Objektgjenkjenning med ABB

Kenneth R. Eikrehagen
Oslo Metropolitan University
Electrical Engineering
Norway, 3048 Drammen
Email: s331475@oslomet.no

Lars-Erik Ulvund
Oslo Metropolitan University
Electrical Engineering
Norway, 0357 Oslo
Email: s331432@oslomet.no

Alexsander Kristensen
Oslo Metropolitan University
Electrical Engineering
Norway, 0160 oslo
Email: s331477@oslomet.no

Erik Johansson
Oslo Metropolitan University
Electrical Engineering
Norway, 0160 Oslo
Email: s330317@oslomet.no

***Sammendrag*—Denne prosjektoppgaven inneholder nesten to prosjekter da roboten som ble brukt til første prosjektet ble ødelagt etter to uker. Gruppen ble godt kjent med NAO sitt brukergrensesnitt gjennom dens egne programmerings verktøy "Choreographe" og Aldebran sitt naoqi-bibliotek i Python 2.7. NAO roboten ble programmert til å finne og gå mot en rød ball før den ble ødelagt.**

Det endelige prosjektet har ved bruk av en industriell robotarm fra ABB og objektgjenkjenning utarbeidet en løsning som sorterer geometriske figurer etter fasong. For å få til dette er det benyttet Python programmering med bruk av OpenCV som henter video fra et kamera som er festet i taket over robotarmen. Socket programming er brukt som kommunikasjonsledd mellom Python og RobotStudio til å sende informasjon om figurtype og dens posisjon. Dette gjorde det mulig for robotarmen å plukke opp figuren og sortere den til korrekt endestasjon.

I. Introduksjon

Dette prosjektet har blitt inspirert av hvordan kunstig intelligens (KI), tingenes internett (IoT) og skytjenester har påvirket robotteknikk, samt hvordan dette kan fortsette å påvirke denne industrien. På leting etter et prosjekt som kunne passe til dette stakk NAO roboten som hører til i laboratoriet for Automatisering, Robotikk og Intelligente Systemer (ARIS-lab) seg frem som et godt utgangspunkt. NAO er konstruert for å skape forståelse om hvordan KI kan brukes i sammenheng med robotteknikk der internett blir brukt som kommunikasjonskanal. To uker inn i prosjektet ble dessverre NAO-roboten diagnostisert til å være defekt, og prosjektet måtte omstilles for å løse en annen oppgave.

Ny prosjektoppgave beholdt omtrent samme fokus. Kombinasjonen KI og robotteknikk med kommunikasjon gjennom et IP-nettverk. En god måte å implementere dette var ved å kombinere et kamera, socket programmering over Transmission Control Protocol (TCP) og en industriell robotarm fra ABB.

Desember 07, 2020

A. NAO prosjekt

I dette prosjektet skulle NAO roboten undersøke synsområdet sitt etter en ball, gå bort til denne ballen, plukke opp ballen og kaste den. Hvis tiden hadde tillatt det skulle den også respondere på stemmekommando.

Prosjektets tre utfordringer:

1) NAO skulle kjenne igjen ballen og gå bort til den
2) Plukke opp ballen og kaste den
3) Reagere på stemmekommando

Dessverre ble roboten diagnostisert til å være defekt to uker inn i prosjektperioden. På dette tidspunktet var første utfordring unnagjort, og en løsning til neste steg var under utarbeidelse. Mye tid gikk med for å finne kilder til hvordan å kunne programmere roboten med Python 2.7 og brukergrensesnittet til Python-NAO-biblioteket (naoqi). Biblioteket naoqi hadde mange gode funksjoner som kunne brukes i prosjektet, med større fokus på robotteknikk enn KI algoritmer. Koden som ble produsert under dette prosjektet finner man i tillegg seksjon A.

B. ABB prosjekt

ABB-robotarmen på ARIS-lab er utrustet med et sugekoppverktøy, og det er montert et kamera rett over robotarmen. Problemstillingen å bruke objektgjenkjenning sammen med robotarmen til å sortere forskjellige geometriske figurer ble raskt unnfanget. De viktigste utfordringene identifisert med dette prosjektet var:

1) Finne en måte å bruke dataen fra kameraet i taket
2) Detektere og finne forskjellige geometriske figurer
3) Få robotarmen til å bevege seg til riktig figur

Videre undersøkelser viste at den beste løsningen var å bruke Python 3.8 (heretter referert til som Python) sammen med OpenCV-biblioteket til å anvende kameraet i taket for objektgjenkjenning. Programvaremessig er Python og RobotStudio brukt for å bevege robotarm; med socket programmering som kommunikasjonsledd. De geometriske figurene som ble brukt for å teste algoritmen var trekant, firkant, sekskant og sirkel.

II. Teori

A. NAO robot

Aldebaran Robotics har laget en liten "humanoid robot" kalt NAO som er designet for å samhandle med mennesker. Den kan gå, danse, snakke og kjenne igjen fjes, stemme og objekt. Denne roboten har nå kommet til sjette generasjon og blir brukt i undervisning, forskning og helsevesen [1]. Den har 25 grader av frihet, 7 berøringssensorer, 4 retningsbestemte mikrofoner og høyttalere, stemmegjenkjenning, 2 2D kameraer og en åpen og fullt programmerbar plattform [2].

