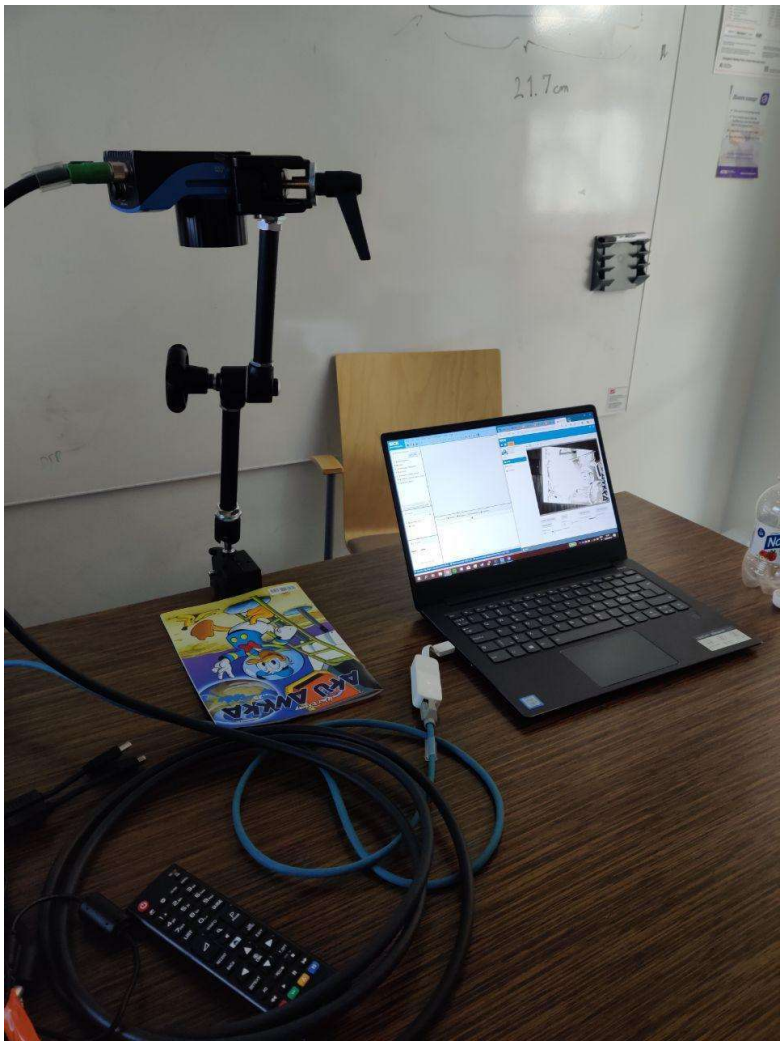
Aalto-yliopisto, Sähkötekniikan korkeakoulu
ELEC-D0301 Protopaja
2019

Raportti

Project #09
Aikakauslehtien lajittelu konenäön avulla



Date: 23.7.2019

Roope Räsänen
Kerkko Karttunen
Mikko Seppi
Juuso Korhonen
Taavi Oja

Information page

Students
Roope Räsänen
Kerkko Karttunen
Mikko Seppi
Juuso Korhonen
Taavi Oja

Project manager
Roope Räsänen

Sponsoring Company
SICK Oy

Starting date
4.6.2019

Submitted date
XX.8.2019

Tiivistelmä

Projektimme tarkoitus oli saada aikaan ohjelma, joka tunnistaa ja jolle on mahdollista opettaa tunnistettavat objektit käyttäen kameran ottamia kuvia syötteenään. Lähtökohtaisesti tunnistettavien objektien oletetaan olevan aikakauslehtien kansia, mutta oikeastaan mitkä tahansa muukin objektit, joissa tunnistettavat muodot ovat oleellisesti yhdessä tasossa, käyvät tarkoitukseen.

Ohjelmointiympäristönä käytimme SICK:in tarjoamaa AppStudiota, joka on suunniteltu toimimaan yhdessä yrityksen ohjelmoitavien laitteiden, kuten Inspector63x-kameran kanssa, jota me käytämme ohjelmassamme. Ohjelmointikielenä AppStudiossa toimii Lua.

Vaatimukset ohjelmallemme olivat valmiudet opettaa lista kuvia aikakauslehdistä ja oikean aikakauslehden tunnistaminen palauttamalla oikea listaindeksi ethernetin välityksellä. Näiden vaatimusten toteutuksessa saimme vapaat kädet.

Toteutuksessa päädyimme hyödyntämään AppStudion valmiita kirjastofunktioita tunnistusalgoritmin rakennuksessa ja AppStudion tarjoamaa graafista käyttöliittymä-rakentajaa opetukseen, tunnistusprosessin seurantaan ja havainnollistamiseen, sekä ajonaikaiseen tunnistusparametrien muuttamiseen ohjelman käyttäjän toimesta.

Vaadittujen opetus- ja tunnistusprosessien lisäksi saimme toteutettua näitä prosesseja tukevia ominaisuuksia, kuten opetusmallien tallennus myöhempää käyttöä varten ja opetusalueen älykäs valinta. Lisäksi toteutimme käyttäjälle mahdollisuuden vaikuttaa opetus- ja tunnistus parametreihin käyttöliittymän kautta, ja näin mahdollisuuden optimoida tunnistus tehokkuutta käytetylle laitteelle sopivaksi. Saimme itsekin optimoitua opetus- ja tunnistusajat melko hyviksi, vaikkakin se ei ollut projektin onnistumisen kannalta olennaista, sillä todennäköisesti todellisessa käytössä suoritusalustana toimiva kamera olisi tehokkaampi.

Abstract

The goal of our project was to build a program, which is able to recognize and to which it is possible to teach a list of objects to be recognized using camera's pictures as an input. Although the objects to be recognized are assumed to be magazine-covers, any other objects, where the identifiable patterns are essentially in one plane, are suitable aswell.

As an IDE (Integrated Development Environment) we used AppStudio offered by SICK, which is designed to work together with the company's embedded systems, such as the Inspector63x-camera, which in use in our project. The programming language of the IDE was Lua.

The specific requirements for our program were the readiness to able to teach a list of magazine-covers and to be able to recognize each of them by outputting an index of the magazine through ethernet. In implementation of these requirements, we got free hands.

In our implementation, we ended up using library-functions while building our recognition-algorithm, and with the usage of the graphical UI-builder for the UI for observation and visualisation of the matching process as well as the ability of the user to change the matching-parameters during runtime.

In addition to the required specifications concerning teach- and matching-processes, we managed to come up with some supporting features, like saving option for the teachmodels and intelligent teach area finder. We also added an option to change the parameters for the teach- and matching-processes, so that the user can get the most out of his or her system. We ourselves also managed to optimize our teach and match-durations to quite good measure, although this was not specified for our project to succeed, since it is likely that the system, which our program was ran on, would be more powerful in real-world use than what was used in our prototype.

Sisällysluettelo

1. Johdanto

Tarkoituksenamme oli tehdä sovellus yrityksen tarjoamalla ohjelmointiympäristöllä, jolle on mahdollista:

1. opettaa lista kategorisoituja referenssikuvia aikakauslehtien kansista
2. tätä listaa hyödyntämällä kategorisoida sille syötetystä kameran livekuvasta oikea lehti (t.s. ulostulona lehden indeksin listasta)

Opetusta ja sovelluksen käyttöä varten meidän tulee rakentaa myös käyttöliittymä, joka onnistuu myös ohjelmointiympäristön työkaluilla. Lisenssit ohjelmointiympäristöön ja testikameran saimme yritykseltä.

Painopiste ei ollut ohjelman suoritustehokkuudessa, vaan tunnistustarkkuudessa, sekä opetus- ja tunnistusprosessien helppokäyttöisyydessä ja seurattavuudessa. Näiden ominaisuuksien toteutuksessa saimme vapaat kädet.

Ajateltuja käyttökohteita ohjelmallemme ovat esimerkiksi lehtipainojen tuotantolinjat tai lehtikioskit. Tuotantolinjalla ohjelma voisi tunnistaa lehtiä, jotka liikkuvat hihnaa pitkin. Tästä tunnistus ohjattaisiin johonkin mekaaniseen lajittelukoneeseen, joka sitten lajittelisi lehdet. Lehtikioskissa työntekijä voisi “skannata” lehdet päivän päätteeksi ja palauttaa myymättömät lehdet lehtitalolle saaden rahaa takaisin.

2. Tavoite

Projektin tavoitteena on kehittää konenäkökameralle sovellus, joka lajittelee aikakauslehtiä. Käytännössä tavoitteena on tuottaa sovellus, joka lajittelee ennalta opetettuja kuvia eri kategorioihin, ja huomioi myös kuvat, joita ei ole aikaisemmin opetettu kameralle ja lajittelee nämä omaan kategoriaan. Lajiteltavat kuvat ovat aikakauslehtien kansista. Kuvia sovellukselle syöttää SICK Oy:n InspectorP -sarjan kamera.

Sovellukselle täytyy pystyä opettamaan lista referenssikuvia lajiteltavista lehdistä. Tavoitteena on myös kohteen mahdollisimman tarkka tunnistus live-kuvasta.

Osana sovellusta on myös Ethernetin välityksellä lähetettävä listan indeksinumero, jota kameran alla oleva lehti vastaa, ja jonka avulla lehdet lajitellaan. Tämä voitaisiin siis lähettää tuotantolinjan mekaaniseen lajittelijaan.

Aikomuksena on kirjoittaa ohjelma LUA -ohjelmointikielellä SICK AppStudio-ohjelmointiympäristössä. Ohjelma käyttää laajaa LUA API kirjastoa, joka tarjoaa kattavan toiminnallisuuden kuvan analysointiin ja graafisen käyttöliittymän rakentamiseen.

Sovelluksen kehityksessä huomioidaan käyttäjäystävällisyys. Uusien referenssikuvien lisääminen tulisi olla vaivatonta ja opetus- ja tunnistusprosesseja tulisi olla helppo seurata. Tätä varten näitä prosesseja tulisi visualisoida jollakin tapaa käyttöliittymässä.

Ohjelman tunnistusnopeutta ei pidetä tärkeänä, koska käytössä olevasta kamerasta on nopeampia malleja, joita oikeassa tuotantomallissa käytettäisiin. Kyse onkin prototyyppi-mallista, josta voi jatkaa sen laajentamista erilaisiin käyttötarkoituksiin. Projektin koodiin ei kuitenkaan tarkoituksella kirjoiteta hidasta koodia, ellei se ole pakollista.

