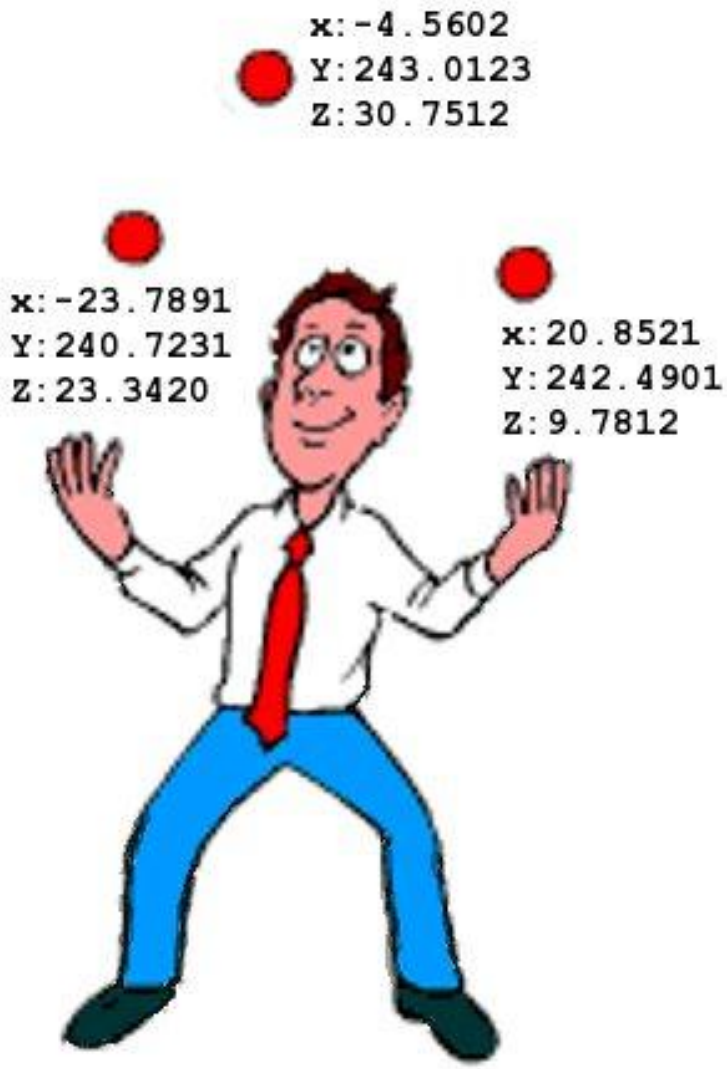
Projektarbete 100p
Rosendalsgymnasiet läsåret 2011/2012
Handledare: Thomas Andersson
Författare: Robert Pettersson, Jakob Larsson, Vidar Swenning

Bollkoll

Avståndsanalys med en webkamera



Sammanfattning

Tanken med detta arbete är att kombinera fysik, matematik och programmering på ett roande sätt. Denna rapport beskriver utförandet och problematiken bakom vårt bildanalysprojekt. Projektet handlar om att från en webkamera ta fram koordinater av ett objekt i det tredimensionella rummet. I rapporten kommer du få läsa om hur vi systematiskt har utarbetat algoritmer från grunden för analyser och beräkningar. Resultatet var över förväntan. En programvara som tar mätvärden flera gånger per sekund med några cm noggrannhet. Detta mynnar senare ut till en spelmotor för rörelsestyrda spel med liknade koncept som Nintendo Wii och Playstation Move.

Innehållförteckning

1. Inledning

1.1 Bakgrund

Sen de första digitala bilderna skapats har en ny värld av möjligheter öppnats upp. Automatisering genom analys av bilder har givit oss många tekniska framsteg. Trots detta är detta område relativt nytt och informationen kring det begränsat. Bildanalys går i stora drag ut på att lära en dator att förstå vad som händer utifrån en stor uppsättning pixlar.

Problemet ligger i att det för datorn bara är en uppsättning pixlar, ett virrvarr av färger som var för sig inte utgör några former. Datorer är starka men inte smarta så självklart uppstår det problem när man ska förklara vad som är intressant bland pixlarna. Lyckas man med att förklara bra för datorn blir resultatet ofta betydligt bättre än med mänskliga ögon. Datorns processor är svår för människan att övervinna.