B. Objektgjenkjenning

Objektgjenkjenning er en datateknologi som benytter datasyn og bildebehandling for å oppdage objekter av en bestemt klasse, f.eks. mennesker, biler eller sykler i digitale bilder og videoer. For å gjenkjenne objekter benyttes det maskinlæring.

1) Maskinlæring: Prosessen med å bruke matematiske datamodeller til å hjelpe en datamaskin med å lære uten direkte instruksjon kalles maskinlæring (ML) [3]. Dette kan være "enklere" tilnærming som baserer seg på forhåndsdefinerte egenskaper eller "kjennetegn" som sier noe om hva det letes etter og hvordan det skal finnes, hvor dataen etterpå blir klassifisert ved å bruke f.eks. SVM (Support Vector Machine).

C. OpenCV

Et programvarebibliotek som ofte blir brukt for konstruksjon av kildekode for maskinlæring og datamaskinsyn er OpenCV. Biblioteket er utviklet for å tilby en felles infrastruktur for datasynapplikasjoner og for å akselerere bruken av maskinoppfattelse i kommersielle produkter. OpenCV sitt grensesnitt inkluderer C++, Python, Java og MATLAB og inneholder over 2500 algoritmer innenfor maskinlæring og datasyn [4].

D. ABB robot arm

ABB robotarm IRB 140 er en relativ liten, men kraftig robotarm som jobber med seks akser. Robotarmen har en rekkevidde på 810mm til akse fem og en løftekapasitet på maks 6 kg. Det finnes nesten ingen begrensninger for hvor mange forskjellige verktøy som kan monteres på enden av robotarmen, deriblant en sugekopp. Basen til robotarmen kan monteres enten til bakken/underlaget, i taket eller til veggen, noe som gjør robotarmen veldig allsidig [5].

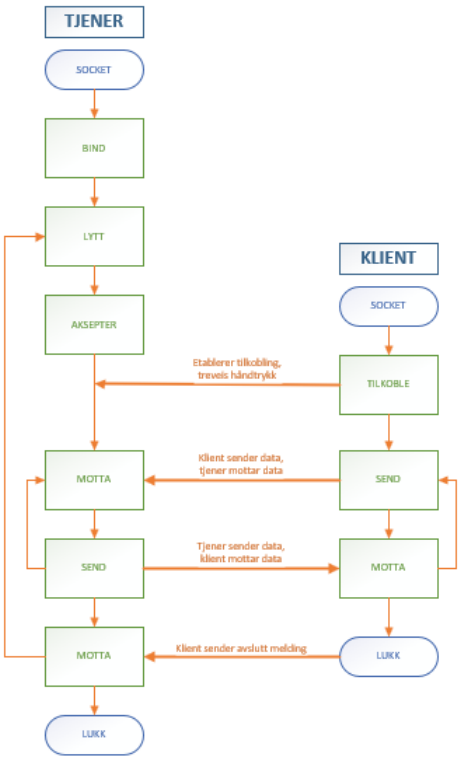
1) Robotstudio: Robotarmen styres av programmer som er laget i ABB sitt program RobotStudio. Programmet overføres til en såkalt FlexPendant, som er koblet til og styrer robotarmen. RobotStudio har en innebygget funksjons som gjør det mulig å simulere ønsket program før det testes på den fysiske roboten.

I RobotStudio er det mulig å modellere gjenstander, figurer og workobjects. En løsning som gjør at simuleringer i programmet enkelt kan gjenspeile virkeligheten. Selve programmet skrives i RAPID, et høynivå programmeringsspråk utviklet for ABB sine industrielle roboter. Programmet bygges opp av flere moduler for å oppnå god struktur. RAPID har en rekke interne biblioteker og funksjoner som gjør prosessen med å konstruere forskjellige programmer relativ enkel [6] [7].

E. Socket programmering

Socket kommunikasjon brukes for å få to noder i et nettverk til å kommunisere med hverandre ved hjelp av meldinger eller pakker. Det kan være et logisk, internt nettverk i en datamaskin, eller et som er fysisk tilkoblet et åpent, ytre nettverk. Det er mulig å sende og motta slike meldinger nesten hvor som helst så lenge man har en internettforbindelse [8].

Nodene defineres som tjener og klient hvor den ene noden (tjeneren) lytter på en bestemt port – den andre (klienten) når ut for å etablere en tilkobling.



