CHALMERS

Ett hjälpmedel för synskadade

TV undertexter som text-to-speech på telefonen

Examensarbete inom Data- och Informationsteknik

William Jönsson
Simon Terling

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

EXAMENSARBETE

Ett hjälpmedel för synskadade

TV undertexter som text to speech på telefonen

William Jönsson
Simon Terling

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2021

Ett hjälpmedel för synskadade
TV undertexter som text to speech på telefonen
William Jönsson
Simon Terling

© William Jönsson, Simon Terling, 2021

Examinator: Sven Knutsson
Handledare: Karl Bäckström

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Omslag:
Någon förklaring

Institutionen för Data- och Informationsteknik
Göteborg 2021

Sammanfattning

Synskadade ser på tv, men kan missa viktiga bitar av dialog då personen inte förstår vad som sägs och inte kan läsa undertexterna. Det finns redan en lösning på problemet, producten Go-box, problemet är att produkten är för dyr i förhållande till behovet. Rapporten kommer gå igenom utvecklingen av en produktprototyp, ett hjälpmedel för synskadade som liknar Go-box som kommer att läsa undertexter med hjälp av text-till-tal. Rapporten kommer även gå igenom möjligheten till att göra produkten billigare än Go-box utan att förlora funktionalitet för att kunna göra produkten tillgänglig för alla som behöver den.
Lösningen består av två huvudsakliga komponenter, en Android app och en enkortsdator som kör ett program som använder datorseende. På grund av hårdvarubegränsningar kan enkortsdatorn inte läsa av live-videoströmmar utan läser bildrutor från en video och försöker i realtid identifiera och läsa texten med hjälp av OCR. Den identifierade texten skickas till appen via Bluetooth och appen läsa upp den mottagna texten med hjälp av text-till-tal. Mjukvaran fungerar korrekt, men enkortsdatorn, Raspberry Pi 4 var inte tillräckligt snabb för att köra programmet.

Nyckelord: datorseende, Android, text-till-tal, Raspberry Pi, Bluetooth, OCR

Abstract

Visually impaired people watch tv, but can sometimes miss crucial parts because they do not understand what's being said and they can't read the subtitles. There is already a solution to the problem, a product called Go-box that solves the problem, but the product is too expensive compared to the benefit of the product. This paper will discuss the development of a product prototype, an aid for visually impaired similarly to Go-box, that will read the subtitles aloud with text-to-speech. It will also investigate the possibility of making the product cheaper than Go-box without compromising functionality, making it an aid available for everyone who needs it. The solution is composed of two main components. An Android app and a microcomputer running a computer vision program. Due to hardware limitations, live video streams are not possible with the microcomputer. Instead the microcomputer reads frames from a video file and tries in real time to identify and read the subtitles with the help of OCR. The identified text will be sent to the app through Bluetooth and the app will read the received text with the help of text-to-speech. The software is working correctly, but the microcomputer, Raspberry Pi, wasn't fast enough to run it.

Keywords: computer vision, Android, text-to-speech, Raspberry Pi, Bluetooth, OCR

Förord

Vi vill tacka vår handledare Andreas Löfman från Nexer för sitt stöd och råd under de möten vi hade under hela projektets gång, och även tips på bra matställen. Vi vill även tacka Nikolai Dahlberg som hjälpte oss att komma i kontakt med Nexer och Andreas. Även stort tack till Karl Bäckström som hjälpte oss med rapporten i slutet av projektet.

William Jönsson och Simon Terling, Göteborg, Maj 2021

Innehållsförteckning

1. Inledning **1**
1.1 Bakgrund 1
1.2 Syfte 1
1.3 Mål 1
1.4 Avgränsningar 2

2. Teknisk bakgrund **3**
2.1 Maskininlärning 3
2.1.1 Neural network 3
2.1.2 PyTesseract-OCR 4
2.2 Python 4
2.2.1 Bibliotek 5
2.2.2 Setup 5
2.3 Android Studio 5
2.3.2 Android Aktivitet 6
2.3.3 Android Fragment 6
2.4 Raspberry Pi 7
2.5 Bluetooth 7
2.6 Textlokalisering EAST metoden 7
2.7 Relaterade Arbeten 8

3. Metod **9**
3.1 Arbetsprocess 9
3.2 Språkval 9
3.3 Öka tillgänglighet 9
3.4 Kommunikation 10
3.5 Insamling av data gällande placering av undertexter 10
3.6 Bildbehandling 10
3.7 Texthantering 12
3.8 Kalibrering och inställningar 12

4. Resultat - Systemkonstruktion **14**
4.1 Tesseract 14
4.2 Tidsprofil 14
4.3 Begränsning av sökområdet för undertexter 15
4.4 Textlokalisering 15
4.5 Bildbehandling 15
4.6 Texthantering 18
4.7 Konfiguration 19
4.8 Kommunikation 21

4.9 Android applikation 23
4.9.1 MainActivity 24
4.9.2 TurnOnBluetoothFragment 24
4.9.3 ConnectToBluetoothDeviceFragment 26
4.9.4 SubtitlesFragment 28
4.9.5 SettingsFragment 28

5. Resultat - Delmål **30**
5.1 Ett program som extraherar text från video 30
5.2 App som läser upp text 31

6. Diskussion **32**
6.1 Utvecklingspotential 32
6.2 Möjlighet till minskad kostnad 33
6.3 Slutsats 33

Referenser **34**

Bilagor **40**
Bilaga 1. Sammanställning av normer för undertexter
Bilaga 2. Förenklat flödesdiagram av texthanteringen
Bilaga 3. Resultat från körning av texthantering

1. Inledning

1.1 Bakgrund

Trots att synskadades syn är gradvis eller helt borta är det fortfarande vanligt att de ser på TV. Mycket kan förstås med hjälp av ljud och tal, men när talet är på ett främmande språk utan möjligheten att läsa undertext missas dialogerna och därmed mycket av handlingen. Projektet fokuserar på att med hjälp av datorseende göra undertexter på TV mer tillgängliga för synskadade.

Idén består av två delar, en Android app och ett program som körs på en Raspberry Pi. Raspberry Pi:n får en videoström och med hjälp av maskininlärning, optical character recognition (OCR), tolkas undertexter i en video. Den detekterade texten ska sedan skickas från Raspberry Pi:n till en mobilapp där undertexten läses upp med hjälp av text-to-speech.

Projektet var en egen idé och ett par företag kontaktades varav Nexer (tidigare Sigma IT) var intresserade av projektidén. Projektet utfördes på Nexers R&D avdelning där Andreas Löfman var vår handledare. Nexer är ett globalt IT-konsultföretag med 1 600 experter i 9 länder med fokus på allt från DevOps[1] till artificiell intelligens inom en mäng olika branscher.[2]

Det finns redan en lösning på problemet med att inte kunna läsa undertexter, Go-box[3], men produkten är inte brett tillgänglig på grund av sitt höga pris i förhållande till behovet.

1.2 Syfte

Syftet med projektet är att öka synskadades tillgänglighet till media som distribueras via television. En prototyp som visar att det är möjligt att skapa ett billigare alternativ till Go-box. Det är inte tänkt bli en produkt för försäljning, utan enbart demonstration av konceptet.

1.3 Mål

Skapa en produktprototyp som ska hjälpa synskadade genom att göra media mer tillgängligt.
Delmål:

- Ett program som extraherar text från video
    - Lokalisera text
    - Extrahera text
    - Köra koden på Raspberry Pi
    - Skicka undertext till app

- App som läser upp text
    - Bluetoothhantering
    - Läsa upp undertext
    - Skicka inställningar till textextraheringsprogrammet

1.4 Avgränsningar

Ursprungsidén var att kunna använda en live-videoström. Det bestämdes senare i projektet att videofiler skulle användas istället för en videoström. Beslutet motiverades med att Raspberry Pi 4 inte har HDMI video-input, repeterbarhet av tester och öka fokus på den tekniska lösningen. För att Raspberry Pi ska kunna använda en live-videoström krävs ytterligare hårdvara. Programmets huvudfunktion är att identifiera undertexter i form av skrift i en videofil oavsett om texten kommer från uppspelad undertext metadata eller textning i video. Programmet är begränsat till att känna igen tecken från latinska alfabetet. Programmet är designat för att klara av svensk textning och ordigenkänning.
Raspberry Pi:n behöver kommunicera med appen. Det var först tänkt att göras via ett lokalt nätverk men har istället gjorts via Bluetooth då det gick snabbare och tar bort krav på en nätverksarkitektur.

Appen görs endast till Android och appen ska inte läsa upp text när den är minimerad.

2. Teknisk bakgrund

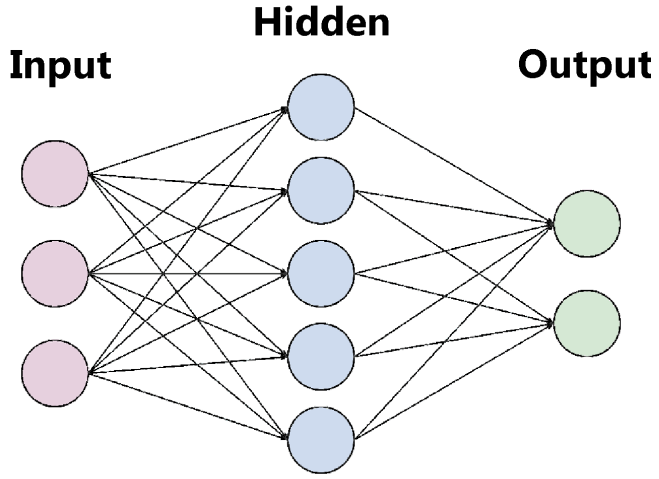
2.1 Maskininlärning

Maskininlärning är en gren inom artificiell intelligens, AI, som med hjälp av algoritmer analyserar data för att lära sig utföra en specifik uppgift. Om en algoritm matas med mer träningsdata eller kan lära sig själv genom att till exempel köra simulationer kommer den att bli bättre på att utföra uppgiften den är tränad för. I vissa fall kan maskininlärning utföra uppgifter som annars hade varit omöjliga att programmera på traditionellt sätt, till exempel bildigenkänning.[4]

AI är att via datorprogram efterlikna mänsklig intelligens. Maskininlärning är ett verktyg för att uppnå detta, men det är inte alltid som det krävs. En AI till ett spel som schack kräver ingen maskininlärning och kan hårdkodas då schack har tydliga och enkla regler. Ibland är det omöjligt att göra hårdkodade lösningar som vid bildigenkänning eftersom bilder av samma sak, till exempel en hund kan se ut på väldigt många sätt. Bildigenkänning kan kräva komplexa regler och då kan det vara lämpligt att använda maskininlärning.

2.1.1 Neural network

Neurala nätverk är ett område inom maskininlärning, där nätverk är uppbyggda av noder som bearbetar data på ett fördefinierat sätt. Data bli inmatad i “input layer” som är det första lagret som sedan skickar vidare datan till “hidden layer” där beräkningar sker som sedan skickat till “output layer” (se figur 1). När neurala nätverk lär sig görs förändringar av kant-vikterna för att optimera precisionen. Kant-vikter är ett tal kopplade till en specifik kant i en graf (se figur 1) och används för att ändra noders indata när data skickas mellan noder.[5]



Figur 1. Enkel visualisering av ett neuralt nätverk med tre lager. Cirklar representerar noder och strecken representerar kanter.

2.1.2 PyTesseract-OCR

PyTesseract är en Python wrapper för Tesseract, ett open-source Optical character recognition (OCR) bibliotek som först utvecklades av HP, sedan 2006 utvecklas det av Google.[6] OCR är en gren inom AI och datorseende där man med hjälp av Deep Learning läser av text från en bild.
För att använda wrappern krävs att Tesseract redan är installerat på datorn.[7]

image_to_string är en metod i PyTesseract som extraherar text från en bild. Metoden är en nyckelkomponent i projektet, kortfattat fungerar metoden på följande sätt:

Det första steget är en “connected component analysis” i vilken konturerna av de funna komponenterna lagras. Sedan skapas “Blobs”, vilket är samlingar av likartade pixlar, genom att slå ihop konturer som överlappar varandra. “Blobs” från föregående steg grupperas och delas upp till ord. Detekteringen av text görs i en “two-pass process”. I första körningen detekteras ord från de grupperade orden i föregående steg. Algoritmen som klassificerar orden tränas då med de nyligen hittade orden. Den andra körningen använder sig av den nytränade klassificeringsalgoritmen för att hitta ord som missats i första körningen. Detta kallas för adaptiv klassificering. De detekterade orden görs sedan om till en textsträng och returneras.[6]

Härefter kommer Tesseract och PyTesseract att användas synonymt om inget annat specificeras.

