figur 29 kan resultatet för PID-regulatorns störningstest ses. Regleringen efter en störning tar en längre tid en förväntat, vilket kan bero på en grov filtrering. Justeringen av ärvärdet efter börvärdesändring är väldigt snabb, vilket är ett mera förväntat resultat av en PID-regulator.



Figur 29: Störningstest och börvärdesändring för PID-regulatorn. Filtreringsvikten är 0,3 här.

4.4.3 Jämförelse med en annan regulator

För att kontrollera att PID-regulatorn faktiskt fungerade och gav pålitliga resultat jämfördes den med en annan regulator, ECA-400, som vanligtvis används för att styra

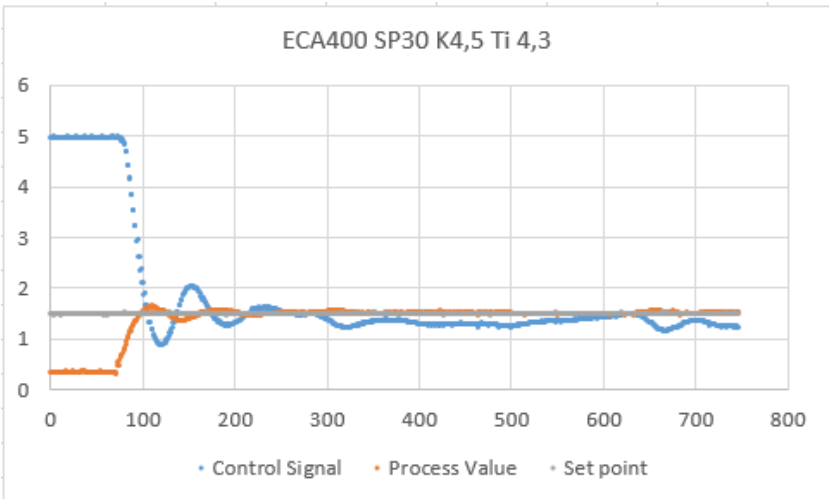
nivåregleringsprocessen. Regulatorerna jämfördes med varandra på så vis att båda regulatorerna ställdes in med samma parametrar och sedan undersöktes resultatet. Processvärdet och styrsignalen mättes också med multimetrar för att visa att de faktiskt var samma för båda regulatorerna. Det viktigaste var ju att processvärdet, vid ett stabilt läge, var samma för båda regulatorerna, eftersom om användaren sätter in ett börvärde så ska ju processvärdet anta detta värde och inte något annat värde. Resultatet blev riktigt bra eftersom båda regulatorerna gav likadana resultat med samma inställningar. Lite skillnader i snabbhet fanns det ju nog men det var förväntat eftersom regulatorerna är olika. De har till exempel inte samma samplingstid.

Två tester blev utförda med följande inställningar:

Tabell 9: Jämförelsetest 1.

Börvärde (SP)	Förstärkning (K)	Integreringstid (Ti)
30 %	4,5	4300 ms

För ECA-400-regulatorn gav inställningarna, som ses i tabell 9, resultatet som kan ses i figur 30:

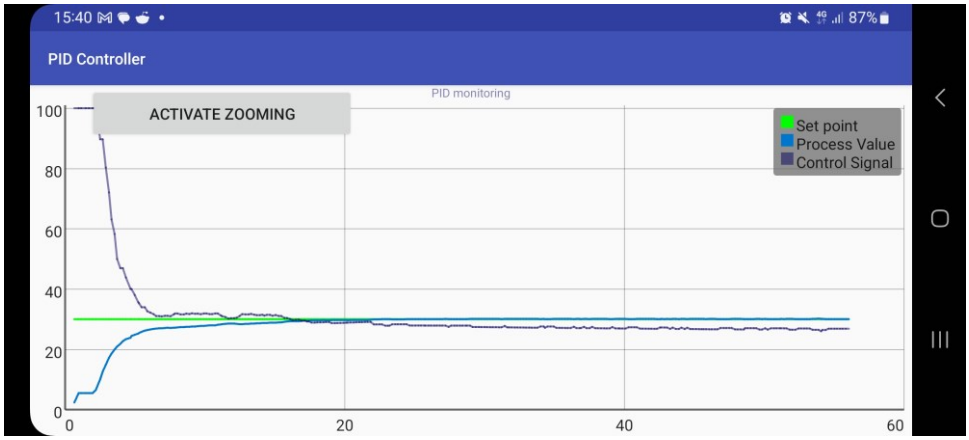


Figur 30: Test för ECA-400 med börvärde 30 %.