Tarkoituksena on käyttää mahdollinen ylimääräinen aika ylimääräisten, opetus- ja tunnistusprosesseja tukevien ominaisuuksien lisäämiseen, sekä nopeuden parantamiseen optimointien kautta.

Projektia demotaan kytkemällä kamera ryhmäläisen tietokoneeseen. Demon katsojat voivat itse asettaa lehden kameran alle, halutessaan opettaa uuden lehden ja seurata tunnistusprosessia käyttöliittymän näyttävältä nettisivulta.

3. Toteutus

3.1 Ohjelmointi

Ohjelmointi tapahtui SICK:in omalla IDE:llä (Integrated Development Environment) SICK AppStudio. Kielenä käytetään Lua-ohjelmointikieltä, jonka päälle on SICK rakentanut mittavan kirjaston. Koska kirjasto oli niin suuri, tuli dokumentaatioon tutustua mittavasti ohjelmoinnin aikana.

Meillä oli käytettävissä SICK:in referenssidokumentit sekä koodiesimerkit. Näiden olemassaolo helpotti ohjelmointia huomattavasti. Tämän lisäksi meillä oli mahdollisuus ottaa yhteys SICK:in support portaaliin. Käytimme tätä mahdollisuutta muutaman kerran ja saimme vastauksen melko nopeasti suurimpaan osaan kysymyksistä.

Käytetty kieli erosi “normaalista” Luasta hyvinkin paljon. Koodia ei jaeta samalla tavalla moduuleihin, kuten Luassa. Sen sijaan käytetään appeja, jotka voivat kommunikoida toistensa kanssa esimerkiksi eventtien kautta.

Otetaan esimerkkinä meidän ohjelmassamme ethernetin kautta ulkoisen laitteen kanssa kommunikoinnin hoitava tcpip-app. TCPIP-serveri on laitettu koodissamme tähän erilliseen appiin. Tämä selkeyttää koodia ja mahdollistaa uudelleenkäyttämisen toisissa projekteissa. tcpip-app yhdistetään itse pointMatcheriin:

```
--TCPIP:n yhdistäminen--
Script.serveEvent('pointMatcher.OnIndexChange', 'OnIndexChange', 'int')
```

Eventit pitää siis “tarjota” siinä appissa, minkä arvoja halutaan käyttää. Kun halutaan ilmoittaa eventille, että arvo on muuttunut, tapahtuu se seuraavalla tavalla:

```
-- Lähetetään lehden indeksi TCPIP appiin
Script.notifyEvent('OnIndexChange', idx)
```

Nyt sitten TCPIP-appissa eventin ilmoitus otetaan vastaan, koska eventti on luotu siellä:

```
Script.register('pointMatcher.OnIndexChange', handleOnIndexChange)
```

Yllä olevasta koodista huomaa, että aina, kun OnIndexChange eventti laukeaa, kutsutaan handleOnIndexChange-funktiota. Tässä tapahtuksessa handleOnIndexChange lähettää saadun parametrin viimeisimmälle yhdistyneelle clientille:

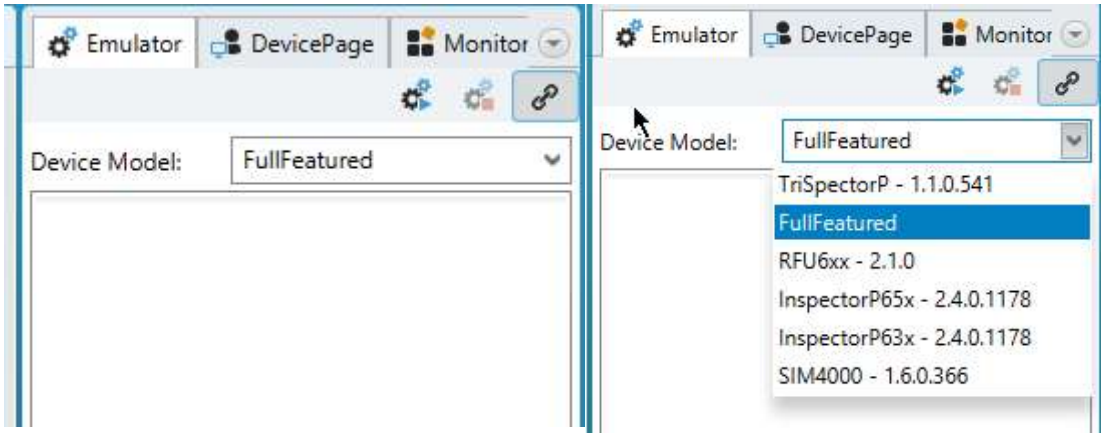
```
local function handleOnIndexChange(index) -- Sends the parameter to the latest connection
  if connection ~= nil and TCPIPServer.Connection.isConnected(connection) then
    TCPIPServer.Connection.transmit(connection, tostring(index))
  end
end
```

Kaiken kaikkiaan ohjelmointi on sujui melko jouhevasti SICK:in antamien työkalujen ansiosta. Vaikka AppStudion käyttö välillä tuotti ongelmia, oli sen kanssa ihan mukava työskennellä. AppStudio tarjoaa ehdotuksia koodiin kaikista kirjastoista suoraan. Ehdotuksissa on lyhyet dokumentaatiot jokaisesta funktiosta. Tämä mahdollistaa koodaamisen ilman, että referenssidokumentti on auki toisella näytöllä.

Ohjelmointi oli todella tärkeässä osassa tässä projektissa, koska käytännössä koko projekti oli pelkkää ohjelmointia. Vaikka koodia ei ole valtavasti, oli siinä paljon töitä, koska kieli oli uusi kaikille. Tämän lisäksi SICK:in kirjastoon tutustuminen vei alussa paljon aikaa ennen siihen tutustumista.

Vaikka AppStudio tarjosikin hyvän staattisen analyysin ja koodiehdotuksia, puuttui siitä monia ominaisuuksia, joita olisimme kaivannut. Esimerkiksi sisäänrakennettu git olisi helpottanut paljon ohjelmointia. Meillä oli alussa ongelma, että AppStudio ei halunnut ajaa koodia, jos workspace oli git-repositio. Ongelma korjattiin poistamalla git-tiedostot kansiosta.

AppStudiossa oli myös monia hyviä ominaisuuksia. Yksi näistä on ehdottomasti emulaattori. Tämän avulla pystyi ajamaan koodia, vaikka ei olisikaan varsinaista laitetta. Emulaattori oli tosin paljon nopeampi, kuin kamera, mikä vääristi suoritustehokkuutta koodia rakennettaessa. Alla on kuva emulaattori-valikosta:



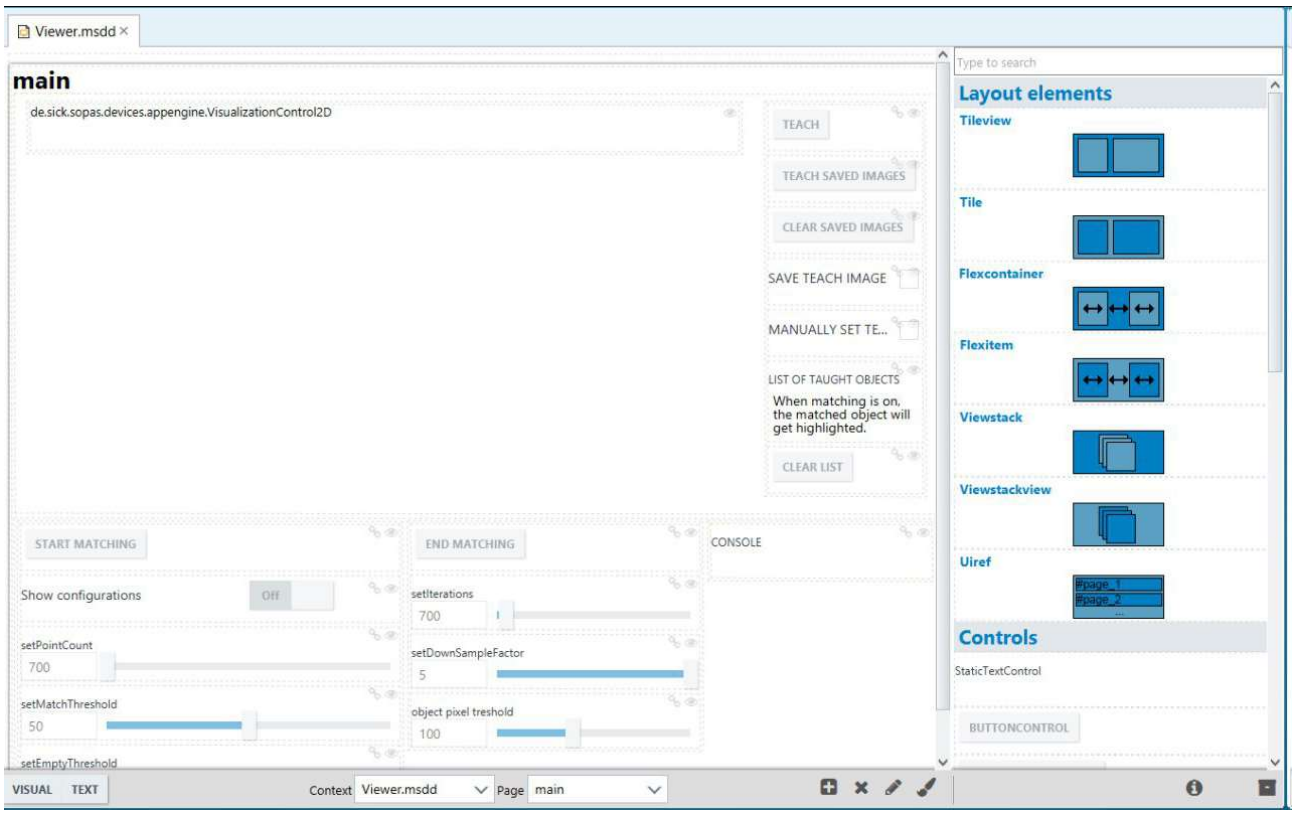
Valinnoissa on monta laitetta, jota voidaan haluta emuloida:

Meillä oli käytössä InspectorP3x, joten valitsimme aina sen, kun varsinaista laitetta ei ollut saatavilla.