2.2 Python

Python är ett interpreterande högnivåspråk och har ett brett användningsområde. Språket är designat för god kod-läsbarhet och stödjer objektorienterad programmering för tydligare logik i större program.[8]

Interpreterande kod, till skillnad från “vanlig” kod kompileras under körtid. Detta påverkar Python-programs körtid negativt, men det går att öka programhastigheten med hjälp av bibliotek. Python stödjer bibliotek skrivna i C och C++ som kan köras nästan lika snabbt som vanlig C kod. Om biblioteken används för tunga beräkningar kan Python användas till beräkningstunga uppgifter.[9][10] Anledningen till att det är viktigt att biblioteken är skrivna i C är att om biblioteken istället implementeras direkt i Python hade körtiden blivit signifikant försämrad. Det hade gjort att Python inte hade varit lämpligt till projektet. Python har även extensiva bibliotek för bildbehandling, vilket används i projektet.

2.2.1 Bibliotek

- OpenCV är ett extensivt bibliotek som används för bildbehandling. Biblioteket har även en inbyggt deep neural network modul som kan användas för att klassificera och lokalisera objekt.[11]
- PyEnchant är ett python-gränssnitt för Enchant, vilket är ett rättstavningsbibliotek som slagit samman många olika bibliotek och program i ett och samma gränssnitt.[12]
- PyTesseract är en wrapper till Tesseract (se 2.1.2 PyTesseract)
- PyBluez är ett python-interface för Bluez. Bluez är ett bibliotek som tillåter Bluetooth-kommunikation och fungera med många linux-system.[13]

2.2.2 Setup

PyTesseract samt Tesseract-OCR kräver att en en träningsfil för svenska inkluderas för att stödja detekteringen av svenska text. Information om hur Tesseract samt testdatan installeras finns på Tesseracts github sida [14]. Projektet använder svensk träningsdata från “tessdata_best” kategorin [15]. Även PyEnchant kräver ytterligare filer för att fungera. PyEnchant har endast engelska ordböcker som standard. För att kontrollera svenska ord behöver en svensk ordbok läggas till i PyEnchants installationsmapp. Ordboken som användes heter sv_SE och kommer från LibreOffice github.[16]

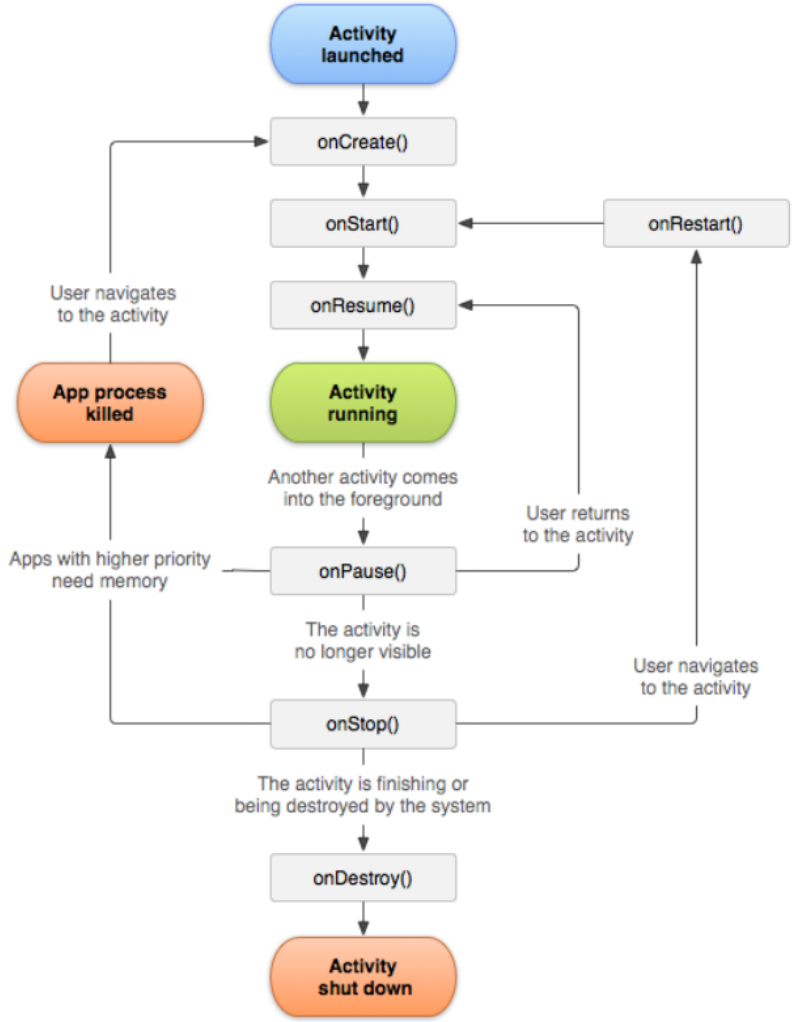
2.3 Android Studio

Android studio är en IDE för att utveckla appar till android. IDE:t är utvecklat av JetBrains och Google som även är utvecklaren av android. Programmet har många avancerade och analytiska funktioner och stödjer utveckling i programmeringsspråken **Kotlin** och Java. Funktioner som gör utvecklingen enklare är verktyg som gradle[17], Android app emulering och flertal lint verktyg[18] inklusive Kite.[19][20] Appen är byggd i Java som under lång tid varit det rekommenderade utvecklingspråket, sedan 2017 har Kotlin varit det rekommenderade språket. Kotlin är ett snabbt växande språk och det av en god anledning. Exempel på fördelar med Kotlin är bättre typsäkerhet samt minskar mängden boilerplate-kod i jämförelse med Java.[21][22] Boilerplate-kod är sektioner som ofta förekommer i koden och är lika varandra med små variationer. Det innebär ofta en betydande mängd kod som måste skrivas av programmeraren för att åstadkomma lite funktionalitet.[23]

Användargränssnittet kan smidigt utformas grafiskt då utvecklaren drar och släpper komponenter för att bygga gränssnittet, utvecklaren har även tillgång till källkoden (XML fil) för full kontroll.

2.3.2 Android Aktivitet

Aktiviteter är grundpelare i alla Android appar där deras enkla livscykel (se figur 2) underlättar utvecklingen.[24] Aktiviteter håller variabler och metoder som kopplas till användargränssnittet för att ge funktionalitet. Aktiviteter är ofta fokuserad på att klara en specifik uppgift och data lagrad i en aktivitet är icke beständig (flyktig) och förloras när aktiviteten avslutas. Appar har ingen generell startmetod som kör all kod utan varje app har en startaktivitet som i sin tur har en egen startmetod, **onCreate()** som är kopplat till dess livscykel. Varje aktivitet är isolerad från andra aktiviteter, behöver en aktivitet information från andra delar av programmet måste informationen skickas till aktiviteten eller sparas med Android preferenser.[25][26]



Figur 2. Visar en Android Aktivitets livscykel

2.3.3 Android Fragment

Fragment är lika aktiviteter. Liksom aktiviteter har fragment logik för att möjliggöra en interaktiv applikation, deras livscykel (se figur 2) är densamma och koden utformas på mycket likartat sätt. Skillnaden är att fragment är mer flexibla där ett eller flera fragment kan användas i en aktivitet. Varje fragment kan ha en specifik uppgift och

är ett smidigt och modulärt sätt att skapa återanvändningsbara funktioner. Med hjälp av fragment kan man ändra en aktivitets utseende under körtid då aktiviteter styr vilka fragment det är som instansieras.[27][28]

2.4 Raspberry Pi

Raspberry Pi är en serie av enkorts mikrodatorer. Då datorn är billig och är snabb relativt till sin storlek är det vanligt att använda den till mindre projekt och prototyputveckling. I projektet kommer Raspberry Pi 4 b användas som har en 1,5 GHz processor och 8 GB ram.[29]

2.5 Bluetooth

Bluetooth utvecklades 1997 och är ett standardiserat sätt för enheter att kommunicera med varandra över korta avstånd.[30] Bluetooth har sedan dess utvecklats och är idag vanligt förekommande och har många tillämpningar. Protokollet som kommer att användas är RFCOMM [31] som är baserat på L2CAP [32] protokollet. Två fördelar med RFCOMM är att enheten svarar när den tagit emot paket, detta kan användas för att säkerställa att strömmad datan har kommit fram. En annan fördel är protokollet stöds av många system.

2.6 Textlokalisering EAST metoden

Att detektera text i ett skannat dokument är inget nytt. Det är däremot svårt att detektera text i bilder som inte är tillrättalagda dokument. EAST (Efficient and Accurate Scene Text detector) är en metod som använder neurala nätverk för att detektera text i bilder oavsett vad det är för bild.[33] EAST används i programmet för att detektera vart i bilden undertexten finns och har visat goda resultat.

2.7 Relaterade Arbeten

2021/05/19 när rapporten skrevs inte hittades inte några rapporter om OCR som hjälpmedel för synskadade. De närmsta relaterade arbetet som hittades var “Detection and Localization of Embedded Subtitles in a Video Stream” av G. ValerySene och S. Jean.[34]

I det projektet har undertexter i videofiler försökt detekteras. Förenklat bygger lösningen på att jämföra olika bildrutor med varandra och se vad som är statiskt i bilden. Det är som statiskt kan eventuellt vara undertext, till skillnad från vårt projekt som använder använder EAST metoden som kan identifiera all text i hela bilden (se 2.1.1 Textlokalisering). En svaghet med metoden i rapporten är att om allt i bildrutan är statiskt kan den inte filtrera ut texten och får därmed ett sämre resultat.

3. Metod

Projektet är uppdelat i två delar, den första är vad som i rapporten kommer att kallas för **bildkoden**. Bildkoden är den del av projektet som lokaliserar, läser av och sedan skickar undertexter vidare till **appen**. Appen är den andra delen av projektet vars uppgift är att att presentera och läsa upp undertexterna som mot tags från bildkoden.

3.1 Arbetsprocess

Projektet har följt ramverket Scrum där en god kommunikation upprätthållits med Nexer. Scrum-reviews samt sprint-planning hölls varannan vecka sedan projektets början där Nexer agerat som produktägare.
För att få översikt i vad som behövde göras gjordes ett scrum board som innehöll alla projektets övergripande delar. Scrum-boardet uppdaterades sedan kontinuerligt för att enkelt kunna följa projektets utveckling. Projektets omfattning var från början liten då majoriteten av tekniker som tänktes användas i projektet var nya. Många funktioner gick snabbare att implementera än planerat och därav utökades omfattningen och nya funktioner lades till.

3.2 Språkval

bildkoden är skriven i Python, vilket är väldigt långsamt prestandamässigt jämfört med många andra språk. Däremot finns många bra bibliotek för bildhantering och maskininlärning som underlättar utvecklingen av programmet. Språkets läslighet, användarvänlighet och det stora utbudet av extensiva bibliotek som dessutom är lättinstallerade gör språket till ett bra val.

Appen är gjord med hjälp av “Android Studio” som tillåter utveckling i Java och Kotlin. Oavsett fördelarna med Kotlin (se 2.3 Android Studio) valdes Java som utvecklingsspråk eftersom Java har använts i stor del av utbildningen och var för oss ett bekant språk.

3.3 Öka tillgänglighet

Go-box är en produkt som ger samma funktion som våran prototyp och finns redan på marknaden och kostar 7192 kr [3] vilket kan vara en hög kostnad för många. Vid kontakt med Synskadades Riksförbund nämnde de att produkten har svårt att klassas som ett hjälpmedel eftersom det är en hög kostnad mot ett behov som inte anses vara tillräckligt stort.[35] För att göra den potentiella produkten tillgänglig för fler måste produktens pris vara lägre. För att minska kostnaden var idéen att minimera mängden hårdvara utan att minska funktionalitet. Go-box har till exempel en IR-sensor, fjärrkontroll och en USB IR-adapter för hörapparater.[36] Om mobilen är uppkopplad till enheten kan telefonen agera som en fjärrkontroll, och mobilen som läser upp texten har redan support för hörapparater.[37]

3.4 Kommunikation

Kommunikationen var från början planerad att ske över ett lokalt nätverk. Både programuppdateringar och kommunikation med mobilen kan utföras över samma kommunikationsmedium. Ett mål är att enheten ska vara billig, ett sätt att minska kostnaden är att minska mängden hårdvara. Go-box har hårdvara som stödjer kommunikation med många olika typer av enheter, prototypen behöver enbart kommunicera med en enhet, mobilen. Som tidigare nämnts var kommunikationen med mobilen planerad att ske över ett lokalt nätverk, nackdelen är att utvecklingen av ett sådant nätverkssystem hade tagit lång tid. Implementeringen blir även svårare om det ska kunna finnas flera enheter på samma nätverk. För att förenkla arbetet valdes Bluetooth, kommunikation över internet skulle implementeras i mån av tid.

