Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

MIKKO TIRKKONEN

ABB IRB 120 -robotin ja konenäön yhdistäminen

SÄHKÖ- JA AUTOMAATIOTEKNIIKAN
KOULUTUSOHJELMA
2020

Tekijä(t)
Tirkkonen, Mikko

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
12/2020

Sivumäärä
66

Julkaisun kieli
Suomi

Julkaisun nimi
ABB IRB 120 -robotin ja konenäön yhdistäminen

Tutkinto-ohjelma
Sähkö- ja automaatiotekniikka

Tiivistelmä
Työn tarkoituksena oli yhdistää Cognexin 5100C-älykamera ABB:n IRB 120 -robottiin. Kamera ja robotti kommunikoivat keskenään Telnet-protokollan avulla.

Älykameralle luotiin kaksi eri ohjelmaa. Yksi tunnistamaan noppia ja niiden väri sekä silmäluku ja toinen, joka tunnisti kuulia ja niiden värin. ABB:n robottia varten luotiin ohjelma, joka tiputti nopat kameran kuvausalueelle ja noukki ne pois kameralta saatujen tietojen perusteella. Lisäksi robotin ohjelma päästi kuulat vierimään kuvausalueelle ja poimi ne kameralta saatujen tietojen perusteella takaisin alustalleen.

Noppia ja kuulia varten 3D-tulostettiin omat alustat. Kameran kuvausaluetta varten suunniteltiin ja valmistettiin oma alusta. Robotin ohjelmaan tehtiin virhetilanteita tunnistavia ominaisuuksia.

Aika ja laiterajoitusten vuoksi sovelluksesta ei tullut noppien osalta niin toimintavarmaa kuin haluttiin. Kuulien osalta sovellus oli varmatoimisempi.

Asiasanat
Automaattiset järjestelmät, konenäkö, robotiikka, tekniset järjestelmät

Author(s) Tirkkonen, Mikko	Type of Publication Bachelor's thesis, AMK	Date 12 2020
	Number of pages 66	Language of publication: Finnish

Title of publication
Connecting ABB IRB 120 robot to machine vision system

Degree program
Electrical and Automation Engineering

Abstract

The objective of the work was to connect a Cognex 5100C series smart camera to an ABB IRB 120 robot. The camera and robot communicated with each other over the Telnet protocol.

Two programs were created for the smart camera. One to recognize dice, their color and eye count and one to recognize colored marbles. The robot was programmed to pick the dice one by one and drop them in the field of view of the camera. Based on data received from the camera, the robot would pick the dice and return them to their point of origin. Additionally, the robot was programmed to let loose the marbles, so they'd drop into the cameras field of view. The robot would then pick the marbles based on their color and return them to their point of origin.

Stands were designed and 3D printed for the dice and marbles. Additionally, a base was designed and made for where they would be dropped. The program for the robot contained some error handling capabilities.

Due to time and material limitations the dice part of the program did not end up as reliable as originally desired. The marbles part ended up more reliable.

Key words
Self-acting systems, machine vision, robotics, technical systems

SISÄLLYS

1 JOHDANTO

Tässä luvussa esitellään työn tilaaja ja käydään lävitse mitä työssä on tarkoitus tehdä.

1.1 Tilaajan esittely

Satakunnan Ammattikorkeakoulu on noin 6000 opiskelijan ja 400 työntekijän monialainen ja kansainvälisesti suuntautunut korkeakoulu. Koulu on painottunut teollisuuskorkeakouluksi. Vahvuusaloina ovat automaatio, robotiikka ja tekoäly sekä merenkulku ja ikääntyvien palvelut. Koulun pääasiallinen kampus sijaitsee Porissa. Tämän lisäksi tiloja on Raumalla, Huittisissa sekä Kankaanpäässä. (Satakunnan Ammattikorkeakoulun www-sivut 2020.)

1.2 Työn esittely

Satakunnan Ammattikorkeakoulun Porin kampus jakaa tilansa K-Supermarketin ja apteekin kanssa. Lisäksi kampuksen vieressä on linja-autoasema sekä juna-asema. Kampuksella on kaikille avoin kahvila. Tämä tuo kampuksen tiloihin paljon koulun ulkopuolista liikennettä. Automaatiolaboratoriossa on useita robotteja, jotka on sijoitettu ikkunoiden taakse niin, että ohikulkijat voivat nähdä niiden työskentelyä. ABB:n IRB 120-robotilla ei tällä hetkellä kuitenkaan ole olemassa mitään demo-ohjelmaa, joka voisi olla ohikulkijoiden iloksi käytössä. Tämä on erityisen valitettavaa, sillä robotin ikkuna on suoraan yhtä K-supermarketin sisääntuloa vastapäätä ja tarjoaisi hyvän mahdollisuuden esitellä robotin kykyjä ohikulkijoille.

Tämän työn tarkoitus on luoda robotille demo tätä tarkoitusta varten. Tarkoitus on, että robotti poimii alustan, jolla on erivärisiä marmorikuulia, ja kaataa ne älykameran näköalueelle olevalle alustalle. Tämän jälkeen robotti palauttaa kuulien alustan paikalleen ja poimii kuulat takaisin sille, kameralta saatujen paikkatietojen perusteella. Ohjelman toisessa vaiheessa, robotti tarttuu toiseen alustaan, jolla on noppia, ja kippaa ne kameran näköalueella olevalle alustalle. Tämän jälkeen robotti palauttaa noppien alustan paikalleen, ja poimii nopat takaisin alustalle silmälukuun perustuvassa järjestyksessä älykameran sille antamien tietojen perusteella.

Älykamerana käytetään Cognexin 5000-sarjan kameraa. Robotin ja älykameran ohjelmien lisäksi demoa varten pitää mallintaa ja valmistaa kuulien ja noppien alustat. Lisäksi pitää valmistaa alue, jolle kuulat ja nopat kaadetaan.

2 LAITTEISTO, OHJELMISTOT JA TEKNOLOGIAT

Työssä käytettiin monia eri ohjelmia ja tekniikoita. Tässä luvussa esitellään ja käydään niitä tarkemmin lävitse.

2.1 Solidworks

Solidworks on Dassault Systemsin valmistama ohjelmistopaketti, joka sisältää 3D-mallinnusohjelman, jota käytetään erityisesti teolliseen suunnitteluun. Ohjelmapaketti sisältää myös muihin tarkoituksiin suunnattuja osia. Kokonaisuudesta löytyy muunmuassa virtapiirien suunniteluun tarkoitettu ohjelma sekä kappaleiden fysiikan simulointiin soveltuva ohjelma. 3D-mallinnusohjelman avulla suunniteltiin kameran kiinnike ja käytetyt alustat. (Solidworks www-sivut 2020.)

2.2 Kamera ja In-Sight Explorer

Satakunnan Ammattikorkeakoulu antoi työtä varten Cognex 5100C -värikameran. Kameran sensori on kooltaan 1/3 tuumaa ja sillä pystyy ottamaan värillisiä kuvia 640x480 resoluutiolla. Kamerassa on ohjelmia varten 32MB muistia ja kuvien käsittelyä varten 64MB muistia. Kameraan oli kiinnitetty 12 mm linssi. (Cognex 2009, 18.)

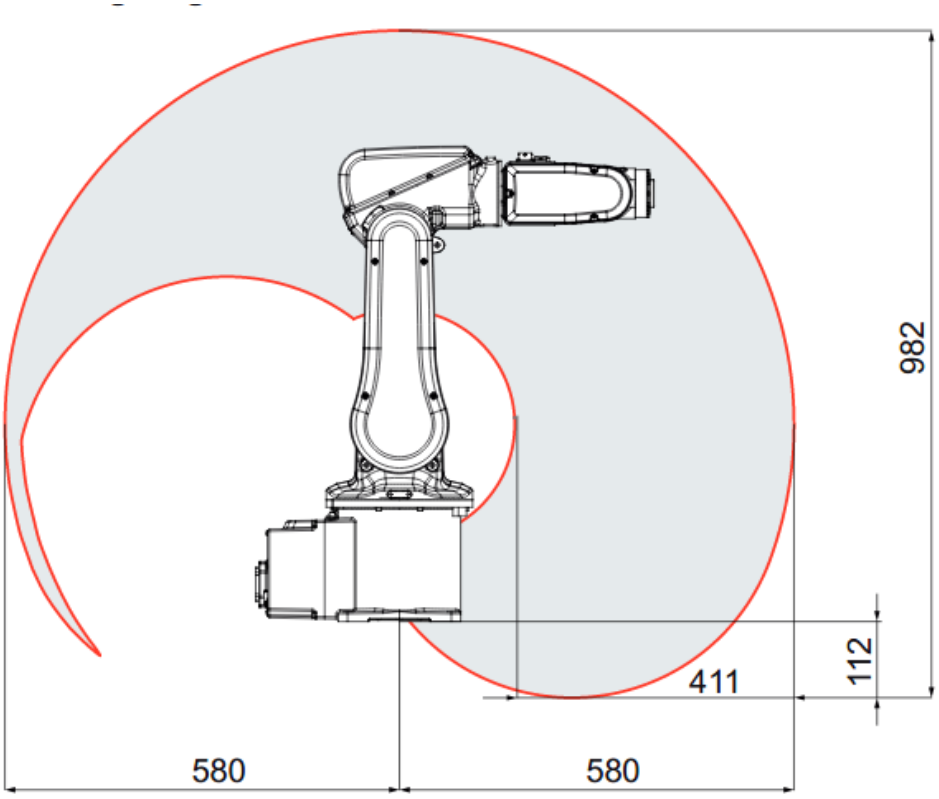
Cognexin kameraan tehdään ohjelmia In-Sight Explorer nimisellä ohjelmistolla. Ohjelmasta on tarjolla monia eri versioita. Työssä käytettiin versiota 4.9.1, sillä se on todettu kameran firmware-version kanssa yhteensopivaksi. In-Sight Explorer on Excel-tyylinen taulukkoon perustuva ohjelmointiympäristö. Taulukon soluihin voidaan raahata erilaisia työkaluja, kirjoittaa kaavoja tai pelkkää tekstiä. Soluihin voidaan myös viitata samalla tavalla kuin Excelissä. Esimerkiksi B15 viittaa sarakkeen B riviin 15. (Leino n.d., 23.)

Raahatessa erilaisia työkaluja taulukkoon on hyvä pitää mielessä, että työkalu voi syöttää ulos tuloksia. Nämä tulokset voivat viedä useampia rivejä ja sarakkeita. Väljyys työkalujen sijoittelussa on siis syytä pitää mielessä, ettei esimerkiksi vahingossa ylikirjoita jotain aiempaa tulosta tai työkalua. (Leino n.d., 42.)

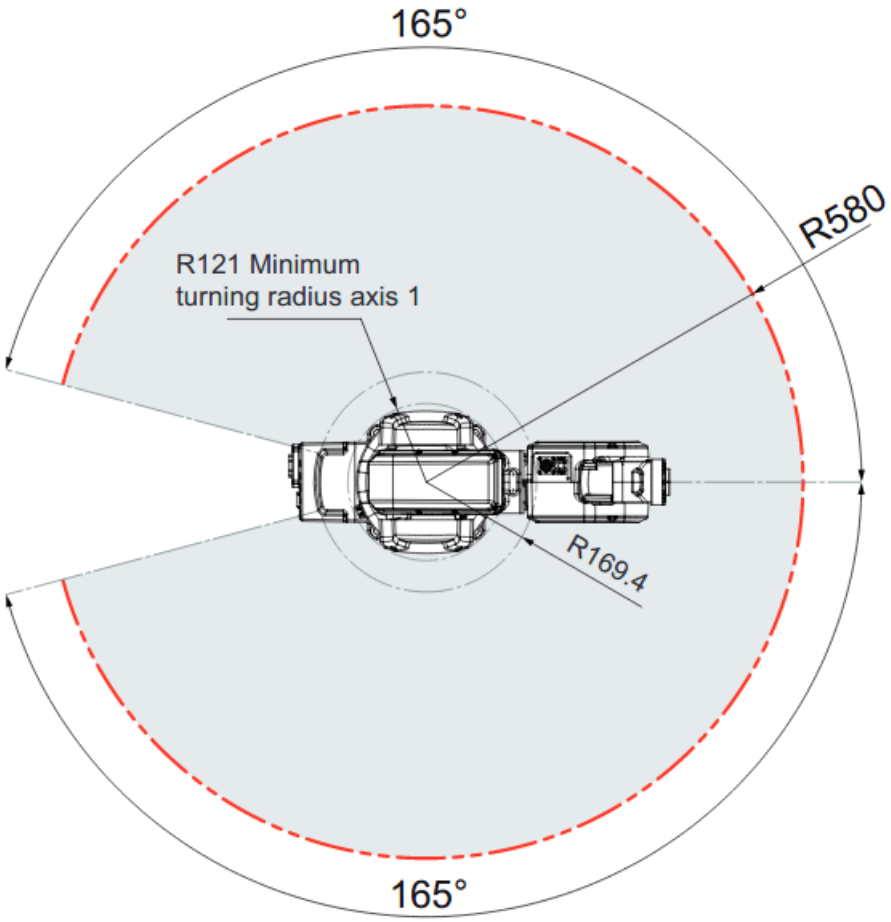
2.3 RobotStudio ja ABB IRB 120 -robotti

RobotStudio on ABB:n tekemä simulointiohjelmisto ja offline-ohjelmointiympäristö, jolla voidaan mallintaa eri ABB-robotteja. Sillä voidaan etukäteen tehdä lähes kaikki robotin ohjelmointi liikeradoista erilaisiin monimutkaisempiin ohjelmiin asti. Ohjelmalla tehdyn koodin voi siirtää fyysiseen robottiin ja ottaa käyttöön.

ABB IRB 120 on kuusiakselinen käsivarsirobotti. Robotti painaa 25 kiloa ja on 700 mm korkea. Sen käsivarren ulottuvuus on 0,58 metriä (kuvat 1 ja 2) ja kantokyky kolme kilogrammaa. Robotti on IP30-suojattu ja sen voi asentaa mihin tahansa kulmaan. Robotin ranteesta löytyy 10 signaalilähtöä ja neljä ilmaläpivientiä. (ABB 2019, 2.)



Kuva 1. Robotin työskentelyalue sivultapäin katsottuna (ABB 2019, 2)



Kuva 2. Robotin työskentelyalue ylähäältäpäin katsottuna (ABB 2019, 2)

2.4 3D-tulostus

3D-tulostuksessa tietokoneella tehdystä 3D-mallista tulostetaan fyysinen esine. Tämä tapahtuu lisäämällä kerroksittain materiaalia, kunnes kappale on valmis. Materiaaleina käytetään yleensä erilaisia muoveja, mutta myös metallin käyttö on mahdollista. Tekniikkaa käytettiin alustojen ja kameran kiinnikkeen valmistukseen. Tulostukseen käytettiin sekä ASA- että PLA-muoveja. (3dprinting.com www-sivut 2020.)

3 ALUSTOJEN JA KIINNIKKEIDEN SUUNNITTELU JA TOTEUTUS

Alun perin tarkoituksena oli, että alustat suunnittelisi ja valmistaisi työn tekijä. Olosuhteiden muutoksen takia puisen alustan sekä noppien alustat suunnitteli työn ohjaaja. Kuulien alustan suunnitteli työn tekijä.

3.1 Pudotusalusta

Pudotusalusta valmistettiin puusta. Siinä on neljä koko alustasta läpi menevää reikää, jotta se saadaan kiinnitettyä robotin työpöytään tukevasti kiinni. Alustassa on mustalla kankaalla vuorattu, syvenevä alue, jolle noppia ja kuulia pudotetaan. Noppien alustoille on omat, matalammat syvennyksensä kuten myös kuulien alustalle. Kuvassa 3 näkyy kaikki alustat.

Kuva 3. Alustakokonaisuus

Puinen alusta toimii koko työn toiminta-alustana. Siihen opetetaan robottia varten oma Object Frame, johon robotin liikkeet sidotaan noppien ja kuulien pudotusta varten. Myös kameran kuvausalueelle tehdään oma Object Frame, joka taas sidotaan alustaan tehtyyn Object Frameen. Tällä tavoin, mikäli alustaa siirretään, tarvitsee vain opettaa uudelleen sen sijainti ja loput Object Framet liikkuvat samassa suhteessa.

3.2 Marmorikuulien alusta

Kuulien alusta on 3D-tulostettu, 10 asteen kulmassa alaspäin viettävä mäki. Mäessä on 12 cm pitkä osuus, jolle kuulia tiputetaan. Tämän jälkeen tulee levennys porttia varten, joka estää kuulia liukumasta kameran näköalueelle. Portin yläpäässä on pyöreä alue, josta robotti tarttuu siihen ja nostaa sen pois paikaltaan, kun kuulien halutaan liukuvan kameran kuvausalueelle. Mäkeä varten tulostettiin kaksi jalkaa, jotka asettavat mäen 10 asteen kulmaan. Kuvassa 3 mäki on vasemmalla, mustan pudotusalueen alapuolella. Tarkempi piirustus alustasta, portista ja kahdesta jalasta löytyy liitteestä 1.

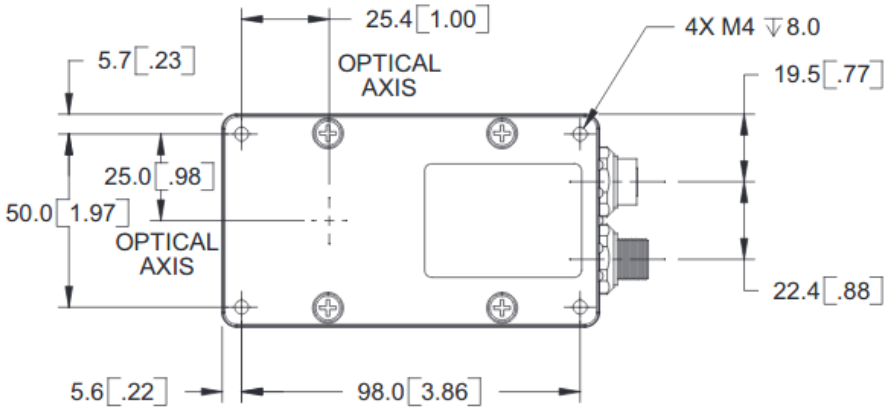
3.3 Noppien alustat

Noppia varten on kaksi 3D-tulostettua alustaa. Alustan keskellä on robottia varten tarttumisalue. Alueen vasemmalla ja oikealla puolella on tilaa kahdelle nopalle. Jokaisella nopalla on oma syvennyksensä, jossa on pieni ramppi, jotta noppa vierii helpommin pois alustalta. Värillisten noppien alustassa syvennykset ovat hieman pienempiä, koska myös nopat ovat valkoisia pienempiä. Kuvassa 3 noppien alustat ovat oikealla, mustan pudotusalueen alapuolella.

3.4 Kameran kiinnike

Kamera kiinnitettiin robotin yläpuolella olevaan alumiiniseen palkkiin. Palkissa itsessään ei ole sopivaa kiinnityskohtaa kameraa varten, joten sellainen oli tehtävä. Kiinnike valmistettiin 3D-tulostamalla. Kappale mallinnettiin Solidworksissa. Se on U:n

muotoinen, jossa on siivekkeet kameran ruuveja varten. Keskikohta menee alumiinisen palkin päälle, jolloin kamera voidaan kiinnittää sen alapuolelle. Kuva 4 näyttää kameran takapuolen tarkat mitat, joihin kiinnike perustuu.



Kuva 4. Piirros kameran takalevyn kiinnityksistä (Cognex 2011, 11)

Reiät ovat M4-kierteellä ja 8,0 mm syviä. Leveyssuunnassa reikien etäisyys on 50,0 mm ja pituussuunnassa 98,0 mm. Näiden mittojen mukaan suunniteltiin valmistettava kiinnike. Tarkempi piirustus kiinnikkeestä löytyy liitteestä 2.

4 FYYSINEN YMPÄRISTÖ JA KYTKENNÄT

Robotti on sijoitettu ikkunan eteen. Kuvassa 5 näkyy myös oikealla ja vasemmalla erikseen asennetut suojaseinät. Robottia ympäröi siis kolmelta sivulta fyysinen este, joka estää robotin työskentelyalueelle kulun. Kuvassa 5 oikealla näkyvä keltainen palkki on valoverho, joka lauetessaan pysäyttää robotin. Valoverho suojaa ainoaa kulkureittiä robotin työskentelyalueelle.

Kuva 5. Kokonaiskuva robotista

4.1 Kuvien tasalaatuisuuden varmistus

Tehtäessä kuvantunnistusta on ensiarvoisen tärkeää, että kuvat ovat tasalaatuisia. Muutos valaistukseen tai ohitse kulkevan ihmisen varjo voivat johtaa siihen, että ohjelman asetuksilla ei enää tunnisteta kuvasta siinä olevia esineitä.

Valaistuksen tasaisena pitäminen vaati lisävalaistuksen asentamista, vaikka tila sinänsä oli hyvin valaistu jo valmiiksi. Robotti on ikkunan takana, jota vastapäätä on

kaupan lasiset sisääntulo-ovet. Näiden ovien kautta ulkoa tuleva valo riitti muuttamaan kuvan valoisuutta sen verran, että aurinkoisena päivänä kappaleet saattoivat löytyä hyvin, mutta pilvisenä päivänä raja-arvoja täytyi säätää, jotta kappaleet löytyivät. Lisävalaistuksen myötä ulkoa tuleva valo vaikutti kuviin vähemmän. Suora auringonvalo oli kuitenkin edelleen riittävän voimakas, että sen kohdalle osuminen oli otettava huomioon kameran ja robotin ohjelmia tehtäessä.

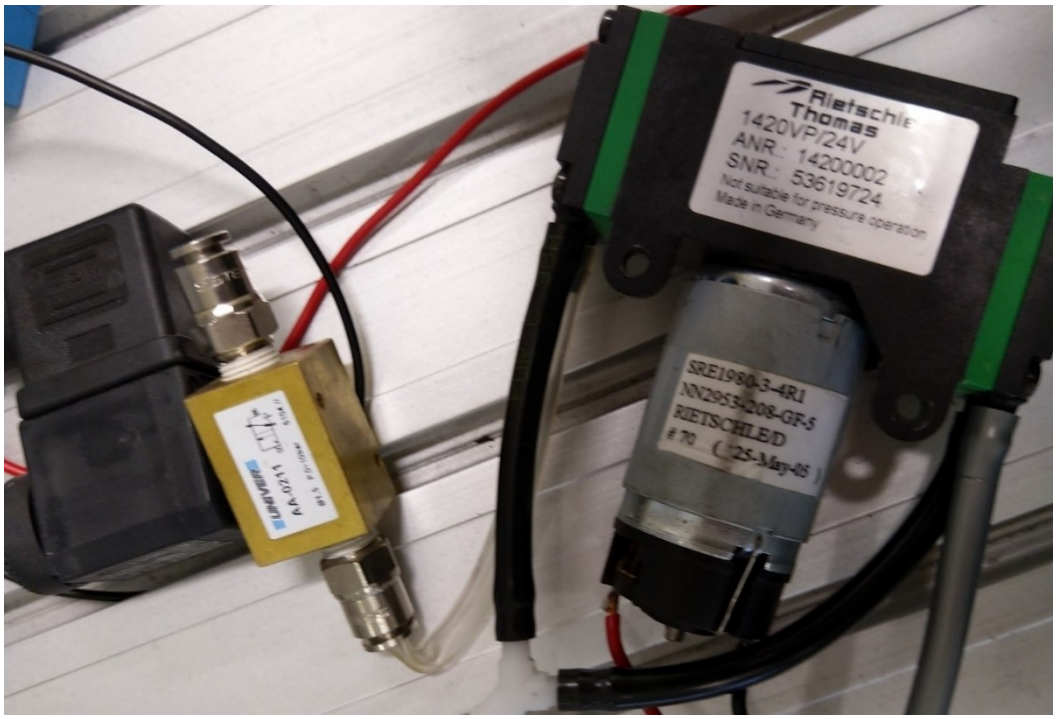
Käytetty tausta ja sen väri on myös tärkeää huomioida. Valkoinen noppa ei erotu valkoiselta taustalta kovin hyvin ja toisaalta varjot näkyvät valkoisella taustalla selkeästi häiriten kuvasta tunnistamista. Tämän vuoksi työhön valittiin musta tausta. Se eliminoi varjot sekä mahdolliset valon heijastukset taustasta.

4.2 Kameran sijoittaminen ja kytkeminen

Kamera ripustettiin robotin työpöydän yläpuolelle alumiinisia palkkeja sekä 3D-tulostettua kiinnikettä käyttäen. Kamera kytkettiin Ethernet-johdolla suoraan robotin kontrolleriin kiinni. Johdon pituus katsottiin sopivaksi, koska ylimääräistä johtoa ei jäänyt suurta määrää.

4.3 Pumpun kytkentä

Imukuppia varten robotti tarvitsi erillisen vakuumipumpun. Robotissa oli aikaisemmin ollut pumppu kytkettynä, joten robotin kontrollerissa oli jo valmiiksi tehty tarvittavat kytkennät, että imun saa päälle. Pumppu tarvitsi vain fyysisesti kytkeä robotin jalustassa olevaan ilmaletkuun sekä robotin pöydän alla olevaan releeseen ja virtalähteeseen. Pumppu on Rietschle Thomas:n valmistama 1420VP/24V-mallinen, johon on kytkettynä Univerin AA-0211-venttiili. Laitteet löytyvät kuvasta 6. Niiden sähkökytkentä löytyy kuvasta 7.