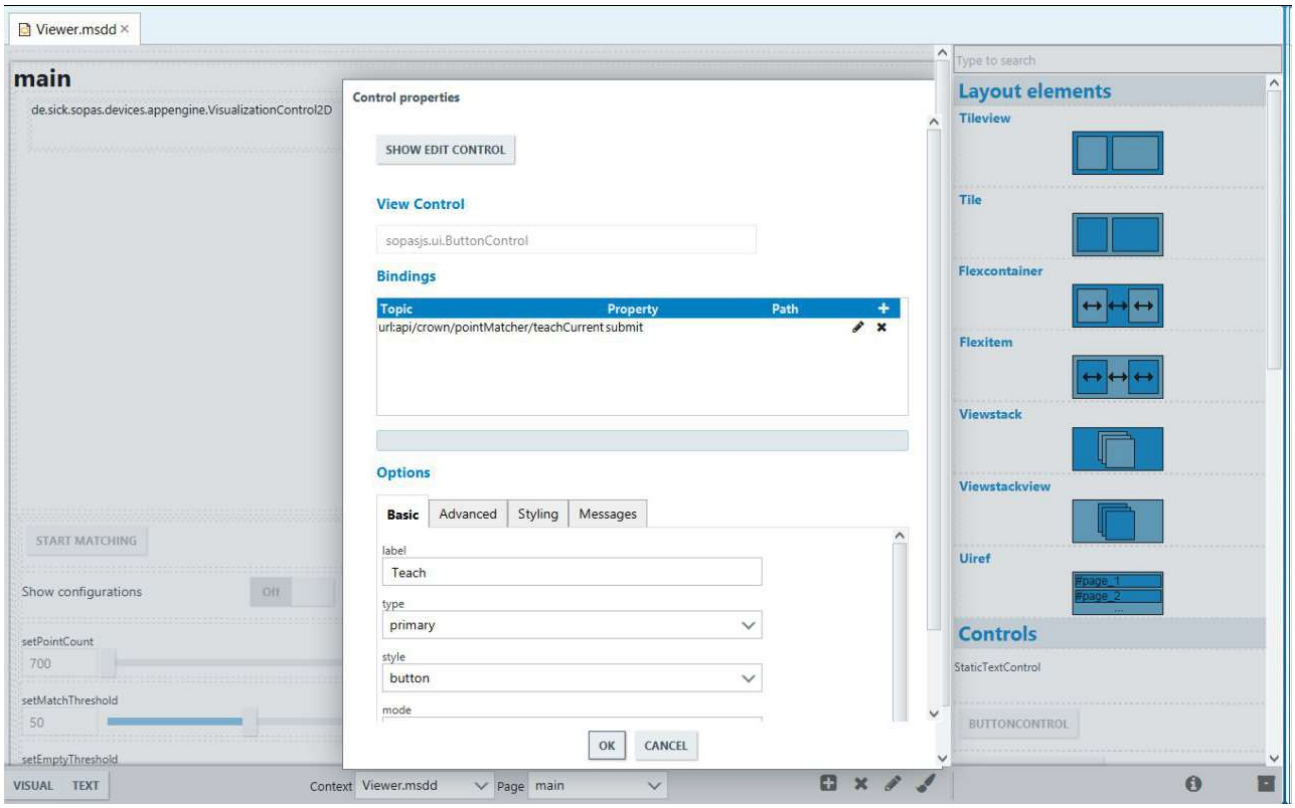
3.1.1 Käyttöliittymän kanssa kommunikointi koodista käsin

Vaikka nappien lisääminen on suunniteltu helpoksi AppSpacessa, oli hieman vaikeaa ymmärtää niiden kanssa kommunikointia koodista. Pitää tehdä monta askelta, että saa käyttöliitymän napin liitettyä johonkin funktioon.

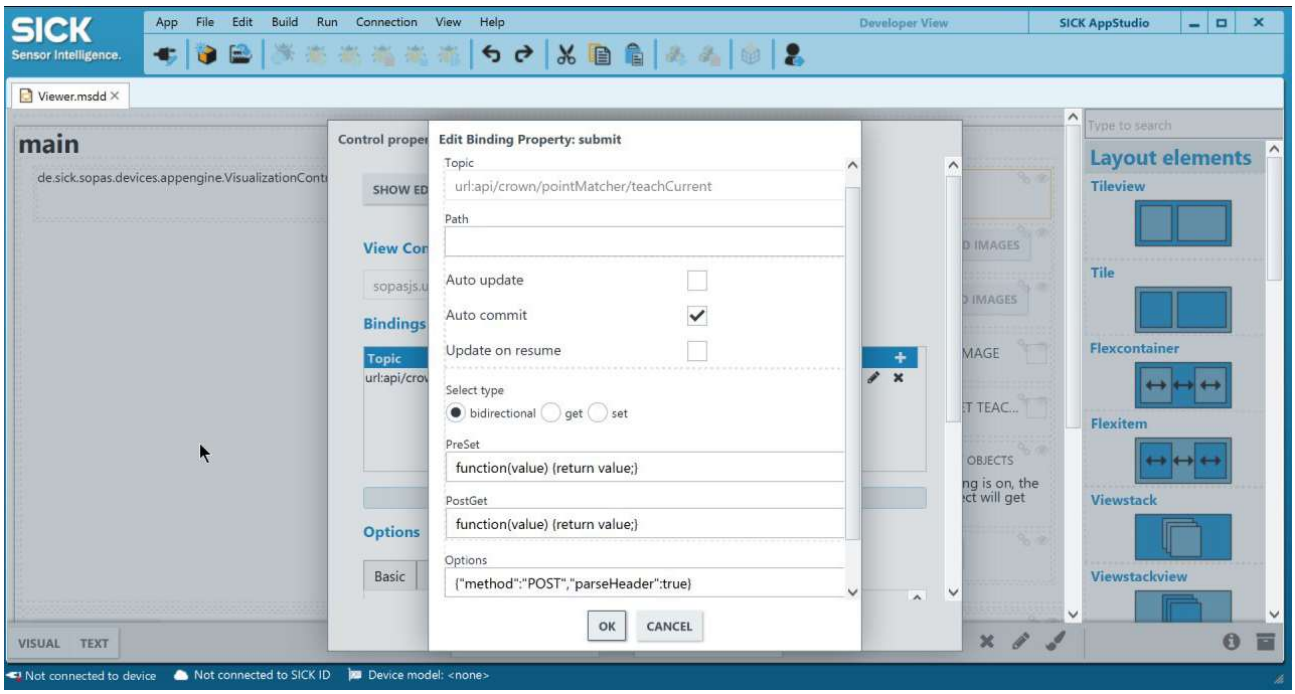
Seuraavissa kuvissa havainnollistamme, kuinka tämä linkitys koodista käyttöliittymään tapahtuu ohjelmassamme. Tämä on toivottavasti avuksi ohjelman mahdolliselle jatkokehittäjälle.



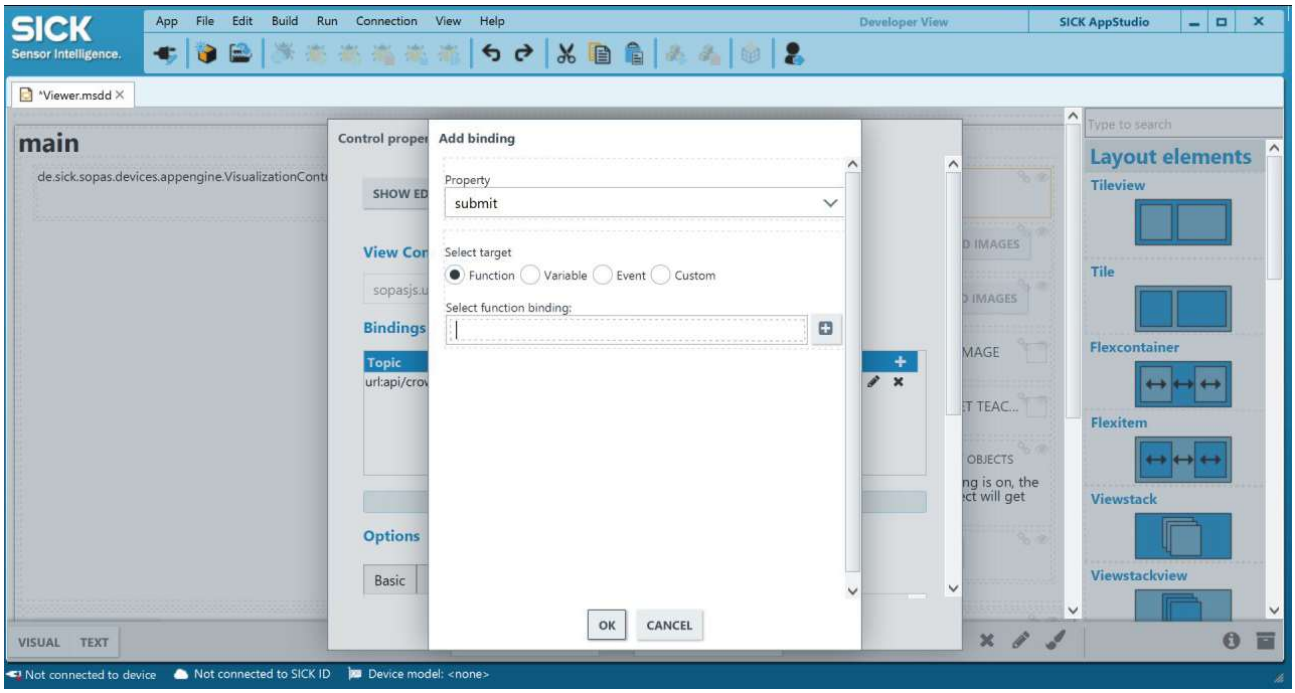
Yllä on kuva AppStudion visuaalisesta käyttöliittymän rakennustyökalusta. Oikealla on lista, josta voi vetää elementtejä, joita itse haluaa sivulleen.