Oavsett val av kommunikationsmedium behövs ett kommunikationsprotokoll. Tanken med det framtagna protokollet var att det skulle vara smidigt att vidareutveckla ifall behovet uppstår.

3.5 Insamling av data gällande placering av undertexter

För att underlätta utvecklingen av bild-programmet samlades data kring placeringen av undertexter in på två sätt. Det första var att läsa på om det fanns regler eller riktlinjer kring placeringen, vilket det fanns i viss mån.[38] Det andra var att studera olika kanaler på TV och notera vilka riktlinjer som följs. Slutligen sammanställdes de normer som fanns för undertexter (se bilaga 1).

3.6 Bildbehandling

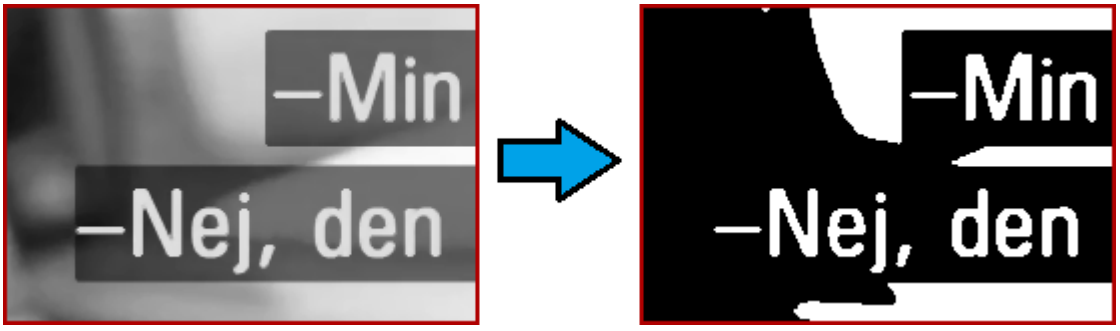
Bildbehandling är vanligt förekommande i samband med datorseende där målet är att isolera och förtydliga bilddatan för ett bättre resultat.[39] Tesseract använder Leptonica library, ett inbyggt bildhanterings bibliotek som används för att processa bilder innan Tesseract läser av texten. Enligt Tesseracts dokumentation är den inbyggda bildhanteringen ofta tillräcklig, men kan ibland vara bristfällig.[40] Då indatan kan vara mycket varierande är det svårt för Tesseract att få konsekventa resultat. För att resultatet skall vara konsekvent och pålitligt användes OpenCV för att bearbeta bilderna innan de skickas till Tesseract. Målet med bearbetningen är att göra stor del av bilden vit med svart text eftersom det visat sig ge bättre resultat med Tesseract.[41] Tesseract ger bättre resultat när bilder på enbart undertexter används. För att kunna göra det live i en video behöver texten först lokaliseras och klippas ut innan bilden kan hanteras.

Bild-programmet använder OpenCV för bildbehandling där följande steg beskriver tillvägagångssättet.

Första steget är att konvertera bilden till **gråskala**, då färg-datan är överflödig. Bildens tre kanaler, RGB, konverteras till en kanal.

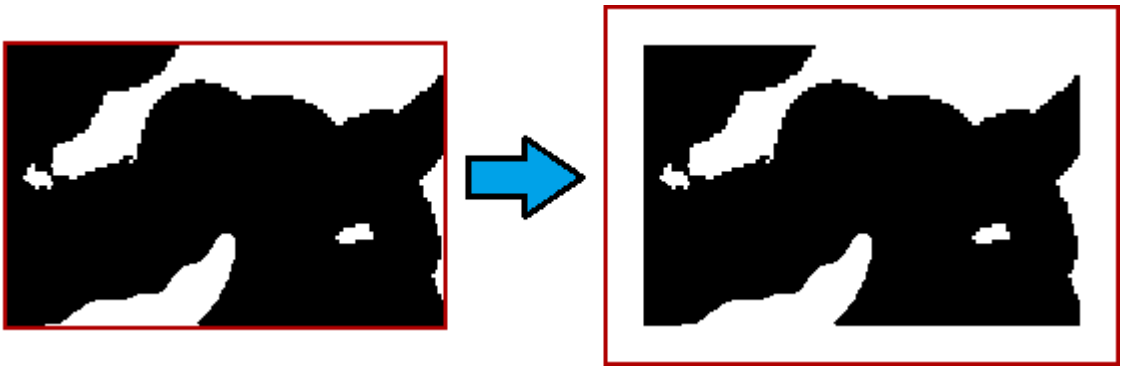
Nästa steg är att applicera en blur-algoritm vilket skapar oskärpa i bilden och på så sätt minskar mängden brus i bilden.

Ett **tröskelvärde** appliceras för att **binärisera** (konvertera pixlarna i en bild till en av två färger, svart eller vit) bilden (se figur 3). Vilket kan beskrivas som att ljusa partier ändras till vitt och mörka partier till svart.

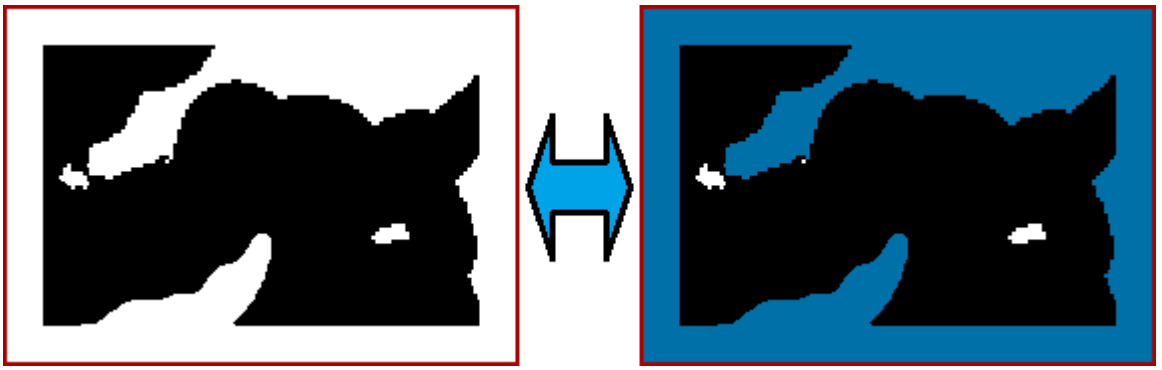


Figur 3. Tröskelvärde applicerat på en bild.

En **kant** av vita pixlar läggs till runt den binäriserade bilden (se figur 4) vilket gör att flera av bildens vita partier sammanfogas (se figur 5).



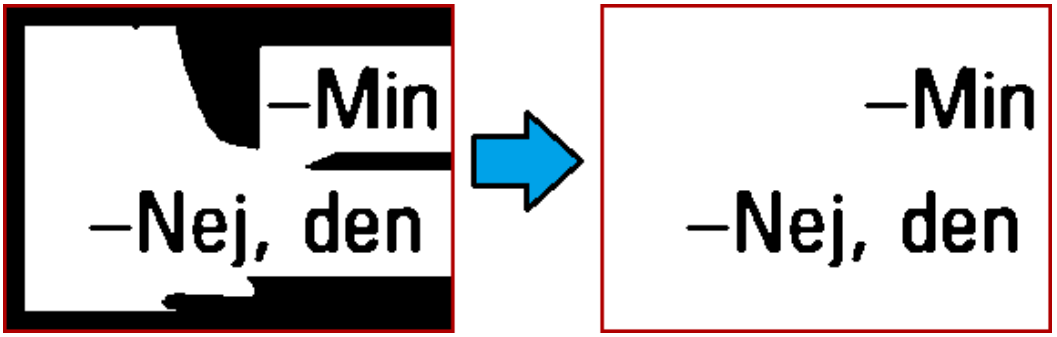
Figur 4. Två bilder där den ena har en kant med vitt runt motivet.



Figur 5. Blå färg visar sammanhängande “vita” partier.

Bildens **färger inverteras** eftersom Tesseract visade sig prestera bättre med svart text mot vit bakgrund.

Flood-fill används med start i övre vänstra hörnet av bilden vilket gör att alla svarta fläckar som är sammankopplade med kanten som lades till i ett tidigare steg ersätts med vit färg. Detta sista steg skapar en mer homogen bild där mycket ointressant information försvinner och skapar färre störmoment för Tesseract (se figur 6).



Figur 6. Två bilder där den ena av bilderna har färre färgfläckar.

3.7 Texthantering

Videodata är mycket varierande och Tesseract läser inte alltid av text korrekt. För att minska antalet felaktiga ord och tecken måste texten bearbetas innan den skickas till appen. Bearbetningen sker på följande sätt:

1. Filtrera bort alla tecken som inte är latinska bokstäver, eftersom Tesseract ofta misstolkar brus i bilden som siffror och specialtecken.
2. Läsa in text under flera bildrutor för att säkerställa att Tesseract läst in korrekt text, kontrollera funna ord mot en ordlista. Ett gränsvärde räknas ut för hur lik texten behöver vara tidigare funnen text.
3. Spara alla ords position samt i hur många bildrutor orden förekommit.
4. Bygg ihop den resulterande texten med hjälp av ords position och hur många gånger orden förekommit där sällan upptäckta ord kasseras.

3.8 Kalibrering och inställningar

För att underlätta debugging samt att för att kunna justera programmets olika parametrar mellan såväl som under körning användes en konfigurationsfil. Kalibreringen är främst ämnad för utvecklare, inte för slutanvändaren. Parametrar som var särskilt användbara var:

- Filsökväg till videofilen programmet skulle använda.
- Gränsvärdet för när en text ska räknas som ny och buffertstorlek i texthanteringen
  (se 3.7 Texthantering).

4. Resultat - Systemkonstruktion

Förenklad beskrivning av bildkodens programflöde:

- Bild läses in från en videofil
- OpenCV används för att lokalisera och klippa ut delen av bilden där texten befinner sig
- Den urklippta bilden behandlas
- Den filtrerade bilden skickas till Tesseract som läser av texten
- Texten skickas till “texthanteringen” som slutligen skickar texten till appen

4.1 Tesseract

Tesseract används för att läsa av texten från bilderna och gav goda resultat, framförallt efter att bilderna förprocessats (se 4.4 Bildbehandling). Nackdelen med Tesseract är körtiden (se 4.2 Tidsprofil). När bildkoden körde på Raspberry Pi 4 skannades 0.19 bilder per sekund. Detta är för långsamt då undertexter oftast blev skannade enbart en gång eller missades. På en snabbare dator skannas 2+ bilder per sekund vilket har har varit tillräckligt för att ge tillförlitliga resultat.

4.2 Tidsprofil

För att få en bra uppfattning om hur bildkoden presterade genererades en tidsprofil med cProfile.[42] Detta bekräftade misstankar som funnits kring att Tesseract var flaskhalsen. Utifrån resultaten från tidsprofil-körningen (se figur 7) kan kan man se att Tesseracts "image_to_string" körs under 53,52 sekunder av tiden vilket motsvarar 97,73 % av programmets totala körtid. Eftersom programmet kräver att undertexter från videon är programmet bromsat av hur snabbt Tesseract kan detektera text.

ncalls	cumtime	percall	filename:lineno(function)
1	54.76	54.76	main.py:1(<module>)
522/1	54.76	54.76	~:0(<built-in method builtins.exec>)
142	53.61	0.3776	main.py:38(imageToText)
142	53.52	0.3769	pytesseract.py:266(run_and_get_output)
142	53.52	0.3769	pytesseract.py:396(image_to_string)

Figur 7. Data där det viktiga är att totala körtiden är 54,76 s och PyTesseract.py(image_to_string) tog kumulativa 53,52 s.

4.3 Begränsning av sökområdet för undertexter

För att snabbare hitta undertexter och undvika detektion av text på skyltar, loggor med mera beskärs bilden innan textlokaliseringen (se 4.5 Textlokalisering). Majoriteten av filmer har textremsan nertill på skärmen och bildkodens standard är att begränsa sökområdet till den nedre delen av bilden. Sökområdet kan ändras via appen till ett av tre alternativ: **“Bottom”, “Top” eller “Whole Picture”** (se 4.9.5 SettingsFragment).