Kuva 6. Pumppu ja venttiili

Kuva 7. Pumpun ja venttiilin sähkökytkentä

Pumpun ja venttiilin (+) johdot (kuva 7, punaiset johdot) kytkettiin rinnakkain releeseen, joka on kytkettynä sen vieressä olevaan virtalähteeseen. Pumpun ja venttiilin (-) johdot (kuva 7, mustat johdot, joissa valkoinen kaulus) kytkettiin virtalähteen (-V) lii-

täntöihin. Robotissa oli valmiina digitaalinen ulostulo, jolla relettä pystyttiin komentamaan päälle ja pois. Kun robotissa oleva digitaalinen ulostulo laitetaan päälle, niin pumppu käynnistyy ja imu lähtee päälle.

4.4 Lisävalon kytkentä

Lisävalo on sarja alumiiniseen palkkiin kiinnitettyjä ledejä. Ledeistä lähti + ja – johdot. Virtaa antamaan otettiin kannettavan muuntaja. Muuntaja ja LED-palkin johdot kytkettiin toisiinsa kytkentäriman avulla. Muuntajan johto oli rakenteeltaan kerroksellinen. Miinusjohdin kulki ulkokuoren alla. Plusjohto kulki erillisen suojakerroksen alla, miinusjohtimen ympäröimänä.

Lisävalo ripustettiin samalla tavalla kuin kameran palkki. Se sijoitettiin hieman sivuun kuvausalueesta, koska suoraan yläpuolelta tullessa, sen luoma valo loi liikaa heijastuksia.

5 ÄLYKAMERAN OHJELMAT

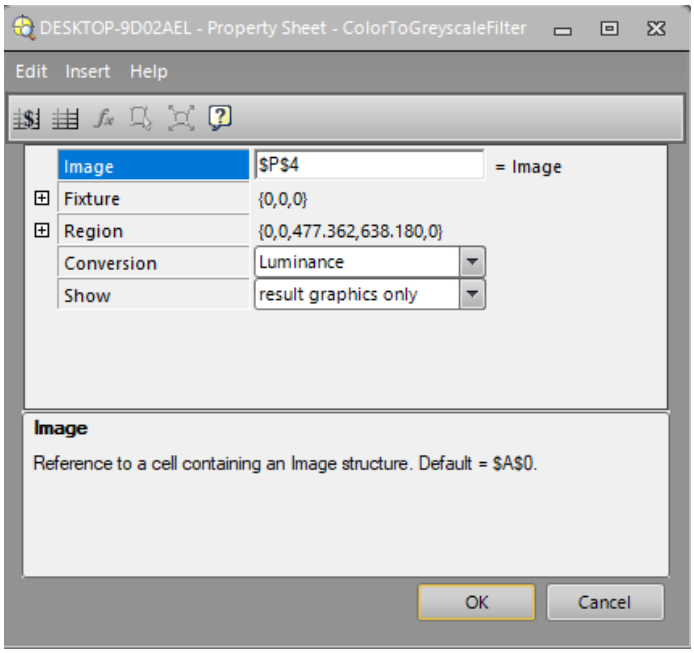
Älykameran ohjelmat toteutettiin In-Sight Explorer -ohjelman versiolla 4.9.1. Ensimmäisen ohjelman tuli tunnistaa erivärisiä noppia, kertoa niiden sijainnin koordinaatistossa, sekä mikä silmäluku minkäkin värisellä nopalla on. Toisessa ohjelmassa kameran tulee tunnistaa värillisiä kuulia ja kertoa niiden väri ja sijainti.

5.1 Noppien ohjelma

Noppien ohjelman tarkoituksena on tunnistaa noppien sijainti, laskea niiden määrä sekä tunnistaa niiden väri ja silmäluku. Jokaisen yksittäisen nopan tiedot kootaan merkkijonoksi, jonka robotin ohjelma lukee kameralta.

5.1.1 Noppien tunnistus

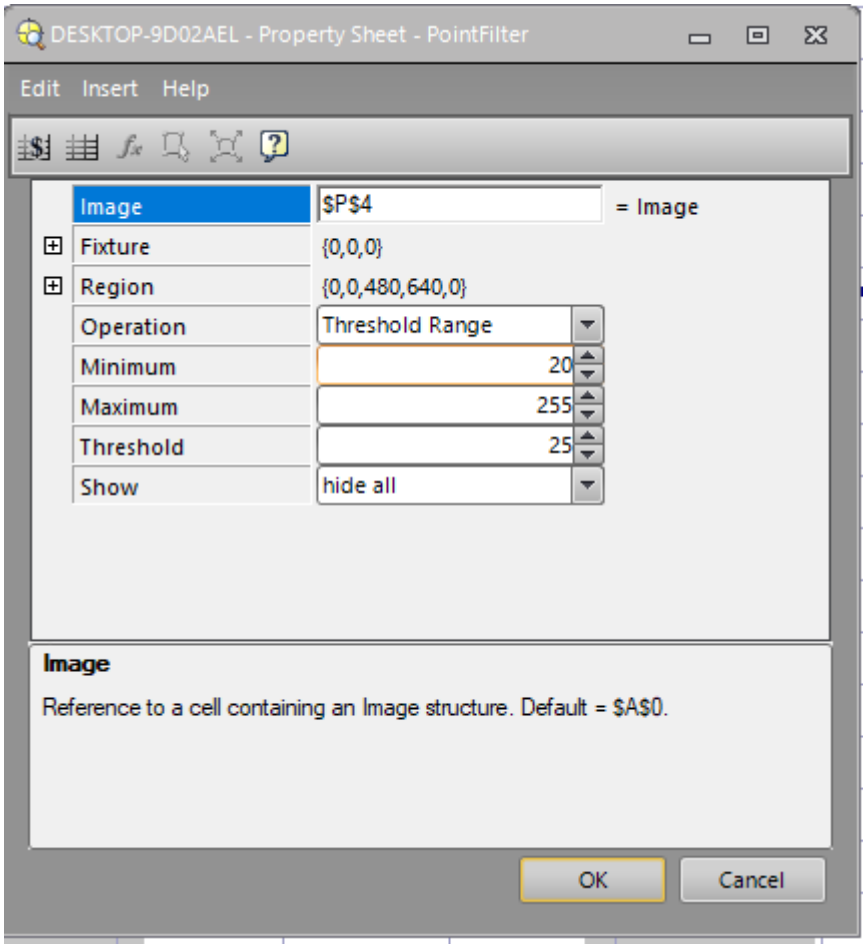
Noppien tunnistukseen käytetään TrainPatMaxPattern- sekä FindPatMaxPattern-työkaluja. Ensimmäisenä kuvasta luodaan harmaasävyinen versio käyttäen ColorToGreyscaleFilter-työkalua (kuva 8). Työkalun saa raahattua ohjelman oikeassa reunassa olevasta työkaluvalikosta, kohdasta Image.



Kuva 8. ColorToGreyscaleFilter-työkalun asetukset

Työkalulle tarvitsee kertoa vain missä solussa sijaitsee kuva, joka halutaan muuttaa harmaasävyiseksi. Tässä tapauksessa kuva sijaitsee solussa P4. Laittamalla $-merkit P:n ja 4:n eteen lukitaan viittaus soluun. Mikäli työkalu siirretään taulukossa toiseen soluun, niin ohjelma ei koita automaattisesti muuttaa viittausta suhteessa kuvan sijaintiin.

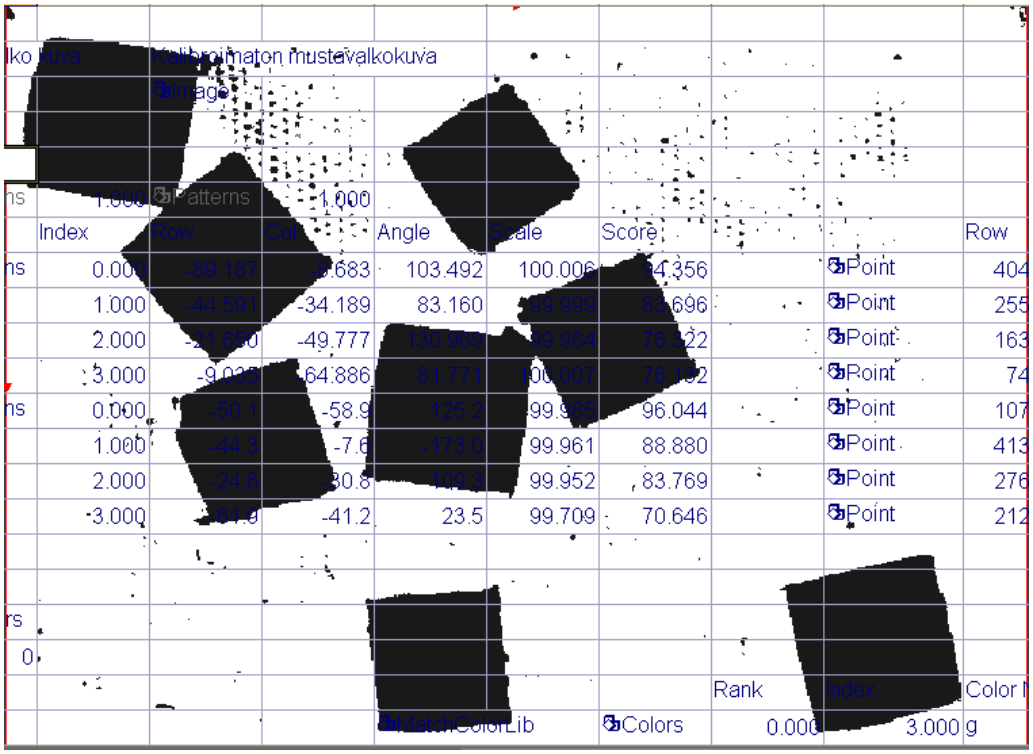
Seuraavaksi taulukkoon tuodaan PointFilter-niminen työkalu. Sillä muutetaan kuva mustavalkoiseksi. Työkalun asetukset löytyvät kuvasta 9.



Kuva 9. PointFilter-työkalun asetukset

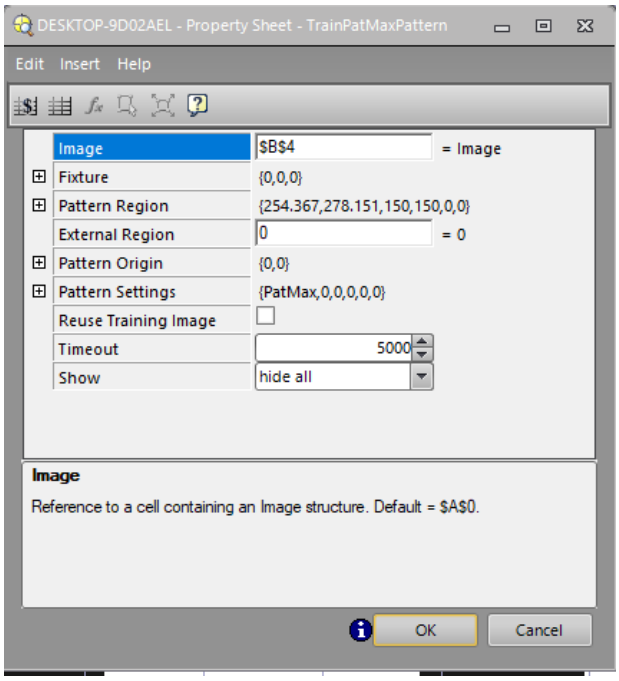
Image-kohdassa viitataan soluun, jossa sijaitsee kuva, joka halutaan muuttaa mustavalkoiseksi. Työkalu tarjoaa erilaisia vaihtoehtoja, miten kuva muutetaan. Tässä tapauksessa käytetään Threshold range -tapaa. Minimi- ja maksimiarvoja, sekä

threshold-arvoa muuttamalla, haetaan sellaiset asetukset, että nopat löytyvät kokonaisina neliöinä kuvasta. Noppien silmälukujen ei tule näkyä löytyneissä kuvioissa. Esimerkki siitä, miltä tämän tulisi näyttää, löytyy kuvasta 10.



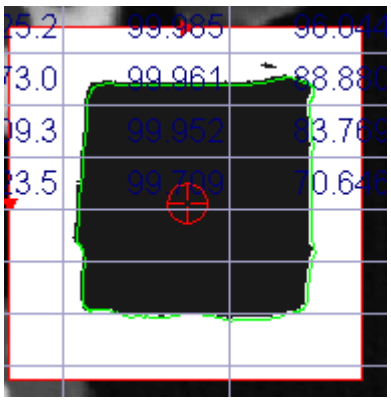
Kuva 10. Noppien hahmot PointFilter-työkalua käyttäen

Koska valkoiset ja värilliset nopat ovat erilaisia, pitää molempien kuvio opettaa erikseen. Kuvion opetukseen käytetään TrainPatMaxPattern-työkalua. Työkalun asetukset löytyvät kuvasta 11 ja esimerkki opetetusta kuviosta kuvasta 12.



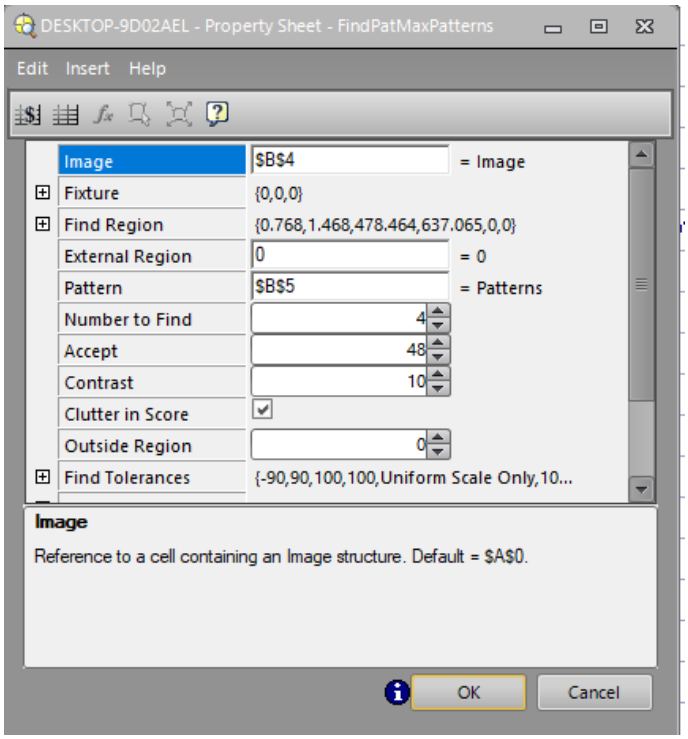
Kuva 11. TrainPatMaxPattern-työkalun asetukset

Image-kohdassa asetetaan työkalu hakemaan kuviota PointFilter-työkalun kuvasta. Tuplaklikkaamalla Pattern Region -kohtaa, saadaan avattua ikkuna, jossa voidaan määritellä alue, jolla opetettava kuvio on. Punaista hakualuetta saa siirrettyä ja kokoa muutettua sen reunoista vetämällä. Alue hyväksytään painamalla Enter-näppäintä. Ensimmäisenä opetettiin valkoisten noppien kuvio.



Kuva 12. Opetettu nopan kuvio.

Samanlainen työkalu luotiin vielä värillisille nopille. Kun molemmat kuviot on opetettu, voidaan alkaa etsiä noppia kuvasta. Tämä tapahtuu FindPatMaxPattern-työkalua käyttäen. Työkalun asetuksista löytyy esimerkki kuvasta 13.



Kuva 13. FindPatMaxPattern-työkalun asetukset

Noppia haetaan jälleen PointFilter-kuvasta. Tämä määritellään image-kohtaan. Find Region -kohdassa määritellään alue, jolta opetettua kuviota haetaan. Tuplaklikkaamalla otsikkoa, aukeaa näkymä, jossa on punainen neliö, joka määrää hakualueen. Punainen neliö tulee suurentaa niin, että se kattaa koko kuvan. Alue hyväksytään painamalla Enter-näppäintä.

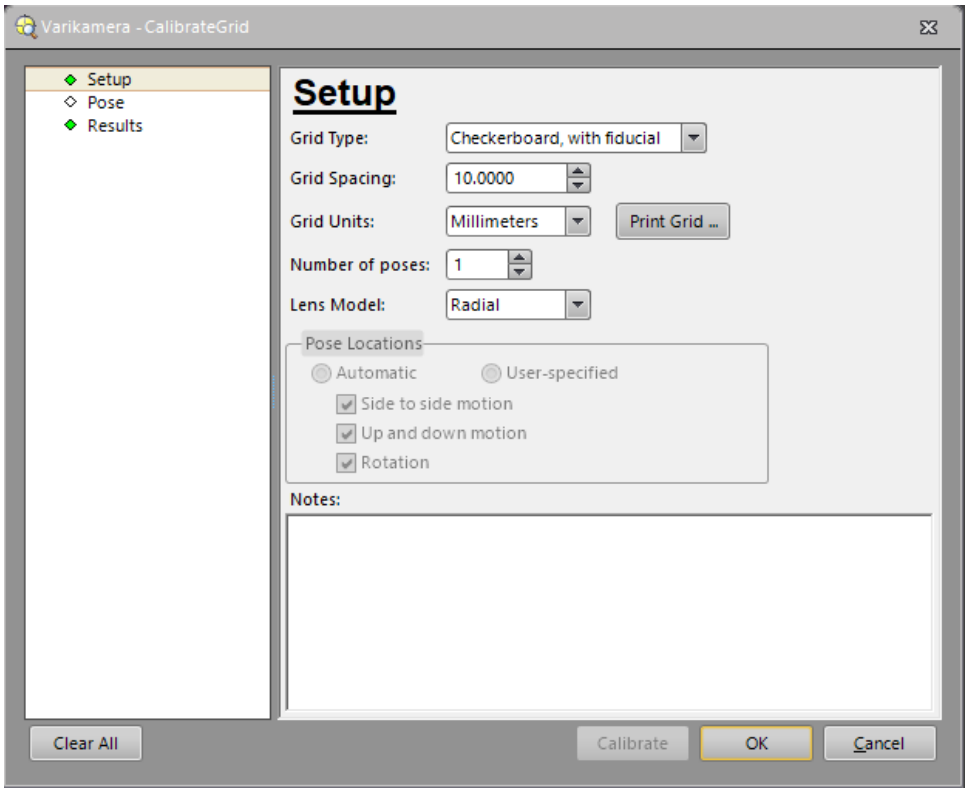
Pattern kohtaan laitetaan viittaus soluun, johon TrainPatMaxPattern-työkalulla on opetettu kuvio (esim. B5). Number to Find -kohdassa määritellään, montako kuviota etsitään. Valkoisia noppia on neljä, joten arvoksi annetaan tämä. Accept-kohdan arvoa muuttamalla voidaan määritellä, kuinka tarkkaan löytyneen kuvion tulee vastata opetettua.

Hyväksytään asetukset painamalla Ok. Työkalu lisää taulukkoon useita rivejä ja sarakkeita, sisältäen löytyneiden kuvioiden sijainnin, kulman, skaalan sekä työkalun antaman pistemäärän. Värillisiä noppia varten luodaan oma työkalu samalla tavalla.

5.1.2 Kuvan kalibrointi

Kuvan kalibroinnilla tarkoitetaan pikselien linkittämistä oikean maailman mittayksiöihin, kuten esimerkiksi millimetreihin. Tämän lisäksi kalibroinnissa voidaan kertoa kameran ohjelmalle, missä suunnassa X- ja Y-akselit ovat. Tämä toimenpide on tehtävä, jotta robotille saadaan siirrettyä noppien sijainti robotin käyttämässä koordinaatistossa.

Kalibrointi tapahtuu CalibrateGrid-nimisellä työkalulla. Työkalun saa raahattua taulukkoon oikealla olevasta työkaluvalikosta. Tämän jälkeen aukeaa kuvan 14 mukainen ikkuna.



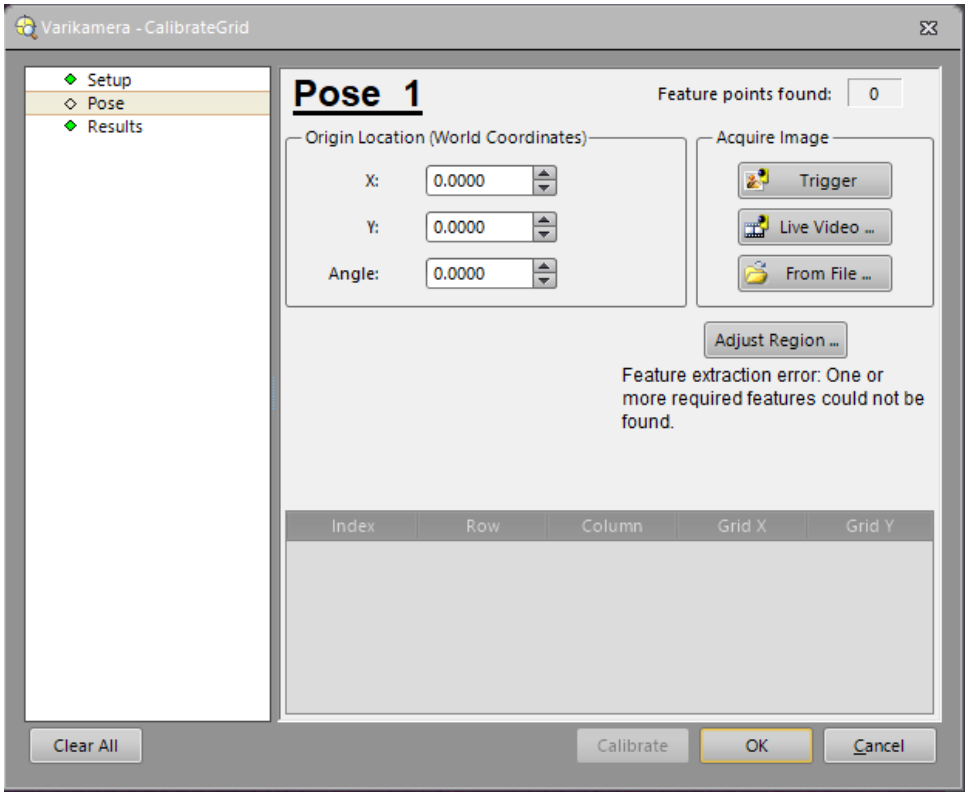
Kuva 14. CalibrateGrid-ikkuna

Aukeavasta Setup-ikkunasta saa tulostettua kalibrointia varten tarvittavan ruudukon. Tulostettavaa ruudukkoa voi muokata tarvetta vastaavaksi. Tässä tapauksessa Grid Type, Grid Spacing ja Grid Units ovat ne, jotka kiinnostavat.

Grid Type -kohdassa valitaan, minkä tyylinen ruudukko tulostetaan. Valitaan kohtaan Checkerboard, with fiducial. Cherkerboard tarkoittaa ruudukkoa, jossa on vuoron perään mustia ja valkoisia neliöitä. Lisäys, with fiducial, tuo ruudukkoon lisäksi kaksi valkoista palkkia, jotka kertovat X- ja Y-akselien suunnan.

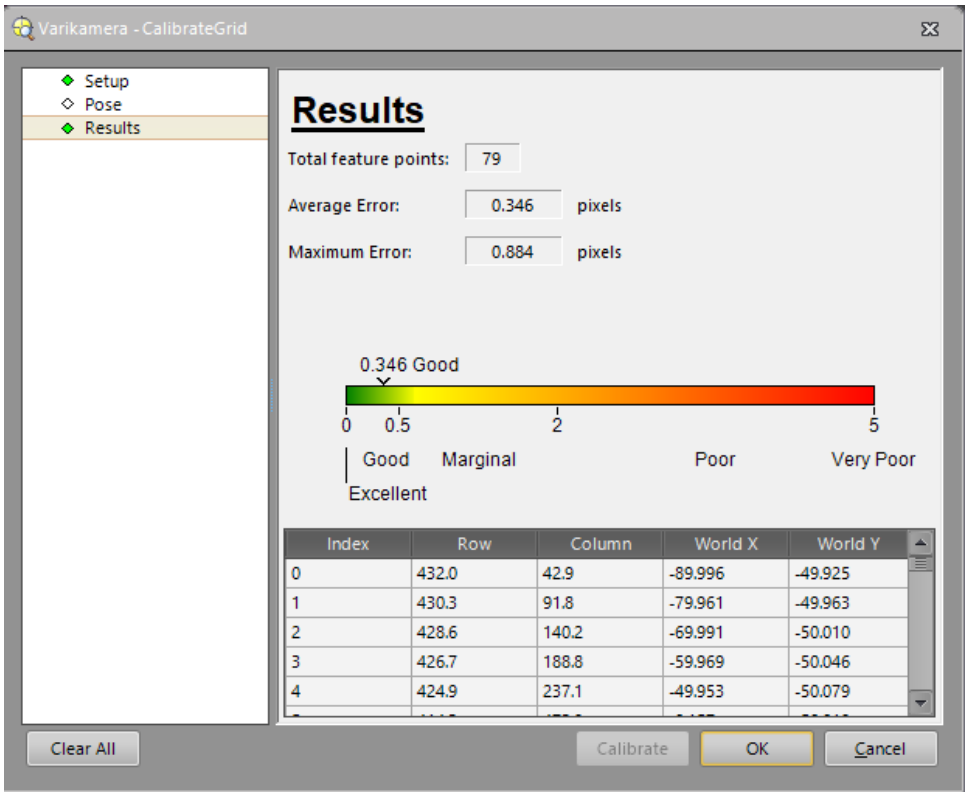
Grid Spacing kohdassa pystyy määrittelemään, kuinka suuri yksi neliö ruudukossa on. Mikäli kameran kuva-alue on laaja, niin isommat neliöt ovat järkeviä, kun taas kuva-alueen ollessa pieni, pienemmät neliöt ovat järkeviä. Työssä käytetyn ruudukon kooksi on valittu 10 mm ja ruudukko on tulostettu niin, että se on A4-paperin kokoinen. Grid Units -kohdassa määritellään, mitä mittayksikkö ruudukossa käytetään. Tähän valittiin millimetrit.