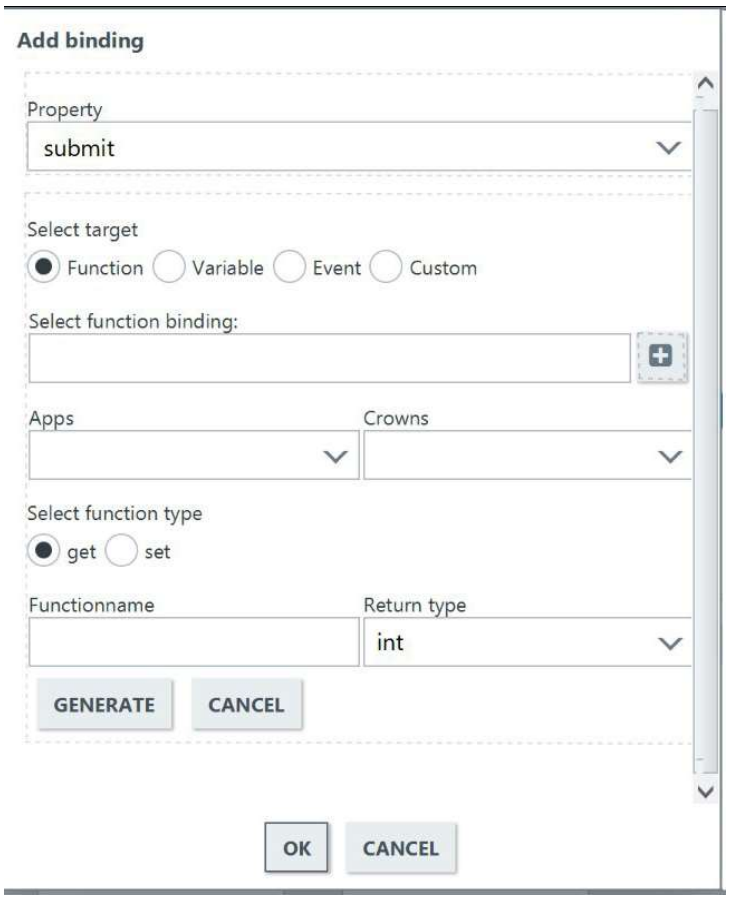
Jokaista nappia voi säätää erikseen monella eri asetuksella. Erilaisia asetuksia voi säätää valikoista. Emme kuitenkaan käyttäneet näitä asetuksia juuri lainkaan. Tärkein asetus on “Binding”, jonka avulla nappi saadaan yhdistettyä funktioon tai eventtiin. Kun avaa bindingin asetukset pienestä kynän kuvakkeesta, aukeaa seuraava näkymä:



Kyseinen näkymä vaikuttaa hieman hankalalta ensimmäistä kertaa. Kuitenkin, kun tutustuu nappeihin, niin asetukset näyttävät loogisilta. Kun halutaan lisätä uusi binding, saadaan seuraava näkymä:

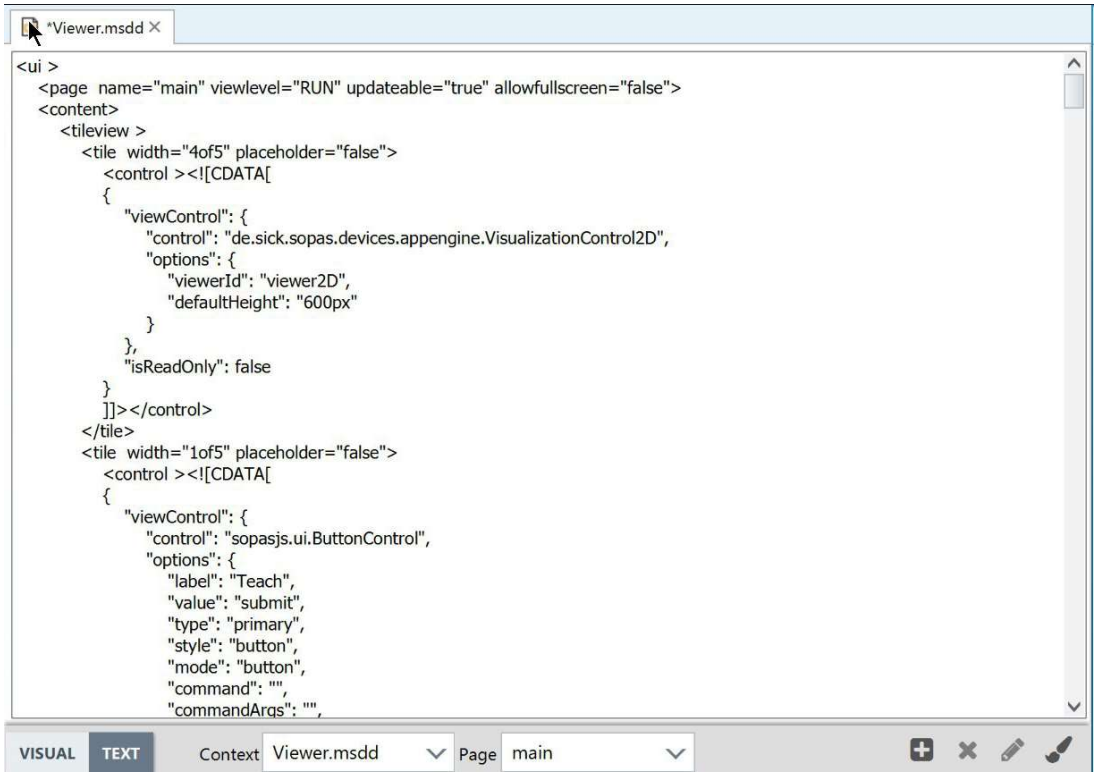


Valitaan siis ensin jokin property, joka täsmää meidän käyttötarvetta. Saatavilla on visible, enabled, writable, value ja submit. Kun halutaan tehdä nappi, valitaan submit. Tämän jälkeen valitaan target. Siinä kerrotaan, että mikä on napin kohde. Vaihtoehtoja ovat function, variable, event tai custom. Käytimme aina joko function tai event targetteja napeissamme. Tämän jälkeen valitaan function binding. Sen voi valita joko suoraan kirjoittamalla sen kenttään, tai painamalla plussaa voi generoida funktion koodiin. Kun painaa plussaa, aukeaa seuraava näkymä:



Täytyy vain valita funktiolle nimi sekä palautusarvon tyyppi. Tämän jälkeen kun painetaan “generate”-nappia, ilmestyy koodiin valmis funktio, johon nappi osoittaa.

Käyttöliittymän rakentamiseen ei ole pakko käyttää visuaalista editoria. On myös mahdollista muokata msdd-tiedostoa suoraan. Koodi näyttää tältä:



Sitten jos haluaa muokata bindingiä jostakin tietystä napista, voi muokata suoraan koodia “helposti”:

```
"binding": [
   {
      "property": "submit",
      "topic": "url:api/crown/pointMatcher/clearSavedData",
      "path": "",
      "autoCommit": true,
      "autoUpdate": 0,
      "updateOnResume": false,
      "direction": "bidirectional",
      "preSet": "function(value) {return value;}",
      "postGet": "function(value) {return value;}",
      "paramPath": "",
      "resultPath": "",
      "init": "",
      "options": {
         "method": "POST",
         "parseHeader": true
      },
      "custom": false
   }
]
```

Tämä on ihan mielipidekysymys, että kumpaa käyttää. Varsinkin jos on ohjelmoinut paljon, saattaa tykätä enemmän suoraan koodin muokkaamisesta sen sijasta, että käyttäisi visuaalista editoria, jota on kieltämättä vähän hankala käyttää. Tämä kysymys myös jakoi ryhmän kahtia. Osa ryhmäläisistä piti enemmän visuaalisesta editorista ja osa halusi muokata suoraan koodia.

Yllä olevassa ui-koodissa laitetaan napin funktioksi clearSavedData. Alla on tämä kyseinen funktio:

```
local function clearSavedData()
  local ret=File.del("/public/teachImg")
  local ret2=File.del("/public/teachSettings")
  local ret3=File.del("/public/teachShapes")
  if ret==true and ret2==true and ret3==true then
    outputConsole("succeeded in deleting previous files")
  else
    outputConsole("error in deleting files")
  end
end
Script.serveFunction("pointMatcher.clearSavedData", clearSavedData)
```

Funktio luodaan normaalisti, mutta luomisen jälkeen se pitää “servata” eli antaa käyttöön muille appeille sekä uille. Tämä prosessi on suoritettu jokaiselle napille erikseen luoden käyttäjälle monipuoliset asetukset.

3.2 Komponentit

Ohjelma koostuu kahdesta pääkomponentista. Ensimmäiseen komponenttiin (pointMatcher) kuuluu ohjelman normaali toiminnallisuus kuten uusien lehtien opettaminen, lehtien tunnistaminen ja muut toiminnot, joita ohjelma tarvitsee käyttöliittymän kanssa kommunikointiin.

Toinen komponenttiin (TCPIP) kuuluu ohjelman ulkoisen kommunikaation vaatimat ominaisuudet. Komponentti pointMatcher käyttää TCPIP komponenttia kun se lähettää tunnistetun lehden indeksin sille, tai vaihtoehtoisesti -1 jos lehteä ei tunnistettu. Komponentti TCPIP lähettää indeksin ethernetin välityksellä eteenpäin, jolloin lehtien tunnistus voidaan yhdistää esimerkiksi koneeseen, joka lajittelee lehdet.

3.2.1. pointMatcher

3.2.1.1. Tiedostot

Kaikki ohjelman tuottamat tiedostot ovat ihmisen luettavassa tekstitiedosto-formaatissa, eikä niitä kirjoiteta binäärissä. Tähän ratkaisuun päädyttiin siksi, että ohjelman tiedostoja olisi helpompi muokata myös ilman ohjelmaa tarpeen tullen, huolehtimatta erityisesti ohjelman käyttämistä tietorakenteista. Kaikki tiedostot kirjoitetaan kameran sisäiseen kansioon "public", jossa muut SICK AppStudion sovellukset ja komponentit voivat käyttää niitä. Tähän ratkaisuun päädyttiin mahdollisen jatkokehityksen helpottamiseksi.

Kaikista opetettavista kuvista, jotka halutaan tallentaa, tallennetaan opetuksen asetukset sekä opetusalue. Kuville annetaan ennen tallennusta uniikki tunniste, joka on UNIX timestamp siitä hetkestä kun kuva opetettiin. Tämä UNIX timestamp on muodossa "mm dd yyyy hh:mm:ss.msecs" (SICK, n.d., p.150). Kamera kuitenkin aloittaa ajanlaskun alusta aina kun kamera käynnistetään uudelleen, joten kyseinen timestamp ei ole välttämättä aina uniikki. Koska alkuperäisenä tarkoituksena oli kuitenkin tunnistaa vain 10 lehteä, on kyseinen tunniste erittäin todennäköisesti välttävä.

Kaikki yhteen kuvaan liittyvä data tallennetaan siten, että sen tunniste (timestamp) on sen nimessä. Esimerkiksi kuvan opetuksen asetukset tallennetaan tiedostoon "public/teachSettings/settings-TIMESTAMP.txt", jossa TIMESTAMP korvataan kuvan UNIX timestamp tunnisteella. Kyseisen tiedoston muoto on seuraava:

"POINT_COUNT ITERATIONS DOWNSAMPLE_FACTOR POSE_VARIABILITY POSE_TYPE"

jossa parametrien erottimena toimii välilyönti.