4.4 Textlokalisering

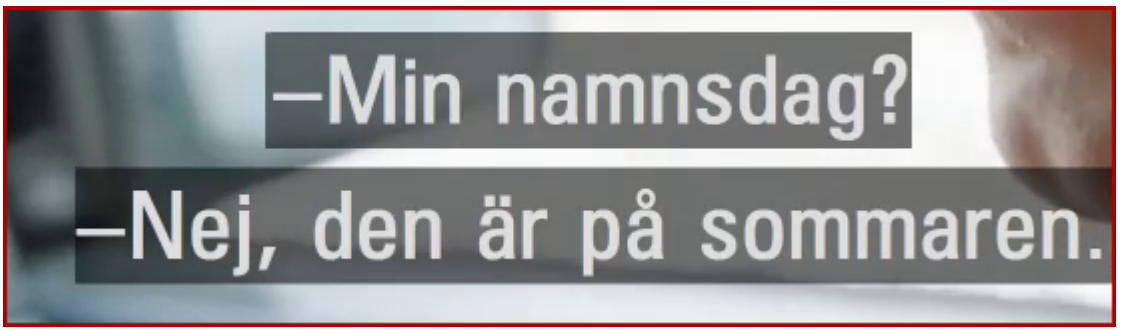
Tesseract ger bättre resultat när bilder på enbart undertexter används. För att kunna ge Tesseract bilder på enbart text behöver texten lokaliseras och skäras ut. För implementationen av textlokaliseringen följdes en guide gjord av Adrian Rosebrock, “OpenCV Text Detection (EAST text detector)”. [43] Metoden gav väldigt goda resultat och med några förändringar kunde den lokalisera undertexter från videofiler i realtid.

Det första steget är att skala om bilderna eftersom EAST kräver att bildstorleken är en multipel av 32, i programmet skalas bilderna om till 320x320. Sedan beräknas förhållandet mellan de nya och de gamla bild-dimensionerna för att användas i ett senare steg. Den nedskalade bilden skickas till OpenCVs blobFromImage vilket behandlar bilden med **Mean subtraction** och matar sedan in bilden i OpenCVs deep neural network med en förtränad EAST textlokaliseringsmodel. Mean subtraction kommer från image subtraction vilket innebär att en bilds pixeldata subtraheras från en annan bilds pixeldata. Mean subtraction använder ett förbestämt statiskt värde. Det statiska värdet är ett medelvärdet som beräknats med hjälp av bilderna som användes för att träna EAST-modellen.[44] EAST textlokaliseringsmodelen returnerar två listor. En med koordinater, vart i bilden texten är lokaliserad och en lista med hur säker modellen är på att det är text. Med hjälp av den datan kan texten detekteras och klippas ut. Dock är texten lokaliserad på den nedskalade bilden, för att extrahera texten från originalbilden skalas textens koordinaterna upp med det förhållande som beräknades tidigare.[43][45][46]

4.5 Bildbehandling

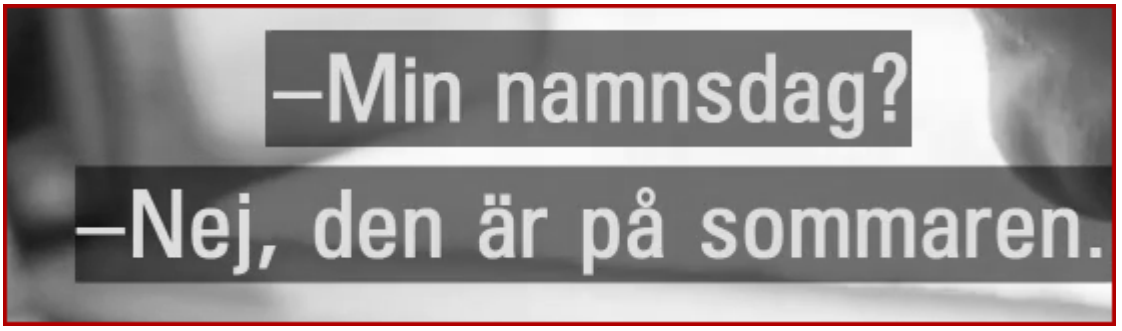
Tesseract har presterat bättre när bildtexten har varit svart och bakgrunden vit. Bilderna i avsnittet är från en körning av bildkoden som illustrerar bildbehandlingen i programmet.

För att få bästa möjliga resultat med OCR behöver ursprungsbilden (se figur 8) behandlas. Bildbehandlingen gjordes i följande steg:



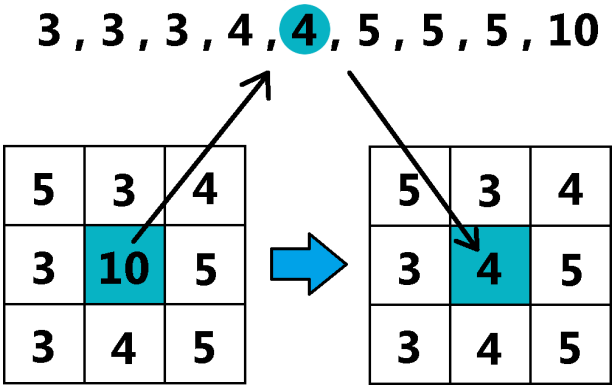
Figur 8. Originalbild, innan bildbehandlingen.

Bilden konverteras till gråskala (se figur 9).

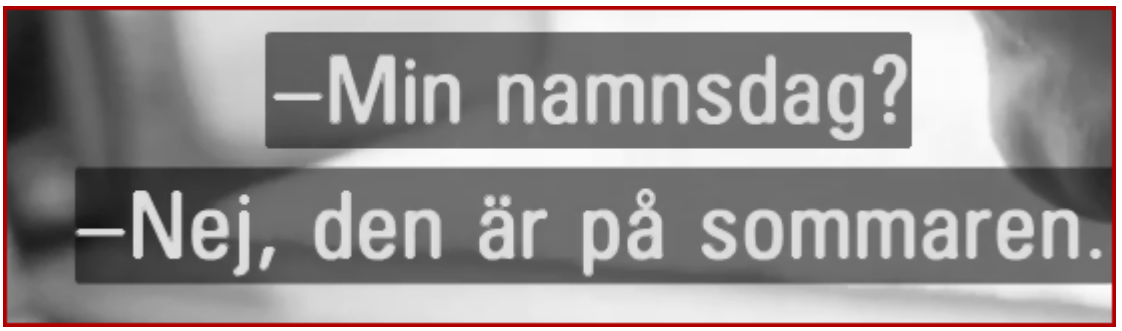


Figur 9. Bild konverterad till gråskala

OpenCVs Median blur [47] appliceras på bilden vilket gör så att alla pixlars värde ändras till medianen av närliggande pixlar (se figur 10) för att reducera brus i bilden (se figur 11).

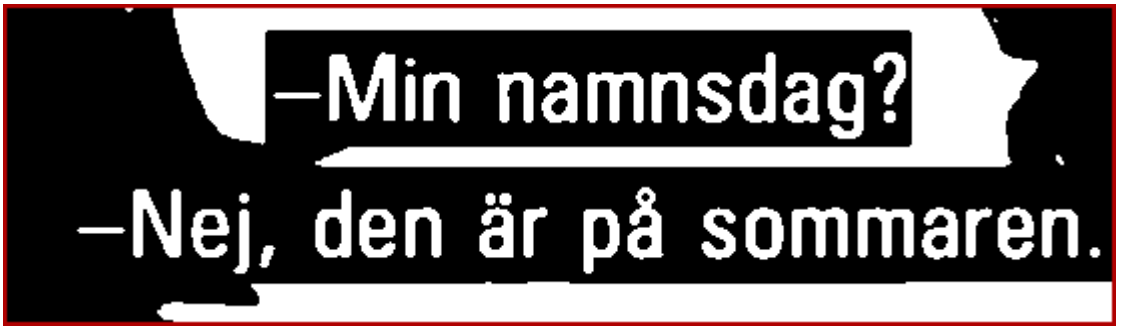


Figur 10. En illustration av hur median blur fungerar för en pixel.



Figur 11. Median-blur applicerad på bilden. Bilden är suddig, små detaljer har reducerats.

Bilden binariseras med hjälp av OpenCV threshold [48] (se figur 12).

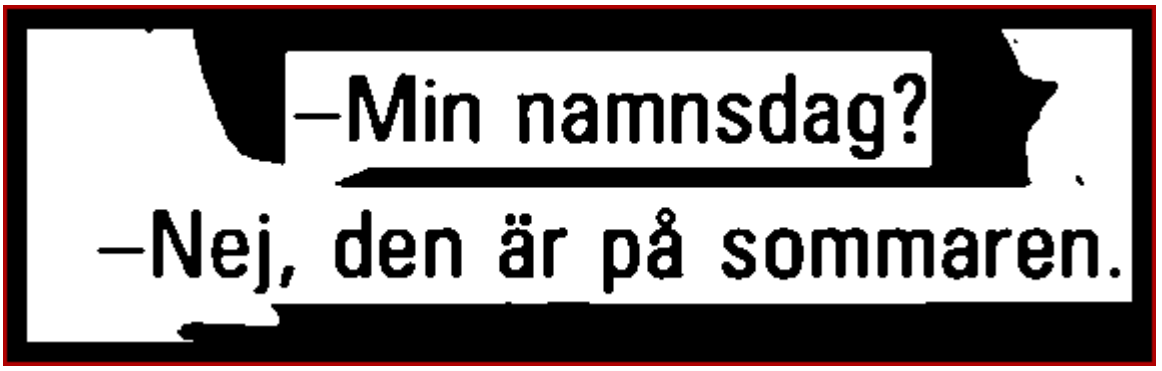


Figur 12. Bild binariserad med tröskelvärde(gränsvärde), pixelvärde blir svart eller vit.

En kant läggs till runt bilden för att i ett senare steg använda flood fill (se figur 13).

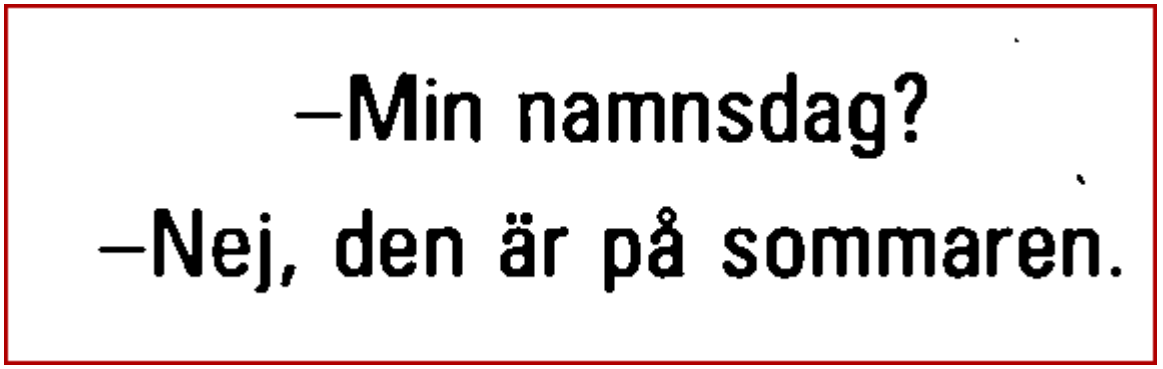
Figur 13. En vit kant runt bilden.

Bildens färger inverteras för att underlätta för OCR (se figur 14).



Figur 14. Inverterade färger

Sist körs OpenCV flood fill [49] (se figur 15).



Figur 15. En helt svartvit bild där texten är svart mot en vit bakgrund. Där ett fåtal störningar av svart färg kan ses på bakgrunden.

4.6 Texthantering

Efter att text har extraherats från bilden filtreras texten med **regex** för att ta bort alla icke latinska bokstäver. Därefter skickades texten till metoden **frame_to_frame_stabilizer**.

Syftet med **frame_to_frame_stabilizer** är att samla information om texten och avgöra om den inlästa texten är ny eller om det är samma undertext som under förra iterationen. Det är för att undvika sändningen av samma text till appen flera gånger och för att säkerställa att rätt text blivit inläst. För att avgöra om texten är ny jämförs den med tidigare funnen text. Texten jämförs med text som finns i den så kallade bufferten som är en lista med text.