Det som från första början endast var bildanalys har idag spridit sig och appliceras nu på alla möjliga tänkbara sätt i alla möjliga tänkbara områden. Allt från medicin och robotar till mineralindustrin och övervakning[1].

Det vanligaste tillvägagångssättet för blidanalys i tre dimensioner involverar två eller flera kameror, alternativt använder man sig av en kamera och en separat avståndssensor vilken kan använda sig av till exempel infraröttljus eller ultraljud. De finns ett antal kommersiella system som använder bild- och avståndsanalys i tre dimensioner. De mest spridda används till tv-spel: Playstation Move[2], Nintendo Wii[3] och Microsoft Kinect[4]. Alla de tre använder sig av sensorer och sändare i kombination med en kamera.

1.2 Syfte

Målet med detta projekt har varit att testa gränserna för vad som är möjligt att göra med en dator. Kan en dator läras att identifiera föremål? En utmaning vi ville testa var ny teknologi endast för enorma företag eller kan den även hanteras av tre gymnasieelever. Vi vill se hurvida det med vanlig billig utrustning över huvud taget går att låta datorn och dagens teknik lösa problem som inte ens den smarta och komplexa människohjärnan riktigt klarar av.

Ny teknologi kan användas på så många sätt och vi vill inte stanna vid ett löst problem utan även använda koordinaterna till en rolig produkt som kan intressera och öka förståelsen för ny teknik.

---

1 http://www.ne.se/lang/datorseende

2 http://www.examiner.com/video-game-in-knoxville/sony-reveals-how-the-playstation-move-works

3 http://www.youtube.com/watch?v=ETAKfSkec6A

4 seattlepi.com/e3-2009-microsoft-at-e3-several-metric-tons-of-press-release

1.3 Frågeställning

Är det möjligt att räkna ut ett objekts position i rummet med hjälp av bildanalys och hur noggrant är det i så fall? Är det möjligt att göra i realtid?

1.4 Avgränsningar

Det här projektet har avgränsats till att endast använda sig av en enda webkamera. En annan avgränsning vi gör är att vi begränsar vårt urval av objekt till enfärgade sfärer. Enfärgade sfärer ter sig lika till form och utseende i alla vinklar.

2. Material, teori och metod

Metoden som används är en deduktiv sådan där vi drar slutsatser samt utarbetar nya idéer och teorier med hjälp av våra tidigare kunskaper inom matematik, fysik och programmering som till stor del kommer från tidigare skolgång.

Vi arbetar enligt metoden: Alla teorier som kommer upp skrivs om till program där de testas varpå de ofta kontrolleras genom visuell tolkning av utdata. Om idén förbättrar resultatet sparas den. På detta sätt kan teorier och idéer snabbt styrkas eller avfärdas beroende på om de ger önskade resultat eller inte.

Till vissa av testerna har vi använt oss av testutrustning som måttband, gradskiva, skjutmått och lampor.

Sfärer av olika slag som testobjekt och datorer med deras inbyggda kameror används för att göra arbetet möjligt.

3. Undersökning/Resultat

För att lättare förstå denna text förklaras här några frekvent använda ord:

Pixel	En digital bild är uppbyggd av flertalet små punkter i olika färger. Tillsammans skapar dessa punkter en bild. Varje punkt kallas pixel.
Färgspann	Varje pixel har ett färgvärde som beskriver dess färg. Med färgspann menar vi avgränsande värden som begränsar urvalet av pixlar. Ett exempel är ett färgspann som endast godtar blåa pixlar.
Bakgrundsbrus	Skiftande färgförändringar i bilden som uppstår i kameran. Vanligt förekommande på till exempel vita vägar.
Fält	En del av bilden där de flesta pixlar har en liknande färg som tillsammans bildar en area.
Färgsystem	En färg kan beskrivas på flera olika sätt genom dess uppbyggnad av olika komponenter. Vilket sätt som används för att beskriva en färg kallas färgsystem.