Kun kuva opetetaan, ja kuvien tallennus on päällä, myös sen alue tallennetaan alueen kulmien koordinaattipareina tiedostoon "public/teachShapes/shape-TIMESTAMP.txt" muodossa:

"teachAreaEnabled;kulma1X,kulma1Y;kulma2X,kulma2Y;...;kulma4X,kulma4Y"

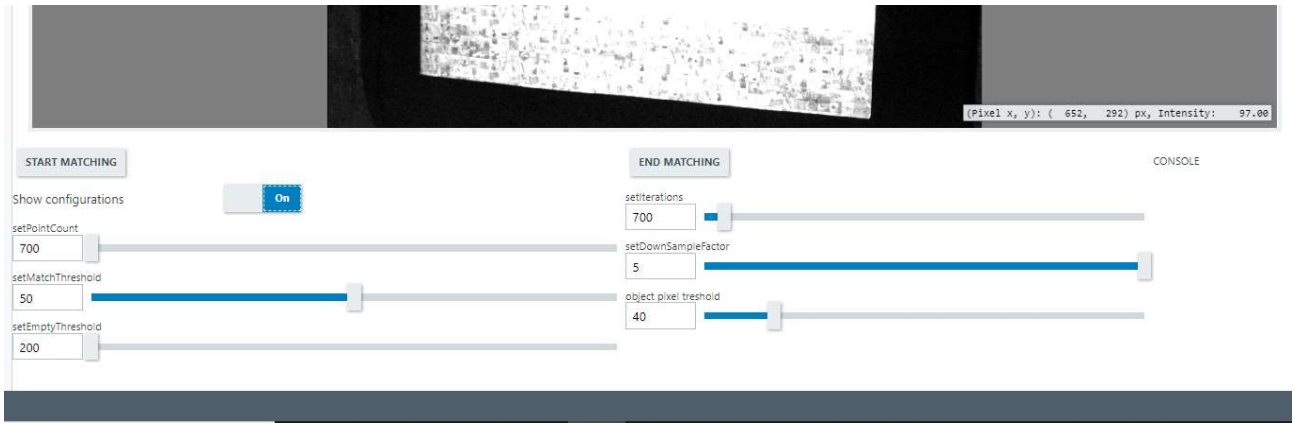
jossa ensimmäinen attribuutti ennen puolipilkku-erotinta on totuusarvo, joka kertoo onko aluen valinta manuaalinen. Kuvat itsessään tallennetaan tiedostoihin "/public/teachImg/TIMESTAMP.bmp", jossa TIMESTAMP on kuvan tunniste. Opetettavat kuvat tallennetaan myös kansioon "/public/runtimeImages/", jotta indeksilistassa voitaisiin tulevaisuudessa linkittää opetettuihin kuviin, ja täten opetettuja kuvia voisi tarkastella painamalla linkistä indeksilistassa.

3.2.1.2. Data

Kun lehtiä opetetaan SICK LUA API:n pointmatcherille, pointmatcher palauttaa opetuksesta mallin, joka tallennetaan pointMatcher komponentin listaan "models". Tähän listaan tallennetaan seuraavainen objekti "{matcher, tunnistusAlue, tunnistusMuoto}", jossa matcher on pointmatcherilta saatu malli. TunnistusAluetta ja tunnistusMuotoa voidaan käyttää tulevaisuudessa

tunnistuksen visualisoinnissa, mutta nykyisellään ohjelma ei niitä käytä. Tämä lista on erittäin keskeisessä asemassa pointMatcher-komponentin toiminnan kannalta, koska sitä käytetään opetuksessa ja tunnistamisessa.

3.2.1.3. Asetukset



Yllä oleva valikko asetuksista avautuu käyttöliittymästä painamalla alakulman Show configurations-nappia

Muuttuja	Arvo	Kuvaus
MATCH_THRESHOLD	kokonaisluku, väliltä 0-1	Kuinka suuri matchin todennäköisyys tulee olla, jotta se otetaan huomioon
EMPTY_THRESHOLD	kokonaisluku	arvo kertoo kuinka monta valoisaa pikseliä kuvassa tulee olla, jotta matchingia yritetään
OBJECT_PIXEL_TRESHOLD	kokonaisluku, väliltä 0-255	Kuinka vaalea pikselin tulee olla, jotta se luetaan valoisaksi pikseliksi
POINT_COUNT	kokonaisluku	Kuinka monta pistettä kuvasta opetetaan ja tunnistetaan pointmatcherilla
ITERATIONS	kokonaisluku > 1	Kuinka monta kertaa tunnistuksessa yritetään sovittaa kuvaa opetettuun malliin ja sen pisteisiin
DOWNSAMPLE_FACTOR	liukuluku > 1	Kuinka paljon kuvamatriisia downsamplataan, eli pienennetään jolloin siitä häviää tiettyjä ominaisuuksia
POSE_VARIABILITY	LOW tai HIGH	Kertoo kuinka suuria skaalauksen, käännön tai

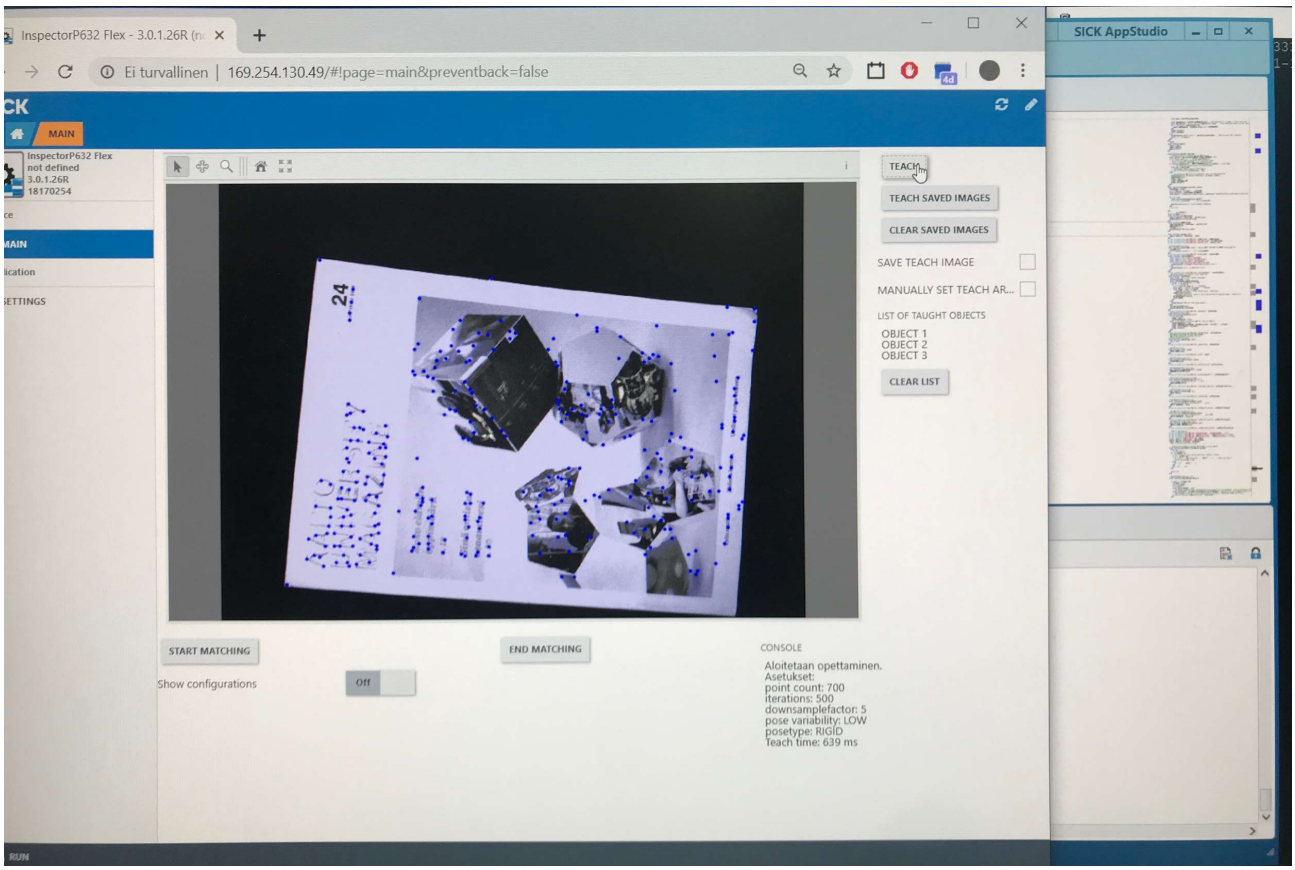
		perspektiivin muutoksia tunnistuksessa tulisi huomioida
POSE_TYPE	RIGID, SIMILARITY, AFFINE, tai HOMOGRAPHY	Kertoo millaisia muunnoksia kuvaan yritetään tehdä
savingOn	totuusarvo	Määrää tallennetaanko opetettavat kuvat
teachAreaEnabled	totuusarvo	Määrää näytetäänkö manuaalinen opetusalueen valinta näytöllä

POSE_VARIABILITY ja POSE_TYPE eri asetusten selitykset:

- POSE_VARIABILITY
  - LOW asetus pystyy huomioimaan ainakin 30% skaalauksen muutokset, kaikki käännökset ja pienet perspektiivin muutokset
  - HIGH asetus pystyy tunnistamaan erittäin hyvin myös suuremmilla skaalauksen ja perspektiivin muutoksilla
- POSE_TYPE
  - RIGID: Etsii kuvasta objekteja, joissa on käännöksen ja perspektiivin muutoksia, ei huomio skaalausta
  - SIMILARITY: Etsii kuvasta objekteja, joissa on käännöksen, perspektiivin ja yhdenmukaisen skaalauksen muutoksia
  - AFFINE: Etsii kuvasta objekteja, joissa on käännöksen, perspektiivin, skaalauksen muutoksia ja vääristymiä
  - HOMOGRAPHY: Yleisin muunnos, joka sallii erittäin paljon muutoksia objekteihin kuvassa

Kaikkia näistä asetuksista pystyy säätämään käyttöliittymästä suoraan koskematta koodiin. Ohjelman käyttöohjeissa (kts. kohta 3. Käyttöohjeet) selitetään myös asetusten merkitykset käyttäjälle, ja demonstroidaan mistä asetuksia voi muuttaa.