Om den lästa texten är tillräckligt lik tidigare buffrad text och bufferten inte är full **registreras** orden, deras position samt hur ofta de förekommer i texten. Registreringen jämför även orden mot en ordbok. Ordboken är samma som **LibreOffice** använder där **PyEnchant** används för att tolka ordboken.[16] Om **PyEnchant** inte finner ordet registreras inte ordet för senare användning. Efter registreringen läggs texten till i bufferten för att kunna användas vid kommande jämförelser. Däremot om texten är lik tidigare funnen text men bufferten är full kallas metoden **stabilize_subtitle** och metoden returnerar en text som skickas till appen.

Skulle texten inte vara tillräckligt lik någon av de buffrade texterna och programmet inte skickat texten till appen än kallas **stabilize_subtitle** och texten skickas. Sedan rensas bufferten, den nya texten registreras och sätts in i den nyligen rensade bufferten.

Se (bilaga 2) för visualisering av förenklat flödesdiagram av texthanteringen.

Buffertens storlek är konfigurerbar, som standard är den satt till 3, vilket betyder att bufferten har plats för 3 undertexter. Gränsvärdet som avgör ifall en text ska räknas som ny är strikt relaterat till hur många procent av bufferten som är fylld. Gränsvärdet är större när bufferten är tom och minskar gradvis eftersom bufferten fylls. Anledningen är för att minska risken för “false positive”, det vill säga att en text räknas som ny när den egentligen ska räknas som “gammal”. Detta minskar risken för att programmet sänder fel text till appen när första extraheringen av en undertext är fel och andra gången korrekt.

Gränsvärdet räknades ut på följande sätt:
a = lägsta tillåtna gränsvärde
b = buffertens storlek
c = antal inlästa texter

$$Gränsvärde \ = \ a \ + \frac{0.9 - a}{b - 1} \ * \ (b \ - \ c)$$

stabilize_subtitle är sista filtrerings-metoden innan texten skickas till appen. Metoden använder de **registrerade** orden för att bygga upp och filtrera bort de ord som sällan upptäckts. För att avgöra vilka ord som kommer skickas används ett annat gränsvärde än det som nämnts ovan där ord under gränsvärdet kasseras. Gränsvärdet beräknas med en procentandel, **WordScoreThreshold** (se 4.7 Konfiguration), av ord-poängen för ordet med högst poäng. Därefter bygger metoden upp meningen med hjälp av ord- och placerings-datan som samlades in i **frame_to_frame_stabilizer** och returnerar texten som därefter skickas till appen.

(se bilaga 3 för ett exempel av texthantering från en programkörning)

4.7 Konfiguration

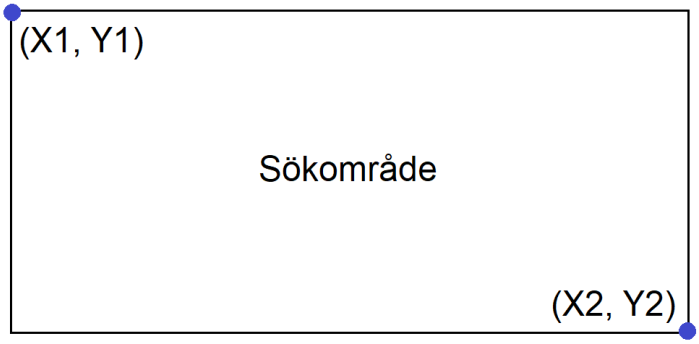
bildkoden använder en konfigurationsfil för att enkelt kunna ändra parametrar och undvika hårdkodade värden. Konfigurationsfilen är indelad i vad som kallas för sektioner och delar där en sektion är ett samlingsnamn för många delar (se figur 16).

```
[Paths]
video_path = C:\Users\Simon\Videos\SVT\DesktopTrim.mp4
east_path = resurser\frozen_east_text_detection.pb
dict_unknown_words = video-text-detection\Dict\unknownWords.txt
custom_dictionary_path = video-text-detection/Dict/customDictionaryWords.txt

[TextDetection]
enchant_dict = sv_SE
VideoSearchLimitX1 = 0
VideoSearchLimitY1 = 0.6
VideoSearchLimitX2 = 0.9
VideoSearchLimitY2 = 1
# How many of the previous frames the current text should be compared to.
# At least one of the previous frames must match to (buffersimilarity*100) % to pass)
# Can take values between 0.01 - 0.89
BufferSimilarity = 0.55
# accepted values 2 - 10
FrameToFrameBufferSize = 5
# How many percentage of the highest scored word each word must have to "pass"
# Accepted values 0.01 - 1
WordScoreThreshold = 0.4
```

Figur 16. Bild över konfigurationsfilen i projektet, där delarna video_path, east_path, dict_unknown_words och custom_dictionary_path kan ses tillhöra sektionen Paths.

- **video_path -** Filsökvägen till videofilen som programmet ska använda. (Filsökvägarna kan vara relativa eller absoluta)
- **east_path -** Filsökvägen till träningsdata för OpenCV (4.5 Textlokalisering)
- **dict_unknown_words -** filsökväg till vart programmet ska spara hittade ord som inte finns i ordboken. Användes främst vid debugging men kan även användas för att analysera och eventuellt lägga till frekvent hittade ord i ordlistan.
- **custom_dictionary_path -** Filsökvägen till den privata ordboken. Det är en textfil där användaren kan skriva in ord som kommer läggas till i ordboken.
- **enchant_dict -** Bestämmer vilken ordbok som kommer användas.
- **VideoSearchLimitX1,**
  VideoSearchLimitY1,
  VideoSearchLimitX2,
  VideoSearchLimitY2 - Vilken del av bilden som kommer skannas för undertexter. Sökområdet är en rektangel där X1, Y1 är övre hörnet och X2, Y2 är det undre hörnet (se figur 17).
- **BufferSimilarity -** Lägsta gränsvärdet för att en text ska räknas som ny i texthanteringen (se 4.6 Texthantering).
- **FrameToFrameBufferSize -** Hur många texter bufferten kan spara. Används i **frame_to_frame_stabilizer** (se 4.6 Texthantering).
- **WordscoreThreshold -** En procentsats som påverkar hur många gånger ett ord måste skannas för att räknas som godkänt. Används i **stabilize_subtitle** (se 4.6 Texthantering).



Figur 17. Förtydligande av sökområde.

4.8 Kommunikation

I bildprogrammet är Bluetooth-kommunikationen uppbyggd med hjälp av biblioteket PyBluez. Pybluez gjorde det smidigt att att implementera Bluetooth-kommunikationen samt att koden fungerar på både Windows och Linux. Nya versioner av Python har officiell support av Bluetooth med hjälp av sockets men är mer svåranvänd.[50]

För att kunna skicka olika typer av information gjordes ett enkelt kommunikationsprotokoll som tillåter kommunikation av undertexter och inställningar mellan enheter.

För att skicka inställningar till enheten med bildkoden ska meddelanden utformas enligt följande: **##config#Section#Part##NewValue**. Första delen, **config**, beskriver att meddelandet är ett konfigurationsmeddelande. **Section** är vilken sektion delen befinner sig i där **Part** är själva delen. **NewValue** är det nya värdet delen ska tilldelas (se 4.7 konfiguration).
Om flera inställningar skall ändras samtidigt skall de separeras med newline(‘\n’) och kan skickas i samma meddelande.
Exempel:
##config#TextDetection#FrameToFrameBufferSize##3 \n
##config#TextDetection#BufferSimilarity##0.45

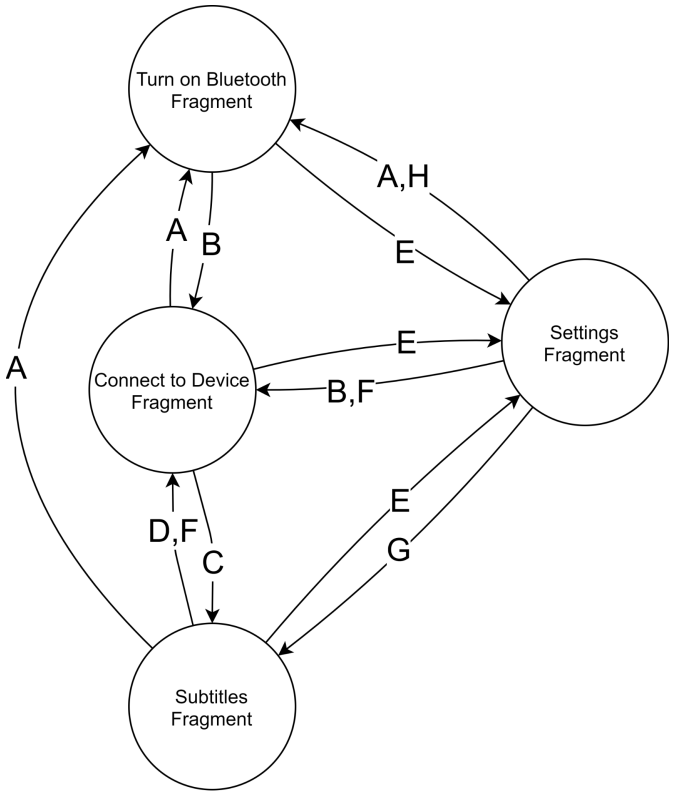
Protokollet stödjer förinställda inställningar som ändrar flera inställningar samtidigt med ett kommandon. Förinställningarna som är implementerade påverkar text sökarean i programmet och är **lower**, **upper** och **full**. **Lower** skannar lägre delen av videorutan(X1=0, Y1= 0.6, X2=0.9, Y2=1), **upper** skannar övre delen av videorutan (X1=0, Y1= 0, X2=0.9, Y2=0.3) och **full** skannar hela videorutan(X1=0, Y1= 0.6, X2=0.9, Y2=1).
Meddelanden för preset ska se ut på följande sätt:
##config#preset#lowert##

Config beskriver att det är ett meddelande av konfigurationstyp, **preset** att det är ett förinställning och **lower** är namnet på förinställningen,

Undertexter som skall läsas upp i appen skickas från bildkoden med taggen ##sub## följt av undertexten. Exempel: ##sub##min namnsdag nej den är på sommaren.

Undertexter och konfigurationskommandon får inte skickas i samma meddelande.

4.9 Android applikation



Figur 18. Diagram över hur användaren navigeras mellan moduler i appen.
A: Bluetooth stängs av → **TurnOnBluetoothFragment**
B: Bluetooth sätts på → **ConnectToDeviceFragment**
C: Enhet ansluts → **SubtitlesFragment**
D: Enhet bortkopplas → **ConnectToDeviceFragment**
E: Inställningar i navigationsfältet trycks → **SettingsFragment**
F: Anslut till enhet i navigationsfältet trycks → **ConnectToDeviceFragment**
G: Undertexter i navigationsfältet trycks → **SubtitlesFragment**
H: Anslut eller Undertexter i navigationsfältet trycks när Bluetooth är av → **TurnOnBluetoothFragment**

Appen är strukturerad med en huvudaktivitet som kan instansiera moduler i form av fragment för att ändra gränssnitt och funktionalitet under körtid. Modulerna används till funktioner som inte kommer visas under hela “appens livstid”. Preferences användes för att kunna spara inställningar gjorda i appen till nästa körning av appen.

Hanteringen av Bluetooth-anslutningar i appen görs av klassen **BluetoothService**. Mycket av koden i **BluetoothService** kommer från exempel-koden “Bluetooth Chat” på Androids Github-sida “connectivity-samples” [51]. Kommunikationprotokollet som används är RFCOMM.

4.9.1 MainActivity

MainActivity är appens huvudaktivitet och grundpelare där övergripande funktionalitet såsom initiering av **BluetoothService**, hantering av mottagna Bluetooth-meddelanden och navigeringsfälts-åtgärder.
Navigationsfältet har tre element vilka det går att trycka på för att navigera mellan appens moduler, elementen är Bluetooth, Undertext och Inställningar (se figur 19).

Figur 19. Appens navigationsfält med numrerade element. 1: Bluetooth, 2:Undertext, 3:Inställningar.

Huvudaktiviteten som alltid kommer att synas i appen har ett navigationsfält överst av skärmen, resterande del av skärmen är reserved för appens utbytbara moduler (se figur 20).