Kun asetukset ovat halutulla tavalla kunnossa, niin tulostetaan ruudukko. Tulostetusta ruudukosta on tämän jälkeen otettava kameralla kuva samassa asennossa kuin kuvattavat kohteet ovat. Kuvasta on pyrittävä saamaan mahdollisimman tarkka, jotta kalibrointi onnistuu hyvin. Asetusikkuna löytyy kuvasta 15.



Kuva 15. Kalibroinnin Pose-näkymä

Kuvan voi ottaa joko niin, että sen tallentaa erilliseen tiedostoon, jonka jälkeen se haetaan Pose-kohdassa From File -nappia painamalla. Kuvan voi myös ottaa livenä Trigger-nappia painamalla. Kun kuva on haettu, painetaan Calibrate-nappia, minkä jälkeen näkyviin tulee kuvan 16 mukainen tulosruutu.



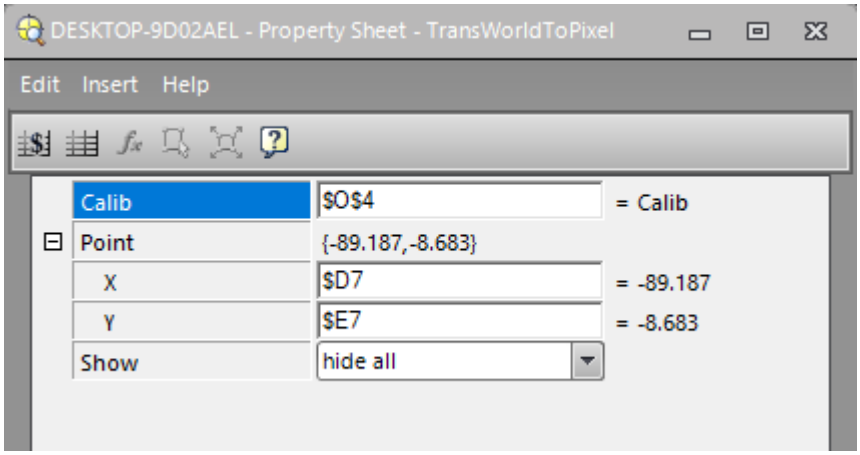
Kuva 16. Kalibroinnin tulosruutu

Tulosruutu kertoo, kuinka hyvä kalibrointikuva on. Mikäli tulos on huono, otetaan uusi, parempi kuva, ja suoritetaan kalibrointi uudelleen. Mikäli tulos on hyväksyttävä, painetaan ok ja taulukkoon ilmestyy kalibroitu kuva siihen soluun, johon työkalu laitettiin.

Kalibroinnin jälkeen on varmistettava, että ColorToGreyScale-, PointFilter-, TrainPatMaxPattern- sekä FindPatMaxPattern-työkalut käyttävät kalibroitua kuvaa. ColorToGreyScale-työkalulla on myös luotava versio, joka on kalibroimaton, sillä sitä käytetään silmälukujen tunnistuksessa.

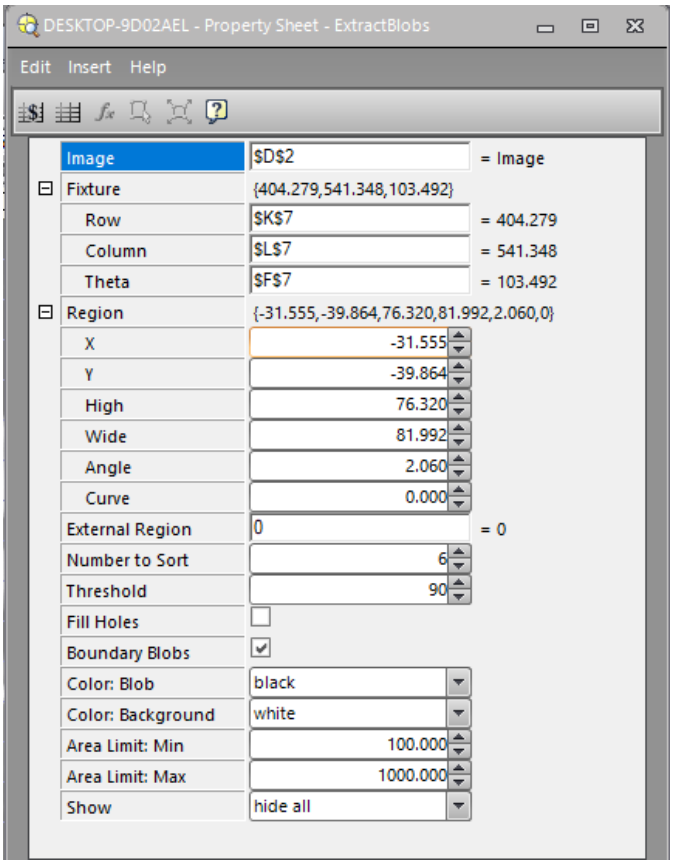
5.1.3 Silmälukujen tunnistus

Noppien silmälukujen tunnistus toteutetaan ExtractBlobs-työkalun avulla. Ennen tätä on kuitenkin työkalua varten luotava X- ja Y-koordinaatteja varten tarvittavat tiedot. Tämä tapahtuu käyttäen TransWorldToPixel-työkalua. Työkaluun haetaan FindPatmaxPattern-työkalujen antamat Col- ja Row-tiedot. Esimerkki työkalun asetuksista löytyy kuvasta 17.



Kuva 17. TransWorldToPixel-työkalun asetukset

Calib-kohtaan annetaan solu, josta löytyy kalibroitu kuva. Point-kohtaa laajentamalla saadaan X- ja Y-koordinaattien solujen paikat. X tarkoittaa Colum-arvoa ja Y tarkoittaa Row-arvoa FindPatMaxPattern-työkalusta. Jokaiselle löytyneelle nopalle tulee luoda oma TransWorldToPixel-työkalunsa. Silmälukujen etsiminen voidaan aloittaa, kun nämä työkalut on tehty. Käytetyn ExtractBlobs-työkalun asetukset löytyvät kuvasta 18.



Kuva 18. Esimerkki ExtractBlobs-työkalun asetuksista

Image kohdassa viitataan kalibroimattomaan harmaasävykuvaan. Region-kohtaa kaksoisklikkaamalla pystytään määrittämään alue, jolta blobeja haeteen. Alue tulee raahata sen nopan ylle, johon työkalu sidotaan. Alueen koko tulee rajata niin, että se on hieman noppaa pienempi, mutta kattaa kuitenkin alueen, jolla silmälukuja esiintyy. Kuvassa 18 työkalu on sidottu ensimmäiseen löytyneeseen valkoiseen noppaan.

Fixture-kohtaa laajentamalla päästään sitomaan hakualue löytyneen nopan sijaintiin. Row-kohdassa viitataan TransWorldToPixel-työkalun antamaan X-arvoon ensimmäisestä löytyneestä valkoisesta nopasta. Y-kohdassa viitataan taas Column-arvoon. Theta on viittaus ensimmäisen löytyneen valkoisen nopan Angle-arvoon.

Number to sort -kohdassa kerrottaan, montako blobia maksimissaan haetaan. Koska käytössä on kuusisivuiset nopat, annetaan kohdan arvoksi 6. Threshold-kohdassa määritellään raja-arvoa blobin löytymiselle. Arvoksi laitetaan 90, jotta vältytään siltä, että työkalu löytäisi esimerkiksi valon heijastuman silmäluvun sijaan.

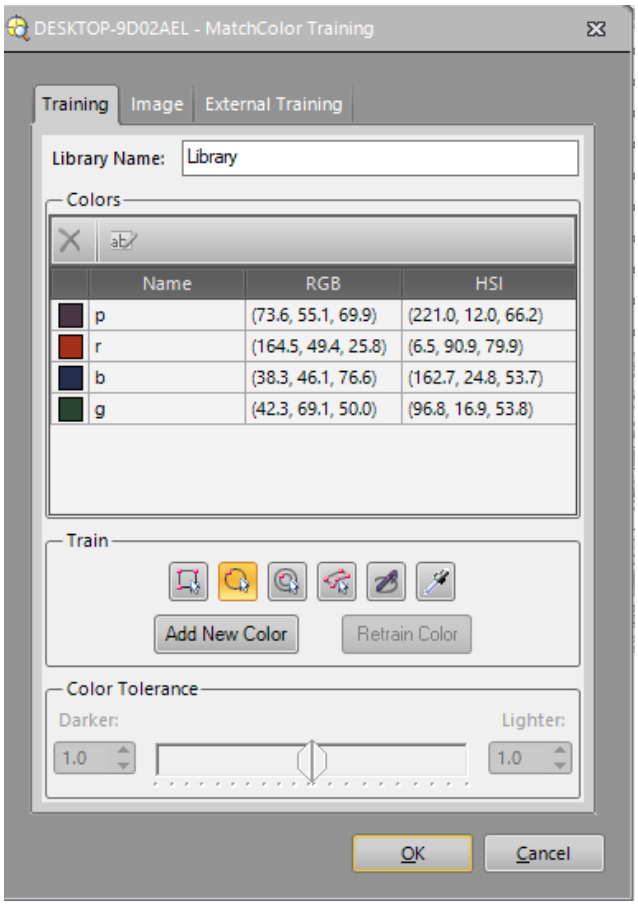
Valkoisen nopan ollessa kyseessä, Color: Blob -kohtaan valitaan black ja Color: Background -kohtaan valitaan white. Värillisten noppien kohdalla näiden kohtien arvot ovat toisinpäin, koska värillisten noppien silmäluvut ovat valkoisia eikä mustia.

Area Limit -kohdilla rajataan vielä, kuinka suuria blobeja haetaan. ExtractBlobs-työkaluja tehdään jokaiselle nopalle omansa. On tärkeää, että noppien hakualueet laitetaan aina vastaamaan sitä löytynyttä noppaa, jonka tietoja haetaan.

5.1.4 Värien tunnistus

Värien tunnistuksessa käytetään TrainMatchColor- ja MatchColor-työkaluja. Työkalut raahataan taulukkoon samalla tavalla kuin muutkin työkalut. TrainMatchColor-työkalulla opetetaan värit ja MatchColor-työkalulla opetettuja värejä tunnistetaan kuvasta. Koska noppien värit on ennalta määrättyjä ja valkoiset nopat ovat isompia kuin värilliset, ei valkoista väriä tarvitse opettaa. Voidaan aina olettaa, että valkoiset nopat ovat siihen tarkoitetulla työkalulla etsittyjä.

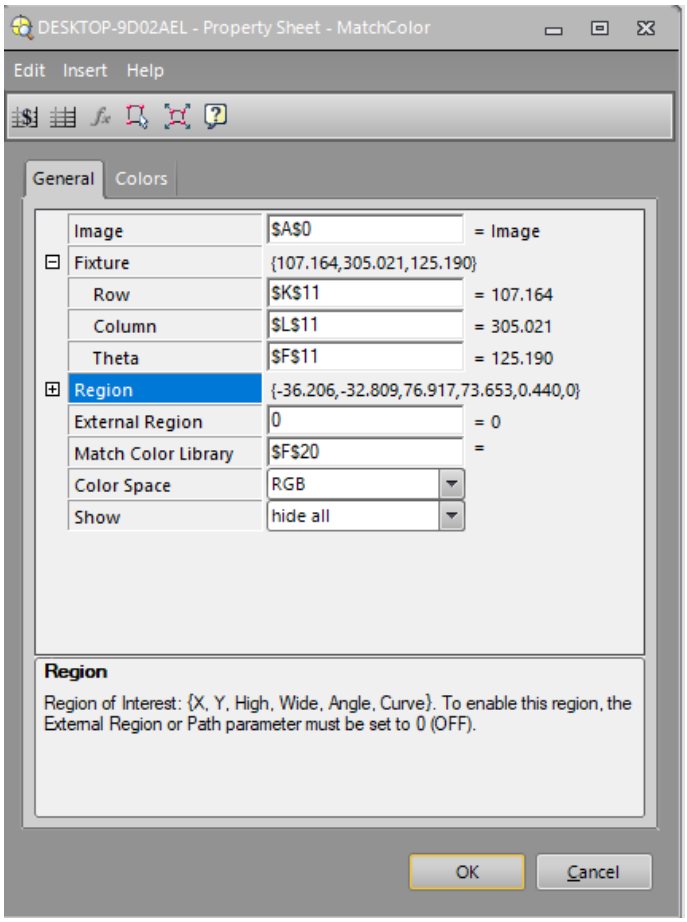
TrainMatchColor-työkalussa uusi väri lisätään painamalla Add New Color -nappia. Tämä aukaisee ikkunan, jossa määritellään, miltä alueelta väri opetetaan. Oletuksena tarjolla on ympyrän muotoinen alue, jonka kokoa ja sijaintia voi muuttaa. Alue sijoitetaan niin, että siinä on mahdollisimman paljon haluttua väriä. Tämän jälkeen alue hyväksytään painamalla Enter-näppäintä. Värin nimen voi muuttaa kaksoisklikkaamalla sen nimeä listassa. Color Tolerance -kohdasta on mahdollista kertoa työkalulle, kuinka paljon tummempi tai vaaleampi väri vielä hyväksytään opetetuksi väriksi. Kuvassa 19 näkyy TrainMatchColor-työkalun asetusikkuna.



Kuva 19. TrainMatchColor-työkalun asetusikkuna

Tarvittaessa Add New Color -napin yläpuolelta löytyy erimuotoisia aluetyökaluja sekä myös työkalu vapaasti piirrettävän alueen käyttöön. Loput opetettavat värit opetetaan samalla periaatteella.

MatchColor-työkalu etsii opetettuja värejä. Tämä työkalu antaa ulos tuloksia, joten sille on hyvä varata tilaa taulukosta. Jokaiselle nopalle luodaan oma MatchColor-työkalu. Kuvassa 20 näkyy Pose-ikkuna.



Kuva 20. MatchColor-työkalun asetukset

Image-kohdassa viitataan soluun, josta löytyy alkuperäinen värikuva. Match Color Library -kohdassa viitataan soluun, johon TrainMatchColor-työkalu sijoitettiin. Näin työkalu saa tietoonsa, mitä värejä sen tulee etsiä.

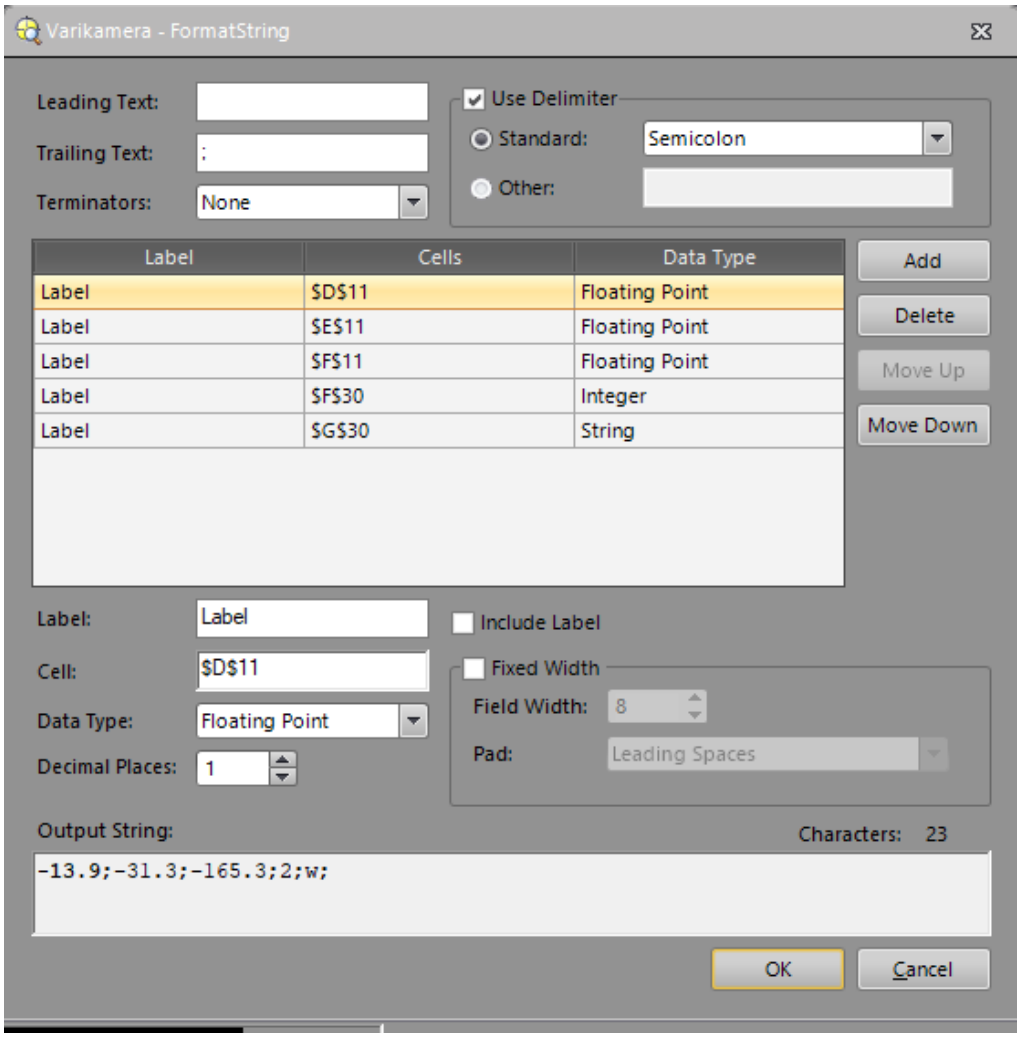
Fixture-kohdassa määritellään mihin löytyneeseen kappaleeseen hakualue sidotaan. Linkitys tapahtuu samalla tavalla kuin ExtractBlobs-työkaluilla silmälukujen haun yhteydessä. Koska tietoja haetaan vain värillisistä nopista, vain näiden tietoihin tehdään linkitys. Työkaluja luodaan siis vain neljä kappaletta.

Region-kohdassa määritellään miltä alueelta värejä etsitään. Hakualue määritellään samalla tavalla, kuin ExtraBlobs-työkaluille silmälukujen etsinnässä.

Colors-välilehdeltä löytyy lista väreistä, joita työkalu etsii. On syytä tarkistaa, että jokaisen värin kohdalla on ruksi ruudussa. Muutoin väriä ei etsitä.

5.1.5 Tietojen koostaminen

Tuloksien koostamiseen käytetään FormatString-työkalua. Sen avulla voidaan koota useammassa solussa oleva tieto yhteen soluun. Kun työkalun raahaa taulukkoon, niin aukeaa kuvan 21 mukainen ikkuna.



Kuva 21. FormatString-asetusikkuna

Tiedot erotellaan toisistaan puolipistettä käyttäen. Syntynyt merkkijono myös päätetään puolipisteellä. Robotin ohjelmassa tiedot haetaan tämän erotusmerkin perusteella. Koottuun merkkijonoon haetaan kalibroidusta FindPatMaxPattern-työkalusta Row-, Column- ja Angle-tiedot sekä nopan silmäluku ja väri. Valkoisten noppien osalta väritieto on kirjattu erikseen soluihin ja värillisten noppien osalta tieto haetaan MatchColor-työkalun tiedoista. Decimal places -kohdassa voidaan määritellä, monenko desimaalin tarkkuudella kukin luku esitetään.

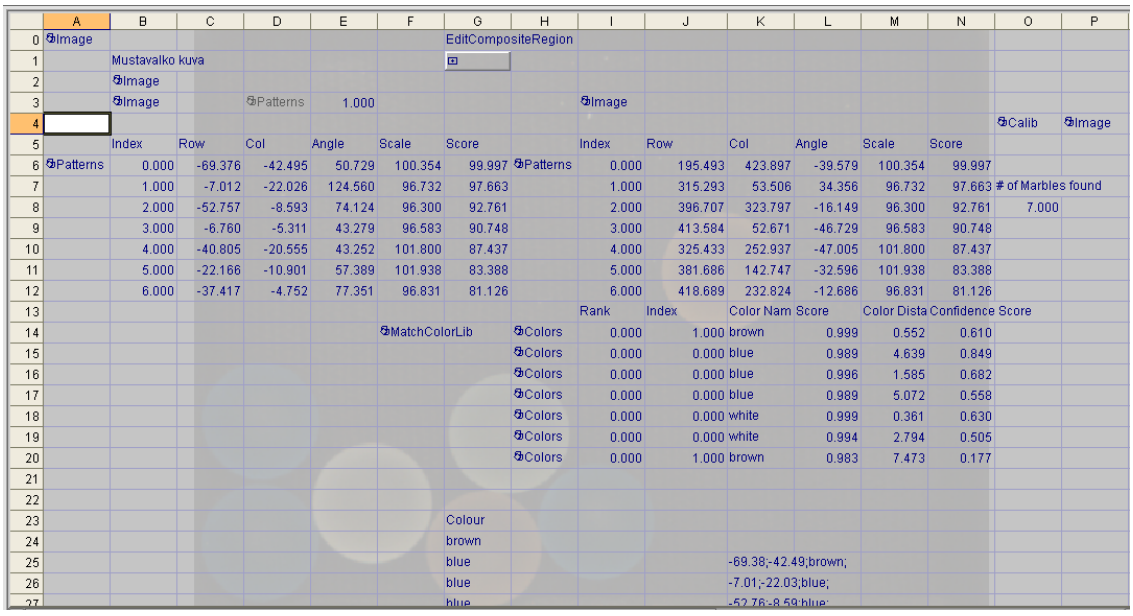
5.1.6 Lisätietoja robotin ohjelmalle

Robotin ohjelma tarvitsee kameralta vielä muutamia lisätietoja. Ensimmäisenä tarvitaan tieto siitä, montako kappaletta kuvasta on löydetty. Noppia on maksimissaan kahdeksan kuvassa, joten siitä vähennetään FindPatMaxPattern-työkalujen Angle-kohdasta löytyvien virheellisten solujen määrä. Mikäli kaikki nopat löytyvät, solu saa arvokseen luvun 7. Mikäli esimerkiksi kaksi noppaa jää jostain syystä uupumaan, niin solu saa arvokseen luvun 6.

Robotin ohjelmalle välitetään myös tieto löytyneiden virheellisten solujen määrästä. Tätä tietoa käytetään robotin ohjelmassa päättämään, tarvitseeko ottaa uusi kuva. Tähän käytetään jälleen CountError-työkalua ja hakualueeksi annetaan molempien FindPatMaxPattern-työkalujen tulosalueet.

5.2 Kuulien ohjelma

Kuulien ohjelma toteutetaan hieman eri tavalla kuin noppien ohjelma. Käytetyt työkalut ovat kuitenkin samoja, TrainPatMaxPattern sekä FindPatMaxPattern. Työn alussa tehdään mustavalkoinen kuva samalla tavalla (ColorToGreyScaleFilter-työkalulla), kuin noppien kanssa. Kuvassa 22 näkyy ohjelma kokonaisuudessaan yleiskuvan saamiseksi.