I figur 30 är y-axeln i volt och x-axeln i antal sampel. Börvärdet var inställt på 1,5 V, vilket är 30 % av maximala börvärdet och nivån blir stabil efter ca 250 sampel som motsvarar en

tid på 25 sekunder. Multimetrarna gav spänningsvärdet 1,497 V för processvärdet och 1,19 V för styrsignalen då nivån hade uppnått ett stabilt läge.

Då PID-regulatorn som skapades i examensarbetet ställdes in enligt tabell 9 så fick man resultatet som ses i figur 31:



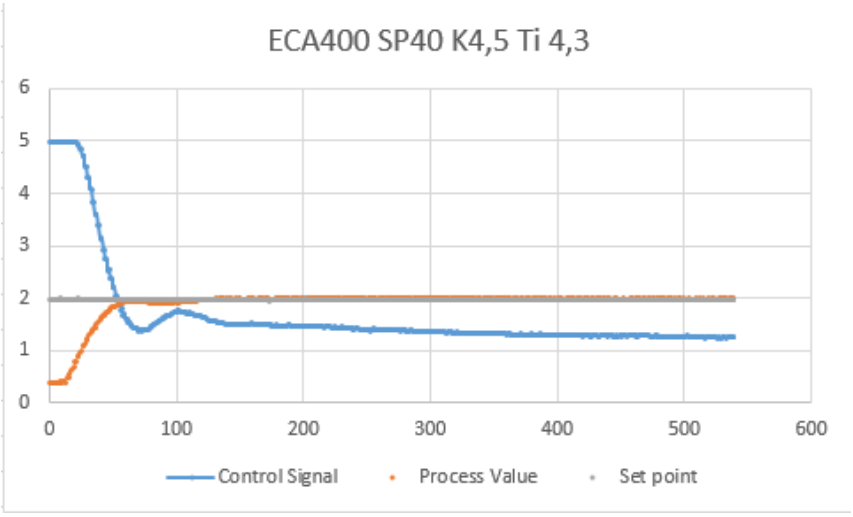
Figur 31: Test för examensarbetets regulator med börväde 30 %.

I testet från figur 31 tog det också ca 25 sekunder att uppnå börvärdet och spänningsmätningarna gav 1,481 V för processvärdet och 1,248 V för styrsignalen. Spänningarna är ganska likadana för både processvärdet och styrsignalen som de var för ECA-400-regulatorn så resultatet är helt pålitligt. Styrsignalen avviker lite från ECA-400:ns styrsignal, vilket kan bero på att mikrokontrollern gav ut styrsignalen som en pwm-signal som kanske inte kan mätas så exakt med multimetern.

Tabell 10: Jämförelsetest 2.

Börvärde (SP)	Förstärkning (K)	Integreringstid (Ti)
40 %	4,5	4300 ms

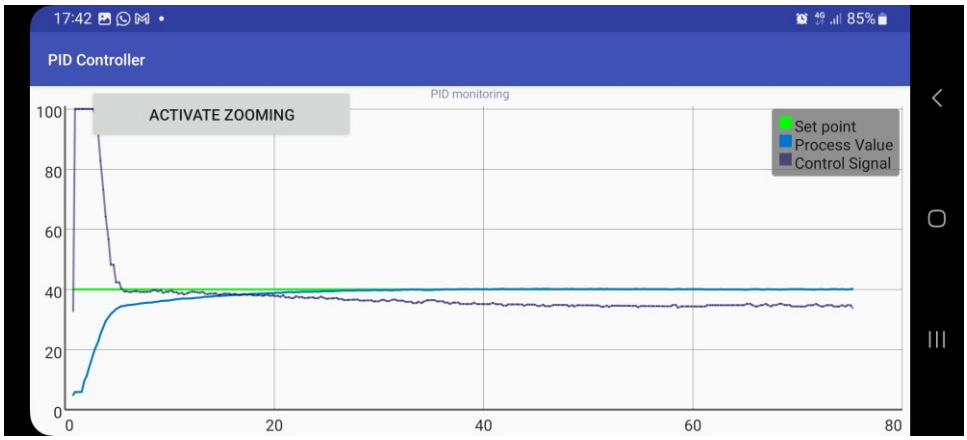
Då ECA-400-regulatorn ställdes in med parametrarna från tabell 10 fick man resultatet som kan ses i figur 22:



Figur 32: Test för ECA-400 med börvärde 40 %.

I figur 32 kan man se att börvärdet är inställt på 2 V vilket är 40 % av maximala börvärdet och processvärdet ser ut att bli stabilt vid ca 100 sampel, som motsvarar 10 sekunder. Multimetrarna gav spänningsvärdet 1,989 V för processvärdet och 1,344 V för styrsignalen då nivån hade uppnått ett stabilt läge.

Examensarbetets PID-regulator gav resultatet som kan ses i figur 33, då regulatorn ställdes in enligt tabell 10:



Figur 33: Test för examensarbetets regulator med börväde 40 %.