3.1 Val av utvecklingsmiljö

I tidigt stadium av projektet togs beslutet att programmet skulle skrivas i programmeringsspråket Java. Det finns många språk som vi skulle kunna använda men vilket språk vi än använder så är det två saker vi måste kunna göra med det. För det första lyckas hämta en bild från webkameran och för det andra översätta vår algoritm till kod. Då det senare går att göra i alla språk så hängde det på det första men då vi inte visste något om webkameror i något programmeringsspråk valde vi Java. Detta valde vi av den enkla anledningen att det är språket vi läser i skolan vilket betyder att vi alla kan det.

För att kunna få in värden från webkameran behövde vi ett tillägg som kunde hjälpa oss med det, ett så kallat bibliotek. Vi började med att försöka med Oracles[5] egna bibliotek Java media framework. Detta fick vi inte att fungera som vi ville. Sökningen fortsatte efter andra gratisbibliotek på internet. Vi testade bibliotek efter bibliotek men inga fungerade. Till slut så hittade vi biblioteket Processing med tilläggsbiblioteket GSVideo. Dessa två fungerade som önskat. Vi nöjde oss då med detta utan att söka vidare efter andra lösningar.

3.2 Ursprungliga undersökningen av bilden

3.2.1 Bakgrundsarbete

Första steget blev att ta reda på vad det fanns för tidigare forskning och algoritmer i detta område men vi fann snabbt att detta hade varit för mycket att hoppas på. Materialet var begränsat, vi fann nästintill inget. Vi kontaktade att därpå Centrum för bildanalys på Uppsala universitet för rådgivning. De anordnade ett möte till oss med Ewert Bengtsson, professor på institutionen. Tillsammans med professor Bengtsson diskuterade vi våra tankar och idéer. Vi insåg vi att det inte fanns några generella vägar att gå när man jobbade med bildanalys. Detta medförde att den största delen av undersökningen har gått ut på att ta fram en bra algoritm. Även om professor Bengtsson gav oss några tips om vad vi borde tänka på så startade vi ändå från ett i stort sett blankt papper.

Vi funderade hur vi skulle angripa problemet. Vår första tanke var att jobba med färger och analysera hur dessa skiljer sig ifrån eller liknar varandra i olika delar av bilden.

3.2.2 Val av färger och färgsystem

Den absolut första algoritmen vi skapade gick ut på att sätta upp ett färgspann och se vilka pixlar som fanns i detta färgspann.[6] Då ett spann av färger kan skapas från olika färgsystem leder det

---

[5] Oracle är företaget bakom programmeringsspråket Java.

[6] Se bilaga 3.2

oss till frågan: vilket system av färger bör vi använda? Oavsett vilket färgsystem vi skulle använda så skulle vi ha samma färger att jobba med, fast graderingen av alla färger går i olika ordningar.

Ett färgsystem som är ganska vanlig är RGB där en färg består av en blandning av rött, grönt och blått. Detta system funkar i många fall helt utmärkt men vi förstod redan från början att det för oss skulle bli ett problem. Om någon tänder eller släcker en lampa kommer det leda till att alla tre komponenter kommer att ändras i hela bilden och därmed blir spannet väldigt opålitligt för just bildanalys. Vi bestämde oss för att göra våra första tester med ett annat färgsystem istället kallat HSB.[7] HSB är ett system där varje färg istället för att bestå av en röd, en grön och en blå del består av en nyans, en mättnad och en ljusstyrka. Fördelen med detta färgsystem är att ett föremål med en viss färg har ungefär samma nyans oavsett hur ljust det är.

Genom att vi lät datorn markera pixlar i en viss nyans fick vi en bild där bollen var markerad. Som vi tidigare hade misstänkt var även vissa delar i bakgrunden markerade. Vi undersökte olika objekt i olika färger kom fram till följande rörande bakgrundsfärger och objektsfärger:

Grön	Klassiska lysrör sänder iväg ett grönaktigt ljus. Stort bakgrundsljus på vita väggar.
Gul	Områden belysta av lampor ser gula ut.
Röd	Återfinns mycket i ansikten och hud.
Blå	Naturlig ljus är blått så i rum med fönster ser vita väggar blåa ut. Återfinns i många vanliga kläder.
Orange	Ansikten ter sig orangea samt områden belysta av lampor.
Rosa	Likande som röd.
Lila	Samma som för Blå fast inte i samma utsträckning

Som följd av dessa upptäckter valde vi att arbeta med lila objekt, de var långt från perfekta men de var det bästa alternativet enligt ovanstående undersökningar. Ofta valde vi att jobba avskilt från naturligt ljus då detta störde. Vidare var lila sfärer objekt vi lätt kunde få tag på.