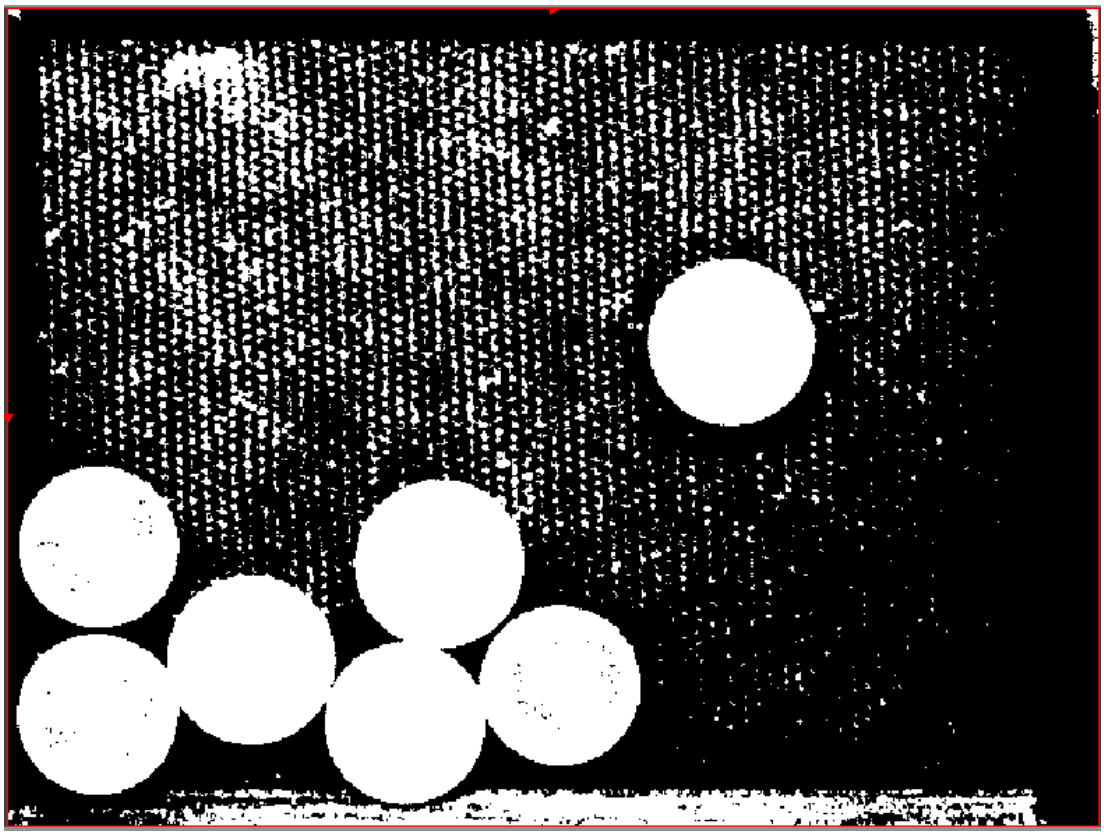


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
0	Image						EditCompositeRegion									
1		Mustavalko kuva														
2		Image														
3		Image		Patterns	1.000				Image							
4															Calib	Image
5		Index	Row	Col	Angle	Scale	Score		Index	Row	Col	Angle	Scale	Score		
6	Patterns	0.000	-69.376	-42.495	50.729	100.354	99.997	Patterns	0.000	195.493	423.897	-39.579	100.354	99.997		
7		1.000	-7.012	-22.026	124.560	96.732	97.663		1.000	315.293	53.506	34.356	96.732	97.663	# of Marbles found	
8		2.000	-52.757	-8.593	74.124	96.300	92.761		2.000	396.707	323.797	-16.149	96.300	92.761	7.000	
9		3.000	-6.760	-5.311	43.279	96.583	90.748		3.000	413.584	52.671	-46.729	96.583	90.748		
10		4.000	-40.805	-20.555	43.252	101.800	87.437		4.000	325.433	252.937	-47.005	101.800	87.437		
11		5.000	-22.166	-10.901	57.389	101.938	83.388		5.000	381.686	142.747	-32.596	101.938	83.388		
12		6.000	-37.417	-4.752	77.351	96.831	81.126		6.000	418.689	232.824	-12.686	96.831	81.126		
13									Rank	Index	Color Nam	Score	Color Dista	Confidence Score		
14						MatchColorLib		Colors	0.000	1.000	brown	0.999	0.552	0.610		
15								Colors	0.000	0.000	blue	0.989	4.639	0.849		
16								Colors	0.000	0.000	blue	0.996	1.585	0.682		
17								Colors	0.000	0.000	blue	0.989	5.072	0.558		
18								Colors	0.000	0.000	white	0.999	0.361	0.630		
19								Colors	0.000	0.000	white	0.994	2.794	0.505		
20								Colors	0.000	1.000	brown	0.983	7.473	0.177		
21																
22																
23							Colour									
24							brown									
25							blue				-69.38;-42.49;brown;					
26							blue				-7.01;-22.03;blue;					
27							blue				-52.76;-8.59;blue;					

Kuva 22. Kuulien ohjelma kokonaisuudessaan

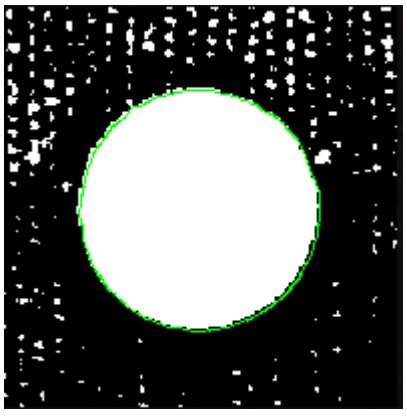
5.2.1 Kuulien tunnistus

Kuulien tunnistamiseksi soluun B3 luodaan PointFilter-niminen työkalu. Tämä luo kuvasta mustavalkoisen version, jossa jokainen kuula erottuu valkoisena ympyränä. Työkalun asetuksista klikataan region-kohtaa ja aukeavasta näkymästä raahataan punainen hakualue niin, että se kattaa koko kuva-alueen. Alue hyväksytään Enteriä painamalla. Valikossa määritellään myös mihin kuvaan (Image) työkalu kohdistetaan. Tähän laitetaan viittaus kalibroituun kuvaan, joka löytyy solusta P4. Kuvassa 23 on esimerkki, miltä kuvausalue näyttää työkalun kanssa.



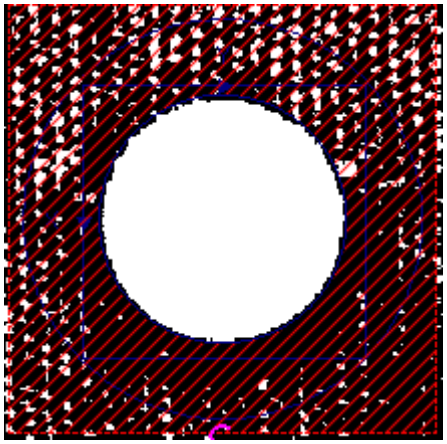
Kuva 23. PointFilterin kuva-alue

Seuraavaksi ohjelmaan tuodaan TrainPatMaxPattern-työkalu. Työkalulle opetetaan yksi valkoinen ympyrä samalla tavalla kuin noppien ohjelmassa. Esimerkki haetusta kuviosta löytyy kuvasta 24. Image-kenttään laitetaan viittaus soluun B3, jossa on PointFilterin luoma kuva.



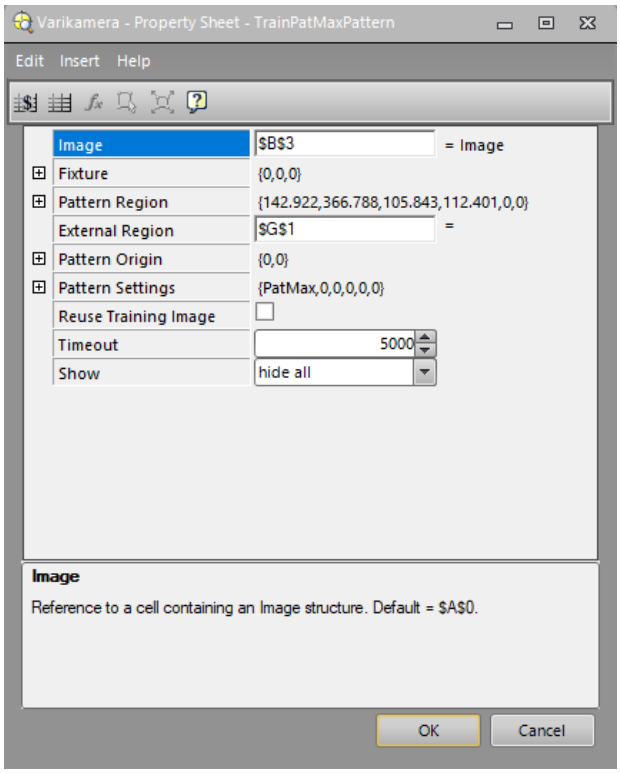
Kuva 24. TrainPatMaxPattern-työkalun kuvio

Soluun G1 luodaan työkalu EditCompositeRegion. Työkalulla saadaan rajattua hakualuetta niin, että siinä on vain valkoinen ympyrä ja sen ympärillä olevat valkoiset pisteet jäävät pois. Tämän tapahtuu ensin raahaamalla punainen neliö niin, että se sisältää valkoisen ympyrän. Tämän jälkeen klikataan neliön sisälle oikealla hiiren napilla ja valitaan aukeavasta valikosta Add subregion → Circle. Raahataan ympyrä neliön sisällä niin, että se sisältää valkoisen ympyrän. Lopputuloksen tulisi näyttään kuvan 25 mukaiselta.



Kuva 25. EditCompositeRegion-työkalun alueet

Hyväksytään alueet painamalla enteriä. Klikataan TrainPatMaxPattern-työkalu jälleen auki ja lisätään ExternalRegion-kohtaan viittaus EditCompositeRegion-työkaluun eli soluun G1. Esimerkki asetuksista löytyy kuvasta 26.



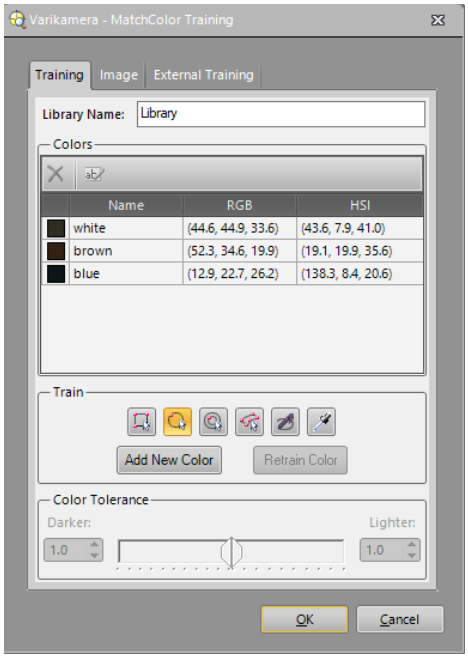
Kuva 26. TrainPatMaxPattern-työkalun asetukset

Soluun A6 lisätään FindPatMaxPattern-työkalu. Image kohtaan laitetaan työkalu viittaamaan soluun B3. Find Region määritellään kattamaan koko kuva-alue. Pattern-kohta laitetaan viittamaan soluun D3, josta löytyy juuri tehty pattern. Number to find -kohdan arvoksi laitetaan 7, joka on kuulien maksimimäärä. Hyväksytään asetukset klikkaamalla ok. Taulukkoon ilmestyy löytyneet patternit koordinaatteineen.

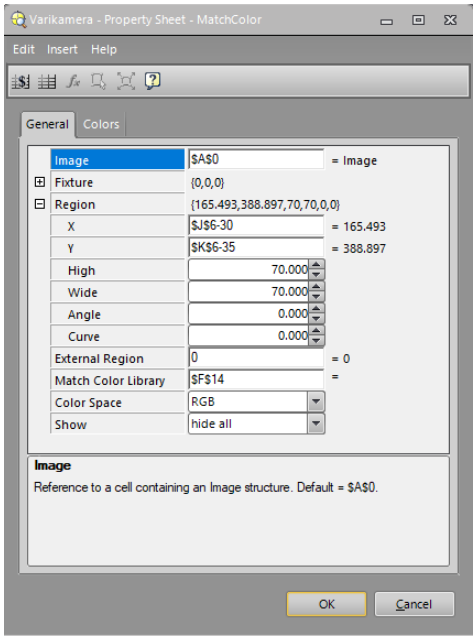
Soluun I3 luodaan kopio jo tehdystä PointFilter-työkalusta ja asetetaan sen Image-kohtaan arvoksi B2. Solusta löytyy kalibroimaton ColorToGreyScaleFilter-kuva. Tämän jälkeen luodaan kopio jo luodusta FindPatmaxPattern-työkalusta. Kopio asetetaan soluun H6. Työkalun image-kohta muutetaan viittaamaan soluun I3. Näin saadaan värien hakua varten kalibroimattomat sijainnit löytyneille patterneille.

5.2.2 Värien tunnistus

Värien tunnistus toteutetaan samalla tavalla kuin noppien kanssa. Käytetään siis TrainMatchColor- ja MatchColor-työkaluja. Kuulien sijaintitietoina käytetään kalibroimattoman FindPatMaxPattern-työkalun tietoja. Värien opetus tapahtuu helpoiten käyttämällä ympyränmuotoista opetusaluetta. Esimerkit työkalujen asetuksista löytyvät kuvista 27 ja 28.



Kuva 27. Kuulien värien opetus



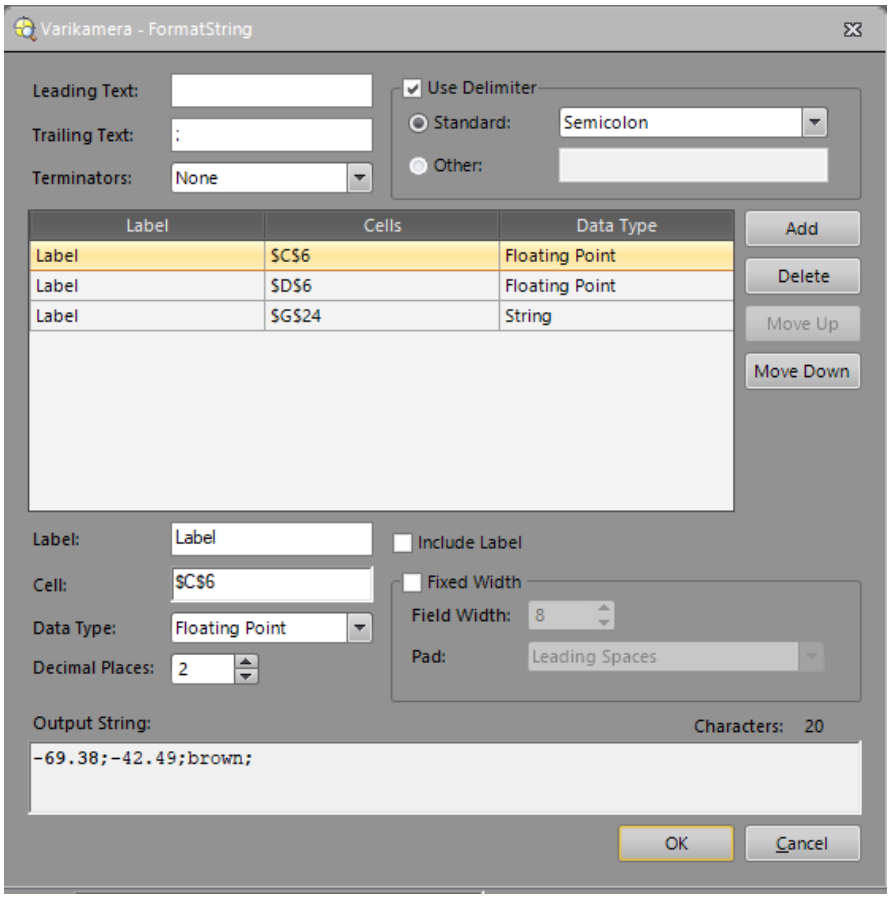
Kuva 28. Esimerkki kuulan värin hakemisesta

5.2.3 Kuulien kuvan kalibrointi

Kuulien kuvan kalibroinnissa käytetään samaa tapaa ja kuvaa kuin noppien kanssa. CalibrateGrid-työkalu sijoitetaan soluun O4 ja CalibrateImage-soluun P4. Näillä kuulien kuva saadaan kalibroitua niin, että kameran antamat pikselilukemat muuttuvat oikeanmaailman mittayksiköiksi. Kuten noppienkin työssä, yksikköinä käytetään millimetrejä.

5.2.4 Paikka- ja väritietojen koostaminen

Paikka- ja väritiedot koostetaan robotin ohjelmaa varten käyttäen FormatString-työkalua. Erottimena käytetään puolipistettä ja jokainen kerätty tieto myös päätetään puolipisteellä. Robotille lähetetään X-, Y- ja väritiedot. Esimerkki tietojen koostamisesta löytyy kuvasta 29. Tiedot lähetetään robotille yhden desimaalin tarkkuudella.



Kuva 29. Tietojen koostaminen

5.3 Telnet-kommunikointi

Kommunikointi toteutetaan Telnet-protokollaa käyttäen Cognexin Native Mode -komennoilla. Telnet-yhteyden luontia varten ei tarvitse tehdä mitään erityisiä asetuksia. On kuitenkin hyvä varmistaa, mitä porttia kamera käyttää kommunikointiin. Tämän saa selville In-Sight Explorer -ohjelmassa menemällä valikkoon Sensor → Network Settings. Aukeavasta valikosta löytyy kohta Telnet Port. Oletuksena arvon pitäisi olla 23. Valikosta saa myös selville kameran IP-osoitteen, jotta yhteyttä voi testata.

Yhteyden testaamiseen käy mikä tahansa Telnet-ohjelmisto. On syytä huomioida, että joissain ohjelmissa Telnet-yhteyden muodostustavaksi pitää valita passiivinen. Muutoin yhteys ei toimi.

Onnistuneen yhteyden luonnin jälkeen kamera kysyy käyttäjätunnusta ja salasanaa. Oletuksena sisään pääsee tunnuksella admin. Salasana on tyhjä. Oikeassa tuotantoympäristössä salasanan asettaminen olisi tietoturvan kannalta järkevää, joskin tällöinkin hyöty olisi kyseenalaista, koska Telnet ei salaa tiedonkulkua millään tavalla. Salasana kulkisi siis verkon ylitse selkokielisenä ja olisi helposti hyökkääjän siepattavissa.

5.4 Native Mode -komennot

Cognex tarjoaa laajan valikoiman niin kutsuttuja Native Mode -komentoja, joilla kameraa voidaan käskyttää. Tämän työn osalta komentoja tarvitaan kuvan ottamiseen, tulosten hakuun, työn vaihtamiseen sekä kameran asettamiseen Online- ja Offline-tiloihin. Komennot eivät erottele isoja tai pieniä kirjaimia. Jokainen komento päättyy Carrier Return (CR)- ja Line Feed (LF)-merkkeihin (ASCII merkit 13 + 10) Telnet-yhteyttä käyttäessä. Jokainen komento palauttaa luvun 1, mikäli komento onnistuu ja tämän jälkeen tulostaa tuloksen. Mikäli komento epäonnistuu, se palauttaa negatiivisen luvun (esim. -2). Mikäli palautettava luku on 0, on komento tuntematon. (Cognex In-Sight Explorer Help 2020.)

Työssä käytetään seuraavia komentoja:

- SW8 - Ottaa kuvan. Palauttaa 1 onnistuessaan.
- SJ[numero] - Asettaa annetun numeroisen työn aktiiviseksi. Kameran pitää olla offline-tilassa. Työn tiedostonimessä tulee olla ensimmäisenä kirjaimena työn numero, jotta komento tunnistaa oikean työn. Sopiva työn nimi on siis esimerkiksi 1dice.job tai 2marbles.job.
- GV[Sarake][Rivi] - Hakee annetun solun arvon. Esimerkiksi komento GVC001 palauttaa solun C1 arvon.
- SO[Numero] - Kameran asettaminen online tilaan [1] tai offline [0] tilaan

6 ROBOTIN OHJELMA

Tämän työn tarkoituksena ei ole toimia johdatuksena RAPID-ohjelmointikieleen tai ohjelmointiin ylipäätänsä. Koodia ei käydä rivi riviltä läpi eikä ohjelmointikielen syntaksia ja muita ominaispiirteitä avata sen tarkemmin. Ohjelmakokonaisuus käydään kuitenkin läpi niin, että sen toimintaperiaate tulee selväksi ja ohjelmointikielen perusteet osaava pystyy sen, ja liitteenä olevan kommentoidun koodin avulla, toteuttamaan vastaavan kokonaisuuden.

6.1 Ohjelman toimintalogiikka

Ennen kuin kirjoittaa riviäkään koodia, on hyvä olla selvillä, mitä ohjelman pitäisi tehdä. Tässä tapauksessa ohjelmalla on selkeitä erillisiä toimintakokonaisuuksia ja askeleita. Selostuksen selkeyttämiseksi alustoille on annettu omat kirjaimensa. A tarkoitta puista alustaa ja siinä olevaa kuvausaluetta, B viitta kuulien alustaan, C valkoisten noppien alustaan ja D värillisten noppien alustaan.

1. Ohjelma käynnistetään ja robotti ottaa yhteyden kameraan Telnetin ylitse.
2. Robotti noukkii nopan alustalta (C).
3. Robotti tiputtaa nopan sille varatulle alustalle (A).
4. Robotti noukkii loput nopat yksitellen alustalta (C), tiputtaa ne alustalle (A) ja toistaa toimenpiteen alustan (D) nopille.
5. Robotti antaa käskyn älykameralle ottaa alustasta (A) kuvan ja palauttaa robotille tiedon noppien silmäluvusta, väristä ja sijainnista.
6. Robotti noukkii nopat alustalta (A) silmäluvun ja värien määräämässä järjestyksessä ja palauttaa ne alustalle (C ja D).
7. Robotti nostaa alustan (B) porttia, jotta kuulat vierivät alustalle (A).
8. Robotti palauttaa alustan (B) portin paikalleen.
9. Robotti antaa käskyn älykameralle ottaa alustasta (A) kuvan ja palauttaa robotille tiedon kuulien väristä ja sijainnista.
10. Robotti noukkii kuulat värin perusteella alustalta (A) ja palauttaa ne niille varatulle alustalle (B).
11. Ohjelma suoritetaan uudelleen alusta alkaen päättymättömänä kierroksena.

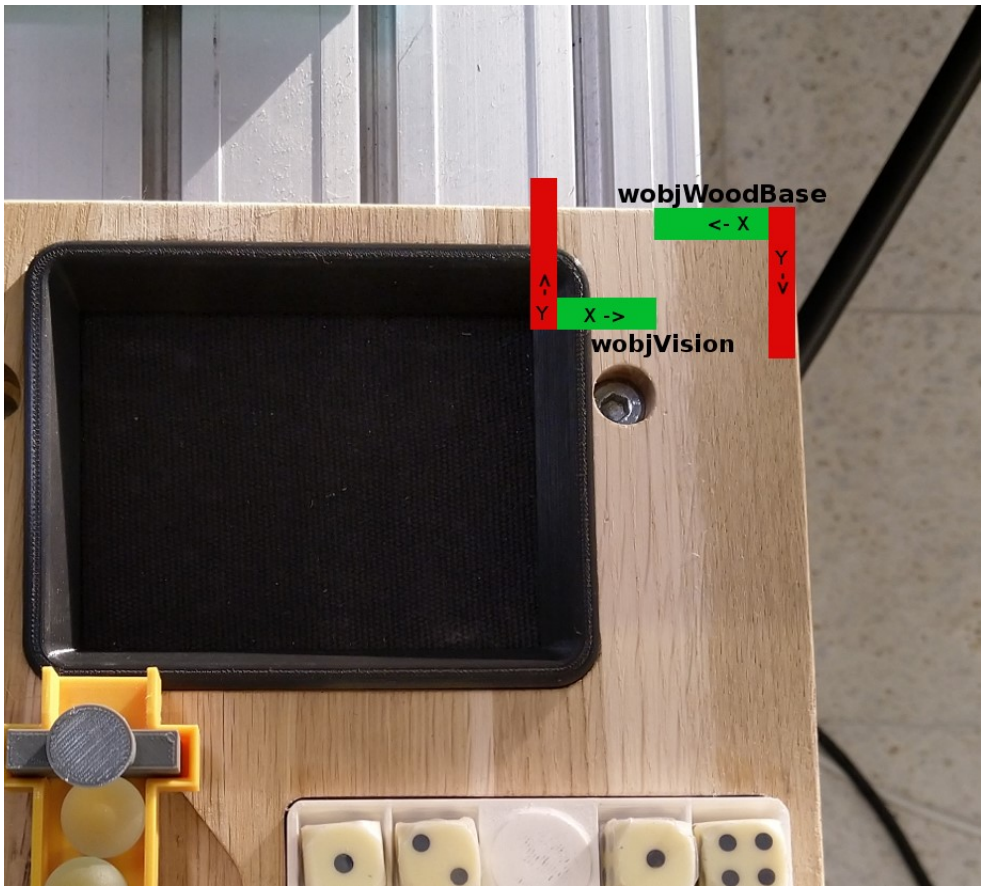
Näin esitettynä ohjelman toiminnallisuus on helppo tajuta. Lisäksi se on pilkottu pienempiin osiin, jotka on helppo toteuttaa yksitellen omina kokonaisuuksinaan.

6.2 Work objectit

Work object eli työkohde on robotille opetettu koordinaatisto, jolla on tietyt ominaisuudet. Työkohteita käytetään yksinkertaistamaan ohjelmointia tilanteissa, joissa työkappale voi siirtyä eri paikkaan. (ABB 2018, 192.)

Esimerkiksi tässä työssä wobjWoodBase on nimensä mukaisesti sidottu puiseen alustaan, jolla kaikki muut työn osat ovat. Kun robotille opetetaan paikkapisteet suhteessa tähän koordinaatistoon, niin mikäli alusta siirtyy, tarvitsee vain opettaa uudelleen, missä wobjWoodBase-koordinaatiston origo on. Yksittäisiä paikkapisteitä ei siis tarvitse erikseen opettaa uudelleen.

Ohjelmaa varten luodaan kaksi erillistä work objectia. wobjWoodBase opetetaan puisen alustan reunan mukaan ja wobjVision opetetaan suhteutettuna wobjWoodBase koordinaatistoon niin, että sen origo vastaa älykameran kalibroinnin yhteydessä määritellyn origon sijaintia. Kuvassa 30 suuntaa antavasti, missä koordinaatistojen origot sijaitsevat.



Kuva 30. Koordinaatistojen origojen sijainnit

Työkoordinaatistojen opetukseen käytettiin kolmipiste menetelmää. Menetelmässä opetetaan X-akselilta kaksi pistettä ja yksi piste Y-askelilta. Tarkempi ohje, miten opetus tapahtuu, löytyy sivulta 194 ABB:n Käyttäjän opas – IRC5 ja FlexPendant dokumentista (2018).

6.3 Paikkapisteet

Paikkapiste on johonkin robotin käytössä olevaan koordinaatistoon opetettu piste. Sen avulla robotille kerrotaan, minne sen tulisi liikkua ja missä asennossa sen akselien tulisi olla pisteeseen päästyään. (ABB 2010, 1176.)