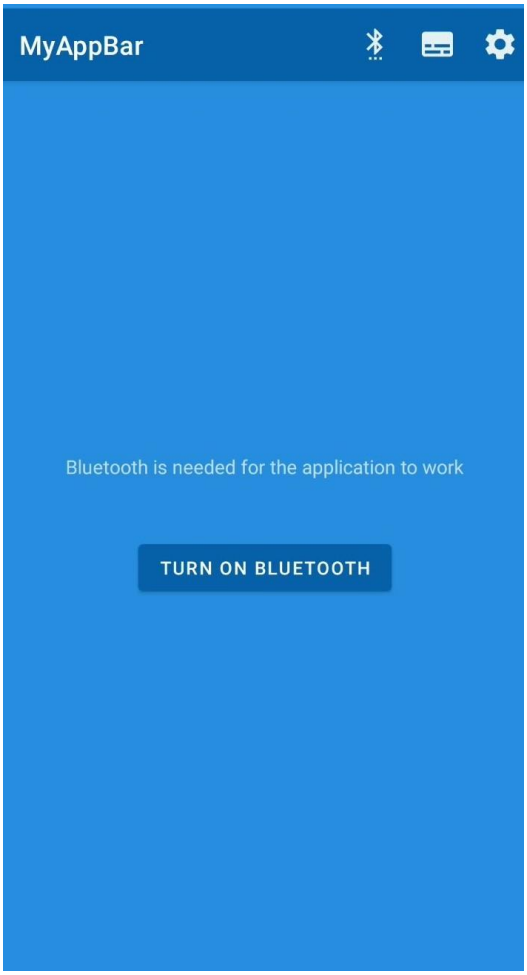
Figur 20. Huvudaktiviteten där moduler blivit inaktiverade.

Appen använder sig av fragment vilka används som utbytbara moduler. I huvudaktiviteten kan en modul i taget vara aktiv där varje modul har en specifik uppgift, till exempel appens **ConnectToDeviceFragment** vilken hanterar sökningen efter och anslutningen till andra Bluetooth-enheter.

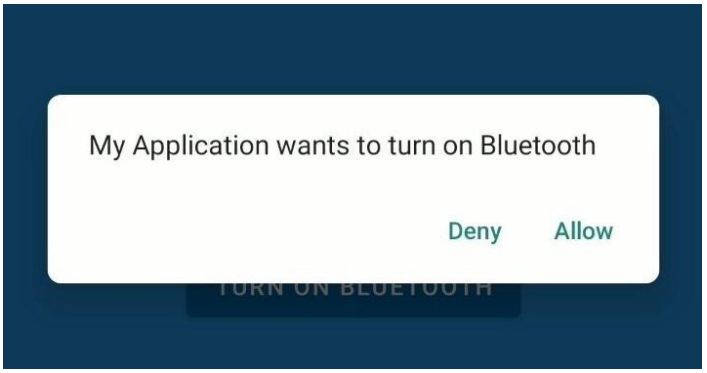
4.9.2 TurnOnBluetoothFragment

TurnOnBluetoothFragment är den modul som visas när när Bluetooth är avstängt. I modulen möts användaren en kort informationstext om varför modulen visas samt en

knapp för att sätta på Bluetooth (se figur 21). Om användaren trycker på knappen görs en tillståndsbegäran av appen att sätta på Bluetooth hos enheten [52] vilket triggar en popup (se figur 22). Om Bluetooth sätts på, antingen via appen eller på något annat sätt så “dirigeras” användaren om till **ConnectToBluetoothDeviceFragment**.



Figur 21. Bild av sätta på Bluetoothmodulen där en knapp för att sätta på Bluetooth visas.



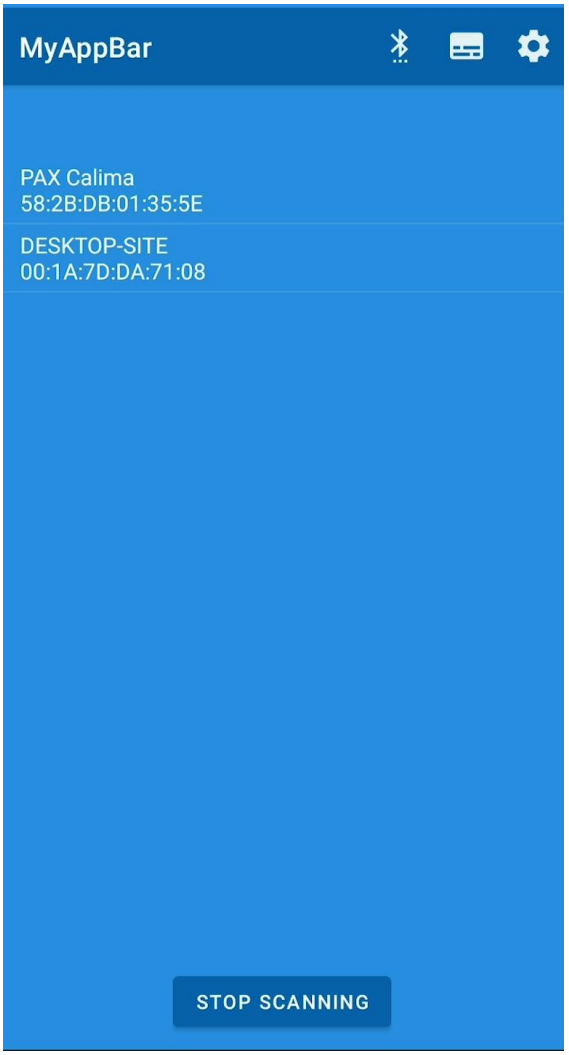
Figur 22. En bild av popup-notifikation i sätta på Bluetoothmodulen vilken frågar användaren om appen tillåts sätta på Bluetooth.

4.9.3 ConnectToBluetoothDeviceFragment

I modulen **ConnectToBluetoothDeviceFragment** möts användaren av en knapp för att skanna efter enheter (se Figur 23).

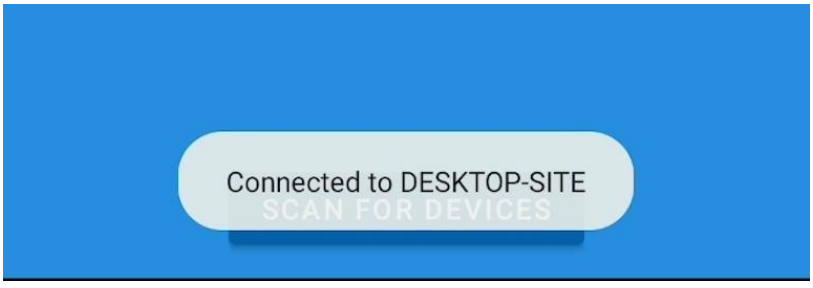
Figur 23. En bild av anslut till enhet modulen. Där en knapp med texten “scan for devices” visas.

Efter att användaren tryckt på knappen “scan for devices” startar appen att söka efter tillgängliga Bluetooth-enheter[53], enheter som upptäcks läggs de till och visas i en dynamisk uppdaterad lista (se figur 24).



Figur 24. En bild av anslut till enhet modulen när två enheter hittats.

När användaren trycker på en av de listade enheterna försöker den ansluta till enheten. När en enhet har etablerat en kommunikation visas en notis på skärmen (se figur 25). Detta gör att undertextsmodulen blir tillgänglig och hamnar i fokus (se 4.9.4 SubtitlesFragment).



Figur 25. En bild av en popup-notifikation i anslut till enhet modulen, som lyder “Connected to DESKTOP-SITE”.

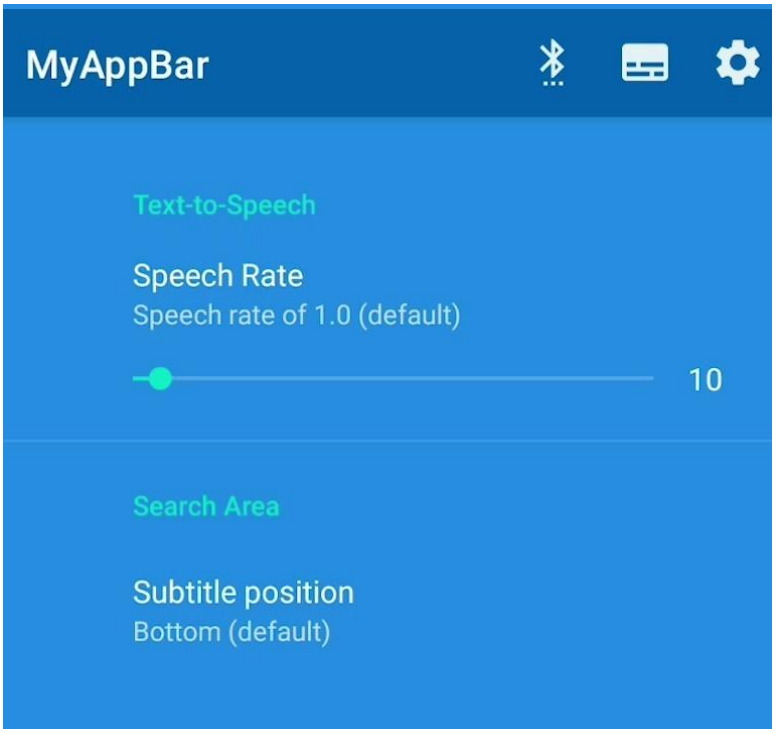
4.9.4 SubtitlesFragment

SubtitlesFragment modulen (se figur 26) visar mottagna undertexter på skärmen och läser upp texten med Androids text-till-tal [54]. Undertexten läses upp med den talhastighet som är satt i inställningsmodulen (se 4.9.5 SettingsFragment).

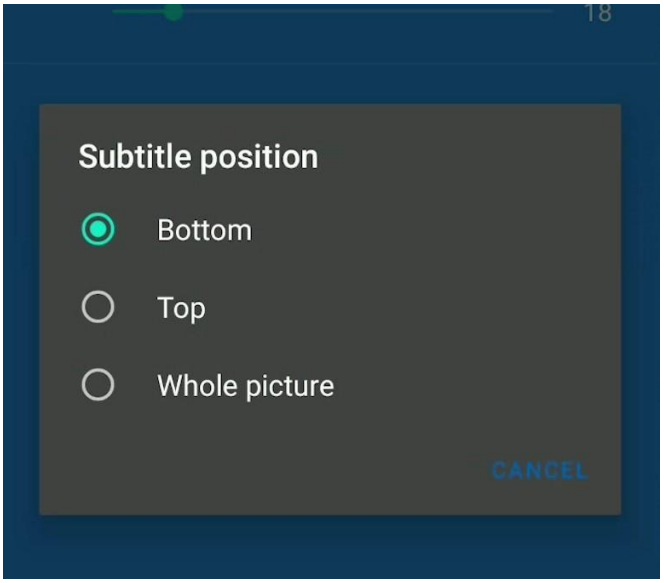
Figur 26. Bild av undertextsmodulen där en undertext visas.

4.9.5 SettingsFragment

Inställnings modulen **SettingsFragment** (se figur 27) är tillgänglig under hela appens livstid. Konfigurationerna som finns tillgängliga är ändring av text-till-tal's talhastighet samt ändra sökområdet för undertexter bildkoden ska använda (se figur 28). När inställningsmodulen stängs skickas sökområdes-konfigurationen med **BluetoothService** om en enhet är ansluten. Inställningarna sparas med Android Preferences [55] för att bevara konfigurationer mellan app körningar.



Figur 27. En bild av inställningsmodulen. Där två inställningar “Speech Rate” och “Subtitle position” visas.



Figur 28. Sökområdes konfigurationsval. Där tre val “Bottom”, “Top” och “Whole picture” visas.

5. Resultat - Delmål

5.1 Ett program som extraherar text från video

Lokalisera text:
Lokalisering av texten fungerar oftast bra, men kan definitivt förbättras. När text lokaliserats beskärs bilden, ibland blir ord avhuggna eftersom textens detekterade position blivit felaktigt. Det utgör inget stort problem då texthanteringen kan hantera och filtrera ut när avhuggna ord avläses av Tesseract, men en förbättrad text-lokaliserare hade fortfarande kunnat förbättra slutresultatet.

Search-Limitation, den som begränsar vart i bilden den söker efter undertext klipper för tillfället ut en del av bilden. Detta gjordes i början för att det var enkelt och ett misslyckat försök till att optimera textlokaliseringen. Search-Limitation kodades innan textlokaliseringen var färdig med antagandet att körtiden kommer påverkas av originalbildens storlek. Med antagandet kan körtiden minskas genom att minska arean som kommer genomsökas. När textlokaliseringen var klar är det tydligt att körtiden inte påverkades av originalbildens storlek/upplösning. Ett bättre alternativ är att istället filtreras text bort om den är utanför sökområdet. På grund av tidsbrist hann det inte bli implementerat.