3.2.3 Första hanteringen av bakgrundsbrus

Detta bakgrundsbrus[8] var något vi behövde handskas med innan vi kunde få några bra resultat. Vi tänkte att genom att gruppera pixlarna i vad vi kallade fält så skulle vi kunna avgöra vilket fält

[7] Även kallat HSV eller på svenska, NMI, nyans mättnad intensitet.
[8] Se 3.2.2

som var vårt objekt och vilka som endast var brus. Ett fält definierade vi som sammansatta pixlar som alla var inom det givna färgspannet. Ett väldigt simpelt sätt att avgöra om ett fält var brus eller inte var att vi testade hur stort fältet var mätt i antalet pixlar det innehöll, var det för litet var det brus.

Att detta sätt att åtskilja objektfält och brusfält hade sina brister upptäckte vi direkt när vi testade algoritmen. När objektet var långt ifrån kameran blev det förlitet för att räknas. Stora mängder av brus räknades fortfarande. Vi förbättrade detta genom att först simulera en rektangel som representerade hur fältet skulle sett ut om den var en rektangel. Detta för att brusfält ofta är mindre kompakta än fälten som innehåller objektet. Problemet var att vi då hittade rektangulärt brus vilket löstes genom att övergå till att simulera cirklar.

För att göra detta tog vi bredden på ett fält och antog att detta var diametern på en cirkel. Räknade sedan på en teoretisk area av denna cirkel. Om detta värde stämde ungefär med antalet pixlar i fältet godkändes fältet. Samma test gjordes utifrån fältets höjd. Om även detta stämde antog vi att fältet var ett av våra objekt.

Vi hade nu en hyfsat stabil algoritm som fungerade så länge som brus inte satt ihop med själva objektet. Fälten i dessa fall var in cirkulära, objekt tillsammans med brus, och klarade därför inte cirkeltestet. Således klassas dessa som brus vilket det delvis faktiskt var. I alla andra fall hade vi dock värden som vi för stunden var nöjda med.

3.3 Två dimensioner blir tre

3.3.1 Grundidén

Att räkna ut tredimensionella koordinater ur endast en tvådimensionell bild är rent generellt ganska svårt eller till och med helt omöjligt. För oss var det dock betydligt lättare då vi hade våra restriktioner om vårt objekt, som att det skulle vara en sfär. Vi behövde förutom det ha tillgång till följande information: objektets koordinater och diameter i bilden, dess diameter i verkligheten samt kamerans vidvinkel i både sidled och höjdled. Vi utgick från en publikation från Harvard[9] som beskriver hur man kan räkna ut avståndet till månen samt några andra generella formler för tredimensionella koordinater. För att mäta upp vidvinkeln på kameran använde vi oss av en gradskiva och ett snöre. Med snöret markerade vi bildens kant och tog mått vid linsen.

Vi antog att sfären skulle se lika stor ut i bilden vid ett visst avstånd från kameran. Storleken skulle alltså vara proportionell mot radien i ett sfäriskt koordinatsystem[10]. Med förhållandet mellan objektets riktiga storlek och dess storlek i bilden kunde den totala distansen till objektet räknas ut. Vi hade nu räknat ut radien i det sfäriska koordinatsystemet.

Fortsättningsvis räknade vi om objektets två-dimensionella position till de två vinklarna i det sfäriska koordinatsystemet. Detta gjordes genom att jämföra objektets relativa position till

9 http://www.cfa.harvard.edu/webscope/activities/pdfs/measureSize.PDF

10 Utvecklingen av ett polärt koordinatsystem. Beroende av två vinklar och radien.

mitten med kamerans vidvinkel. Ett objekt som var en fjärdedel till höger om mitten av bilden skulle till exempel ha en vinkel i det tredimensionella rummet med samma storlek som en fjärdedel av vidvinkeln.