Kappaleiden pudotukseen ja takaisin paikalleen laittoon liittyvät paikkapisteet opetetaan suhteutettuna wobjWoodBase-koordinaatistoon. Ainoat pisteet, jotka opetetaan wobjVision-koordinaatistoon suhteutettuna, ovat kappaleiden poimintaan ja pudotukseen käytetyt pisteet.

Pisteiden opetukseen löytyy ohje ABB:n dokumentista Käyttäjän Opas – IRC5 ja FlexPendant sivulta 166 alkaen (2018). Pisteitä opettaessa on tärkeää pitää mielessä, että robottia liikutetaan ja opetetaan silloin, kun käytössä on oikea työkoordinaatisto.

6.3.1 wobjWoodBase-paikkapisteet

Kappaleiden pudotukseen ja paikalleen nostoon liittyvät paikkapisteet opetettiin wobjWoodBase-koordinaatiston suhteen. Tämä pitää sisällään kuulien portin nostoon liittyvät paikkapisteet sekä kuulien takaisin paikalleen pudotusta varten tehdyt pisteet. Myös noppien pudotukseen liittyvät pisteet opetettiin tämän koordinaatiston suhteen, poissulkien kuvausalueen yläpuolella oleva piste. Noppien takaisin alustalle pudotukseen liittyvät pisteet opetettiin myös tämän koordinaatiston suhteen.

6.3.2 wobjVision-paikkapisteet

pVisPick on paikkapiste, joka opetetaan wobjVision-työkoordinaatiston suhteen. Se sijoitetaan wobjVision-koordinaatiston origoon. Paikkapistettä käytetään kameran näköalueella olevien kappaleiden poimintaan. Paikkapistettä siirretään käyttäen Off-komentoa, sekä antamalla sille parametreinä kameralta saadut sijainti tiedot.

```
MoveL    Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index},-
         12),v500,fine,tool0\WObj:=wobjVision;
```

Yllä oleva liikekäsky siirtää robotin kuulasta saadun X- ja Y-koordinaattien osoittamaan sijaintiin ja menee 12 millimetriä alemmaksi, kuin opetettu piste.

pVisDrop-paikkapiste on noppien yksitellen pudottamiseen opetettu piste. Sitä siirrellään Offs-komennolla, jotta vältyttäisiin siltä, että nopat putoaisivat toistensa päälle.

6.4 Ohjelmassa käytetyt funktiot

Tässä luvussa käydään tarkemmin lävitse työssä käytettyjä funktioita. Ohjelmassa käytetään RAPID-kielen omia funktioita tiettyjen asioiden tekemiseen, joten niiden käyttötarkoitus ja toiminta on syytä käydä läpi, jotta koodista saa helpommin selvää.

6.4.1 SocketCreate

SocketCreate-komentoa käytetään luomaan uusi socket TCP/IP- tai UDP/IP-yhteyden luomista varten. Tätä varten luodaan uusi muuttuja, jonka tyyppi on socketdev. Komento hyväksyy vain socketdev-tyyppisiä muuttujia.

Esimerkkikoodi:

```
VAR socketdev socket1;
...
SocketCreate socket1;
```

Kaikki Socket-komennot vaativat, että robotin kontrollerissa on enabloitu PC Interface -ominaisuus. (ABB 2010, 460.)

6.4.2 SocketConnect

SocketConnect-komennolla luodaan yhteys toiseen laitteeseen. Komennolla kerrotaan SocketCreate-komennolla luodulle socket-muuttujalle kohdelaitteen IP-osoite sekä portti numero, johon yhteyttä koitetaan luoda.

Esimerkkikoodi:

```
SocketConnect socket1, "192.168.0.1", 23;
```

Edellinen komento luo yhteyden IP-osoitteeseen 192.168.0.1 ja käyttää porttia 23. Mikäli nämä tiedot sijoitetaan muuttujiin, tulee IP-osoitteen olla muotoa merkkijono ja portin numeerista tietoa. (ABB 2010, 457.)

6.4.3 SocketSend

SocketSend-komennolla voidaan lähettää dataa toiselle laitteelle. Toimiakseen komento vaatii, että SocketCreate-komennolla on luotu socket kommunikaatiota varten ja yhteys toiseen laitteeseen on luoto SocketConnect-komennolla. (ABB 2010, 469.)

Esimerkkikoodi:

```
SocketSend socket1 \Str := "Hello world";
```

Edellinen komento lähettää “Hello World” -merkkijonon toiselle laitteelle.

6.4.4 SocketReceive

SocketReceive-komentoa käytetään datan vastaanottoon toiselta laitteelta. Vastaanotetut viestit voivat olla tyypiltään merkkijonoja (string), raakadataa, aikakoodeja tai bittimäärien lukemista. Toimiakseen komento vaatii, että SocketCreate-komennolla on luotu socket kommunikaatiota varten ja yhteys toiseen laitteeseen on luotu SocketConnect-komennolla. (ABB 2010, 464.)

Esimerkkikoodi string-tyyppisen datan vastaanotosta:

```
VAR string str_data;
...
SocketReceive socket1 \Str := str_data;
```

Koodi lukee vastaanotetun datan str_data-nimiseen muuttujaan. Huomattavaa on, että RAPID-kielessä luettavan merkkijonon maksimipituus on 80 merkkiä. (ABB 2010, 464.)

6.4.5 StrPart

StrPart (String Part) -komento etsii annetusta merkkijonosta määritellyltä merkkiväliltä löytyvät merkit ja muodostaa niistä uuden merkkijonon.

Esimerkkikoodi:

```
VAR string part;
part := StrPart("Robotics",1,5);
```

Ensin määritellään merkkijonotyyppinen muuttuja part. Tämän jälkeen muuttujaan luetaan StrPart-komennolla ensimmäiset viisi merkkiä sanasta “Robotics”. Part-muuttuja saa siis arvokseen “Robot”. Annettu merkkijono voi olla tyypiltään muuttuja, kunhan se on merkkijonotyyppiä. Myös alku- ja loppu-arvot voidaan antaa muuttujilla, kunhan ne ovat tyypiltään numeerisia. (ABB 2010, 1005.)

6.4.6 TPWrite & TPErase

TPWrite-komennolla kirjoitetaan viesti robotin flexpendantin näytölle. Tämä on kätevä tapa antaa käyttäjälle palautetta siitä, mitä ohjelma tekee.

Esimerkkikoodi:

```
TPWrite "Execution started";
```

Komento kirjoittaa flexpendantin näytölle merkkijono “Execution started”. Komentoon on mahdollista lisätä tietoa muuttujista, olivat ne sitten tyypiltään numeerisia tai merkkijonomuotoisia. (ABB 2010, 568.)

TPErase-komennolla pyyhitään flexpendantin näytölle jo kirjoitetut viestit. Tämä on kätevä tapa vähentää pendantille lähetettyjen viestien aiheuttamaa tietovyöryä. Komento ei ota vastaan erillisiä argumentteja. (ABB 2010, 556.)

6.4.7 StrLen

StrLen (String Length) -komennolla voidaan selvittää annetun merkkijonon pituus. Tämä tieto voidaan sijoittaa uuteen muuttujaan, joka on numeerista muotoa.

Esimerkkikoodi:

```
VAR num len;
len := StrLen("Robotics");
```

Esimerkissä len-muuttuja saa arvokseen 8, koska “Robotics” merkkijonossa on 8 merkkiä. Len-muuttujaa voidaan puolestaan tämän jälkeen käyttää esimerkiksi StrPart komennon yhteydessä. (ABB 2010, 996.)

Esimerkkikoodi:

```
part := StrPart("Robotics",2,len);
```

Part-muuttuja saisi esimerkkikoodissa arvokseen “obotics”, sillä haettavien merkkien väliksi tulisi merkistä 2 merkkiin 8.

6.4.8 StrFind

StrFind (String Find) -komennolla voidaan etsiä määriteltyä merkkiä alkaen määritellystä kohdasta merkkijonoa. Komento palauttaa numeerisena arvona, monesko merkki merkkijonossa vastaa etsittyä merkkiä. (ABB 2010, 994.)

Esimerkkikoodi:

```
VAR num found;
found := StrFind("Robotics",1,"aeiou");
```

Ensimmäisenä argumenttina StrFind ottaa merkkijonon, josta merkkiä etsitään. Toisena argumenttina annetaan kohta merkkijonossa, josta etsintä aloitetaan. Kolmantena annetaan merkit, joita etsitään. Esimerkissä found-muuttuja saa arvon 2, sillä toisena merkkinä on o-kirjain, ja tämä on kirjainten joukossa, jota etsitään. Antamalla komennolle eri määreitä kolmantena argumenttina, on myös mahdollista etsiä ensimmäistä numeromuotoista merkkiä (STR_DIGIT), välilyöntiä (STR_WHITE) tai merkkiä, joka ei ole määriteltyjen etsittävien joukossa ("aeiou"\NotInSet). (ABB 2010, 994.)

Käyttämällä StrFind-, StrLen- ja StrPart-komentoja yhdessä on mahdollista pilkkoa pitkiäkin merkkijonoja yksittäisiksi tiedon paloiksi, kunhan merkkijonossa käytetään jotain tiedossa olevaa erottelu merkkiä.

6.4.9 StrToVal

StrToVal (String To Val) –komento muuttaa string-tyyppisen muuttujan minkä tahansa muun tyyppiseksi muuttujaksi.

Esimerkkikoodi:

```
VAR bool ok;
VAR num nval;
ok := StrToVal("3.85",nval);
```

Muutos laitetaan boolean-tyyppisen muuttujan taakse. Näin saadaan helposti varmistettua, onnistuiko muutos, sillä muuttuja ok saa arvon TRUE vain, mikäli muutos onnistuu. StrToVal-komento ottaa ensimmäisenä argumenttina string-tyyppisen muuttujan tai suoraan merkkijonon. Toisena arguenttina tulee muuttuja, mihin muunnettu tieto tallennetaan. Esimerkissä nval-muuttuja saa siis arvokseen 3.85. (ABB 2010, 1010.)

6.4.10 NumToStr

NumToStr (Numeric To String) –komento muuttaa numeerisen arvon stringiksi.

Esimerkkikoodi:

```
VAR string str;
str := NumToStr(0.38521,3);
```

NumToStr ottaa ensimmäiseksi argumentiksi numeerisen arvon joko suoraan numeron tai muuttujaa kutsumalla. Toisella argumentilla määritellään, monenko desimaalin tarkkuudella numero esitetään. Tässä tapauksessa str-muuttuja saa arvokseen “0.385” eli numero muutetaan stringiksi kolmen desimaalin tarkkuudella. (ABB 2010, 904.)

6.4.11 Liikekäskyt MoveL, MoveJ ja offs

MoveL-käsky liikuttaa robottia lineaarisesti niin, että työkalupiste liikkuu suoraa viivaap pitkin määriteltyyn pisteeseen. Komento saa parametreinä paikkapisteen, mihin robotti ajetaan (p1), nopeuden (v1000), tarkkuuden (z30) sekä käytetyön työkalupisteen (tool2). (ABB 2010, 264.)

Esimerkki komennosta:

```
MoveL p1, v1000, z30, tool2;
```

MoveJ-komentoa käytetään, kun robotin ei tarvitse liikkua annettuun pisteeseen lineaarisesti, vaan robotti voi itse päättää niveltensä liikuttamisen ja siten sen, miten pyydettyyn pisteeseen liikutaan. Liikekäskyä käyttäessä robotin liike on siis epälineaarista. Komento ottaa vastaan samat parametrin, kuin MoveL komento. (ABB 2010, 253.)

Offs-komentoa käytetään ilmoittamaan robotille uusi piste suhteessa jo opetettuun pisteeseen. Sitä käytetään yhdessä MoveL- tai MoveJ-komentojen kanssa. Komento saa parametreinä tunnetun paikapisteen sekä muutoksen suhteessa tunnetun paikkapisteen X-, Y- tai Z-koordinaatteihin. (ABB 2010, 906.)

Esimerkki komennosta:

MoveL Offs(p2, 0, 0, 10), v1000, z50, tool1;

Komento ajaa robotin lineaarisesti 10 mm ylemmäksi Z-suunnassa, kuin mitä paikkapiste p2 on alun perin opetettu. Komennon hyöty on siinä, että uutta paikkapistettä ei välttämättä tarvitse erikseen opettaa. Tämä on hyödyllistä esimerkiksi, kun samankokoisia kappaleita pitää latoa vierekkäin. Jokaiselle kappaleelle ei tarvitse opettaa omaa paikkapistettä, vaan riittää, että on opetettu yksi paikkapiste ja robottia käsketään liikkumaan sen suhteen seuraavia kappaleita ladottaessa.

6.5 Ohjelman koodi

Ohjelma on toteutettu useiden aliohjelmien avulla. Se pohjautuu Cognexin esimerkkikoodiin, joskin siihen on tehty huomattavia lisäyksiä ja muutoksia. Rivittämätön esimerkkikoodi löytyy In-Sight Explorerin help-kannasta. Ohjelman koodi löytyy kokonaisuudessaan liitteestä 3. (Cognex Support Site www-sivut 2020.)

6.5.1 Main()

Jokaisessa RAPID-ohjelmassa on Main-ohjelma, jota robotti suorittaa. Tapa, jolla ohjelma on toteutettu tässä työssä johtaa siihen, että Main-kohdassa kutsutaan lähinnä vain muita aliohjelmia, jotka hoitavat kaiken robotin liikkeistä, kameraan yhdistämiseen ja tietojen lukemiseen kameralta. Osion rakenne vastaakin hyvin pitkälti luvussa 7.1 esiteltyä toimintalogiikkaa.

Esimmäisenä kutsutaan **ConnectToInsight**-ohjelmaa, joka yhdistää robotin kameraan Telnetin ylitse. Seuraavaksi kutsutaan **ChangeJobDice-**ohjelmaa, joka varmistaa, että kameraan on ladattu noppien ohjelma. Sitten kutsutaan ohjelmaa **DropDice2**, joka ohjaa robotin pudottamaan nopat kameran kuvausalueelle. Tämän jälkeen kutsutaan oh-

jelmaa **GetVisionDataDicev2**, joka ohjeistaa kameraa ottamaan kuvan ja robottia lukemaan tulokset ohjelmaan. Seuraavaksi kutsutaan ohjelmaa **PickDice**, joka ohjaa robotin poimimaan nopat ja palauttamaan ne alustoilleen.

Tämän jälkeen kutsutaan ohjelmaa **ChangeJobMarbles**, joka lataa kameraan marmorikuulien ohjelman. Tätä seuraa **MarbleDropSlide-**ohjelma, joka ohjeistaa robottia pudottamaan kuulat kameran näköalueelle. Kuulien sijaintitedot haetaan kutsumalla ohjelmaa **GetVisionDataMarbles**. Kuulat poimitaan takaisin alustalleen kutsumalla **MarblePickSlide-**ohjelmaa. Yksi ohjelman sykli on tämän jälkeen ajettu läpi. Ohjelman ajo voidaan aloittaa alusta.

6.5.2 ConnectToInsight()

ConnectToInsight-ohjelma alkaa luomalla socket-tyyppinen muuttuja (SocketCreate ComSocket), johon määritellään kameran IP-osoite sekä portti, johon yhteyttä koitetaan luoda (SocketConnect). Tietoja vastaanotetaan komennolla SocketReceive ComSocket. Tiedot tuleva merkkijonona, joten ne luetaan merkkijono (string) -tyyppiseen muuttujaan. Tämän jälkeen ohjelma tarkistaa, että vastaanotettu merkkijono vastaa käyttäjänimeä pyytävää kehotetta. Mikäli näin ei ole, niin ohjelma kirjoittaa robotin pendantille virheilmoituksen ja ohjelman suoritus pysäytetään.

Mikäli käyttäjänimeä pyytävä kehote on paikallaan, ohjelma lähettää käyttäjänimen ja salasanan. Tämä tapahtuu komennolla SocketSend ComSocket \Str:="admin"+CRLF;. Salasana lähtee samanlaisella komennolla, joskin lähetettävä merkkijono on tyhjä, joten lähetetään vain komento CRLF.

Lopuksi ohjelma tarkistaa, että kameralta saadaan “User Logged In” merkkijono. Näin varmistetaan, että sisäänkirjautuminen on onnistunut.

6.5.3 CheckStatus()

CheckStatus on pieni apuohjelma, jonka avulla tarkistetaan komentojen onnistunut suoritus. Se tarkistaa, että lähetetyn komennon jälkeen kameran vastaus on “1”+CRLF,

eli merkki komennon onnistumisesta. Mikäli kamera vastaa jotain muuta, ohjelma kirjoittaa virheilmoituksen robotin pendantille ja ohjelman suoritus pysäytetään. Tätä apuohjelmaa hyödynnetään lähes jokaisessa muussa ohjelmassa.

6.5.4 ChangeJobDice() & ChangeJobMarbles()

ChangeJobDice-ohjelma vaihtaa kamerassa olevan työn noppien työksi. Ensimmäisenä ohjelma tarkistaa, onko kamera Online- vai Offline-tilassa. Työn vaihto onnistuu vain Offline-tilassa, joten ohjelma asettaa kameran Offline-tilaan, mikäli se on tarpeen. Tämä käy komennolla SocketSend ComSocket \Str:="SO0"+CRLF;. Komennon onnistuminen tarkistetaan kutsumalla CheckStatus-ohjelmaa.

Kun kamera on Offline-tilassa, aktiivinen työ asetetaan komennolla SocketSend ComSocket \Str:="SJ1"+CRLF;. Komennon onnistuminen tarkistetaan jälleen CheckStatus-ohjelmaa kutsumalla. Onnistumisesta kirjoitetaan myös robotin pendantille viesti.

Kun työ on onnistuneesti vaihdettu, kamera asetetaan jälleen Online-tilaan. Tämä tapahtuu komennolla SocketSend ComSocket \Str:="SO1"+CRLF;. Onnistuminen tarkistetaan jälleen CheckStatus-ohjelmaa kutsumalla.

ChangeJobMarbles-ohjelma toimii muutoin samoin kuin ChangeJobDice, mutta se lataa kameran käyttöön kuulien ohjelman. Tämä tapahtuu komennolla SocketSend ComSocket \Str:="SJ2"+CRLF;. Muutoin ohjelman toiminta on täysin identtinen ChangeJobDice-ohjelman kanssa.

6.5.5 DropDice2()

DropDice2-ohjelma sisältää tarvittavat liikekäskyt noppien pudottamiseen alustalle sekä imun päälle ja pois laittamiseen. Opetettuja paikkapisteitä on lähestymispiste noppien yläpuolelle, nopan nostopiste sekä noppien alustalle pudottamista varten tehty

piste. Noppien noukkimiseen alustalta on vain yksi piste, jota siirretään offs-komennolla sen mukaan, monesko noppa on noukittavana. Näiden pisteiden avulla robottia ohjataan käyttäen joko liikerataa (MoveL) tai (MoveJ).

6.5.6 GetVisionDataDice()

GetVisionDataDice-ohjelma on vastuussa tietojen hakemisesta kameralta. Haetut X-, Y-, Angle-, silmäluku- ja nopan väri -tiedot tallennetaan jokainen omaan taulukko (array) -tyyppiseen muuttujaan (esim. arrX, arrY). Tiedonhaun toteuttamiseen koitettiin kahta eri tapaa. Ensimmäinen tapa on toteutukseltaan suoraviivaisempi, kun taas tapa kaksi vaatii hieman enemmän ohjelmoinnin ymmärtämistä. Näistä tapa kaksi todettiin nopeammaksi, joten sitä käytettiin lopullisessa ohjelmassa. Tässä kuitenkin käydään molemmat toteutustavat lävitse, sillä kumpikin on toimiva. Käytettävän tavan valinta riippuu omasta sovelluksesta ja sen tarpeista.

Tapa 1: Jokaisen tiedon haku yksitellen
Tavassa 1 varsinaiset tiedonhakukomennot sijoitetaan while-silmukaan, jota suoritetaan, kunnes boolean-tyyppinen muuttuja bPartFound saa arvon TRUE. Muuttuja saa arvon TRUE, kun löydettyjen osien lukumäärä vastaa jokaisella suorituskierroksella kasvavaa Index-muuttujan arvoa.

Seuraavaksi ohjelma tarkistaa, että yhteys kameraan on muodostettu. Tämän jälkeen se ohjeistaa kameraa ottamaan kuvan, lähettämällä native mode -komennon SW8. Tämän jälkeen ohjelma pyytää tiedon solun B018 arvosta. Soluun on tallennettu tieto, kuinka monta #ERROR-arvoista solua kameran ohjelmassa on löytyneiden kappaleiden alueella. Mikäli erroreita ei ole, ohjelma jatkaa suorittamista. Mikäli erroreita on, otetaan uusi kuva, kunnes saadaan sellainen, jossa ei ole error-arvoisia soluja.

Tämän jälkeen ohjelma pyytää tiedon solun O008 arvosta. Tähän soluun on tallennettu tieto, montako kappaletta kameran ohjelma on löytänyt. Tämä tieto tallennetaan muuttujaan PartsFound.

Tämän jälkeen alkaa FOR-silmukka, joka käy pyytämässä kameralta tiedot solu kerrallaan. Silmukkaa suoritetaan, kunnes muuttujan i arvo vastaa muuttujan PartFound arvoa.

Esimerkkikoodi yhden X-arvon hausta:

```
SocketSend ComSocket \Str:="gvd0" + CellsRangeStartstr + CRLF;

! Read the data
SocketReceive ComSocket \Str:=stReceived;
StringLength:=StrLen(stReceived)-5;
XData:= StrPart(stReceived, 4, StringLength);

...
bOK:=StrToVal(xData,nXData);
arrX{Index}:=nXData;
```

Ohjelma lähettää käskyn hakea tieto solusta D0 + CellsRangeStarstr-muuttujan arvo. Ohjelman suorituksen alussa, muuttujan arvo on 11, joten ensimmäinen tieto löytyy solusta D011. Jokaisen silmukan lopussa muuttujan arvoa kasvatetaan yhdellä. Näin seuraavalla suorituskerralla tieto haetaan yhtä solua alempaa (D012, D013 jne.). Kameran vastaus tallennetaan muuttujaan stReveived, josta se sijoitetaan muuttujaan XData niin, että viestin mukana tullut turha tieto (esim. viestin lopussa oleva rivinvaihto komento) jää pois.

FOR-silmukan lopussa muuttuja XData muutetaan numeeriseksi ja sijoitetaan muuttujaan nXData. Tämä numeerinen muuttuja taas sijoitetaan arrX-taulukkoon Index-muuttujan määrittelemään paikkaan. Index-muuttujan arvoa kasvatetaan jokaisella silmukan suorituskerralla. Näin kolmannen kappaleen X-koordinaatti tallentuu taulukon kohtaan 3, ensimmäisen kohtaan 1 jne.

Y-, Angle-, silmäluku- ja väritiedot haetaan samalla periaatteella. Ainoa muutos on solun sijainti, josta tieto löytyy, sekä muuttuja, johon saatu tieto sijoitetaan.

On huomioitava, että jokaiseen kameralle tehtyyn kyselyyn kuluu tietty määrä aikaa. Mikäli kyselyitä on suuri määrä voi syntyä tilanne, jossa robotti on pitkänkin aikaa tekemättä mitään. Se, onko tämä ongelma, riippuu ohjelman käyttötarkoituksesta. Esimerkkikoodi tämän tavan toteutuksesta löytyy liitteestä 3.

Tapa 2: Tietojen haku koostetusti

Tavassa kaksi jokaisen nopan tiedot koostetaan kameran ohjelmassa omaksi merkkijonoksi, jonka robotin ohjelma lukee. Tämän jälkeen robotin ohjelma purkaa merkkijonon yksittäisiin osiin ja tallentaa saadun tiedon niille varattuihin taulukoihin. Tällä tavalla verkon yli menevien kyselyjen määrä saadaan vähennettyä neljästäkymmenestä kahdeksaan. Tämä nopeuttaa huomattavasti ohjelman toimintaa.

Ohjelman rakenne on muutoin sama kuin tavassa yksi, mutta tiedonhaku tapahtuu vain yhdestä sarakkeesta. Koostetut tiedot löytyvät sarakkeesta K riviltä 31 alaspäin. Tiedot haetaan siis komennolla SocketSend ComSocket \Str:="gvk0" + CellsRangeStartstr + CRLF;.

CellsrangeStartstr-muuttujalle on annettu arvoksi 31. Muuttujan arvoa kasvatetaan jokaisella toistorakenteen suorituskerralla. Vastauksena saadaan merkkijono, joka voi olla esimerkiksi ”-14.5;-31.2;-177.2;4;w;”. Purettuna merkkijono tarkoittaa, että kappaleen sijainti on X-suunnassa -14.5, Y-suunnassa -31.2, sen kulma on -177.2, silmäluku on 4 ja väri on w eli valkoinen. Seuraavaksi robotin ohjelma pilkkoo saadun merkkijonon sen sisältämiin yksittäisiin tietoihin.