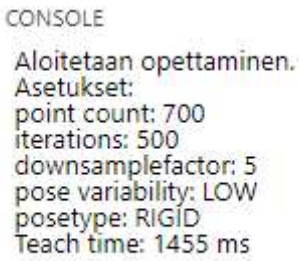
3.2.1.4. Käyttöliittymä

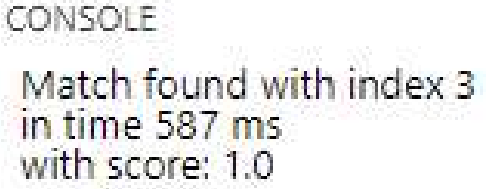


Käyttöliittymä on toteutettu SICK AppStudiossa WYSIWYG (What You See Is What You Get) tyyppisellä editorilla, joka tuottaa HTML-tyyppistä koodia, joka on yhteensopiva AppStudion kanssa.

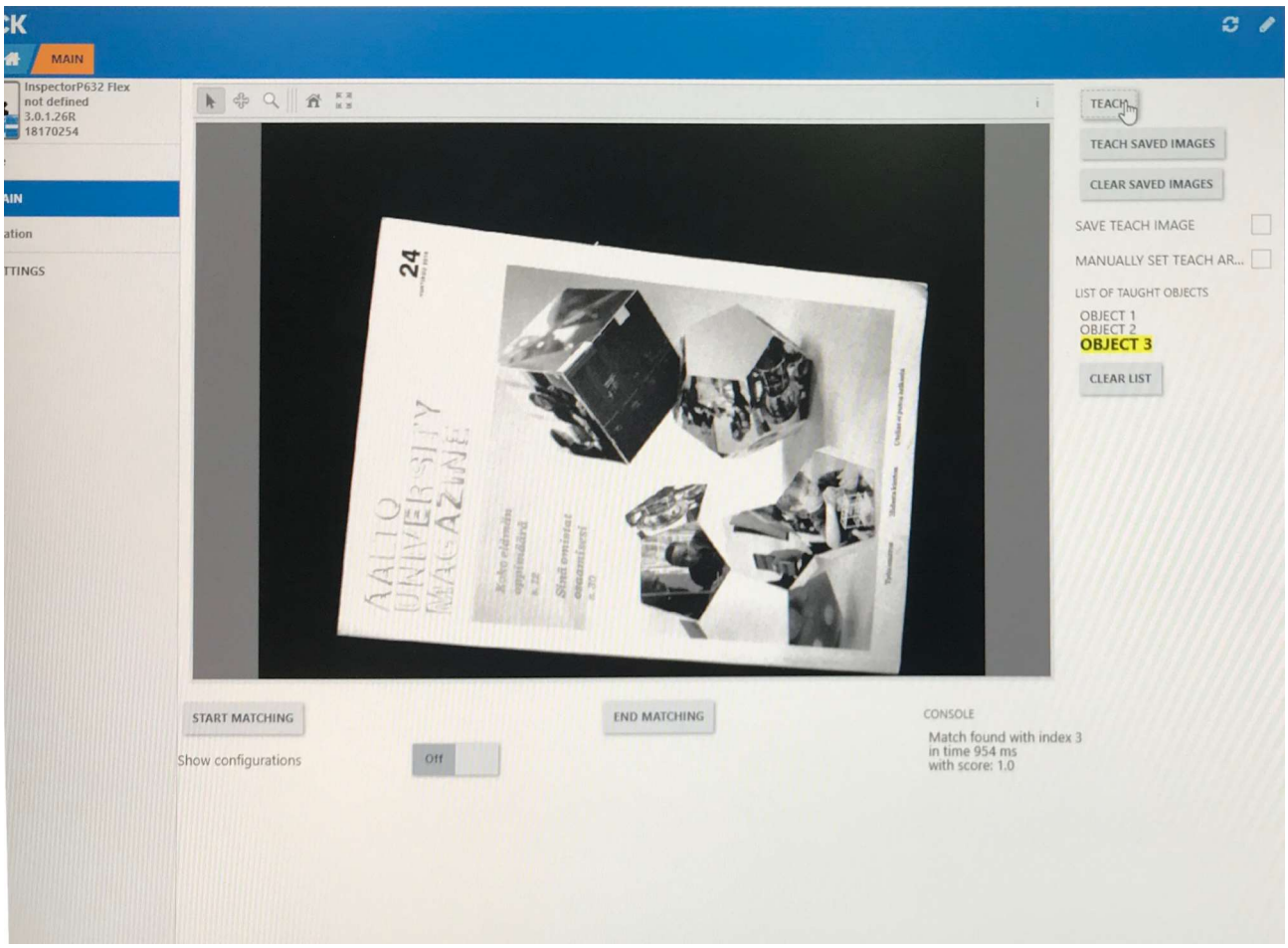
Käyttöliittymän muutokset, kuten painikkeiden painaminen, kutsuvat kyseiseen painikkeeseen liitettyä funktiota pointMatcherissa parametrinaan uusi arvo. Tämä arvo käsitellään ja asetetaan ohjelman asetuksiin. Tässä mielessä monet graafiseen käyttöliittymään yhdistetyt painikkeet kutsuvat ns. setter-funktioita, joiden ainoa tarkoitus on asettaa uusi arvo jollekin muuttujalle.

Käyttöliittymässä on itsessään komponentteja neljä: opetuksen ja tunnistuksen painikkeet, asetusten muuttaminen, konsoli ja indeksilista. Näistä komponenteista asetusten muuttaminen ja tunnistuksen painikkeet ovat pitkälti setter-funktioiden kaltaisia toiminnallisuudeltaan, ja muuttavat siis ohjelman suorituksen tilaa.





Konsoli (kts. yllä olevat kuvat) toimii AppStudion konsolin korvikkeena käyttöliittymässä, ja siihen tulostetaan erilaisia tietoja ohjelman toiminnallisuuteen liittyen, sekä vasteita käyttöliittymän painikkeille. Esimerkiksi kun uutta lehteä opetetaan, sen opetuksen tiedot kuten opetukseen kulunut aika näytetään konsolissa käyttäjälle. Konsoli toimii ilmoittamalla käyttöliittymälle konsolin tekstin muunnoksesta "OnNewConsoleText" eventillä (vrt. kohta 3.1 Ohjelmointi: ‘pointMatcher.OnIndexChange’-event ja sen toiminta).



Indeksilistassa näytetään kaikki opetetut lehdet ja niiden indeksi. Kun lehtien tunnistus on päällä, indeksilistassa väritetään tunnistettu lehti korostavalla värillä, jotta on ilmeistä mikä lehti on tunnistettu. Indeksilista toimii samalla periaatteella kuin konsoli, mutta sisältää enemmän HTML muotoilua.

Käyttöliittymän käytöstä kerrotaan enemmän kohdassa 7. Käyttöohjeet.

3.2.1.5. Lehtien opettaminen ja tunnistaminen

Kaksi ohjelman pääasiallista toiminnallisuutta on lehtien opettaminen ja tunnistaminen. Näitä toiminnallisuuksia vastaa funktiot teach ja match. Molemmat ottavat ainoana parametrinaan opetettavan/tunnistettavan kuvan.

Lehtien opettamisessa asetetaan ensin pointMatcherin globaaleista muuttujista opettamisen asetukset (kts. 5.1.3). Tämän jälkeen, jos automaattinen opetusalueen tunnistus on käytössä, ohjelma etsii suurimman valoisan pikselialueen ja valitsee tämän opetusalueeksi. Jos käytössä on manuaalinen opetusalueen valinta, ohjelma muuntaa valitun alueen koordinaatteja mm. matriisimuutoksilla, jotta aluetta voidaan käyttää opetukseen. Saatuaan opetusalueen, ohjelma opettaa pointMatcherille kuvasta valitun alueen ja pointMatcherin instanssi tallennetaan models listaan.

Opetusalue visualisoidaan graafiseen käyttöliittymään ja näytetään kuvan päällä pointMatcherin valitsemat opetuspisteet. Opettamisessa kuva ja opetusalue myös tallennetaan, jos käyttäjä on valinnut käyttöliittymässä haluavansa tallentaa opetettavat kuvat. Tämän jälkeen käyttöliittymän indeksilista päivitetään.

Ennen lehtien tunnistamista, kameran kuvan vaaleiden pikseleiden määrää tarkastellaan, ja jos niiden määrä on suurempi kuin EMPTY_THRESHOLD, kuva lähetetään match fuktiolle tunnistamista varten. Täten tyhjiä kuvia, joissa ei ole lehtiä, ei tunnisteta ohjelmalla.

Lehtien tunnistuksessa ohjelma käy läpi models listaa ja kutsuu jokaisen pointMatcher instanssin match funktiota, joka palauttaa tunnistuksen laadun/tarkkuuden (score, väliltä 0-1) ja muunnoksen pointMatcherille opetettuun skaalaan, perspektiiviin ja käännökseen. Kun koko models on käyty läpi, ohjelma ottaa instanssin, jolla on suurin tarkkuus (score), jonka perusteella match funktio palauttaa parametrina annetussa kuvassa olevan lehden indeksin. Jos suurimman tarkkuuden omaava pointMatcher instanssi ei kuitenkaan ylitä MATCH_THRESHOLD arvoa, funktio palauttaa -1 indeksin, koska kuvassa ei ole tarpeeksi varmaa tunnistusta yhdestäkään opetetusta mallista.

SICK APIn pointmatcherilta saatua muunnosta käytetään tunnistuksen visualisoinnissa graafiseen käyttöliittymään, kun opetetun mallin pisteet muunnetaan vastaamaan pisteitä funktiolle syötetyssä kameran kuvassa.

3.2.2. TCPIP

Yhtenä projektin vaatimuksina oli lähettää tunnistetun lehden indeksi ethernetin välityksellä ohjelmasta. Päätimme käyttää tähän TCP/IP protokollalla toimivaa palvelinta, joka lähettää siihen yhdistäville asiakasohjelmille (client) indeksejä kun ohjelma tunnistaa lehtiä. Päätimme myös erottaa tämän toiminnallisuuden omaan appiin AppStudiossa (kuten mainittu kohdassa 4. Ohjelmointi).