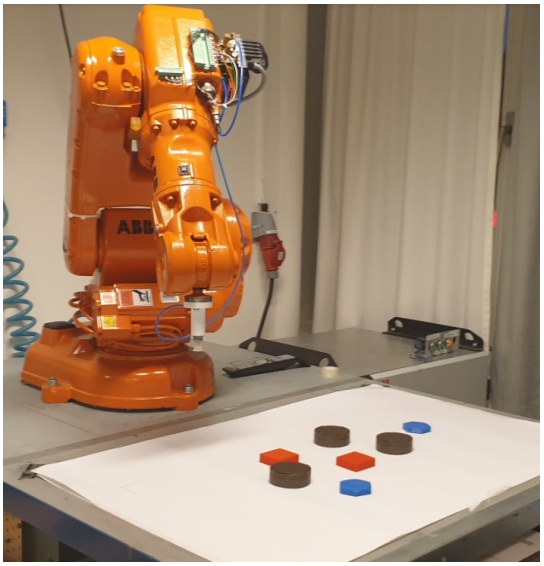
Figur 1. Flytskjema som viser hvordan socet programmering med TCP fungerer

Kommunikasjonsprotokollen som normalt brukes er TCP, hvor man har god kontroll på at meldingene mottas i den formen de ble sendt. Mottakeren kan bekrefte meldingen – og pakker som ikke når frem overføres på nytt av avsender, dette blir godt illustrert på figur 1.

1) Transport Control Protocol (TCP): Både tjener og klient etablerer socket. Tjeneren binder socket-en til spesifisert adresse og port, og lytter passivt etter klientens forsøk på tilkobling. Når tjeneren aksepterer en innkommende forespørsel om tilkobling gjøres et treveis håndtrykk for å bekrefte at kommunikasjonen er etablert. Meldinger kan så sendes fritt mellom nodene. Klientens socket lukkes ofte etter utført operasjon, slik at tjeneren kan gå tilbake til passiv lyttemodus, men dette avhenger litt av hensikt og applikasjon.

III. Metode

I denne prosjektoppgaven er objektgjenkjenning sammen med ABB-robotarmen brukt til å sortere forskjellige geometriske figurer som vist på figur 2 på neste side. OpenCV og Python-programmering er benyttet for gjennomføring av objektgjenkjenning. ABB-robotarm med sugekopp-verktøy er anvendt for den fysiske sorteringen av figurene. Hvordan robotarmen skulle oppføre seg ble programmert gjennom RAPID i RobotStudio. For å etablere kommunikasjon mellom Python og RobotStudio over internett måtte det benyttes socket programmering.



Figur 2. Robotarmen som ble brukt sammen med figurene den skulle sortere

Algoritmen som ble konstruert ble testet med at robotarmen skulle sortere 4 forskjellige geometriske figurer. Figurene bestod av trekant, firkant, sirkel og sekskant. For enkelheten sin skyld ble benevnelsen forkortet til respektive TRI, SQR, CRC og HEX for videre kommunikasjon med robotarmen.

For utstyrsliste se tillegg seksjon B tabell B og B.

A. ABB-robotarm

Til å sortere de ulike geometriske figurene er det benyttet en ABB-robotarm, og programmet som kjøres på FlexPendant er skrevet i RAPID. Robotarmen ble utstyrt med et sugekopp-verktøy som flyttet de forskjellige figurene fra synsfeltet til kameraet og over til en predefinert "endestasjon".

For å verifisere arbeidsområdet til robotarmen ble det plasserte et hvit ark på bordet foran robotarmen. Kameraet i taket ble skrudd på og prosjektert på en pc-skjerm. En firkant figur ble plassert i hvert hjørnene i henhold til hva kameraet kunne detektere og hjørnene ble markert på det hvite arket.

B. Objektgjenkjenning

OpenCV ble brukt sammen med Python for å kunne fange og prosessere video av ønsket arbeidsområde. For å kommunisere med kamera i taket var det også nødvendig å installere en ekstern driver [9], da dette kameraet ikke var "plug and play".

Det som kjennetegner en geometrisk figur er hvor mange kanter den har. OpenCV har en funksjon som kan detektere antall kanter til objekter i bildet. Dette kalles "edge detection" og er en gren som tilhører objektgjenkjenning. Denne funksjonen utførte først objektlokalisering for å finne figurene, deretter ble det telt opp hvor mange kanter figurene hadde. Antall kanter ble deretter analysert for å kjenne igjen hvilke typer geometriske figurer den så på bildet. Ut ifra dette kunne man også kalkulere senterpunktet til hver figur. De foreløpige verdiene måtte deretter skaleres fra pikseldimensjon til metriske verdier. Dette ble gjort ved å beregne forholdet mellom høyden i bildet med tilsvarende høyde på arbeidsområdet. I forbindelse med videre kommunikasjon til robotarmen var det nødvendig å etablere et referansepunkt. Det viste seg at OpenCV analyserte bildet som en matrise ved at den startet i øverste venstre hjørnet, og jobbet seg bortover rad for rad. Referansen for videre kommunikasjon måtte derfor være øverst til venstre i bildet.