Prosessi alkaa selvittämällä, missä jokainen erottajamerkkinä toimiva puolipiste sijaitsee. Tähän käytetään StrFind-funktiota.

```
nXPos:=StrFind(stReceived, 3, ";");
nYPos:=StrFind(stReceived, nXpos+1, ";");
nAnglePos:=StrFind(stReceived, nYpos+1, ";");
nEyeCountPos:=StrFind(stReceived, nAnglePos+1, ";");
nColourPos:=StrFind(stReceived, nEyeCountPos+1, ";");
```

StrFind palauttaa numerona tiedon siitä, kuinka monentena etsitty merkki on merkkijonossa. Näin saadaan selvitettyä erotusmerkkien sijainnit. Seuraavan merkin etsintä aloitetaan aina yhden edellistä löytynyttä merkkiä edempää. Kun erotusmerkkien sijainnit on saatu selville, voidaan merkkijono pilkkoa sen yksittäisiin osiin.

```
xData:=StrPart(stReceived, 3, nXPos-2);
yData:=StrPart(stReceived, nXPos+1, nYPos-nXPos-1);
sAngle:=StrPart(stReceived, nYPos+1, nAnglePos-nYPos-1);
sEyeCount:=StrPart(stReceived, nAnglePos+1, nEyeCountPos-
nAnglePos-1);
sDiceColour:=StrPart(stReceived, nEyeCountPos+1, nColourPos-
nEyeCountPos-1);
```

Pilkkomiseen käytetään StrPart-funktiota. Merkkijono on tallennettu muuttujaan stReceived. Etsintäalue määräytyy erotinmerkkien sijainnin avulla. Esimerkiksi yData-muuttujaan tieto etsitään yksi merkki eteenpäin nXPos-erottimen sijainnista ja lopetetaan yksi merkki ennen nYPos-erottimen sijainnista, kun siitä on vähennetty nXPos-erottimen sijainti. Samalla logiikalla käydään koko merkkijono lävitse. Lopuksi ykisttäiset tiedot tallennetaan niille varattuihin talukoihin.

```
bOK:=StrToVal(xData,nXData);
arrX{Index}:=nXData;
bOK:=StrToVal(yData,nYData);
arrY{Index}:=nYData;
bOK:=StrToVal(sAngle,nAngle);
arrAngle{Index}:=nAngle;
bOK:=StrToVal(sEyeCount,nEyeCount);
arrDiceNumber{Index}:=nEyeCount;
arrDiceColour{Index}:=sDiceColour;
```

Tiedon sijainti taulukossa määräytyy Index-muuttujan arvon mukaan. Sen arvo ensimmäisellä toistorakenteen suorituksella on 1 ja sitä kasvatetaan jokaisella kierroksella.

Tavasta yksi poiketen verkkokyselyjä syntyy tällä tavalla vain niin monta, kuin noppia on alustalla. Robotin ohjelma on huomattavasti nopeampi pilkkomaan merkkijonoja verrattuna verkkokyselyn vaatimaan aikaan. Robotin hakiessa tietoja yksitellen ohjelmalla saattoi kestää jopa seitsemän tai kahdeksan sekuntia ennen, kuin kaikki tiedot oli haettu. Koostetulla tavalla saman tiedon keruuseen menee vain sekunti tai kaksi. Hyöty ohjelman toimintanopeuteen on siis huomattava ja tämän vuoksi tapa valittiin lopulliseen ohjelmaan.

6.5.7 PickDiceRoutine() ja PickDiceRouticeC()

PickDiceRoutine- ja PickDiceRouticeC-ohjelmien tarkoitus on vähentää koodin toistoa. PickDiceRoutine sisältää valkoisien noppien poimintaa varten vaadittavat liikekäskyt ja vastaavasti PickDiceRoutineC sisältää värillisiä noppia varten liikkeet. Ohjelmat ovat erikseen, koska nopat ovat erikokoisia, joten työkalua täytyy tuoda värillisiä noppia varten alemmaksi kuin valkoisia poimiessa. Ohjelmia kutsutaan PickDice-ohjelmassa, kun noppa pitää poimia kameran näkemältä alueelta.

6.5.8 PickDiceConditionsW() ja PickDiceConditionsC()

PickDiceConditionsW- ja PickDiceConditionsC-ohjelmat sisältävät noppien alustalle pudotukseen vaadittavat liikekäskyt, sekä pudotuspisteen siirrot. Pudotuspisteen sijaintia muutetaan nXOffs- ja nYOffs-muuttujilla riippuen siitä, monesko poimittava noppa on kyseessä. Koska ohjelma poimii aina valkoiset nopat ensin ja tämän jälkeen värilliset, sijainnit voidaan koodata pysyvämmin, kuin jos noppien poimintajärjestys muuttuisi ohjelmaa suorittaessa.

6.5.9 PickDice()

Ensimmäinen toimenpide PickDice-ohjelmassa on laskea kulma, jonka verran robotin tulee kääntää noppaa, jotta se on pudotusalustaan nähden suorassa kulmassa. Tämä tarvitsee tehdä, koska nopille on varattu neliönmuotoiset kolot alustalla. Mikäli noppaa ei käännetä oikein, se ei putoa sille varattuun koloon, vaan jää vinoon. Tällöin nopan poiminta uudella suorituskerralla epäonnistuu.

Kulman laskenta tapahtuu FOR-toistorakenteen sisällä olevilla WHILElauseilla. Laskennan tulos tallennetaan arrAngle-taulukkoon, kunkin nopan kohdalle.

PickDice-ohjelma käy löydetyt nopat lävitse ja poimii ne määrätyssä järjestyksessä. Ensimmäisenä poimitaan valkoiset nopat, suurimmasta silmäluvusta pienimpään. Tämän jälkeen värilliset nopat järjestyksessä punainen, lila, sininen, vihreä. Seuraavassa on esimerkkikoodi oikean nopan löytämiseksi:

```
FOR i FROM 1 TO PartsFound DO
! Only pick dice if Colour is white and eyecount is 6
IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=6 THEN
        PickDiceRoutine;
        Incr nDicePicked;
        PickDiceConditionsW;
    ENDIF
    Incr Index;
ENDFOR
```

FOR-silmukkaa suoritetaan, kunnes i:n arvo vastaa PartsFound-muuttujan arvoa, ts. kunnes kaikki löytyneet kappaleet on käyty läpi. FOR-silmukan sisällä oleva IF-silmukka toteutuu vain, jos nopan väri on valkoinen ja sen silmäluku on 6. Mikäli nämä ehdot täyttyvät, ohjeistetaan robotti poimimaan noppa saaduista koordinaateista ja tiputtamaan se alkuperäiselle alustalle. Tämä tapahtuu kutsumalla PickDiceRoutinesekä PickDiceConditionsW-ohjelmia. Lopuksi kasvatetaan Index-muuttujan arvoa, jotta seuraavalla kierroksella käydään läpi eri nopan tiedot. Näitä silmukoita on omansa jokaista nopan silmälukua varten.

Värillisten noppien läpikäynti toimii täysin samoin käyttäen värillisille nopille tarkoitettuja ohjelmia. Noppien poimintajärjestys määräytyy niiden värin mukaan, ensimmäisenä punainen, sitten lila, sininen ja vihreä.

6.5.10 MarbleDropSlide()

MarbleDropSlide-ohjelma ajaa robotin kuulia sisältävän mäen portille. Robotti tarttuu porttiin, nostaa sen ja odottaa 1,5 sekuntia, että kuulat ehtivät varmasti liukua mäestä kuvausalueelle. Tämän jälkeen robotti palauttaa portin paikalleen. Ohjelma koostuu pelkistä liikekäskyistä sekä imun päälle ja pois laittamisesta.

6.5.11 GetVisionDataMarbles()

GetVisionDataMarbles-ohjelma toimii täysin samalla periaatteella kuin GetVisionDataDice. Kuulista kuitenkin tarvitaan vähemmän tietoja kuin nopista. Pyöreillä kappaleilla ei ole väliä, missä kulmassa (angle) ne ovat eikä niillä ole silmälukuja. Ainoat tiedot, mitkä kiinnostavat, ovat X- ja Y-koordinaatit sekä kuulan väri. Kyselyjä on siis vähemmän, mutta ne toteutetaan täysin samalla tavalla kuin noppien kohdalla.

6.5.12 MarblePickSlide()

MarblePickSlide-ohjelma toimii täysin samalla periaatteella kuin PickDice-ohjelma. Ohjelma toimii samalla FOR-silmukalla kuin värillisten noppien poiminta. Ainoa muutos on käytetyt paikkapisteet. Pudotuspiste on kuulien mäen yläpäässä. Kuulat poimitaan värijärjestyksessä, ensimmäisenä valkoiset, tämän jälkeen ruskehtavat ja tämän jälkeen siniset.

Koska kuulia ei tarvitse sijoittaa määrättyyn paikkaan, voidaan ohjelman lopuksi tehdä vielä varmistus, että robotti on noukkinut kaikki kuulat. Tämä tarkistus on sijoitettu Main-ohjelmaan.

```
MarbleDropSlide;
GetVisionDataMarbles;
MarblePickSlide;
GetVisionDataMarbles;
   IF PartsFound = 0 THEN
      TPWrite "No Parts Found. Continuing program.";
```

```
ELSE
   WHILE PartsFound > 0 DO
   MarblePickSlide;
   GetVisionDataMarbles;
   ENDWHILE
ENDIF
```

Main-ohjelmassa kutsutaan MarblePickSlide-ohjelman jälkeen uudelleen GetVision-DataMarbles-ohjelmaa. Mikäli alustalta ei löydy kappaleita, kirjoittaa ohjelma tästä robotin pendantille viestin. Mikäli kappaleita löytyy, robotti tekee uuden yrityksen niiden poimimiseksi kutsumalla MarblePickSlide- ja GetVisionDataMarbles-ohjelmia. Näitä ohjelmia kutsutaan niin kauan, kunnes alusta on tyhjä.

7 ONGELMIA JA HAASTEITA

Työtä tehdessä kohdattiin erilaisia ongelmia ja haasteita. Osaan ratkaisu on esitetty jo aikaisemman tekstin seassa, mutta jotkin ongelmat olivat sen laatuisia, että niitä on hyvä tarkastella lähemmin.

7.1 COVID-19

Hallitus tiedotti 16.3.2020 poikkeustilasta. Syynä Suomeenkin levinnyt koronavirus, COVID-19. Tämän seurauksena Satakunnan Ammattikorkeakoulu sulki kampuksensa. Sulkemisen myötä menetettiin pääsy ABB IRB 120 -robotille. Tämä teki työn tekemisestä mahdotonta alkuperäisen suunnitelman mukaan.

15.4.2020 neuvottelu työtä ohjaavien henkilöiden kanssa johti siihen, että työn aihetta muutettiin niin, että toteutus tehtäisiin täysin RobotStudion avulla. Kameran työtä tekevä oli ottanut kotiin jo valmiiksi, joten tämä vaikutti hyvältä ratkaisulta. Etätyöskentely toi kuitenkin mukanaan omia haasteita. Kommunikointi sähköpostin avulla on luonnollisesti hitaampaa kuin kasvotusten puhuminen.

Lopulta kuitenkin tuli vastaan tilanne, jossa jouduttiin toteamaan, että työn tekeminen suunnitellusti Robotstudion avulla oli mahdotonta. Ohjelma ei yksinkertaisesti taipunut tarvittaviin asioihin. Esimerkiksi nopan luonti simulointiympäristöön kameralta saadun sijaintitiedon perusteella oli ongelma, johon ei löytynyt ratkaisua. Kameraa ei myöskään saatu keskustelemaan Robotstudion kanssa Telnetin ylitse.

Lokakuussa 2020 päätettiin, että palataan toteuttamaan työ alkuperäisen suunnitelman mukaan, eli oikealla robotilla. Koronavirustilanne oli helpottanut sen verran, että kampus oli jälleen aukeamassa ja robotille pääsy onnistui jälleen. Muutokset toteutustapaan johtivat työn viivästymiseen ja osaltaan siihen, että lopputuloksesta ei tullut aivan alkuperäisen suunntelman mukaista.

7.2 Robotin imukuppiongelmia

Ensimmäisellä yrityksellä robotin imukuppi ei tarttunut noppien ja kuulien alustoissa oleviin tartunta kohtiin. 3D-tulostetun kappaleen pinta oli sen verran epätasainen, että imukuppi ei saanut tiivistä kosketusta. Ratkaisu oli hioa tartuntapinnat sileäksi hienolla hiekkapaperilla.

Tämän jälkeen jouduttiin kuitenkin toteamaan, että robotin imukuppi oli liian iso. Se meni liikaa ohi kappaleista eikä näin ollen saanut kunnon imua aikaiseksi ja kappaleet jäivät poimimatta. Pienemmän imukupin kanssa ongelmaksi puolestaan muodostui noppien silmälukujen kuopat. Niistä syntyi rakoja, erityisesti silmäluvun ollessa kuusi, joiden vuoksi imukuppi ei tarttunut noppiin. Ongelma ratkaistiin teippaamalla nopat läpinäkyvällä teipillä. Tämä ei kuitenkaan ratkaissut pienemmän imukupin toista ongelmaa. Se ei jaksanut poimia kuulien alustaa ja isompien noppien kanssa imukuppi petti noppien valuessa alustalta pois. Kuulien osalta ongelma ratkaistiin suunnittelemalla niiden alustalle pudotus uudelleen. Käyttöön otettiin liukumäki, jonka päässä oli portti, jonka robotti nosti pois, jolloin kuulat valuivat kuvausalueelle.

Noppien osalta ongelma ratkaistiin poimimalla ne yksitellen alustalta ja pudottamalla ne kuvausalueelle. Tämän tavan ongelma on kuitenkin se, että nopat voivat pudota toistensa päälle tai jäädä alustan reunaa vasten outoon kulmaan. Jos alustalle kaadettaisiin useampia noppia samaan aikaan, niiden toisiinsa törmäily ehkäisisi tätä ongelmaa. Työn puitteissa ei kuitenkaan ollut aikaa hankkia parempaa imukuppia tai koittaa saisiko koko alustan noston toimimaan tehokkaammalla pumpulla tai injektorilla.

7.3 Valaistus

Lisävalosta huolimatta valaistus jäi edelleen ongelmaksi. Auringonvalon osuessa kohdalle kameran kuvista tulee käyttökelvottomia. Tätä pyrittiin kiertämään luomalla robottiin virheentarkistus. Robotti pysähtyy edelleen, mutta jatkaa toimintaansa, kun saa jälleen hyvän kuvan.

Ongelman ratkaisemiseksi riittäisi, jos kuvausalueen saisi varjoon. Haastetta kuitenkin tuo se, että ikkuna, josta valo tulee, on myös ikkuna, josta ohikulkijoiden pitäisi pystyä

katsomaan robotin työskentelyä. Ikkunaa ei voi siis kokonaan peittää. On mahdollista, että kuvausalueelle saisi sopivan varjon suhteellisen pienellä peitealueella, mutta sitä ei työn puitteissa ehditty tutkimaan.

8 YHTEENVETO

Työn tuloksena saatiin aikaiseksi kaksi älykameran ohjelmaa. Ensimmäinen tunnistaa kuvasta noppia ja koostaa löytyneistä kappaleista robottia varten tiedon sijainnista, väristä ja silmäluvusta. Toinen ohjelma tunnistaa kuvasta kuulia ja koostaa löytyneistä kappaleista robotin ohjelmaa varten tiedon kappaleiden sijainnista ja väristä.

Robottia varten syntyi ohjelma, joka ottaa kameraan yhteyden Telnet-protokollaa käyttäen. Ohjelma pudottaa nopat kameran kuvausalueelle, käskyttää kameraa ottamaan kuvan ja tämän jälkeen poimii nopat takaisin alustalleen kameralta saatujen tietojen perusteella. Samanlainen toiminnallisuus toteutettiin myös marmorikuulia varten.

Kokonaisuuden toimintavarmuutta ei saatu alkuperäisen suunnitelman tasolle. Kameran ohjelmien ongelmaksi muodostui muuttuva valaistus, kun taas robotin toimintaa vaikeutti noppien silmälukujen kuopat sekä imukupin ja pumpun yhdistelmän riittämätön teho. Näitä ongelmia pyrittiin parhaan mukaan kompensoimaan ohjelmia tehdessä, mutta työn puitteissa aika ei riittänyt kaikkia ongelmia korjaamaan.

Työn lopputulos on sinänsä toimiva kokonaisuus, joka ei kuitenkaan ole valmis ajettavaksi tuntikausia ilman ongelmia. Ohjelmat toimivat, mutta fyysinen ympäristö ja välineistö eivät ole suotuisia varmalle toiminnalle. Mikäli nämä kaksi asiaa saadaan kuntoon, niin kokonaisuuskin toimii.

LÄHTEET

3dprinting.com www-sivusto. Viitattu 14.12.2020. https://3dprinting.com.

ABB, ABB IRB 120 Data Sheet, 2019.

ABB, Käyttäjän Opas – IRC5 ja FlexPendant, 2018.

ABB, Technical reference manual, RAPID instructions, Functions and Data types, 2010.

Cognex, In-Sight 5100,5100C,5401,5400C,5403,5400 Specification Sheet, 2009.

Cognex, In-Sight 5000 Series Vision System Installation Manual Addendum, 2011.

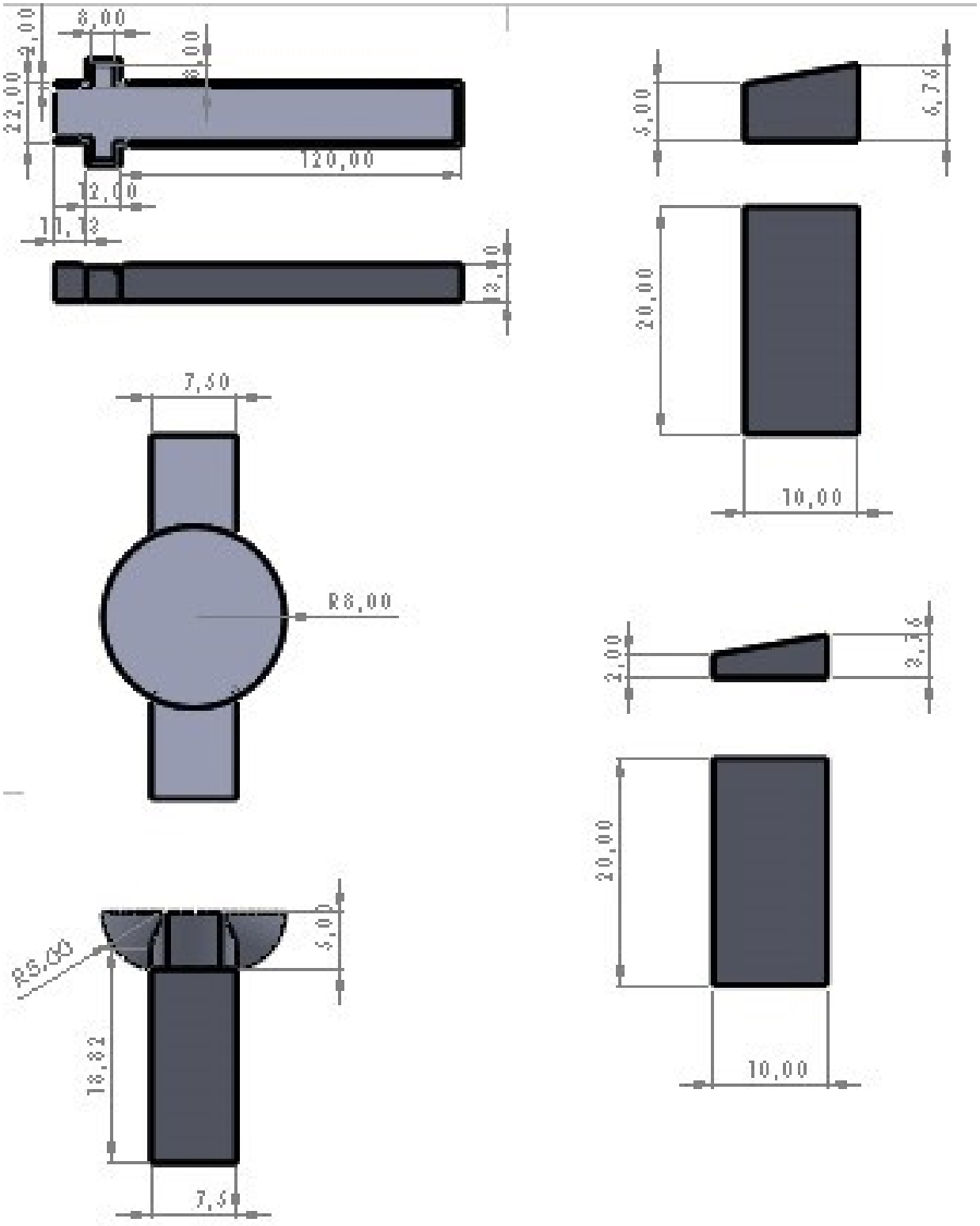
Cognex Support Site. Viitattu 10.12.2020. https://support.cognex.com

Cognex, In-Sight Explorer Help, Communications Reference, Native Mode Commands, 2020.

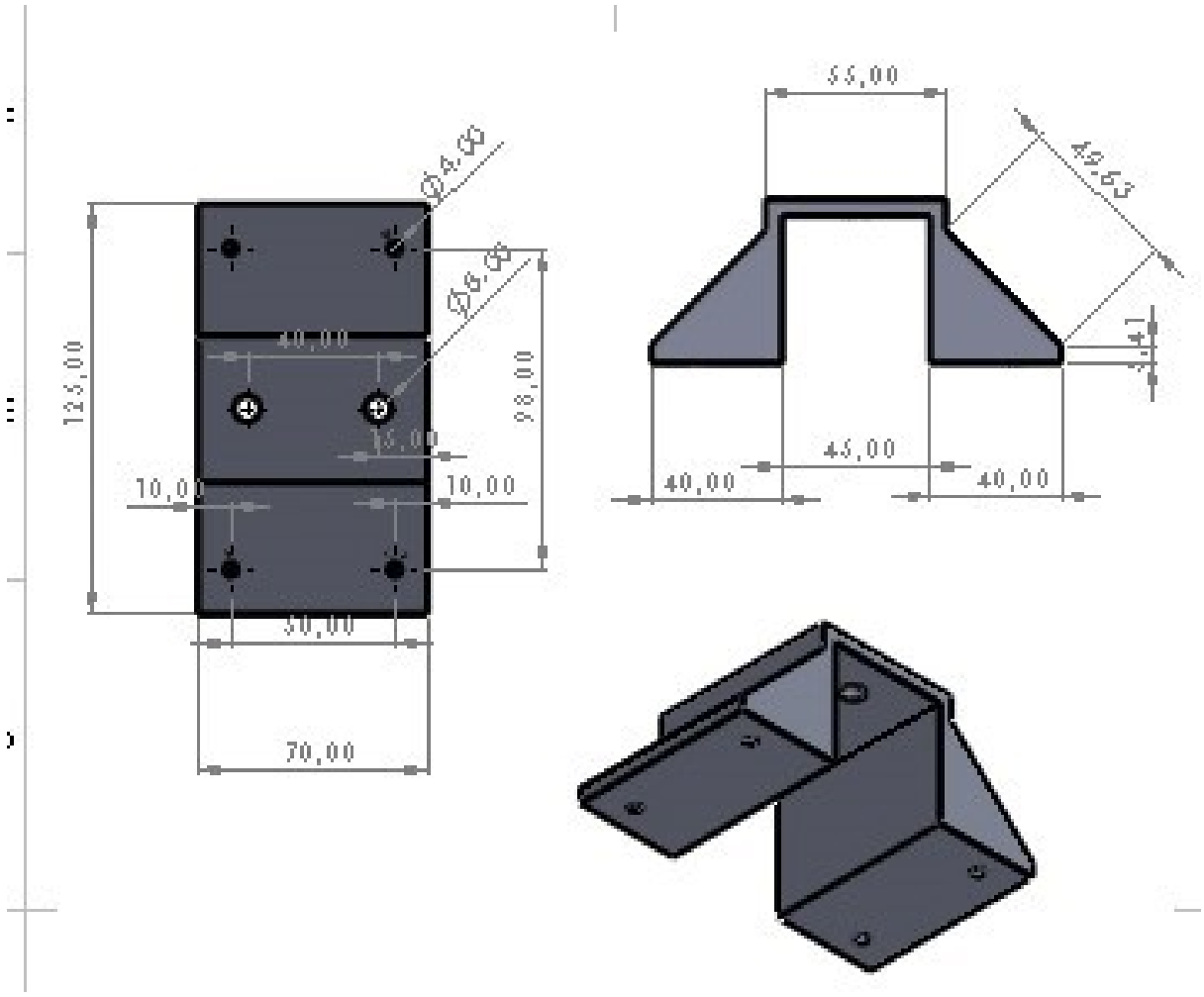
Leino, M. Luentomateriaali: Älykamera ja koodien lukeminen.
Satakunnan Ammattikorkeakoulun www-sivut. Viitattu 25.02.2020.
https://www.samk.fi/