Extrahera text:
Extraheringen av text gick över förväntan. De steg som ingår i extraheringen av text är lokalisering av text, bildbehandling, Tesseract och texthantering.

Bildbehandlingen kan filtrera ut undertexten med bra precision. När bildrutor har många små detaljer kan det bli brus kvar i bilden vilket försvårar arbetet för Tesseract. Det finns metoder i OpenCV som kan reducera bruset men är beräkningstunga och har markant påverkan på programmets körtid.

Tesseract var det som tog mest tid under en körning och går det inte att påskynda utan hårdvaruacceleration.

Trots att Tesseract inte alltid läser av undertexterna rätt kan texthanteringen filtrera ut och bygga upp undertexter exakt eller väldigt likt undertexten som visas på TV:n (bortsett från specialtecken).

Köra koden på Raspberry Pi:
Koden gick att köra på Raspberry Pi men presterade mycket sämre än vad som förväntades. Den långsamma körhastigheten på Raspberry Pi gör programmet obrukbart.

Skicka undertext till app:
Kommunikationen fungerar bra men det finns problem som inte lösts. Till exempel om bildprogrammet tappar kontakten med appen kommer programmet inte kunna återställa uppkopplingen under körning.

5.2 App som läser upp text

Bluetoothhantering:
Hanteringen av Bluetooth-anslutningar fungerar som det var tänkt från början, men i slutet av arbetet uppmärksammades funktionalitet som hade förenklat för användaren. Till exempel spara tidigare anslutna enheter och inte tappa anslutningen till nuvarande enheten om användaren byter till “anslut till enhet” modulen. Då tiden inte räckte till implementeras inte den saknade funktionaliteten.

Läsa upp undertext:
Implementeringen av text-till-tal var enklare än vad som hade förväntats och gav goda resultat.

Skicka inställningar till textextraheringsprogrammet:
Skicka inställningar från app till bildkoden (extraheringsprogrammet) fungerar och är lätt att utvidga för utökad funktionalitet i både app och bild-kod.

6. Diskussion

6.1 Utvecklingspotential

En stor del i vårt projekt bygger på textigenkänning. Möjlighet att påverka Tesseract är begränsad och den är redan väldigt bra. Däremot finns förbättringspotential för bildbehandlingen och text detekteringen. Enligt rapporten i (2.6 Relaterade arbeten) är deras lösning effektiv och om så är fallet kan det vara värt att byta ut textigenkänningsmetoden. Ifall bilderna i deras rapport är representativa är lösningen för textens position mer exakt än vad vår är vilket kan minimera fel som avklippta ord eller "*identifiera text*" där det inte finns text. Deras lösning är att inte läsa av text som till exempel skyltar eftersom det är text i rörelse. Då vår lösning identifierar all text, inklusive skyltar och annan text som inte önskas översättas.

Texthanteringen kan även förbättras. Ett problem är att den tar bort nummer och kolon. Punkter gör inte så mycket men kolon och nummer kan orsaka problem. Om en tid angiven i digital tid t.ex. "14:15" kommer den tar bort både numren och kolon. Ett annat potentiellt problem kan vara **feta undertexter**. Det är inte ett problem som påverkat vårt arbete men Tesseract har enligt många källor, inklusive sin egna dokumentation svårigheter att känna igen fetstilt text. Bildhanteringen kan lösa problemet, men en simpel vanlig lösning löser inte problemet. Om vanlig icke fet text kommer kan delar av texter försvinna. Det kan vara en funktion som man kan sätta på och stänga av, men då är det inte lika användarvänligt.

En annan del av projektet som kan förbättras eller snarare sagt förändras är kommunikationen. Som nämnts tidigare var kommunikationen från början planerad att ske över nätverk, men är implementerad med Bluetooth. Om en nätverkskommunikation implementeras behövs ej Bluetoothadaptern vilket är närmre till vår vision om produkten.

Ett modernt sätt att spara undertexter idag är att ha texterna i en separat fil istället för inbränt i videon, ifall användaren har tillgång till texten borde filen kunna användas som källa för undertexten och bli uppläst.

Enheten var planerad att vara "plug'n'play" där det enbart är att koppla in enheten till en videokälla, nätverket och koppla upp sig mot enheten via appen. Programmet stödjer en video-ström, eller borde göra det. Det har varit svårt att testa då hårdvaran inte stödjer HDMI-video input. Då hårdvaran inte stödjer det kan hårdvaran inte vara plug'n'play, det krävs hårdvara som klarar HDMI-input. Även om den hade HDMI-input hade det inte fungerat korrekt, det krävs snabbare hårdvara med hårdvaruacceleration. Programmet behöver också skrivas om för att att testerna ska köras på grafikkortet. Ett ännu bättre resultat kan uppnås genom att göra om allt i ett högpresterande språk som C++. Om det någon gång ska bli en riktig produkt är ett

program i C++ och specialiserad hårdvara designad för ändamålet ett bra tillvägagångssätt för att minimera produktionskostnaderna i längden. Även om en produkt är billig att producera är det svårt att veta om produkten kommer ge någon vinst. Det är en liten marknad, vilket kan vara en anledning till att Go-box kostar så mycket.

bildkoden och appen kan utökas att ge support för fler språk. En funktion som enkelt kan läggas till är en meny i appen för att välja språk för både text-till-tal och bildkoden. Detta gör att den smala marknaden nämn ovan blir lite större.

6.2 Möjlighet till minskad kostnad

Med vad som gjorts i projektet är vi övertygade om att en produkt som är billigare än Go-box är möjlig. Det är lite svårt att avgöra då hårdvaran var olämplig, men Nvidia gör enkortsdatorer i samma prisklass som Raspberry Pi som är specialiserade på hårdvaruacceleration, AI och datorseende. Nästa steg i produktutvecklingen hade alltså varit att hitta tillräckligt bra hårdvara och ifall den inte stödjer HDMI-input, konvertera bildströmmen till något som stöds.

6.3 Slutsats

Allt fungerade inte som det var tänkt från början såsom körningen av bildprogrammet på Raspberry Pi samt att bildprogrammet använder sig av videofiler istället för live-videoströmmar. Utöver det fungerar bildprogrammet och appen bättre än förväntat. Både bildprogrammet och appen upplevs som stabila, där avlästa undertexter sällan blir fel. Trots det är det tydligt vad som kan förbättras såsom hårdvaruacceleration, textlokalisering och utökat språkstöd. Med allt som projektet inneburit och det resultat vi kommit fram till tycker vi att projektet är lyckat.

Referenser

Länk till repot: https://gitlab.com/examensarbete1/aid-visually-impaired.git

[1] Microsoft Azure, “Vad är DevOps?”, 2021. [Online]. Tillgänglig: https://azure.microsoft.com/sv-se/overview/what-is-devops/, hämtad: 2021-06-04.

[2] Nexer, “Om oss”, 2020. [Online]. Tillgänglig: https://nexergroup.com/sv/om-oss/, hämtad: 2021-05-19.

[3] Iris Hjälpmedel, “Iris Go-box - läser upp textremsan på din TV”, 2020. [Online]. Tillgänglig: https://www.irishjalpmedel.se/go-box-laser-upp-textremsan-pa-din-tv/article, hämtad: 2020-12-20.

[4] M. Tom, *Machine Learning*. New York: McGraw Hill, 1997. ISBN 0070428077.

[5] DeepAI, “ Weight (Artificial Neural Network)“, 2021. [Online]. Tillgänglig: https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network, hämtad: 2021-06-04.

[6] R. Smith, "An Overview of the Tesseract OCR Engine," Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), 2007, pp. 629-633, doi: 10.1109/ICDAR.2007.4376991.

[7] R. Smith, Z. Podobny, “Tesseract wrappers”, 2021. [Online]. Tillgänglig: https://tesseract-ocr.github.io/tessdoc/AddOns.html#tesseract-wrappers, hämtad: 2021-05-20.

[8] Python Software Foundation, “Why does Python use indentation for grouping of statements?”, 2021. [Online]. Tillgänglig: https://docs.python.org/3/faq/design.html#why-does-python-use-indentation-for-grouping-of-statements, hämtad: 2021-05-20.

[9] J. Weinstein, “Make Python Hundreds of Times Faster With a C-Extension”, *Medium*, Jul. 2020. [Online]. Tillgänglig: https://medium.com/swlh/make-python-hundreds-of-times-faster-with-a-c-extension-9d0a5180063e, hämtad: 2021-05-20.

[10] N. Batchelder, “The structure of .pyc files”, *Ned Batchelder*, Apr. 2008. [Online]. Tillgänglig: https://nedbatchelder.com/blog/200804/the_structure_of_pyc_files.html, hämtad: 2021-05-20.

[11] OpenCV, “OpenCV”, 2021. [Online]. Tillgänglig: https://opencv.org/, hämtad: 2021-05-20.

[12] R. Kelly, “PyEnchant Introduction”, *“PyEnchant”*, 2011. [Online]. Tillgänglig: https://pyenchant.github.io/pyenchant/, hämtad: 2021-05-20.

[13] BlueZ, “About”, 2016. [Online]. Tillgänglig: http://www.bluez.org/about/, hämtad 2021-05-20.

[14] R. Smith, Z. Podobny, “Tesseract OCR”, 2021. [Källkod]. Tillgänglig: https://github.com/tesseract-ocr/tesseract, hämtad: 2021-05-20.

[15] R. Smith, Z. Podobny, “tessdata_best – Best (most accurate) trained models” 2021. [Källkod]. Tillgänglig: https://github.com/tesseract-ocr/tessdata_best, hämtad: 2021-05-20.

[16] LibreOffice, “LibreOffice Dictionaries”, 2021. [Källkod]. Tillgänglig: https://github.com/LibreOffice/dictionaries, hämtad: 2021-05-20.

[17] Grade, “What is Gradle?”, 2021. [Online]. Tillgänglig: https://docs.gradle.org/current/userguide/what_is_gradle.html, hämtad: 2021-05-24.

[18] R. Bellairs, “What Is Lint Code? And Why Is Linting Important?”, “*Perforce*”, Mar. 2019. [Online]. Tillgänglig: https://www.perforce.com/blog/qac/what-lint-code-and-why-linting-important, hämtad: 2021-05-24.

[19] Android Developers, “Android Studio”, 2021. [Online]. Tillgänglig:https://developer.android.com/studio, hämtad: 2021-05-20.

[20] Kite, “Kite - Free AI Coding Assistant and Code Auto-Complete Plugin”. 2021. [Online]. Tillgänglig: https://www.kite.com/, hämtad: 2021-05-20.

[21] Android Developers, “Kotlin and Android”, 2021. [Online]. Tillgänglig: https://developer.android.com/kotlin, hämtad 2021-05-20.

[22] P. Sagar, “Java Vs. Kotlin: Which One Will Be the Best in 2019?”, *DZone*, Jun. 2020. [Online]. Tillgänglig: https://dzone.com/articles/java-vs-kotlin-which-one-will-be-the-best-in-2019, hämtad: 2021-05-20.

[23] Educative, “What is a boilerplate code?”, 2021. [Online]. Tillgänglig: https://www.educative.io/edpresso/what-is-a-boilerplate-code, hämtad: 2021-05-24.

[24] Android Developers, “Activity Lifecycle”, 2021. [Online]. Tillgänglig: https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle, hämtad: 2021-05-20.

[25] Android Developers, “Activity”, 2021. [Online]. Tillgänglig: https://developer.android.com/reference/android/app/Activity, hämtad: 2021-05-20.

[26] Android Developers, “Introduction to Activities”, 2021. [Online]. Tillgänglig: https://developer.android.com/guide/components/activities/intro-activities, hämtad: 2021-05-20.

[27] Android Developers, “Fragments”, 2021. [Online]. Tillgänglig: https://developer.android.com/guide/fragments, hämtad: 2021-05-20.

[28] Android Developers, “Fragment”, 2021. [Online]. Tillgänglig: https://developer.android.com/reference/android/app/Fragment, hämtad: 2021-05-20.

[29] Raspberry Pi, “Raspberry Pi 4 Tech Specs“, 2021. [Online]. Tillgänglig: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/, hämtad: 2021-05-20.

[30] J. Kardach, “Tech History: How Bluetooth got its name”, *“EE Times”*, Maj. 2008. [Online]. Tillgänglig: https://www.eetimes.com/tech-history-how-bluetooth-got-its-name/, hämtad: 2021-05-20.