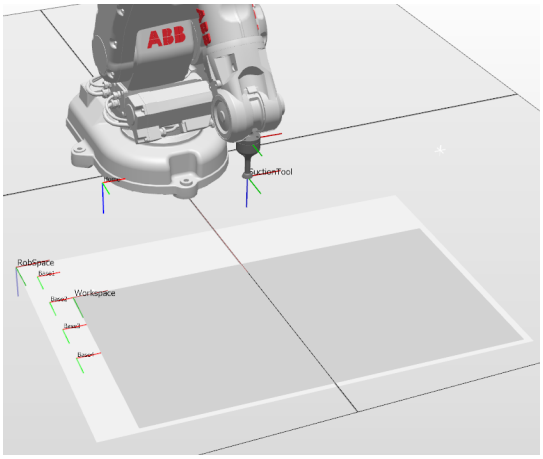
C. Socket programmering

For å oppnå kommunikasjon mellom Python koden som tok seg av objektgjenkjenningen og RobotStudio som beveger robotarmen ble det benyttet socket programmering. Python programmet ble satt opp til å være en klient, og robotarmen ble konfigurert til å være server. Før dataen kunne sendes videre til robotarmen måtte alle numeriske verdier konverteres til datatypen "string". Dette ble gjort i en egendefinert funksjon som tok i mot antall identifiserte kanter, samt piksel koordinater til senter i figurene og returnerte dette som en string. Beskjeden som ble sendt inneholdt benevnelsen av hvilken type figur samt senter koordinatene til figuren.

D. RobotStudio

For at ABB robotarmen skulle plukke opp figurene var det viktig å synkronisere arbeidsområdet til kameraet med arbeidsområdet til robotarmen. Kameraet var montert slik at på videoen som ble vist var hjørnet oppe til venstre for kameraet tilsvarende hjørnet nede til høyre for basen til robotarmen. Programmet som ble laget i Python sendte koordinater med utgangspunkt fra hjørnet oppe til venstre. Der X-aksen strakk seg langs øvre kant og Y-aksen nedover langs bildets venstre kant.

Løsningen som ble valgt var å lage et workobject i RobotStudio som tilsvarer arbeidsområdet til kameraet. Arbeidsområdets vinkel i forhold til arbeidsbordet ble justert ved hjelp av en trigonometrisk funksjon, vinkel $A = tan^{-1}\frac{b}{a}$, og er vist i figur 3.



Figur 3. Arbeidsområde definert i forhold til bildeutsnitt brukt i OpenCV

Siden kameraet er montert i taket oppfatter den figurene kun i 2D. Det vil si at kameraet ikke kan detektere høydene på figurene som vises i arbeidsområdet. Høyden på de forskjellige figurene må derfor pre defineres for hver figur.

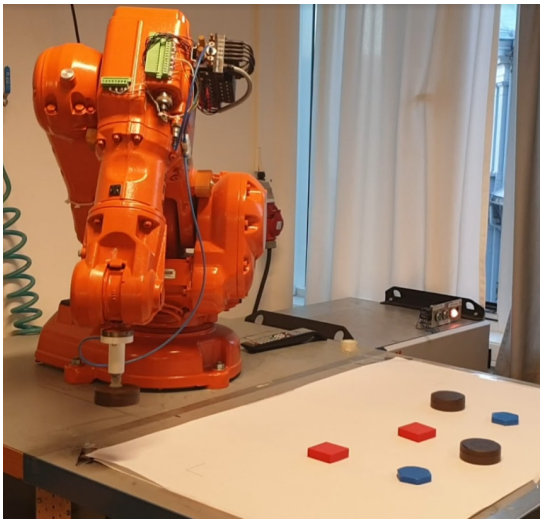
For å unngå at de sorterte geometriske figurene ikke skulle forveksles som figurer som skulle sorteres. Ble "endestasjonene" satt til å være utenfor synsfeltet til kameraet.

E. Figurene

For å vise at prosjektet med objektgjenkjenning kunne kjenne igjen forskjellige geometriske figurer falt valget ned på 4 forskjellig figurer. Hver med forskjellig høyde. En trekant med en høyde på 20 mm, firkant med høyde 15 mm, sekskant med høyde 10 mm og sirkel med høyde 24 mm. De geometriske figurene firkant, trekant og sekskant ble tegnet i Tinkercad [10] og 3D-printet med bruk av PLA. PLA er et termoplastisk materiale som er relativt lufttett. Dette gjør det lettere for sugekopp-verktøyet til robotarmen å plukke opp de geometriske figurene. Sirkelen som er benyttet er en gammel snusboks som er laget av plast. PLA ble valgt fordi det var gratis og tilgjengelig på MakerSpace.

IV. Resultater

Robotarmen brukte et sugekoppverktøy til å løfte de geometriske figurene under sortering som vist i figur 4.



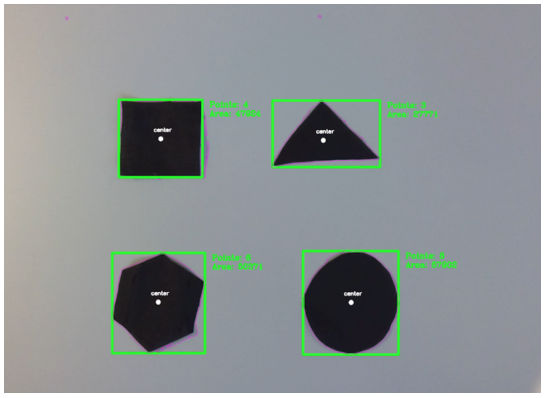
Figur 4. Her er et bilde av robotarmen som starter å sortere figurene

Hvordan programmet vi konstruerte oppfattet det som ble fanget på kameraet ser man på figur 5 på neste side.