Solidworks www-sivusto. Viitattu 14.12.2020. https://www.solidworks.com

LIITE 1


8,00
2,00
22,00
8,00
120,00
12,00
11,12
2,00
5,00
6,76
20,00
10,00
7,50
R8,00
2,00
2,36
20,00
10,00
5,00
R8,00
18,82
7,5

LIITE 2


Ø4,0
Ø6,0
125,00
40,00
98,00
10,00
10,00
10,00
50,00
70,00
55,00
49,63
3,41
40,00
40,00
40,00

```
MODULE MainModule
    ! DATA DECLARATIONS

    !PERS num nXOffs:=85;
    !PERS num nYOffs:=-34;
    !PERS num nAngle:=0;
    !PERS num nDiceEyeNumber:=0;



    PERS string sDiceColour;
    VAR string stReceived;
    VAR socketdev ComSocket;
    VAR socketstatus status;

    ! Targets and workobjects

    PERS wobjdata wobjWoodBase:=[FALSE,TRUE,"",[[-
222.979,357.438,138.06],[0.999864,-0.0120949,-0.0112245,-
0.000164848]],[[0,0,0],[1,0,0,0]]];
    CONST robtarget pDicePick:=[[60.12,141.73,6.47],[0.0119179,0.137182,-
0.990167,-0.0246699],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDicePickAp-
proach:=[[59.63,139.61,36.43],[0.0130061,0.136978,-0.99002,-
0.0304754],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDicePDropAp-
proach:=[[125.67,18.49,17.81],[0.00801804,0.143475,-0.989615,-0.00346656],[1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDicePDrop:=[[112.16,24.36,53.29],[0.0415405,0.134757,-
0.857536,-0.494719],[1,0,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !PERS wobjdata wobjVision:=[FALSE,TRUE,"",[[-
168.046,373.441,146.58],[0.000142805,-
0.0112404,0.0121054,0.999864]],[[0,0,0],[1,0,0,0]]]; // Vanha
    !PERS wobjdata wobjVision:=[FALSE,TRUE,"",[[-
167.500,377.500,143.021],[0.000134751,-
0.0112953,0.0120521,0.999864]],[[0,0,0],[1,0,0,0]]]; // Uusi
    PERS wobjdata wobjVision:=[FALSE,TRUE,"",[[-
169.500,379.000,143.021],[0,0,0,1]],[[0,0,0],[1,0,0,0]]];
    CONST robtarget pDicePPAp-
proach:=[[96.52,55.22,24.65],[0.0271153,0.377102,0.925358,0.0277742],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !CONST robtarget pVisPick:=[[0.82,-1.59,-1.50],[0.028142,0.376225,0.92589,-
0.0197344],[1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pVisPick:=[[0,0,-1.59],[8.8023E-06,-0.556017,-0.831171,-
5.35815E-06],[1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !CONST robtarget pVisPick:=[[0,0,-1.50],[0,0,1,0],[1,-1,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget pDicePPVisDrop:=[[97.16,141.84,14.25],[0.00259371,-
0.78855,0.614751,0.0162358],[1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDicePPVisDropAp-
proach:=[[97.33,141.66,21.85],[0.00259431,-0.788553,0.614747,0.0162365],[1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleDumpApproach:=[[161.69,156.22,26.96],[0.0181029,-
0.33694,-0.94125,-
0.0138445],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleDumpPick:=[[161.69,156.22,6.93],[0.0181015,-
0.336917,-0.941258,-
0.0138578],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleDumpDropAp-
proach:=[[111.30,63.99,30.47],[0.0272998,0.37712,0.925346,0.0277417],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleDumpDrop:=[[97.07,103.03,39.25],[0.14729,-
0.346427,-0.904085,-0.202298],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleVisDropApproach:=[[-104.46,-
110.57,26.18],[0.0279545,0.37616,0.925926,-0.019597],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleVisDrop:=[[-104.45,-
110.56,10.25],[0.0279442,0.37615,0.92593,-0.0195813],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMar-
bleVisPick:=[[45.95,17.85,9.01],[0.0280381,0.376196,0.925906,-0.0197155],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDicePickAp-
proach2:=[[59.63,174.34,26.29],[0.0130089,0.137036,-0.990015,-
0.0303527],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDicePick2:=[[59.63,174.35,11.18],[0.0130032,0.137021,-
0.990017,-0.0303817],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pDiceDrop2:=[[138.86,15.39,4.92],[0.340472,-0.192139,-
0.843533,-0.368258],[1,-1,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pGateLiftApproach:=[[143.53,119.41,39.95],[0.0189296,-
0.0785827,0.996518,0.0204739],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pGateLift:=[[143.53,119.40,23.63],[0.0189346,-
0.0785977,0.996517,0.0204626],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleSlideDrop:=[[150.10,211.17,40.28],[0.00608596,-
0.0786908,0.989908,0.117702],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pMarbleSlideDropAp-
proach:=[[150.09,160.40,60.25],[0.00608817,-0.0787102,0.989908,0.117686],[1,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget pAlignPoint:=[[58.44,21.34,4.17],[0.0202063,-
0.996899,0.0741581,0.0168911],[1,0,3,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+0
9]];
    !PERS tooldata tGripper:=[TRUE,[[0,0,159],[1,0,0,0]],[2,[65,0,0],[1,0,0,0],0,0,0]];
```

```
    !PERS tooldata tVision:=[TRUE,[[0,0,0],[0,0,0,1]],[5,[1, 0, 0],[1,0,0,0],0,0,0]];
    CONST robtarget pHome:=[[559.54,-55.01,192.22],[0.00605055,0.364584,-
0.931025,-0.0153432],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !CONST robtarget pVisionPos:=[[0,0,-
90],[0,1,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !PERS wobjdata woVisionCalib :=
[FALSE,TRUE,"",[[521.266,0.973149,207.66],[0.703255,-0.000254116,9.89283E-
05,-0.710937]],[[0,0,0],[1,0,0,0]]];
    CONST string CRLF:="\0D\0A";
    !PERS tooldata tPen := [TRUE,[[116.214,-
1.37941,159.021],[1,0,0,0]],[3,[5,0,0],[1,0,0,0],0,0,0]];
    CONST robtarget pVisDrop:=[[4.67,-10.41,-
0.70],[0.0673432,0.554232,0.828555,-0.0422865],[1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


    ! Arrays

    VAR num arrX{9};
    VAR num arrY{9};
    VAR num arrDiceNumber{9};
    VAR num arrAngle{9};
    VAR num arrMarbleX{9};
    VAR num arrMarbleY{9};
    VAR string arrMarbleColour{9};
    VAR string arrDiceColour{9};

    ! Numeric variables

    VAR num PartsFound;
    VAR num Index:=1;
    VAR num nDicePicked:=0;
    PERS num nXOffs;
    PERS num nYOffs;
    PERS num nAngle;
    PERS num nDiceEyeNumber;

    ! Booleans

    VAR bool bFailedRunCheck;


    PROC Main()

        ! Connect to the camera

        ConnectToInSight;

        !#####################
        ! Start Dice sequence. For position data, use GetVisionDataDicev2. It's faster.
```

```
        ! DiceDrop2 for individually dropping dice, DiceDrop for dropping dice with
slotted base
        !######################

        ChangeJobDice;
        DiceDrop2;
        GetVisionDataDicev2;
        PickDice;

        !######################
        ! Start Marbles sequence.
        ! MarbleDrop and PickMarbles for the non-slide version.
        !######################

        ChangeJobMarbles;

        MarbleDropSlide;
        GetVisionDataMarbles;
        MarblePickSlide;
        ! Check if any marbles remain on the platform and pick them up if there are
        bFailedRunCheck:= TRUE;
        GetVisionDataMarbles;
          IF PartsFound = 0 THEN
            TPWrite "No Parts Found. Continuing program.";
          ELSE
            WHILE PartsFound > 0 DO
            MarblePickSlide;
            GetVisionDataMarbles;
            ENDWHILE
          ENDIF

        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;
        SocketClose ComSocket;
        TPWrite "Program done";
        bFailedRunCheck:= FALSE;
        WaitTime 2;
    ENDPROC

    PROC MarbleDrop()
        ! Used with the old base to drop all the marbles at once
        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;
        MoveL pMarbleDumpApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveL Offs(pMarbleDumpPick,0,0,-8),v500,fine,tool0\WObj:=wobjWood-
Base;
        Set Imu;
        WaitTime 1;
        MoveL pMarbleDumpApproach,v500,z50,tool0\WObj:=wobjWoodBase;
        MoveL pMarbleDumpDropApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveJ pMarbleDumpDrop,v1000,fine,tool0\WObj:=wobjWoodBase;
        WaitTime 1;
```

```
        MoveJ pMarbleDumpDropApproach,v1000,z50,tool0\WObj:=wobjWoodBase;
        MoveL pMarbleDumpApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveL pMarbleDumpPick,v500,fine,tool0\WObj:=wobjWoodBase;
        Reset Imu;
        WaitTime 1;
        MoveL pMarbleDumpApproach,v1000,z50,tool0\WObj:=wobjWoodBase;
        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;
        WaitTime 1;
    ENDPROC

    PROC MarbleDropSlide()
        ! used with the new slide design to lift the gate and let the marbles fall
        MoveL pHome,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveL offs(pGateLiftApproach,0,0,+3),v1000,fine,tool0\WObj:=wobjWood-
Base;
        MoveJ offs(pGateLift,0,0,+3),v200,fine,tool0\WObj:=wobjWoodBase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pGateLiftApproach,0,0,+13),v200,fine,tool0\WObj:=wobjWood-
Base;
        WaitTime 1.5;
        MoveJ offs(pGateLift,0,0,+3),v200,fine,tool0\WObj:=wobjWoodBase;
        Reset Imu;
        WaitTime 1;
        MoveJ offs(pGateLiftApproach,0,0,+3),v200,fine,tool0\WObj:=wobjWood-
Base;
        MoveL pHome,v1000,fine,tool0\WObj:=wobjWoodBase;

    ENDPROC

    PROC MarblePickSlide()
        ! Used with the new slide deisgn to pick the marbles back up
        VAR string PartsFoundStr;
        VAR bool bOK;

        Index:=1;

        FOR i FROM 1 TO PartsFound DO
            !TPWrite "For loopissa";
            ! Only pick marble if Colour is white
            IF arrMarbleColour{Index}="white" THEN
                !TPWrite "IF loopissa";
                !WaitTime 2;

                MoveL offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+24),v500,fine,tool0\WObj:=wobjVision;

                MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-2,-
11),v500,fine,tool0\WObj:=wobjVision;
                Set imu;
```

```
            WaitTime 1;
            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+24),v500,fine,tool0\WObj:=wobjVision;

            MoveJ Offs(pMarbleSlideDropAp-
proach,0,0,+10),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ pMarbleSlideDrop,v200,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
          ENDIF
          Incr Index;
        ENDFOR

        Index:=1;

        FOR i FROM 1 TO PartsFound DO
          !TPWrite "For loopissa";
          ! Only pick marble if Colour is brown
          IF arrMarbleColour{Index}="brown" THEN
            !TPWrite "IF loopissa";

            MoveL offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+24),v500,fine,tool0\WObj:=wobjVision;

            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-2,-
11),v500,fine,tool0\WObj:=wobjVision;
            Set imu;
            WaitTime 1;
            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+24),v500,fine,tool0\WObj:=wobjVision;

            MoveJ Offs(pMarbleSlideDropAp-
proach,0,0,+10),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ pMarbleSlideDrop,v200,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
          ENDIF
          Incr Index;
        ENDFOR

        Index:=1;

        FOR i FROM 1 TO PartsFound DO
          !TPWrite "For loopissa";
          ! Only pick marble if Colour is blue
          IF arrMarbleColour{Index}="blue" THEN
            !TPWrite "IF loopissa";

            MoveL offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+24),v500,fine,tool0\WObj:=wobjVision;
```

```
            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-2,-
11),v500,fine,tool0\WObj:=wobjVision;
            Set imu;
            WaitTime 1;
            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+24),v500,fine,tool0\WObj:=wobjVision;

            MoveJ Offs(pMarbleSlideDropAp-
proach,0,0,+10),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ pMarbleSlideDrop,v200,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
          ENDIF
          Incr Index;
        ENDFOR

        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;

    ENDPROC

    PROC DiceDrop2()
        ! Used to drop dice one at a time into the cameras field of view
        nXOffs:=0;
        nYOffs:=0;

        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-1),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-30,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;
        nXOffs:=nXOffs-20;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-1),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-40,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;
```

```
        nXOffs:=nXOffs-40;
        nYOffs:=-1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-1),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-50,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;
        nXOffs:=nXOffs-20;
        nYOffs:=-1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-1),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-60,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;
        nXOffs:=0;
        nYOffs:=36;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-2),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+15),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-70,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;
        nXOffs:=nXOffs-20;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-2),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+15),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-30,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
```

```
        WaitTime 1;
        nYOffs:=nYOffs-1;
        nXOffs:=nXOffs-42;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-2),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+15),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-40,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;
        nYOffs:=nYOffs-1;
        nXOffs:=nXOffs-20;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveJ offs(pDicePPVisDrop,nXOffs,nYOffs,-2),v500,fine,tool0\WObj:=wob-
jWoodbase;
        Set Imu;
        WaitTime 1;
        MoveJ offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+15),v500,fine,tool0\WObj:=wobjWoodbase;
        MoveL offs(pVisDrop,-70,+25,+30),v500,fine,tool0\WObj:=wobjVision;
        Reset Imu;
        WaitTime 1;


        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;

    ENDPROc

    PROC DiceDrop()
        ! Used to drop the dice if picking up their entire base
        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;
        MoveL pDicePickApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveL Offs(pDicePick,0,0,-4),v500,fine,tool0\WObj:=wobjWoodBase;
        Set Imu;
        WaitTime 2;
        MoveL pDicePickApproach,v200,z50,tool0\WObj:=wobjWoodBase;
        MoveL pDicePDropApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveJ Offs(pDicePDrop,+5,0,-10),v500,fine,tool0\WObj:=wobjWoodBase;
        WaitTime 1;
        MoveJ pDicePDropApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveL pDicePickApproach,v500,z50,tool0\WObj:=wobjWoodBase;
        MoveL Offs(pDicePick,0,0,0),v1000,fine,tool0\WObj:=wobjWoodBase;
        Reset Imu;
        !WaitTime 1;
        MoveL pDicePickApproach,v500,z50,tool0\WObj:=wobjWoodBase;
```

```
        MoveJ pDicePickApproach2,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveJ Offs(pDicePick2,0,0,-4),v500,fine,tool0\WObj:=wobjWoodBase;
        Set Imu;
        WaitTime 2;
        MoveJ offs(pDicePickApproach2,0,0,+10),v1000,fine,tool0\WObj:=wob-
jWoodBase;
        MoveJ offs(pDicePDropApproach,0,0,+10),v1000,fine,tool0\WObj:=wob-
jWoodBase;
        MoveJ pDicePDropApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveJ Offs(pDicePDrop,+5,0,-10),v500,fine,tool0\WObj:=wobjWoodBase;
        WaitTime 1;
        MoveJ pDicePDropApproach,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveJ offs(pDicePickApproach2,0,0,+15),v1000,fine,tool0\WObj:=wob-
jWoodBase;
        MoveJ Offs(pDicePick2,0,0,-4),v500,fine,tool0\WObj:=wobjWoodBase;
        Reset Imu;
        WaitTime 1;
        MoveJ pDicePickApproach2,v1000,fine,tool0\WObj:=wobjWoodBase;
        MoveL pHome,v1000,z50,tool0\WObj:=wobjWoodBase;
        WaitTime 1;
    ENDPROC

    PROC PickDiceRoutine()
        ! Move robot to pick the dice and lift it up. This one is for the white dice.

        MoveJ offs(pVisPick,arrX{Index},arrY{Index}-
1,+40),v500,fine,tool0\WObj:=wobjVision;

        MoveJ Offs(pVisPick,arrX{Index},arrY{Index}-1,-
11),v500,fine,tool0\WObj:=wobjVision;
        Set imu;
        WaitTime 2;
        MoveJ Offs(pVisPick,arrX{Index},arrY{Index}-
1,+40),v500,fine,tool0\WObj:=wobjVision;

    ENDPROC

    PROC PickDiceRoutineC()
        ! Move robot to pick the dice and lift it up. This one is for the colored dice.


        MoveJ offs(pVisPick,arrX{Index},arrY{Index}-
1,+40),v500,fine,tool0\WObj:=wobjVision;

        MoveJ Offs(pVisPick,arrX{Index},arrY{Index}-1,-
12.5),v500,fine,tool0\WObj:=wobjVision;
        Set imu;
        WaitTime 2;
        MoveJ Offs(pVisPick,arrX{Index},arrY{Index}-
1,+40),v500,fine,tool0\WObj:=wobjVision;
```

```
    ENDPROC

    PROC PickDiceConditionsW()
        ! FOR WHITE DICE. Called after PickDiceroutine.
        ! Tells the robot where to drop off the picked dice based on how many dice have
been picked so far.
        IF nDicePicked=3 THEN
            nXOffs:=-60;
            nYOffs:=+1;
            TPWrite "Kulma3: "\Num:=arrAngle{Index};

            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,5),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
        ELSEIF nDicePicked=1 THEN
            nXOffs:=+1;
            nYOffs:=+1;
            TPWrite "Kulma1: "\Num:=arrAngle{Index};
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,5),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
        ELSEIF nDicePicked=2 THEN
            nXOffs:=-19.5;
            nYOffs:=+0.5;
            TPWrite "Kulma1: "\Num:=arrAngle{Index};
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,5),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
```

```
        ELSEIF nDicePicked=4 THEN
            nXoffs:=-80;
            nYOffs:=-1;
            TPWrite "Kulma4: "\Num:=arrAngle{Index};
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,5),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+18),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            nYOffs:=nYOffs+36;
            nXOffs:=-1;
        ENDIF

    ENDPROC

    PROC PickDiceConditionC()
        ! FOR COLORED DICE. Called after PickDiceroutineC.
        ! Tells the robot where to drop off the picked dice based on how many dice have
been picked so far.
        IF nDicePicked=7 THEN
            nXOffs:=-61.5;
            nYOffs:=+36;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,3),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
        ELSEIF nDicePicked=6 THEN
            nXOffs:=-19.5;
            nYOffs:=+36;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,3),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
```

```
        ELSEIF nDicePicked=5 THEN
            nXOffs:=-1.5;
            nYOffs:=+36;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,3),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
        ELSE
            nXOffs:=-80;
            nYOffs:=+36;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
            MoveJ RelTool(offs(pDicePPVisDrop,nXOffs,nYOffs,3),0,0,0\Rz:=ar-
rAngle{Index}),v500,fine,tool0\WObj:=wobjWoodBase;
            Reset Imu;
            WaitTime 1;
            MoveJ RelTool(offs(pDicePPVisDropAp-
proach,nXOffs,nYOffs,+24),0,0,0\Rz:=arrAngle{In-
dex}),v500,fine,tool0\WObj:=wobjWoodBase;
        ENDIF

    ENDPROC

    PROC PickDice()
        ! Pick the dice from the cameras field of view

        nXOffs:=0;
        nYOffs:=+3;
        nAngle:=0;
        Index:=1;

        ! Calculate the angle offsets so dice can be aligned with their drop area
        FOR i FROM 1 TO PartsFound DO

            WHILE arrAngle{Index}<0 DO

                arrAngle{Index}:=arrAngle{Index}+90;

            ENDWHILE

            WHILE arrAngle{Index}>90 DO

                arrAngle{Index}:=arrAngle{Index}-90;
```

```
    ENDWHILE
    Incr Index;
ENDFOR


Index:=1;

! Go through the white dice and pick them from largest to smallest eyecount
FOR i FROM 1 TO PartsFound DO
    TPWrite "White FOR LOOP";
    ! Only pick dice if Colour is white and eyecount is 6
    IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=6 THEN
        PickDiceRoutine;

        Incr nDicePicked;

        PickDiceConditionsW;


    ENDIF
    !Incr i;
    Incr Index;

ENDFOR
!i:=1;
Index:=1;

FOR i FROM 1 TO PartsFound DO

    TPWrite "White FOR LOOP";

    ! Only pick dice if Colour is white and eyecount is 5
    IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=5 THEN

        PickDiceRoutine;

        Incr nDicePicked;
        PickDiceConditionsW;


    ENDIF
    !Incr i;
    Incr Index;

ENDFOR
!i:=1;
Index:=1;

FOR i FROM 1 TO PartsFound DO
```

```
    TPWrite "White FOR LOOP";
    ! Only pick dice if Colour is white and eyecount is 4
    IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=4 THEN

        PickDiceRoutine;

        Incr nDicePicked;
        PickDiceConditionsW;

    ENDIF
    !Incr i;
    Incr Index;

ENDFOR
!i:=1;
Index:=1;

 FOR i FROM 1 TO PartsFound DO
    TPWrite "White FOR LOOP";
    ! Only pick dice if Colour is white and eyecount is 3
    IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=3 THEN
        PickDiceRoutine;

        Incr nDicePicked;

        PickDiceConditionsW;


    ENDIF
    !Incr i;
    Incr Index;

ENDFOR
!i:=1;
Index:=1;

FOR i FROM 1 TO PartsFound DO

    TPWrite "White FOR LOOP";
    ! Only pick dice if Colour is white and eyecount is 2
    IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=2 THEN

        PickDiceRoutine;

        Incr nDicePicked;
        PickDiceConditionsW;

    ENDIF
    !Incr i;
```

```
    Incr Index;

ENDFOR
!i:=1;
Index:=1;

FOR i FROM 1 TO PartsFound DO

    TPWrite "White FOR LOOP";

    ! Only pick dice if Colour is white and eyecount is 1
    IF arrDiceColour{Index}="w" AND arrDiceNumber{Index}=1 THEN

        PickDiceRoutine;

        Incr nDicePicked;
        PickDiceConditionsW;

    ENDIF
    Incr Index;
ENDFOR
Index:=1;

! Switch to picking coloured dice in order of red, purple, blue, green

FOR i FROM 1 TO PartsFound DO
    TPWrite "RED FOR LOOP";
    ! Only pick dice if Colour is red
    IF arrDiceColour{Index}="r" THEN

        PickDiceRoutineC;

        Incr nDicePicked;
        PickDiceConditionC;
    ENDIF
    !i:=i+1;
    Incr Index;

ENDFOR
!i:=1;
Index:=1;

FOR i FROM 1 TO PartsFound DO
    TPWrite "PURPLE FOR LOOP";
    ! Only pick dice if Colour is purple
    IF arrDiceColour{Index}="p" THEN

        PickDiceRoutineC;

        Incr nDicePicked;
```

```
            PickDiceConditionC;
        ENDIF
        !i:=i+1;
        Incr Index;

    ENDFOR
    !i:=1;
    Index:=1;

    FOR i FROM 1 TO PartsFound DO
        TPWrite "BLUE FOR LOOP";
        ! Only pick dice if Colour is blue
        IF arrDiceColour{Index}="b" THEN

            PickDiceRoutineC;

            Incr nDicePicked;
            PickDiceConditionC;
        ENDIF
        !i:=i+1;
        Incr Index;

    ENDFOR
    !i:=1;
    Index:=1;

    FOR i FROM 1 TO PartsFound DO
        TPWrite "GREEN FOR LOOP";
        ! Only pick dice if Colour is green
        IF arrDiceColour{Index}="g" THEN
            TPWrite "GREEN IF LOOP";

            PickDiceRoutineC;

            Incr nDicePicked;
            PickDiceConditionC;
        ENDIF
        !i:=i+1;
        Incr Index;

    ENDFOR
    !i:=1;
    Index:=1;

    MoveJ pHome,v1000,z50,tool0\WObj:=wobjWoodBase;

ENDPROC

PROC PickMarbles()
    ! Used to pick the marbles with the old stand
```

```
    VAR num MarblesPicked:=0;

    nXOffs:=0;
    nYOffs:=0;
    nAngle:=0;

    Index:=1;

    FOR i FROM 1 TO PartsFound DO

      ! Only pick marble if Colour is white
      IF arrMarbleColour{Index}="white" THEN


        MoveL offs(pVisPick,arrMarbleX{Index},arrMarbleY{In-