Figur 33 visar resultatet för examensarbetets regulator med börvärdet på 40 %. I det här testet tog det ganska mycket mera tid att uppnå börvärdet, då man jämför med ECA-400:an men spänningsvärdena var igen väldigt nära varandra. Multimetrarna visade spänningsvärdena 1,976 V för processvärdet och 1,358 V för styrsignalen.

5 Slutsatser

PID-regulatorn som skapades i examensarbetet fungerade riktigt bra och likaså fungerade Androidapplikationen bra. PID-regulatorn hade lite ostabil styrsignal när derivatan aktiverades men det är ändå ganska sällan som man använder derivatadelen och det går ju också att ändra på filtreringen i inställningarna, vilket gjorde att derivatadelens styrsignal blev stabilare. Androidapplikationen blev inte helt perfekt och kan nog krascha och ha andra problem men den brukar börja fungera igen om man stänger ner den och öppnar igen. Det finns helt klart mycket som kan förbättras för både Android- och Arduinoapplikationerna men koden finns på GitHub och det är bara att ladda ned den där ifrån och göra de ändringar man vill. Det är ju ändå ganska lätt att ladda upp ny kod både till Androidenheter via Android Studio och till Arduinoneheter via Arduino IDE.

Det var meningen att regulatorn skulle få en auto tune-knapp för automatisk inställning av regulatorn men den blev aldrig implementerad. Det fanns många idéer hur auto tune kunde ha implementerats, som till exempel att använda relämetoden, att göra en börvärdesändring på den oreglerade processen och räkna ut parametrar med hjälp av hastighetsförstärkningen eller att göra pulstester, men det visade sig vara mera komplicerat än det förväntades vara.

6 Källförteckning

App Manifest Overview. (Februari 2023). Hämtat från Developers Android: https://developer.android.com/guide/topics/manifest/manifest-intro

ArduinoBLE. (Mars 2023). Hämtat från Arduino.cc: https://www.arduino.cc/reference/en/libraries/arduinoble/

Bauer, J. (den April 25 2016). *Bauerjj/Android-Simple-Bluetooth-Example.* Hämtat från Github: https://github.com/bauerjj/Android-Simple-Bluetooth-Example

Bluetooth. (Mars 2023). Hämtat från Wikipedia.org: https://en.wikipedia.org/wiki/Bluetooth

Burns, R. S. (2001). *Advanced Control Engineering.* Elsevier Science & Technology.

Configure your build. (Mars 2023). Hämtat från Developers Android: https://developer.android.com/studio/build

Developers Android Guides. (February 2023). Hämtat från Developers Android: https://developer.android.com/guide

Exponential smoothing. (Mars 2023). Hämtat från Wikipedia.org: https://en.wikipedia.org/wiki/Exponential_smoothing

Gehring, J. (den 7 Augusti 2013). *Jjoe/GraphView.* Hämtat från Github: https://github.com/jjoe64/GraphView

Gupta, N. (2013). *Inside Bluetooth Low Energy.* Boston, Massachusetts, Förenta Staterna: Artech House. Hämtat från https://tritonia.finna.fi/novia/Record/abo_electronic_novia.9913432523905972

Hägglund, T. (2019). *Praktisk processreglering.* Lund: Studentlitteratur AB.

Installing the SAMD21 core for MKR boards. (Mars 2023). Hämtat från Arduino.cc: https://docs.arduino.cc/software/ide-v1/tutorials/getting-started/cores/arduino-samd

Junginger, M. (den 8 December 2021). *greenrobot/EventBus.* Hämtat från Github: https://github.com/greenrobot/EventBus

Nano 33 Iot. (Mars 2023). Hämtat från Arduino.cc: https://docs.arduino.cc/hardware/nano-33-iot

Overview of Arduino IDE 1. (Mars 2023). Hämtat från Arduino.cc: https://docs.arduino.cc/software/ide-v1/tutorials/Environment

PID controller: Origins. (Mars 2023). Hämtat från Wikipedia.org: https://en.wikipedia.org/wiki/PID_controller

Sigvardsson, S. (den 23 Juni 2019). Optimering av kompressorstyrning. Sudsvall, Norrland, Sverige.

Thomas, B. (2016). *Modern Reglerteknik.* Stockholm: Liber AB.

Wahlfrid, J. (2007). *Realisering och inställning av PID-regulatorer.* Malmö: Malmö Högskola Teknik och Samhälle.

Wasserman, J. (den 12 July 2016). *joelwass/Android-BLE-Connect-Example.* Hämtat från Github: https://github.com/joelwass/Android-BLE-Connect-Example/find/master

Bilaga 1: Android kod

Bilaga 2: Arduino kod