V. Diskusjon

A. Simulering og test i RobotStudio

Tekststrengen som ble sendt fra Python programmet ble i første omgang kun lest i sin helhet av programmet i robotstudio. Når den ble mottatt av programmet i robotstudio viste seg derimot at det var en utfordring med at Python programmet sendte samme tekststreng gjentatte ganger. Dette førte til at robotarmen prøvde å hente samme figur flere ganger selv om



Figur 5. Denne figuren viser hvordan vårt program detekterer og gjenkjenner figurene som er i kameraet

den allerede var flyttet. Løsningen på problemet var å sende tekststrengen inn i en rekke if-setninger. If-setningene sjekket om ny tekststreng var lik den forrige tekststrengen. De sjekket også om x-koordinatene var lik forrige x-koordinat eller om y-koordinatene var lik forrige y-koordinat. Hvis en av tilfellene var sant skulle programmet i RobotStudio se bort i fra denne tekststrengen og vente på neste tekststreng.

En kombinasjon av at kameraet ikke var montert vinkelrett på arbeidsområdet og at lyset i rommet var dynamisk endrende, kunne skyggene til objektene noen ganger plukkes opp som egne kanter. Dette resulterte i at en trekant noen ganger ble registrert som en firkant og at sirklene ikke ble detektert i det hele tatt. Dette kunne utbedres noe ved å justere på parametrene for sensitivitet i Python programkoden. På grunn av at kameraet modellerer virkeligheten med piksler var det ikke til å komme unna at en sirkel som ideelt har 0 kanter faktisk får kanter. Med dette kameraets oppløsning ble 8 kanter registrert som en sirkel. Sirkel-problematikken er løst ved å tillate rom for avvik slik at antall kanter for en sirkel kunne være større enn 7, men mindre enn 10. Det måtte også settes en øvre grense slik at ikke alt på arbeidsområde kunne tolkes som en sirkel.

OpenCV støtter hovedsakelig bare plug and play enheter, noe som skapte problemer for kommunikasjon mellom openCV og kameraet. Det viste seg at kameraet hadde flere sensorer som ble knyttet sammen gjennom kamera driveren og programvaren, noe openCV ikke klarte å gjøre. Synsvinkelen til kameraet ble derfor noe dårligere enn ønsket. Bildet fra kameraet var også speilvendt som gjorde ting vanskelig da openCV behandler dataen med utgangspunkt i det øverste venstre hjørnet. Det var derfor nødvendig å speilvende tilbake bildet før bildeprosesseringen kunne anvendes. Speilvendingen av bildet viste seg å være veldig prosesskrevende som resulterte i en treg videostrøm for bildeprosesseringen. En løsning på dette kunne vært å bruke en datamaskin med en kraftigere grafikkprosessor til å håndtere bildeprosesseringen.

B. Test med ABB robotarm

Som følge av gjentatte tester med ABB robotarmen viste det seg at Python programmet detekterte selve robotarmen som en

figur og prøvde å sortere denne. Det ble derfor nødvendig å legge inn en "pause-posisjon" for robotarmen utenfor synet til kameraet, slik at python programmet kunne hente ny posisjon til neste geometriske figur som skulle sorteres.

Det viste seg ganske fort at trekant figuren som var laget fungerte dårlig. Den var for liten og ujevn i kantene. Noen ganger ble den registrert som en annen geometrisk figur og robotarmen prøvde å sortere den til en annen "endestasjon" enn for trekanter. Det viste seg også at størrelsen på trekanten gjorde det vanskelig for sugekoppen å få tak i den. Derfor ble etterhvert trekanten ekskludert fra gjennomføringen av prosjektet.

Mellom hver test i ARIS-laben viste det seg at kameraet i taket hadde flyttet seg litt. Det medførte at arbeidsområdet til ABB robotarmen også hadde flyttet seg. Det var derfor nødvendig med en justering av arbeidsområdet før det var mulig å kjøre programmet. Det hadde vært ønskelig med et kamera som var bedre montert i taket og som står vinkelrett mot basen til robotarmen.

C. Forslag til forbedring av oppgaven

I dette prosjektet ble det brukt et 2D kamera festet i taket over arbeidsområdet. Dette begrenset synsvinkelen til kun å oppfatte hvor objektene lå i xy-planet. Det var dog ikke mulig å oppfatte høyden på objektene. En utbedring på dette kunne blant annet ha vært å montere et ekstra kamera som hadde kunnet oppfatte høyden på objektene slik at høyden ikke hadde måttet forhåndsdefineres.

For objektgjenkjenning kunne det alternativt blitt trent opp en modell til å gjenkjenne figurene i stedet for å gjenkjenne antall kanter. Dette hadde resultert i en mer nøyaktig bilde-gjenkjenning hvor flere objekter kunne blitt implementert, men det hadde samtidig krevd flere arbeidstimer noe det ikke var tilstrekkelig mange av i dette prosjektet.