3.2.2.1 TCP/IP protokolla ja palvelin

TCP/IP yhteyden aikaansaamiseksi tcpip appi luo uuden instanssin TCPIPServer:istä. TCPIPServer on SICK LUA APIn tarjoama rajapinta uusien TCP/IP protokollalla toimivien palvelimien luomiseen. Komponentti asettaa uudelle instanssille portin 2120 ja muut tarvittavat konfiguroinnit, jotta palvelin voi lähettää indeksejä.

3.2.2.2 Tapahtumien käsittely

Itse indeksien lähetys tapahtuu tapahtumien havainnoinnin kautta. Appi tcpip odottaa "OnIndexChange" eventtiä pointMatcherilta, jonka havaittuaan se lähettää palvelimeen viimeisimmäksi yhdistetylle asiakasohjelmalle indeksin, jonka se sai pointMatcherilta.

Enemmän tapahtumien käsittelystä tcpip komponentissa kerrotaan kohdassa 4. Ohjelmointi.

3.3 Käyttöohjeet käyttöliittymään

3.3.1. Lehtien opetus

1. Teach

- Opettaa lehden kameralle. Jos manuaalinen opetus alue ei ole valittuna (5.), opetusalue määrittyy automaattisesti.

2. Tech saved images

- Opettaa aiemmin opetettuja ja tallennettuja kuvia. Kuvat tallentuvat opetettaessa, kun SAVE TEACH IMAGE (4.) on valittuna.

3. Clear saved images

- Poistaa tallennetut kuvat ja opetusalueet.

4. Save teach image

- Kun tämä on valittuna, lehti tallentuu sitä opettaessa ja lehdet voidaan opettaa uudestaan TEACH SAVED IMAGES (2.) napista.

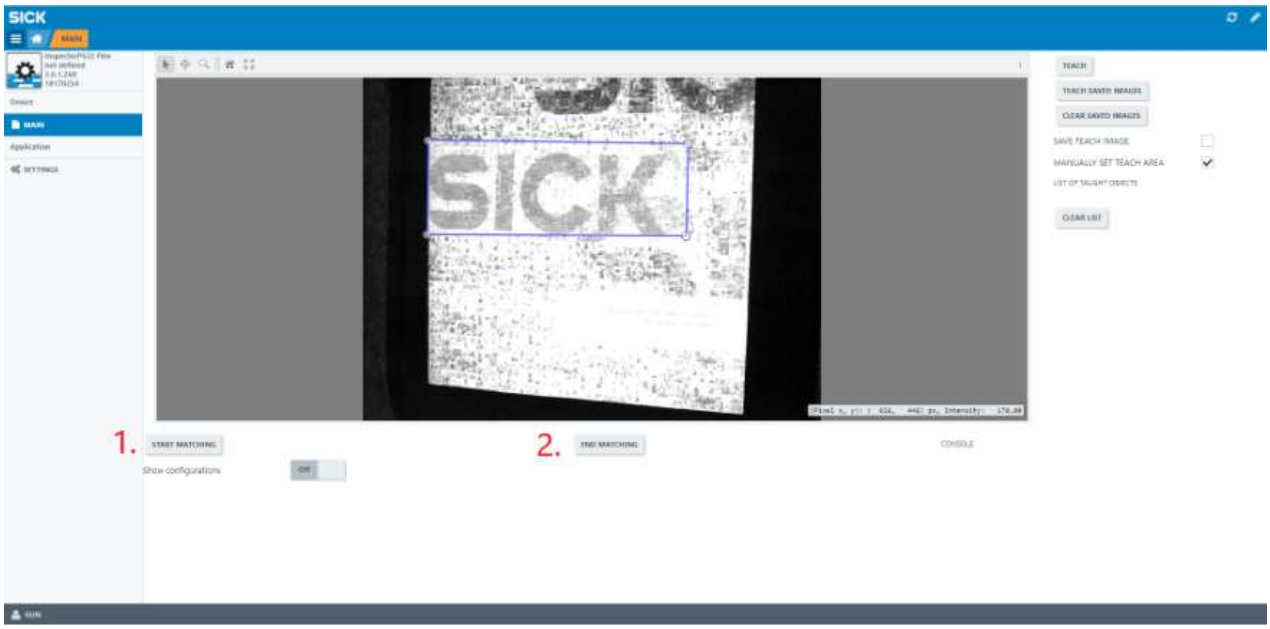
5. Manually set teach area

- Kun tämä on valittuna, ilmestyy kuvaan muokattava opetusalue, jonka voi siirtää kohtaan, jonka haluaa opettaa. Kun tämä ei ole valittuna, opetusalue määrittyy automaattisesti.

6. Clear list

- Poistaa opetetut mallit ja tyhjentää indeksi listan.

3.3.2. Lehtien tunnistus



1. Start matching

- Aloittaa lehtien tunnistamisen.

2. End matching

-Lopettaa lehtien tunnistamisen.

3.3.3. Asetukset

1. Show configurations

- Laittaa asetukset näkyviin tai pois näkyvistä.

2. SetPointCount

- Määrittää tunnistuspisteiden määrän.

3. SetMatchThreshold

- Määrittää kuinka suuri tunnistettavuus prosentti täytyy olla, jotta lehti tunnistetaan.

4. setEmptyThreshold

- Määrittää että montako valoista pikseliä täytyy olla, jotta tunnistusta yritetään.

5. setiterations

- Määrittää kuinka monta kertaa kuvaa yritetään sovittaa opetettuun malliin.

6. setDownSampleFactor

- Määrittää miten yksityiskohtainen tunnistuksesta tulee.

7. object pixel treshold

- Määrittää mikä tunnistetaan valoisaksi pikseliksi.

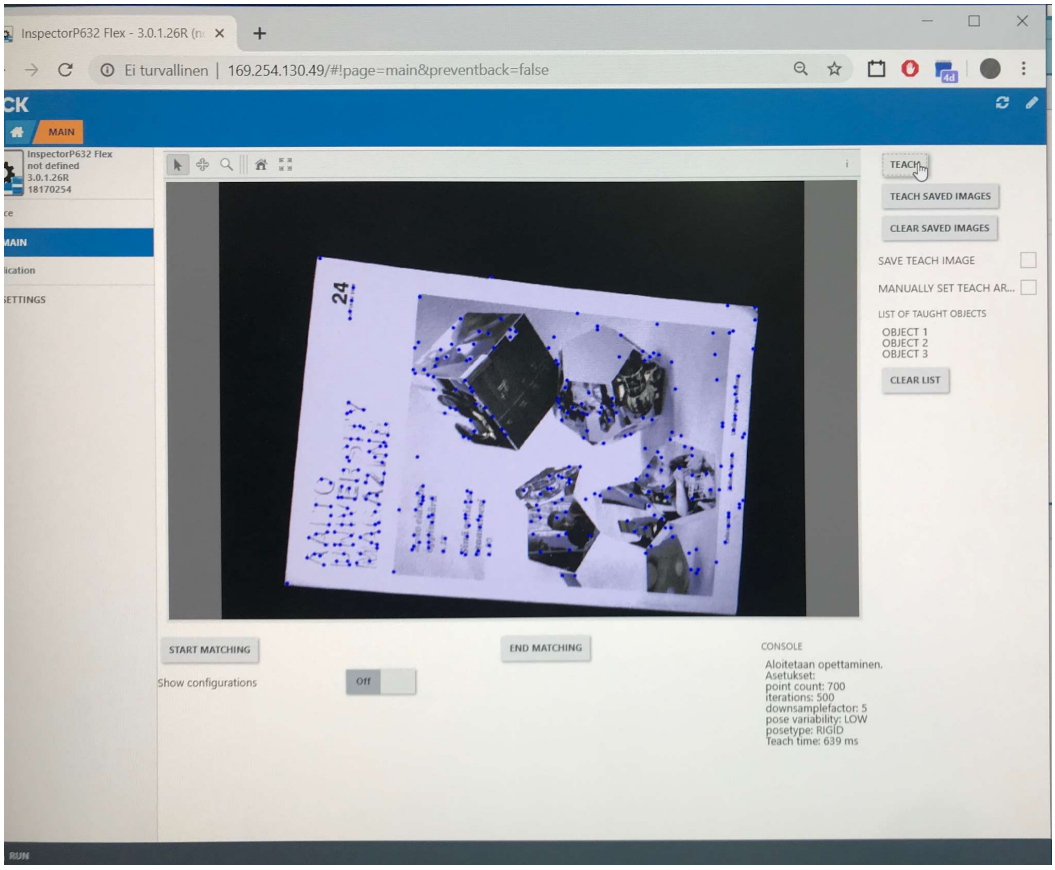
4. Tulokset

Projektin aikana olemme saaneet projektissa käytettävän kameran toimimaan halutulla tavalla sekä luomaan ohjelman, joka toteuttaa projektin tavoitteet.