[31] *RFCOMM WITH TS 07.10, Serial Port Emulation*, vol. 12, 2012. [Online]. Tillgänglig: https://www.bluetooth.com/specifications/specs/rfcomm-1-2/, hämtad: 2021-05-20.

[32] *Bluetooth Network Encapsulation Protocol (BNEP) Specification*, 2003. [Online]. Tillgänglig: https://www.bluetooth.com/specifications/specs/bluetooth-network-encapsulation-protocol-1-0/, hämtad: 2021-05-20.

[33] X. Zhou et al., "EAST: An Efficient and Accurate Scene Text Detector," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2642-2651, doi: 10.1109/CVPR.2017.283.

[34] Valery G., Jean S. (2020) Detection and Localization of Embedded Subtitles in a Video Stream. In: Gervasi O. et al. (eds) Computational Science and Its Applications – ICCSA 2020. ICCSA 2020. Lecture Notes in Computer Science, vol 12254. Springer, Cham. https://doi.org/10.1007/978-3-030-58817-5_10

[35] Har haft muntlig kontakt med Synskadades Riksförbund. De nämnde att möjligheten till att få hjälpmedel dyrare än Go-box finns, men hjälpmedel evalueras med kostnad mot behov och de har inte hört om någon som fått ett liknande hjälpmedel.

[36] Bruksanvisning Go-box, Sverige: Iris Hjälpmedel AB, u.d. [Online]. Tillgänglig: https://www.irishjalpmedel.se/Dynamics/Documents/def96341-34cf-437a-b064-5aa57dbe99f9/Go-box_bruksanvisning.pdf, hämtad: 2021-05-19.

[37] Google, “Connect hearing aids to your device”, 2021. [Online]. Tillgänglig: https://support.google.com/accessibility/android/answer/9426888?hl=en, hämtad: 2021-05-20.

[38] Medietextarna, “Riktlinjer för undertextning i Sverige”, Journalistförbundet, vol. 1, 2020. [Online]. Tillgänglig: https://www.medietextarna.se/wp-content/uploads/2020/03/Riktlinjer-f%C3%B6r-undertextning-i-Sverige-1-2020.pdf, hämtad: 2021-05-20.

[39] L. Velho, A.C. Frery, J. Gomes, *Image Processing for Computer Graphics and Vision*, vol. 2, London: Springer, 2009. https://doi.org/10.1007/978-1-84800-193-0

[40] R. Smith, Z. Podobny, “Improving the quality of the output”, 2021. [Online]. Tillgänglig: https://Tesseract-ocr.github.io/tessdoc/ImproveQuality.html, hämtad: 2021-05-20.

[41] Stack Overflow, “Using Tesseract to recognize license plates”, 2013. [Online]. Tillgänglig: https://stackoverflow.com/questions/19268648/using-tesseract-to-recognize-license-plates/19418347, hämtad: 2021-05-20.

[42] Python Software Foundation, “The Python Profilers”, 2021. [Online]. Tillgänglig: https://docs.python.org/3/library/profile.html, hämtad: 2021-05-20.

[43] A. Rosebrock, “OpenCV Text Detection (EAST text detector)”, *“PyImageSearch”*, Aug. 2018. [Online]. Tillgänglig: https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/, hämtad: 2021-03-16.

[44] A. Rosebrock, “Deep learning: How OpenCV’s blobFromImage works“, *“PyImageSearch”*, Nov. 2017. [Online]. Tillgänglig: https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/, hämtad: 2021-05-20.

[45] J. Benabderrazak, “OpenCV EAST model and Tesseract for detection and recognition of text in natural scenes”, Apr. 2020. [Online]. Tillgänglig: https://jaafarbenabderrazak-info.medium.com/opencv-east-model-and-tesseract-for-detection-and-recognition-of-text-in-natural-scene-1fa48335c4d1, hämtad: 2021-05-20.

[46] K. Yuan, D. He, X. Yang, Z. Tang, D. Kifer and C. L. Giles, "Follow The Curve: Arbitrarily Oriented Scene Text Detection Using Key Points Spotting And Curve Prediction," 2020 IEEE International Conference on Multimedia and Expo (ICME), 2020, pp. 1-6, doi: 10.1109/ICME46284.2020.9102770.

[47] OpenCV, “OpenCV: Image Filtering | medianBlur()”, 2021. [Online]. Tillgänglig: https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9, hämtad: 2021-05-20.

[48] OpenCV, “Image Thresholding”, 2021. [Online]. Tillgänglig: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html, hämtad: 2021-05-20.

[49] OpenCV, “Miscellaneous Image Transformations | floodFill()”, 2021. [Online]. Tillgänglig: https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#ga366aae45a6c1289b341d140839f18717 , hämtad: 2021-05-20.

[50] Python Software Foundation, “socket — Low-level networking interface”, 2021. [Online]. Tillgänglig: https://docs.python.org/3/library/socket.html, hämtad: 2021-05-20.

[51] Google, “Connectivity Samples Repository | Bluetooth Chat Service” 2019. [Källkod]. Tillgänglig: https://github.com/android/connectivity-samples/blob/main/BluetoothChat/Application/src/main/java/com/example/android/bluetoothchat/BluetoothChatService.java, hämtad: 2021-05-20.

[52] Android Developers, “Request app permissions”, 2021. [Online]. Tillgänglig: https://developer.android.com/training/permissions/requesting, hämtad: 2021-05-20.

[53] Android Developers, “Find Bluetooth devices”, 2021. [Online]. Tillgänglig: https://developer.android.com/guide/topics/connectivity/bluetooth/find-bluetooth-devices, hämtad: 2021-05-20.

[54] Android Developers, “TextToSpeech”, 2021. [Online]. Tillgänglig: https://developer.android.com/reference/android/speech/tts/TextToSpeech, hämtad: 2021-05-20.

[55] Android Developers, “androidx.preference”, 2021. [Online]. Tillgänglig: https://developer.android.com/reference/androidx/preference/package-summary, hämtad: 2021-05-20.

Bilagor

Bilaga 1. Sammanställning av normer för undertexter

Följande regler är en rekommendation, ej strikta regler. Författare till undertexter behöver inte strikt följa dessa, men det rekommenderas då det underlättar för läsaren.

Uppfattade regler för undertexter inom svensk tv:

Undertexter är horisontellt centrerade eller vänsterställda med offset från vänsterkant

Undertexter är nederkanten av bildrutan om inte “pop-up”s dyker upp och då flyttas undertexterna uppåt så mycket som “pop-up”en kräver.

Undertexter är max två rader

Undertexter har vit text med mörk bakgrund med opacitet på 90%

Siffror mellan 1 - 12 skrivs med bokstäver, 13 och över skrivs de med siffror
exempel:
7 –> sju
12 - > tolv
13 -> 13
40 –> 40

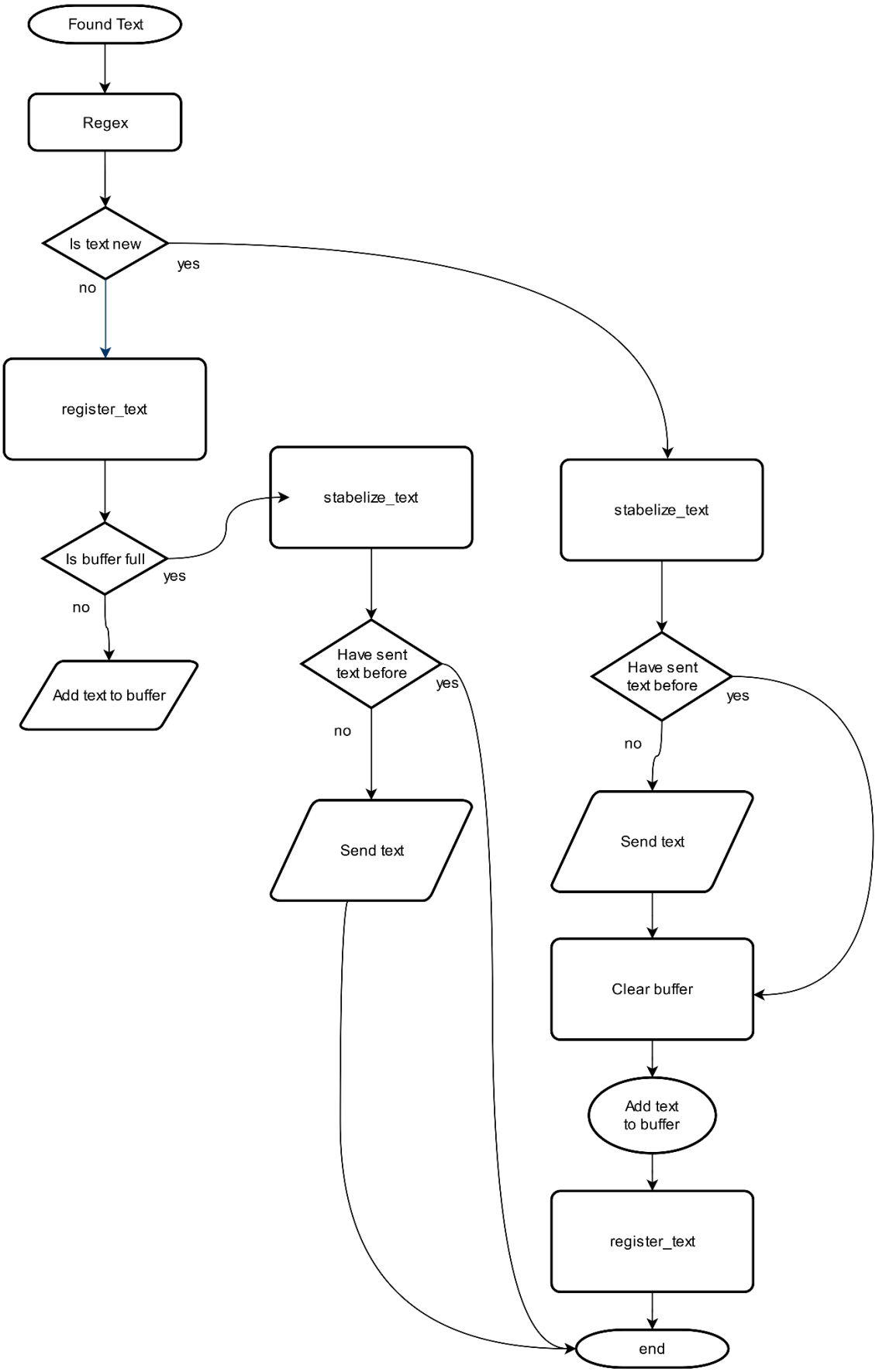
Riktlinjer från Medietexterna:

Undertexters exponeringstid skiljer sig åt beroende på längd. Om undertexten är några enstaka ord räcker det med 1,5 - 2 sekunder, en rad 3 sekunder, två rader 5-6 sekunder. Undertexter bör inte visas i mer än 7-8 sekunder.

Undertexter ska exponeras samtidigt som någon säger något och sedan tas bort några bildrutor eller sekunder efter att samtalet tystnat. Detta gäller så länge inget scenbyte eller annan replik hindrar. Det ska vara 2-4 bildrutor mellan undertexter.

Undertexter bör undvika utfyllnadsord. Det går snabbare att läsa och ger tittaren mer tid att se vad som händer i videon.

Bilaga 2. Förenklat flödesdiagram av texthanteringen



Bilaga 3. Resultat från körning av texthantering

Flödet nedan visar en körning av bild-programmets avläsningar från sex stycken olika bildrutor där undertexten som visades löd: “men också visa att det finns en skörhet i naturen.”. Skillnaden mellan bildrutorna var bakgrunden som ändrades.

Inställningar: **FrameToFrameBufferSize**: 5, **BuffertSimilarity**: 0.55, **WordScoreThreshold**: 0.4
(se 4.7 konfiguration)

Avläst text från Tesseract:

1. | —-men också visa
   4” f—] ma .
   att det finns en skörhet i naturen.

2. men också visa
   att det finns en skörhet i naturen.

3. j-men också visa)
   att det finns en skörhet i naturen.

4. . .v Vanéèd
   . s—-men också visa
   att det finns en skörhet i naturen.

5. men också visa
   att det finns en skörhet i naturen.

6. omen € också visa
   qu
   att det finns en skörhet i naturen.

Eftersom buffertstorleken (**FrameToFrameBufferSize**) är satt till 5 för denna körningen kommer bufferten att vara full när texten hittas en sjätte gång. När detta händer kallas metoden **stabilize_subtitle**, vilken returnerar: “men också visa att det finns en skörhet i naturen” som sedan skickas till mobilen.