For å øke bildefrekvensen kunne det blitt benyttet en kraftigere datamaskin. Dette ville forbedret nøyaktigheten og samtidig fikse mye av problemene som oppstod underveis, f.eks. utfordringen ved at Python programmet sendte samme informasjons-streng flere ganger

VI. Konklusjon

Det ble lagt mye tid i undersøkelser rundt det første prosjektet som ble igangsatt: finne riktig bibliotek, nøkkelkode til Choreagraphe, documentasjon om NAO osv. Prosjektet var kommet godt i gang da roboten ble ødelagt. Etter en god idemyldring var det mulig å omstille seg og finne en oppgave som beholdt samme grunntanke. Dette ga en myk overgang til prosjekt nummer to.

OpenCV viste seg å være et meget godt verktøy å bruke for nykommere i bildeprosessering og kunstig intelligens. Emnet *ELVE3610 Robotteknikk* har gitt innføring i å lage funksjoner i Python, samt bli kjent med hvordan funksjoner ser ut. Dette ga en bedre forutsetning til å tolke funksjonene i OpenCV-biblioteket og vite hvordan å utnytte disse. Hadde prosjekt-perioden vært lenger eller hadde det vært mulig å trene en egen maskinlærings modell til å kjenne igjen de geometriske figurene i steden for å bruke kantdeteksjon, kanskje til og med laget egenspesifiserte figurer den kjente igjen.

En av laboratorieoppgavene i kurset innebar å bruke socket programmering som et kommunikasjonsledd mellom roboten og en ekstern datamaskin ved hjelp av funksjoner i RAPID (RobotStudio) og et Python-script. Dette konseptet ble godt implementert i prosjektet og gjorde det mulig å kombinere KI med ABB-robotarmen.

I arbeidet med programmeringen i RobotStudio satt vi på mye god kunnskap som var opparbeidet gjennom lab-oppgavene i faget. Likevel gikk det mye tid til å sette sammen programmodulene og få programmet til å kommunisere på en god måte med Python-scriptet. Ved testing viste seg at den virtuelle verden og den virkelige verden ikke alltid er lik. Det var derfor nødvendig med noe finjustering. Dette kom spesielt frem når vi skulle synkronisere kameraets arbeidsområde til robotarmens arbeidsområde.

Referanser

[1] IEEE, "Nao." https://robots.ieee.org/robots/nao/. Accessed 02-12-2020.

[2] SoftBanks, "Nao[6]." https://www.softbankrobotics.com/emea/en/nao. Accessed 02-12-2020.

[3] Microsoft, "Maskinlæring." https://azure.microsoft.com/nb-no/overview/what-is-machine-learning-platform/. Accessed 04-12-2020.

[4] OpenCV, "Opencv." https://opencv.org/about/. Accessed 04-12-2020.

[5] ABB, "Abb-robotarmen." https://new.abb.com/products/robotics/industrial-robots/irb-140. Accessed 03-12-2020.

[6] ABB, "Robotstudio." https://library.e.abb.com/public/244a8a5c10ef8875c1257b4b0052193c/3HAC032104-001_revD_en.pdf. Accessed 03-12-2020.

[7] ABB, "Robotstudio." https://new.abb.com/products/robotics/robotstudio. Accessed 03-12-2020.

[8] N. Jennings, "Socket programering." https://realpython.com/python-sockets/. Accessed 02-12-2020.

[9] ABB, "Robotstudio." https://realpython.com/python-sockets/. Accessed 03-12-2020.

[10] ABB, "Robotstudio." https://www.tinkercad.com. Accessed 03-12-2020.

TILLEGG

A. *NAO - kode*

NAO snakker og går

```
import sys
PATH = 'path/to/NAO/library'
sys.path.append(PATH)
from naoqi import ALProxy

naoIP = "127.0.0.0"
PORT = 9559

def setupNAO(name):
    proxy = ALProxy(name,naoIP,PORT)
    return proxy

snakke = setupNAO("ALTextToSpeech")
move = setupNAO("ALMotion")
positur = setupNAO("ALRobotPosture")

snakke.say("Hello. Do you want to play?")
#Sett inn voice recognition

move.post.moveTo(0.5,-0.12,-0.28) #.post gjor at man
     kan gjore flere ting samtidig
#move.moveToward(1,0,0)
snakke.say("I am walking one meter")
move.waitUntilMoveIsFinished()
snakke.say("Done walking")
positur.goToPosture("Sit",1.0)


print(move.__dict__) #hva move objektet inneholder
#print(dir(move))
```