dex},+5),v500,fine,tool0\WObj:=wobjVision;

        MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index},-
15),v500,fine,tool0\WObj:=wobjVision;
        Set imu;
        WaitTime 1;
        MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+5),v500,fine,tool0\WObj:=wobjVision;
        IF MarblesPicked=0 THEN
          nXOffs:=0;
          nYOffs:=0;
          MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
          MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
          Reset Imu;
          WaitTime 1;
          MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
          Incr MarblesPicked;

        ELSEIF MarblesPicked=1 THEN
          nXOffs:=nXOffs-20;
          nYOffs:=nYOffs-7;
          MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
          MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
          Reset Imu;
          WaitTime 1;
          MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
          Incr MarblesPicked;
        ENDIF
      ENDIF
```

```
        Incr Index;
    ENDFOR

    Index:=1;

    FOR i FROM 1 TO PartsFound DO
        ! Only pick marble if color is brown
        IF arrMarbleColour{Index}="brown" THEN

            nXOffs:=0;
            nYOffs:=0;

            MoveL offs(pVisPick,arrMarbleX{Index},arrMarbleY{In-
dex},+5),v500,fine,tool0\WObj:=wobjVision;

            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index},-
15),v500,fine,tool0\WObj:=wobjVision;
            Set imu;
            WaitTime 1;
            MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{In-
dex},+5),v500,fine,tool0\WObj:=wobjVision;

            IF MarblesPicked=2 THEN

                nXOffs:=nXOffs+20;
                nYOffs:=nYOffs-5;
                MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
                MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
                Reset Imu;

                WaitTime 1;

                MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;

                Incr MarblesPicked;
                nXOffs:=0;
                nYOffs:=0;

            ELSEIF MarblesPicked=3 THEN

                nXOffs:=nXOffs+25;
                nYOffs:=nYOffs-27;
                MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
                MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
                Reset Imu;
```

```
            WaitTime 1;

            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;

            Incr MarblesPicked;
            nXOffs:=0;
            nYOffs:=0;
          ENDIF
        ENDIF
        Incr Index;
      ENDFOR

      Index:=1;

      FOR i FROM 1 TO PartsFound DO
        ! Only pick marble if colr is blue
        IF arrMarbleColour{Index}="blue" THEN

          MoveL offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
1,+5),v500,fine,tool0\WObj:=wobjVision;

          MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-1,-
15),v500,fine,tool0\WObj:=wobjVision;
          Set imu;
          WaitTime 1;
          MoveL Offs(pVisPick,arrMarbleX{Index},arrMarbleY{Index}-
2,+10),v500,fine,tool0\WObj:=wobjVision;

          IF MarblesPicked=4 THEN
            ! Fifth marble drop
            nXOffs:=nXOffs-25;
            nYOffs:=nYOffs-27;
            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,+5),v500,fine,tool0\WObj:=wobjVision;
            MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
            Reset Imu;
            WaitTime 2;
            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
            nXOffs:=0;
            nYOffs:=0;
            Incr MarblesPicked;

          ELSEIF MarblesPicked=5 THEN
            ! Sixth marble drop
            nXOffs:=nXOffs-25;
            nYOffs:=nYOffs-47;
```

```
            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,+5),v500,fine,tool0\WObj:=wobjVision;
            MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
            Reset Imu;
            WaitTime 1;
            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
            nXOffs:=0;
            nYOffs:=0;
            Incr MarblesPicked;

        ELSEIF MarblesPicked=6 THEN
            ! Seventh marble drop
            nXOffs:=nXOffs+25;
            nYOffs:=nYOffs-47;
            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,+5),v500,fine,tool0\WObj:=wobjVision;
            MoveJ offs(pMarbleVisDrop,nXOffs,nYOffs,-
5),v500,fine,tool0\WObj:=wobjVision;
            Reset Imu;
            WaitTime 1;
            MoveJ offs(pMarbleVisDropAp-
proach,nXOffs,nYOffs,0),v500,fine,tool0\WObj:=wobjVision;
            nXOffs:=0;
            nYOffs:=0;
            Incr MarblesPicked;
        ENDIF

      ENDIF
      Incr Index;
    ENDFOR

  ENDPROC


  PROC GetVisionDataDice()
    ! Slower method of getting the dice data. Don't use this. it's here just as an ex-
ample of how it's done.
    ! gets each bit of data about the dice, individually, one at a time
    VAR string XData:="";
    VAR string YData:="";
    VAR string AngleData:="";
    VAR string EyeNumber:="";
    VAR string DiceColour:="";
    VAR num nXData;
    VAR num nYData;
    VAR num nAngleData;
    VAR num nEyeNumber;
    VAR num CellsRangeStart:=11;
```

```
VAR num CellsRangeStart2:=30;
VAR string CellsRangeStartstr;
VAR num NumCharacters:=9;
VAR bool bOK;
VAR bool bPartFound;
VAR num StringLength;
VAR string PartsFoundStr;

Index:=1;

TPErase;
! Check if connection to camera is OK
TPWrite "Searching For Parts";
status:=SocketGetStatus(ComSocket);
IF status<>SOCKET_CONNECTED THEN
   TPErase;
   TPWrite "Vision Sensor Not Connected";
   Return ;
ENDIF

! Instruct In-Sight to Acquire an Image
! and not return until complete
SocketSend ComSocket\Str:="sw8"+CRLF;
CheckStatus;

! Check how many parts have been found
TPWrite "gvo008"+CRLF;
!WaitTime 5000;
SocketSend ComSocket\Str:="gvo008"+CRLF;
!CheckStatus;

! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
PartsFoundStr:=StrPart(stReceived,2,3);
bOK:=StrToVal(PartsFoundStr,PartsFound);
!WaitTime 5000;

bPartFound:=FALSE;
WHILE bPartFound=FALSE DO


   ! Get the values from all the parts from the individual cells
   FOR i FROM 1 TO PartsFound DO

      ! Get X value
      CellsRangeStartstr:=NumToStr(CellsRangeStart,0);
      TPWrite "gvd0"+CellsRangeStartstr+CRLF;
      !WaitTime 5000;
      SocketSend ComSocket\Str:="gvd0"+CellsRangeStartstr+CRLF;
```

```
!CheckStatus;

! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
TPWrite " ";
!IF StrPart(stReceived,1,1) <> "1"+CRLF THEN
!TPErase;
!TPWrite "Vision Error!";
!Stop;
!ENDIF
TPWrite "String Length="\Num:=StrLen(stReceived);
StringLength:=StrLen(stReceived)-5;
TPWrite stReceived;
XData:=StrPart(stReceived,4,StringLength);
TPWrite XData;
! Parse the data string
!XData:= StrPart(stReceived, 1, 8);

! Get the Y value

TPWrite "gve0"+CellsRangeStartstr+CRLF;
!WaitTime 5000;
SocketSend ComSocket\Str:="gve0"+CellsRangeStartstr+CRLF;
!CheckStatus;
! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
TPWrite " ";
!IF StrPart(stReceived,1,3) <> "1"+CRLF THEN
!TPErase;
!TPWrite "Vision Error!";
!Stop;
!ENDIF
TPWrite "String Length="\Num:=StrLen(stReceived);
StringLength:=StrLen(stReceived)-5;
TPWrite stReceived;
YData:=StrPart(stReceived,4,StringLength);

! GET ANGLE DATA
TPWrite "gvf0"+CellsRangeStartstr+CRLF;
!WaitTime 5000;
SocketSend ComSocket\Str:="gve0"+CellsRangeStartstr+CRLF;
!CheckStatus;
! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
TPWrite " ";
!IF StrPart(stReceived,1,3) <> "1"+CRLF THEN
!TPErase;
```

```
!TPWrite "Vision Error!";
!Stop;
!ENDIF
TPWrite "String Length="\Num:=StrLen(stReceived);
StringLength:=StrLen(stReceived)-5;
TPWrite stReceived;
AngleData:=StrPart(stReceived,4,StringLength);

! GET THE EYE NUMBER FOR THE DICE

CellsRangeStartstr:=NumToStr(CellsRangeStart2,0);
TPWrite "gvf0"+CellsRangeStartstr+CRLF;
!WaitTime 5000;
SocketSend ComSocket\Str:="gvf0"+CellsRangeStartstr+CRLF;
!CheckStatus;
! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
TPWrite " ";
!IF StrPart(stReceived,1,3) <> "1"+CRLF THEN
!TPErase;
!TPWrite "Vision Error!";
!Stop;
!ENDIF
TPWrite "String Length="\Num:=StrLen(stReceived);
StringLength:=StrLen(stReceived)-5;
TPWrite stReceived;
EyeNumber:=StrPart(stReceived,4,StringLength);
!DiceColour:= StrPart(stReceived, 24, 1);

! GET THE COLOUR OF THE DICE

TPWrite "gvg0"+CellsRangeStartstr+CRLF;
!WaitTime 5000;
SocketSend ComSocket\Str:="gvg0"+CellsRangeStartstr+CRLF;
!CheckStatus;
! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
TPWrite " ";
!IF StrPart(stReceived,1,3) <> "1"+CRLF THEN
!TPErase;
!TPWrite "Vision Error!";
!Stop;
!ENDIF
TPWrite "String Length="\Num:=StrLen(stReceived);
StringLength:=StrLen(stReceived)-5;
TPWrite stReceived;
DiceColour:=StrPart(stReceived,4,StringLength);
```

```
            TPWrite XData;
            !WaitTime 5000;

            ! Put read data into arrays
            bOK:=StrToVal(xData,nXData);
            arrX{Index}:=nXData;

            bOK:=StrToVal(yData,nYData);
            arrY{Index}:=nYData;

            bOK:=StrToVal(AngleData,nAngleData);
            arrAngle{Index}:=nAngleData;

            bOK:=StrToVal(EyeNumber,nEyeNumber);
            arrDiceNumber{Index}:=nEyeNumber;

            arrDiceColour{Index}:=DiceColour;

            ! Grow the value of Index, CellsRangeStart and CellsRangeStart2 for the
next loop
            Incr Index;
            Incr CellsRangeStart;
            Incr CellsRangeStart2;

        ENDFOR
        TPErase;

        ! Write all the found parts and value to the pendant
        TPWrite "1st dice: "+ValToStr(arrX{1})+", "+ValToStr(arrY{1})+",
"+ValToStr(arrAngle{1})+", "+ValToStr(arrDiceNumber{1})+", "+ValTo-
Str(arrDiceColour{1});
        TPWrite "2nd dice: "+ValToStr(arrX{2})+", "+ValToStr(arrY{2})+",
"+ValToStr(arrAngle{2})+", "+ValToStr(arrDiceNumber{2})+", "+ValTo-
Str(arrDiceColour{2});
        TPWrite "3rd dice: "+ValToStr(arrX{3})+", "+ValToStr(arrY{3})+",
"+ValToStr(arrAngle{3})+", "+ValToStr(arrDiceNumber{3})+", "+ValTo-
Str(arrDiceColour{3});
        TPWrite "4th dice: "+ValToStr(arrX{4})+", "+ValToStr(arrY{4})+",
"+ValToStr(arrAngle{4})+", "+ValToStr(arrDiceNumber{4})+", "+ValTo-
Str(arrDiceColour{4});
        TPWrite "5th dice: "+ValToStr(arrX{5})+", "+ValToStr(arrY{5})+",
"+ValToStr(arrAngle{5})+", "+ValToStr(arrDiceNumber{5})+", "+ValTo-
Str(arrDiceColour{5});
        TPWrite "6th dice: "+ValToStr(arrX{6})+", "+ValToStr(arrY{6})+",
"+ValToStr(arrAngle{6})+", "+ValToStr(arrDiceNumber{6})+", "+ValTo-
Str(arrDiceColour{6});
        TPWrite "7th dice: "+ValToStr(arrX{7})+", "+ValToStr(arrY{7})+",
"+ValToStr(arrAngle{7})+", "+ValToStr(arrDiceNumber{7})+", "+ValTo-
Str(arrDiceColour{7});
```

```
        TPWrite "8th dice: "+ValToStr(arrX{8})+", "+ValToStr(arrY{8})+",
"+ValToStr(arrAngle{8})+", "+ValToStr(arrDiceNumber{8})+", "+ValTo-
Str(arrDiceColour{8});

        ! Check if all the parts have been found and, if so, end the while loop by set-
ting bPartFound to TRUE
        IF PartsFound=Index-1 THEN
          TPWrite "PARTS FOUND!";
          bPartFound:=TRUE;
        ELSE
          bPartFound:=FALSE;
          UIMsgBox "PART NOT FOUND! Search again?";
        ENDIF

      ENDWHILE
    ENDPROC

    PROC GetVisionDataDicev2()
      ! Uses the ondenced string from the camera to grab more all the data for a single
dice at once
      ! use this one. It's faster.
      VAR string XData:="";
      VAR string YData:="";
      VAR string sAngle:="";
      VAR string sEyeCount:="";
      VAR string sDiceColour:="";
      VAR num nXData;
      VAR num nYData;
      VAR num nAngle;
      VAR num nEyeCount;
      VAR num CellsRangeStart:=31;
      VAR string CellsRangeStartstr;
      VAR bool bOK;
      VAR bool bPartFound;
      VAR num StringLength;
      VAR string PartsFoundStr;
      VAR string sErrorCount;
      VAR num nXPos;
      VAR num nYPos;
      VAR num nAnglePos;
      VAR num nEyeCountPos;
      VAR num nColourPos;
      VAR num nErrorCount;

      Index:=1;
      ! Check to see the robot is connected to the camera
      TPErase;
      TPWrite "Searching For Parts";
      status:=SocketGetStatus(ComSocket);
      IF status<>SOCKET_CONNECTED THEN
```

```
    TPErase;
    TPWrite "Vision Sensor Not Connected";
    Return ;
ENDIF

! Instruct In-Sight to Acquire an Image
! and not return until complete
SocketSend ComSocket\Str:="sw8"+CRLF;
CheckStatus;

! Check to see if there are any errors. If there are errors, try for a new picture
TPWrite "Checking for errors!";
SocketSend ComSocket\Str:="gvb018"+CRLF;
SocketReceive ComSocket\Str:=stReceived;
StringLength:=StrLen(stReceived);
sErrorCount:=StrPart(stReceived,2,StringLength-3);
bOK:=StrToVal(sErrorCount,nErrorCount);
TPWrite sErrorCount;
WAitTime 3;

WHILE nErrorCount>0 AND bFailedRunCheck = FALSE DO
    TPWrite "Errors found! Taking new picture every 30 seconds to recover!";
    WaitTime 30;
    SocketSend ComSocket\Str:="sw8"+CRLF;
    CheckStatus;
    SocketSend ComSocket\Str:="gvb018"+CRLF;
    SocketReceive ComSocket\Str:=stReceived;
    StringLength:=StrLen(stReceived);
    sErrorCount:=StrPart(stReceived,2,StringLength-3);
    bOK:=StrToVal(sErrorCount,nErrorCount);
ENDWHILE

! Check how many parts have been found
TPWrite "gvo008"+CRLF;

SocketSend ComSocket\Str:="gvo008"+CRLF;
!CheckStatus;

! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
PartsFoundStr:=StrPart(stReceived,2,3);
bOK:=StrToVal(PartsFoundStr,PartsFound);

bPartFound:=FALSE;
WHILE bPartFound=FALSE DO

    ! Get the values from all the cells
    FOR i FROM 1 TO PartsFound DO
```

```
            ! Get the condenced strings value
            CellsRangeStartstr:=NumToStr(CellsRangeStart,0);
            TPWrite "gvk0"+CellsRangeStartstr+CRLF;
            !WaitTime 5000;
            SocketSend ComSocket\Str:="gvk0"+CellsRangeStartstr+CRLF;


            ! Read the data
            SocketReceive ComSocket\Str:=stReceived;
            !CheckStatus;

            StringLength:=StrLen(stReceived);
            ! Parse the string so individual bits of information can be extracted
            nXPos:=StrFind(stReceived,3,";");
            nYPos:=StrFind(stReceived,nXpos+1,";");
            nAnglePos:=StrFind(stReceived,nYpos+1,";");
            nEyeCountPos:=StrFind(stReceived,nAnglePos+1,";");
            nColourPos:=StrFind(stReceived,nEyeCountPos+1,";");
            ! Deposite the individual bit of data into variables
            xData:=StrPart(stReceived,3,nXPos-3);
            yData:=StrPart(stReceived,nXPos+1,nYPos-nXPos-1);
            sAngle:=StrPart(stReceived,nYPos+1,nAnglePos-nYPos-1);
            sEyeCount:=StrPart(stReceived,nAnglePos+1,nEyeCountPos-nAnglePos-
1);
            sDiceColour:=StrPart(stReceived,nEyeCountPos+1,nColourPos-
nEyeCountPos-1);
            ! Convert numerical data from string to numerical and store it into the ar-
rays
            bOK:=StrToVal(xData,nXData);
            arrX{Index}:=nXData;

            bOK:=StrToVal(yData,nYData);
            arrY{Index}:=nYData;

            bOK:=StrToVal(sAngle,nAngle);
            arrAngle{Index}:=nAngle;

            bOK:=StrToVal(sEyeCount,nEyeCount);
            arrDiceNumber{Index}:=nEyeCount;

            arrDiceColour{Index}:=sDiceColour;

            Incr Index;
            Incr CellsRangeStart;
        ENDFOR
        ! If no parts are found, error
        IF PartsFound=Index-1 THEN
            TPWrite "PARTS FOUND!";
            bPartFound:=TRUE;
        ELSE
```

```
            bPartFound:=FALSE;
            UIMsgBox "PART NOT FOUND! Search again?";
          ENDIF
        ENDWHILE
        ! Print the data to the pendant
        TPWrite "1st dice: "+ValToStr(arrX{1})+", "+ValToStr(arrY{1})+", "+ValToStr(arrAngle{1})+", "+ValToStr(arrDiceNumber{1})+", "+ValToStr(arrDiceColour{1});
        TPWrite "2nd dice: "+ValToStr(arrX{2})+", "+ValToStr(arrY{2})+", "+ValToStr(arrAngle{2})+", "+ValToStr(arrDiceNumber{2})+", "+ValToStr(arrDiceColour{2});
        TPWrite "3rd dice: "+ValToStr(arrX{3})+", "+ValToStr(arrY{3})+", "+ValToStr(arrAngle{3})+", "+ValToStr(arrDiceNumber{3})+", "+ValToStr(arrDiceColour{3});
        TPWrite "4th dice: "+ValToStr(arrX{4})+", "+ValToStr(arrY{4})+", "+ValToStr(arrAngle{4})+", "+ValToStr(arrDiceNumber{4})+", "+ValToStr(arrDiceColour{4});
        TPWrite "5th dice: "+ValToStr(arrX{5})+", "+ValToStr(arrY{5})+", "+ValToStr(arrAngle{5})+", "+ValToStr(arrDiceNumber{5})+", "+ValToStr(arrDiceColour{5});
        TPWrite "6th dice: "+ValToStr(arrX{6})+", "+ValToStr(arrY{6})+", "+ValToStr(arrAngle{6})+", "+ValToStr(arrDiceNumber{6})+", "+ValToStr(arrDiceColour{6});
        TPWrite "7th dice: "+ValToStr(arrX{7})+", "+ValToStr(arrY{7})+", "+ValToStr(arrAngle{7})+", "+ValToStr(arrDiceNumber{7})+", "+ValToStr(arrDiceColour{7});
        TPWrite "8th dice: "+ValToStr(arrX{8})+", "+ValToStr(arrY{8})+", "+ValToStr(arrAngle{8})+", "+ValToStr(arrDiceNumber{8})+", "+ValToStr(arrDiceColour{8});
        !WaitTime 5;
    ENDPROC

    PROC GetVisionDataMarbles()
        ! Get marbles information from the camera
        VAR string XData:="";
        VAR string YData:="";
        VAR string sMarbleColour:="";
        VAR num nXData;
        VAR num nYData;
        VAR num CellsRangeStart:=25;
        VAR string CellsRangeStartstr;
        VAR string sErrorCount;
        VAR bool bOK;
        VAR bool bPartFound;
        VAR num StringLength;
        VAR string PartsFoundStr;
        VAR num nXPos;
        VAR num nYPos;
        VAR num nColourPos;
        VAR num nErrorCount;
```

```
nXOffs:=0;
nYOffs:=0;
nAngle:=0;
Index:=1;
! Check to see if robot is connected to the camera
TPErase;
TPWrite "Searching For Parts";
status:=SocketGetStatus(ComSocket);
IF status<>SOCKET_CONNECTED THEN
   TPErase;
   TPWrite "Vision Sensor Not Connected";
   Return ;
ENDIF

! Instruct In-Sight to Acquire an Image
! and not return until complete
SocketSend ComSocket\Str:="sw8"+CRLF;
CheckStatus;

! Check to see if there are any errors. If there are errors, try for a new picture
TPWrite "Checking for errors!";
SocketSend ComSocket\Str:="gvb018"+CRLF;
SocketReceive ComSocket\Str:=stReceived;
StringLength:=StrLen(stReceived);
sErrorCount:=StrPart(stReceived,2,StringLength-3);
bOK:=StrToVal(sErrorCount,nErrorCount);

WHILE nErrorCount>0 AND bFailedRunCheck = FALSE DO
   TPWrite "Errors found! Taking new picture every 30 seconds to recover!";
   WaitTime 30;
   SocketSend ComSocket\Str:="sw8"+CRLF;
   CheckStatus;
   SocketSend ComSocket\Str:="gvb018"+CRLF;
   SocketReceive ComSocket\Str:=stReceived;
   StringLength:=StrLen(stReceived);
   sErrorCount:=StrPart(stReceived,2,StringLength-3);
   bOK:=StrToVal(sErrorCount,nErrorCount);
ENDWHILE

! Check how many parts have been found
TPWrite "gvo008"+CRLF;
!WaitTime 5000;
SocketSend ComSocket\Str:="gvo008"+CRLF;
!CheckStatus;

! Read the data
SocketReceive ComSocket\Str:=stReceived;
TPWrite stReceived;
```

```
PartsFoundStr:=StrPart(stReceived,2,3);
bOK:=StrToVal(PartsFoundStr,PartsFound);
!WaitTime 5000;

IF PartsFound=0 THEN
   TPWrite "No parts found!";

ELSE

   bPartFound:=FALSE;
   WHILE bPartFound=FALSE DO

      ! Get the values from all the cells
      FOR i FROM 1 TO PartsFound DO

         ! Get the condenced strings value
         CellsRangeStartstr:=NumToStr(CellsRangeStart,0);
         TPWrite "gvk0"+CellsRangeStartstr+CRLF;
         !WaitTime 5000;
         SocketSend ComSocket\Str:="gvk0"+CellsRangeStartstr+CRLF;


         ! Read the data
         SocketReceive ComSocket\Str:=stReceived;

         StringLength:=StrLen(stReceived);
         ! Parse the string
         nXPos:=StrFind(stReceived,3,";");
         nYPos:=StrFind(stReceived,nXpos+1,";");
         nColourPos:=StrFind(stReceived,nYpos+1,";");
         ! Parsed data into variables
         xData:=StrPart(stReceived,3,nXPos-3);
         yData:=StrPart(stReceived,nXPos+1,nYPos-nXPos-1);
         sMarbleColour:=StrPart(stReceived,nYPos+1,nColourPos-nYPos-1);
         ! Variables into numeric values and into arrays
         bOK:=StrToVal(xData,nXData);
         arrMarbleX{Index}:=nXData;

         bOK:=StrToVal(yData,nYData);
         arrMarbleY{Index}:=nYData;

         arrMarbleColour{Index}:=sMarbleColour;

         TPWrite "X: "+ValToStr(arrMarbleX{Index})+" Y: "+ValTo-
Str(arrMarbleY{Index})+" Colour: "+arrMarbleColour{Index};
         !WaitTime 3;

         Incr Index;
         Incr CellsRangeStart;
      ENDFOR
```

```
        IF PartsFound=Index-1 THEN
          TPWrite "PARTS FOUND!";
          bPartFound:=TRUE;
        ELSE
          bPartFound:=FALSE;
          UIMsgBox "PART NOT FOUND! Search again?";
        ENDIF
      ENDWHILE
    ENDIF
ENDPROC

PROC CheckStatus()
    SocketReceive ComSocket\Str:=stReceived;
    IF stReceived<>"1"+CRLF THEN
      TPErase;
      TPWrite "Vision Error!";
      Stop;
    ENDIF
    RETURN ;
ENDPROC


PROC ConnectToInSight()
    ! Connect the robot to the camera using Telnet
    SocketCreate ComSocket;
    SocketConnect ComSocket,"192.168.125.203",23;
    SocketReceive ComSocket\Str:=stReceived;
    TPErase;
    TPWrite stReceived;

    IF StrPart(stReceived,StrLen(stReceived)-5,6)<>"User: " THEN
      TPErase;
      TPWrite "Vision Login Error (User Prompt)";
      Stop;
    ENDIF

    !Send the Username
    SocketSend ComSocket\Str:="admin"+CRLF;

    SocketReceive ComSocket\Str:=stReceived;
    TPWrite stReceived;

    IF stReceived<>"Password: " THEN
      TPErase;
      TPWrite "Vision Login Error (Password Prompt)";
      Stop;
    ENDIF

    ! Send Password
```

```
    SocketSend ComSocket\Str:=CRLF;

    SocketReceive ComSocket\Str:=stReceived;
    TPWrite "";
    TPWrite stReceived;

    IF stReceived<>"User Logged In"+CRLF THEN
        TPErase;
        TPWrite "Vision Login Error (Final Login)";
        Stop;
    ENDIF
ENDPROC

PROC ChangeJobDice()
    ! Change camera job to the dice job
    ! Check if camera is online or offline
    SocketSend ComSocket\Str:="GO"+CRLF;
    SocketReceive ComSocket\Str:=stReceived;
    stReceived:=StrPart(streceived,1,1);
    TPWrite "Camera status:"+stReceived;
    WaitTime 1;
    ! Set camera offline if it's online
    IF stReceived="1" THEN
        TPWrite "Camera online. Setting it offline";
        SocketSend ComSocket\Str:="SO0"+CRLF;
        CheckStatus;
        TPWrite "Camera offline";
        WaitTime 1;
    ENDIF

    ! Load program 1 which is the dice program
    TPWrite "Loading camera job 1";
    SocketSend ComSocket\Str:="SJ1"+CRLF;
    CheckStatus;
    TPWrite "Done!";
    WaitTime 1;


    ! Set camera back online
    TPWrite "Setting camera back online";
    SocketSend ComSocket\Str:="SO1"+CRLF;
    CheckStatus;
    TPWrite "Done";
    WaitTime 1;
ENDPROC

PROC ChangeJobMarbles()
    ! Change camera job to the marbles job
    ! Check if camera is online or offline
    SocketSend ComSocket\Str:="GO"+CRLF;
```

```
        SocketReceive ComSocket\Str:=stReceived;
        stReceived:=StrPart(streceived,1,1);
        TPWrite "Camera status:"+stReceived;
        WaitTime 1;
        ! Set camera offline if it's online
        IF stReceived="1" THEN
            TPWrite "Camera online. Setting it offline";
            SocketSend ComSocket\Str:="SO0"+CRLF;
            CheckStatus;
            TPWrite "Camera offline";
            WaitTime 1;
        ENDIF

        ! Load program 2 which is the marbles program
        TPWrite "Loading camera job 2";
        SocketSend ComSocket\Str:="SJ2"+CRLF;
        CheckStatus;
        TPWrite "Done!";
        WaitTime 1;


        ! Set camera back online
        TPWrite "Setting camera back online";
        SocketSend ComSocket\Str:="SO1"+CRLF;
        CheckStatus;
        TPWrite "Done";
        WaitTime 1;
    ENDPROC

ENDMODULE
```