4.1. Kamera

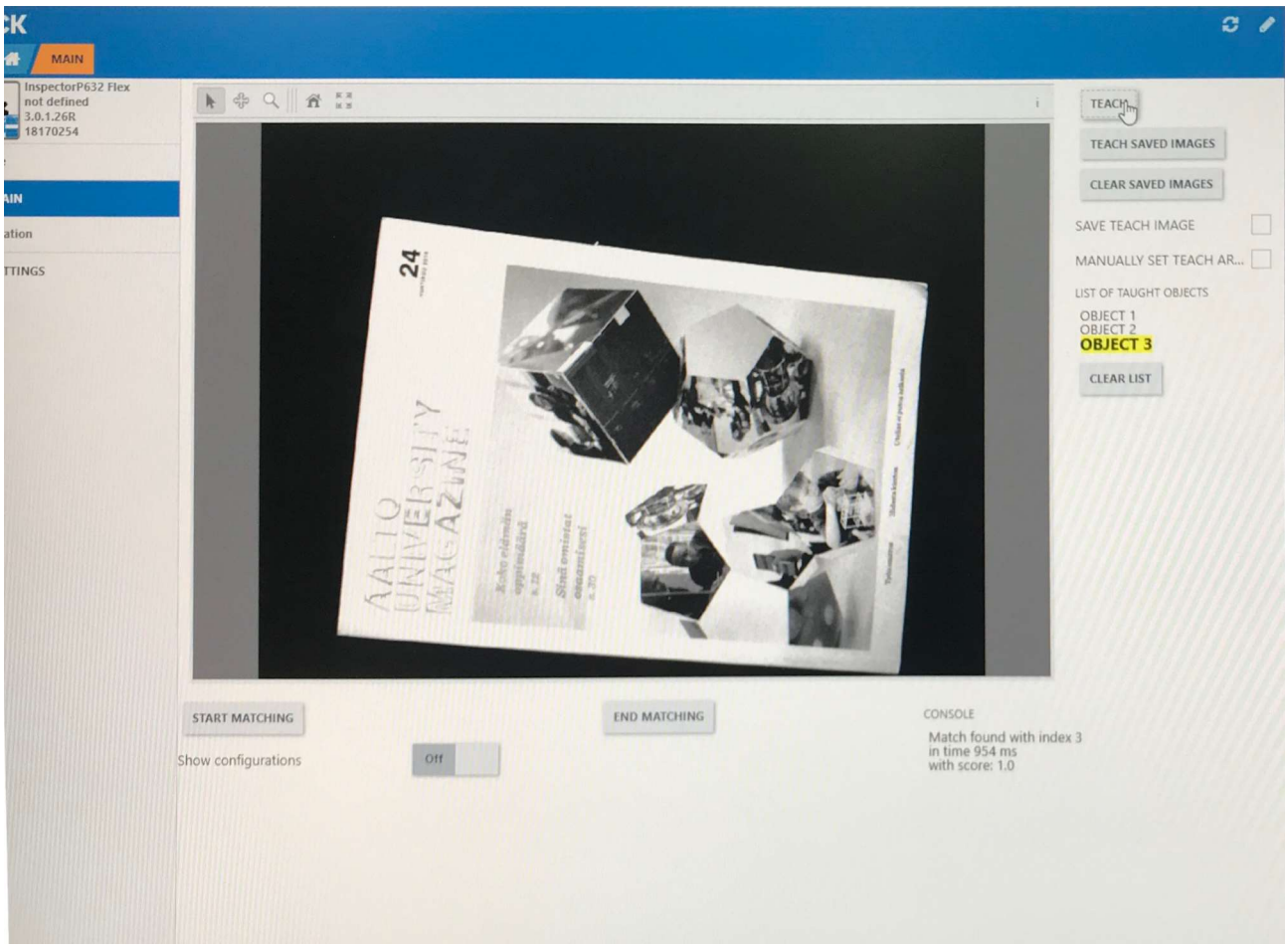
Projektia varten hankitulla telineellä saamme kameran asetettua vaakatasoon n. 42 cm korkeuteen, jolloin kameran kuvaus-alue on n. 28 cm x 18 cm. Oikeassa käyttökohteesssa kamera olisi tätä korkeammalla kuvaus-alueen laajentamiseksi.

4.2. Ohjelma



Ohjelma tallentaa, “opettaa”, referenssilehdistä haluttujen parametrien mukaiset pistepilvet (ylläolevassa kuvassa sinisellä), joita kohtaan vertailemalla tunnistettavan lehden vastaavaan pistepilveen ohjelma pystyy palauttamaan kyseisen lehden listaindeksin tai ilmoituksen ettei lehteä tunnistettu.

Opettaessa valittavissa on automaattinen, läiskiin perustuva, opetusalueen tunnistus sekä manuaalinen opetusalueen tunnistus.



Ohjelma palauttaa käyttäjälle vertailuista tietoja: vertailuista suurimman arvon saaneen lehden indeksin, kaikkiin vertailuihin kuluneen ajan sekä vertailun arvon (0.0-1.0). Arvolla 1.0 kaikki verrattavien pistepilvien pisteet ovat identtiset ja 0.0 pistepilvillä ei ollut yhtään yhteistä pistettä. Lehtien vertailuaika riippuu asetuksista sekä lehtien lukumäärästä. Lisäksi ohjelma lähettää TCP/IP protokollan avulla vertailutuloksen palvelimelle, josta tulosta voidaan hyödyntää esim. lajittelijan ohjailussa.

Ohjelman tunnistustarkkuus on korkea, jopa alhaisilla asetuksilla, joskin korkeammat asetukset voivat tuoda tunnistukseen joustavuutta. Lehtien tunnistaminen kameran eri asetuksilla otetuista kuvista onnistuu vaihtelevasti. Matalilla asetuksilla referenssikuvat tulisi ottaa kameralla samoilla kameran asetuksilla, kameran fyysisellä asennolla sekä lehden suuntauksella, sillä tunnistettavan lehden koon, kulman ja perspektiivin korjaaminen ei ole täysin luotettavaa.

5. Projektitoiminta

5.1. Tavoitteet saavuttaminen

Saimme projektin tehtyä melkein kuukautta ennen deadlinea yrityksen antamien vaatimusten mukaan. Tämän vuoksi lisäsimme ominaisuuksia, joita ei alkuperäisessä tehtävänannossa ollut. Saimme yritykseltä positiivista palautetta projektin kulusta sekä ylimääräisistä ominaisuuksista. Saimme myös yritykseltä ylimääräisiä vaatimuksia ja saimme nekin tehtyä määräaikaan mennessä.

5.2. Aikataulu

Jaoimme työtunnit tasaisesti kesän ajaksi, koska oli mahdotonta tietää, milloin ryhmän jäsenet halusivat pitää lomia. Meillä oli siis melko vapaa aikataulu, mutta saimme silti tehtyä projektin helposti aikataulussa. Projektisuunnitelmassa tehty aikataulu ei siis noudata toteutunutta tuntijakoa. Käytimme paljon aikaa kesän alussa projektiin, mikä mahdollisti lomien pitämisen ja rentoutumisen kesän lopussa.

Työjako meillä oli melko vapaa. Koska oli vain yksi kamera, käytännössä yksi tai kaksi pystyi tekemään projektia samalla aikaa. Aina, kun halusi tehdä jotain, annettiin kamera tälle henkilölle. Saimme kuitenkin lisenssit AppStudioon jokaiselle henkilölle, joten kaikki pystyivät käyttämään omaa tietokonettaan projektin työstämiseen.

5.3. Riskianalyysi

Alla olevat riskit listattiin projektisuunnitelmassa:

Riski	**Taso**	**Miten otettu huomioon**
Ajan loppuminen	Pieni	Projektin deadlinet ovat asetettu siten, että vaikka niihin ei kerkeäisikään, ei aika lopu heti kesken.
Kameran rikkoutuminen	Suuri	Tätä ei voida ottaa huomioon suunnittelussa. Jos näin käy, ei projektia voi käytännössä jatkaa.
Henkilövahingot	Keskisuuri	Jos esimerkiksi joku ryhmän jäsen sairastuu erittäin vakavasti moneksi viikoksi, tulee muiden ryhmäläisten tehdä tämän sairaan ryhmäläisen työt.
Sabotaasi	Pieni	Kalliita esineitä pidetään jonkin ryhmäläisen kotona sen sijasta, että niitä pidettäisiin julkisella paikalla, kuten sähköpajassa.
Ohjelman toimimattomuus	Pieni	SICK:llä on hyvät dokumentaatiot, joiden avulla saadaan koodi varmasti toimimaan.

Vaikka lista on melko kattava, ainakin yksi riski puuttui. Välillä kun referenssidokumentti ei ollut tarpeeksi selvä jostakin asiasta, otimme yhteyden SICK:in support portaaliin. Vastausta sai odotella joskus kauankin ja yhteen kysymykseen emme saaneet vastausta. Tämän riskin olisi siis voinut lisätä listaan.

Mikään riskeistä ei onneksi toteutunut. Projektitoimintamme oli melko turvallista ja olimme hyvin varovaisia kameran kanssa. Tämän vuoksi kameramme ei mennyt rikki ja riskeiltä säästyttiin. Kukaan ryhmäläisistä ei myöskään sairastunut, joten henkilövahingoilta säästyttiin.

6. Yhteenveto ja johtopäätökset

Projektimme eteni koko kesän hyvin ja pysyimme hyvin aikataulussa kokoajan. Meillä ei ilmennyt mitään suuria vaikeuksia projektin teossa eikä myöskään ryhmätyöskentelyssä. Kaikilla projektiryhmän jäsenillä riitti aikaa kesällä työstää projektia sekä käydä ryhmätapaamisissa, assaritapaamisissa sekä tapaamisissa yritysyhteistyö henkilöiden kanssa.

Opimme projektin edetessä paljon Lua- ohjelmointikielestä, jolla projekti tehtiin. Kenellekkään ryhmän jäsenistä Lua ei ollut tuttu ohjelmointikieli, joten meillä meni aluksi oma aikansa siihen tutustuessa. Onneksi meillä oli käytössämme kattava dokumentaatio Lua kielestä, joka auttoi meitä käyttämään sitä. Opimme myös Inspector63x-kameran käyttöä ja kuinka sitä voi ohjelmoida käyttäen SICK Appstudiota. Kameran kanssa meillä oli aluksi omat haasteensa, mutta niiden jälkeen kameran käyttö ja ohjelmointi onnistui hyvin. Opimme myös SICK Appstudio ohjelmointiympäristön käyttöä, jossa teimme projektin. Appstudionkin kanssa meillä oli vähän vaikeuksia, sillä se oli meille uusi ohjelmointiympäristö. Varsinkin käyttöliittymään nappien sekä muiden sellaisten lisääminen tuotti aluksi vaikeuksia, mutta niistäkin selvisi kun tajusimme sen toimintaperiaatteen.

Projektin aikana opimme myös ryhmätyötaitoja, sekä kuinka tällaista vähän isompaa projektia kannattaisi lähteä tekemään ryhmänä. Pidimme, joka viikko vähintään yhden ryhmätapaamisen, jossa katsoimme mitä edellisen viikon aikana on saatu aikaan sekä mitä seuraavalla viikolla pitäisi saada aikaan. Opimme myös projektin esitystaitoja, sillä meidän täytyi esitellä ja kertoa yritysyhteyshenkilöille ja vastuuassistentillemme mitä olemme saaneet aikaiseksi tapaamisissa. Myös loppunäytöksesssä esittelimme projektin muille kurssilaisille, kurssihenkilökunnalle sekä muille vieraille, joka myös antoi meille lisää esiintymiskokemusta.

Projekti siis kaiken kaikkiaan sujui hyvin ja saimme esitellä valmiin projektin loppunäytöksessä. Tietysti ohjelmaa olisi vielä voinut parannella ja joitakin bugeja ohjelmasta voi löytyä, mutta saimme kuitenkin kaikki yrityksen asettamat vaatimukset täytettyä sekä joitakin lisäominaisuuksia lisättyä. Ryhmämme on kuitenkin suhteellisen tyytyväinen projektin lopputulokseen ja protopaja kurssi oli meille kaikille positiivinen ja opettavainen kokemus.

Liitteet

Lähteet

SICK Oy. (n.d.). CROWN Lua API Reference Manual. V2D6xxP - V 2.4.0.1178.