NAO lokaliserer en rød ball og går til den

```
motion = nao.ALProxy("ALMotion", naoIP, port)
stand = nao.ALProxy("ALRobotPosture", naoIP, port)
tts    = nao.ALProxy("ALTextToSpeech", naoIP, port)
videoProxy = nao.ALProxy("ALVideoDevice",naoIP, port
    )

#Henter bilde
subscriber = videoProxy.subscribeCamera("demo", 0,
    3, 13, 1)
imageNao = videoProxy.getImageRemote(subscriber)

motion.moveInit()
try:
    stand.goToPosture("StandInit",0.8)
except (ValueError, RuntimeError):
    pass

def locate_ball():
  headMov = NAOconfig.loadProxy("ALMotion")
  i = 0
  a = [-1.5,0,1.5]
  b = [2.0,4.0,6.0]
  isAbsolute = True
  headMov.stiffnessInterpolation("HeadYaw", 1.0,
    1.0)
  targetName = "RedBall"
  diameter = 0.03
  distanceX = 0.03
  distanceY = 0.0
  angleWz = 0.0
  thresholdX = 0.1
  thresholdY = 0.1
  thresholdWz = 1.0
  effector = "None"
  mode = "Move"
  tracker = NAOconfig.loadProxy( "ALTracker" )
  tracker.setEffector(effector)
  tracker.registerTarget(targetName, diameter)
  tracker.setRelativePosition([-distanceX, distanceY
    , angleWz, thresholdX, thresholdY, thresholdWz])
  tracker.setMode(mode)
  t_end = time.time() + 30
  tracker.track(targetName)
  for j in range(3):
        headMov.angleInterpolation("HeadPitch", a[j
    ], b[j], isAbsolute)
        for i in range(3):
            headMov.angleInterpolation("HeadYaw", a[
    i], b[i], isAbsolute)
            i+=1
            if tracker.isTargetLost():
                continue
            else:
                break
        j += 1

  headMov.angleInterpolation("HeadYaw", 0, 8.0,
    isAbsolute)

  if time.time() == t_end:
    tracker.stopTracker()
  tracker.unregisterTarget(targetName)
  headMov.stiffnessInterpolation("HeadYaw", 0.0,
    1.0)

tts.say("Trying to find ball")

tts.say("Try again")
```

B. *Utstyrsliste*

Software

Program	Type
RobotStudio (RAPID)	Programmeringsmiljø
Python	Programmeringsspråk
OpenCV	Bibliotek
socket	Bibliotek

Figur 6. Oversikt over hvilke software vi brukte under dette prosjektet

Hardware

Utstyr	Type	Fabrikant
Robotarm	IRB 140, 6kg, 0.81m	ABB
Kamera	UI-3360CP	iDS
Datamaskin	Laptop	Windows
FlexPendant	Kontroller	ABB
Sugekopp	Verktøy til robotarm	

Figur 7. Oversikt over hvilke hardware som ble benyttet under dette prosjektet

C. *ABB - kode*

Objektgjenkjenning

```
import cv2
import numpy as np
from math import floor

def stackImages(scale,imgArray):
    '''
    Denne funksjonen legger bilder lagvis paa
    hverandre. Det er nyttig
```

```
    hvis man vil se live hvordan programmet
    handteres det den ser.
    Brukes med cv2.imshow()
    '''
    rows = len(imgArray)
    cols = len(imgArray[0])
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    if rowsAvailable:
        for x in range(0,rows):
            for y in range(0,cols):
                if imgArray[x][y].shape[:2] ==
    imgArray[0][0].shape[:2]:
                    imgArray[x][y] = cv2.resize(
    imgArray[x][y], (0,0),None,scale,scale)
                else:
                    imgArray[x][y] = cv2.resize(
    imgArray[x][y], (imgArray[0][0].shape[1],
    imgArray[0][0].shape[0]),None,scale,scale)
                if len(imgArray[x][y].shape) == 2:
    imgArray[x][y] = cv2.cvtColor(imgArray[x][y],
    cv2.COLOR_GRAY2BGR)
        imageBlank = np.zeros((height,width,3), np.
    uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows
        for x in range(0,rows):
            hor[x] = np.hstack(imgArray[x])
        ver = np.vstack(hor)
    else:
        for x in range(0,rows):
            if imgArray[x].shape[:2] == imgArray[0].
    shape[:2]:
                imgArray[x] = cv2.resize(imgArray[x
    ],(0,0),None,scale,scale)
            else:
                imgArray[x] = cv2.resize(imgArray[x
    ], (imgArray[0].shape[1], imgArray[0].shape[0]),
     None,scale,scale)
            if len(imgArray[x].shape) == 2: imgArray
    [x] = cv2.cvtColor(imgArray[x], cv2.
    COLOR_GRAY2BGR)
        hor = np.hstack(imgArray)
        ver = hor
    return ver


def getContours(imgContour, img):
    '''
    Denne funksjonen identifiserer hvilken
    geometrisk figurer bilde har
    ved aa telle antall kanter den ser samt hvor
    mange figurer det finnes i bilde.
    Det blir ogsaa kalkulert senterpunktet til
    figurene. Dette blir tegnet paa bilde
    hvis man vil se hva funksjonen gjoer visuelt.
    Funksjonen returnerer antall kanter til figuren
    nederst til venstre og
    koordinatene til senterpunktet.
    '''
    contours, hierarchy = cv2.findContours(
    imgContour,cv2.RETR_EXTERNAL,cv2.
    CHAIN_APPROX_NONE)
    edges = 0
    center = []

    for cnt in contours:
        area = cv2.contourArea(cnt)
        areaMin = 5000
        areaMax = 35000
        #find center
        M = cv2.moments(cnt)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        if area>areaMin and area<areaMax:
            cv2.drawContours(img,contours
    ,-1,(255,0,255,5))
            cv2.circle(img,(cX,cY),7,(255,255,255 )
    ,-1)
            cv2.putText(img, "center", (cX - 20, cY
    - 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
     255, 255), 2)
            peri = cv2.arcLength(cnt,True)
            approx = cv2.approxPolyDP(cnt,0.02*peri,
    True)
            x,y,w,h = cv2.boundingRect(approx)
            cv2.rectangle(img, (x,y), (x + w, y + h)
    , (0,255,0), 5)
            cv2.putText(img, "Points: " + str(len(
    approx)), (x + w + 20, y + 20), cv2.
    FONT_HERSHEY_COMPLEX, 0.7, (0,255,0), 2)
            cv2.putText(img, "Area: " + str(int(area
    )), (x + w + 20, y + 45), cv2.
    FONT_HERSHEY_COMPLEX, 0.7, (0,255,0), 2)
            edges = len(approx)
            center = (cX,cY)
    return edges, center

def number2string(number):
    '''
    Konverterer nummeriske verdier til string
    og returnerer stringen
    '''
    StrNumber = str(number)
    length = len(StrNumber)
    if length == 3:
        return StrNumber
    elif length == 2:
        return '0' + StrNumber
    elif length == 1:
        return '00' + StrNumber

def pixel2metric(pixel):
    '''
    Her blir piksel verdier konvertert til
    millimeter
    og returnerer millimeter verdien
    '''
    forholdstall = 409.0/1080.0 # mm/pixel
    metric = floor(pixel*forholdstall) # Metric er
    millimeter
    return metric # ceil runder opp til naermeste
    heltall

def ObjectAnalysis(edges, centerPoint):
    '''
    Tar i mot antall kanter som er identifisert samt
     piksel
    koordinatene til senterpunktet. Tallene blir saa
     konvert til
    string og det blir returnert figur
    identificasjon, x og y
    koordinater til senterpunktet som en string.
    Eksempel melding: SQRX050Y233
    '''

    xCoor = centerPoint[0] # X koordinat
    yCoor = centerPoint[1] # Y koordinat

    xCoor = pixel2metric(xCoor) # konverterer til mm
     eller cm
    yCoor = pixel2metric(yCoor) # konverterer til mm
     eller cm

    xCoor = number2string(xCoor) # konverterer til
    string
```

```
    yCoor = number2string(yCoor) # konverterer til string

    shape = ''
    if edges == 3:
        shape = 'TRI'
    elif edges == 4:
        shape = 'SQR'
    elif edges == 6:
        shape = 'HEX'
    elif edges > 7 and edges <11:
        shape = 'CRC'
    msg = ''
    if shape!='':
        msg = shape + 'X' + xCoor + 'Y' + yCoor
    return msg

def imageProcess():
    '''
    Denne funksjonen tar et bilde fra videokameraet idet funksjonen
    blir kalt. Deretter bestemme hvilke figurer som er i bilde
    velge figuren som er nederst til venstre. Saa kalkulerer den
    sentrum av figuren og tilslutt sender tilbake en  string med
    figur type samt x og y koordinater til sentrum
    '''
    food = "" # Initialiserer beskjed som skal returneres
    success, img = cap.read()
    img = cv2.flip(img,0)
    imgContour = img.copy()

    imgBlur = cv2.GaussianBlur(img,(7,7),1)
    imgGray = cv2.cvtColor(imgBlur,cv2.COLOR_BGR2GRAY)

    threshhold1 = 55
    threshhold2 = 50
    imgCanny = cv2.Canny(imgGray,threshhold1,threshhold2)
    kernel = np.ones((5,5))
    imgDil = cv2.dilate(imgCanny,kernel,iterations=1)
    figur,center = getContours(imgDil,imgContour)
    food = ObjectAnalysis(figur, center)
    return food
```

Klient koden

```
from center_contour import*
import socket

#-------- Kamera settings --------#
# Initialiserer opploesningen til kameraet
frameWidth = 1920
frameHight = 1080
FPS = 1 # Frames Per Second
cap = cv2.VideoCapture(2) # Starter kameraet fra COM  port 2
cap.set(3, frameWidth)
cap.set(4, frameHight)
cap.set(5, FPS)
#-------- Socket programmering --------#
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

HOST_IP = '192.168.12.97'
port = 2222
encoding = 'utf-8' # Definerer hvilket bibliotek den  skal bruke for tolking av beskjeden

client.connect((HOST_IP,port)) # Socket oppkobling

print(f"Gained access to {HOST_IP}") # Verifiserer at den har tilgang til verten

while True:
    data = client.recv(1024) #Definerer at vi kan motta 1024bit.
    print("\n"+data.decode(encoding))
    motatt = data.decode(encoding)
    msg = ""

    if motatt == "Feed me!" or motatt == "Feed me! Feed me!":
        msg = imageProcess()
        if msg == "":
            msg = imageProcess()

        client.send(bytes(msg, encoding))
        print(f"beskjed sendt = {msg}")

client.close()
```