Med alla komponenter som krävdes för att uttrycka objektets position med sfäriska koordinater kunde vi efter omräkning även få fram kartesiska koordinater[11].

3.3.2 Felaktigt antaganden angående optik

Till en början verkade det som vi fått riktigt bra värden, skillnaderna mellan de uträknade och de uppmätta värdena var så litet som några centimeter när det totala avståndet var runt en meter. Värdena verkade bra tills vi testade att ha objektet långt från kameran samt långt ut i kanten av bilden varpå koordinaterna inte alls stämde med samma noggrannhet. Efter utförliga tester av avstånden märkte vi att felvärdena var mycket större vid högre vinklar från origo. Vid detta tillfälle så började vi misstänka att något var fel med vårt antagande om konstant storlek på konstant avstånd från kameralinsen. För att testa denna tes satte vi upp ett experiment.

Vi fäste en fiskelina mellan bollen och kameran. Vi programmerade datorn så att den kontinuerligt skrev ut storleken, bredden och höjden på fältet den hittade. Enligt den första tesen skulle dessa mått vara konstanta om vi rörde oss på samma avstånd mellan kameran och bollen. Fiskelinan var spänd för att detta skulle uppnås. Resultaten bekräftade att denna tes inte alls stämde. Nu utnyttjade vi en av fördelarna med att undersöka med en dator. Vi reviderade algoritmen så att den istället antar att ett objekts storlek är konstant på ett konstant djup i bilden. Efter revideringen av algoritmen testade vi igen på samma sätt men räknade ut avståndet till bollen ,de fiskelinan representerade, i sista hand. Resultaten blev nu mycket mer tillfredsställande utan att vi förstod riktigt varför.

Som mycket annat kring vårt forskningsområde hittade vi inte någon information rörande detta utan i slutänden fick vi själva analysera och fundera kring bilder, kameror och linser.

För att komma underfund med hur det funkade så tänkte vi på hur en bild skulle se ut om det var som vi först trott. Vi tänkte även på användningsområdet för kameran. Om avståndet mellan kamera och objekt skulle visas i bredden i bilden skulle allt se böjt ut. En person som videochattar, viket är kamerans huvudsakliga användningsområde, och gungar fram och tillbaka i sidled över skärmen skulle få ett huvud som ökade och minskade i storlek. Sådant är inte fallet.

Ser vi en tvådimensionell bild tryckt på en sfär ser den böjd ut. I detta fall är det tvärt om vi ser en tredimensionell bild tryckt på ett plan. För att det inte ska se böjt ut så måste man i kamerans lins göra det kompensera för detta. Om detta stämde får objekt på samma djup har samma storlek. Det vill säga ett föremål som är på en meters framför kameran är alltid lika bred även om avståndet mellan linsen och föremålet är längre. Detta kände rimligt och skulle kunna förklara våra resultat.

---

11 Koordinater som visar avståndet till origo för de tre dimensionerna på formen (x,y,z).

3.3.3 Matematik

Med korrigeringen i algoritmen är nu matematiken som följande:

d_r = diameter på riktigt
d_b = diameter i bilden uttryckt i pixlar
k_w = kamerans vidvinkel i sidled i grader
k_h = kamerans vidvinkel i höjdled i grader
m_x = objektets mittpunkt i bilden
m_y = objektets mittpunkt i bilden
w = bildens bredd i pixlar
h = bildens höjd i pixlar
Övriga beteckningar är tillfälliga variabler för att lättare kunna följa beräkningarna.

$p_x = m_x / w$
$p_y = m_y / h$
$v_x = (0{,}5 - p_x) * k_w + 90$
$v_y = (0{,}5 - p_y) * k_h + 90$
$r_x = v_x * pi / 180$
$r_y = v_y * pi / 180$

$d_v = d_b / w * k_w$
$y = 180 * d_r / (pi * d_v)$
$a = y/(\sin(r_x)\sin(r_y))$
$x = a * \cos(r_x)\sin(r_y)$
$z = a * \cos(r_y)$

Objektets koordinat är punkten (x,y,z) där kameran är på punkten (0,0,0).

3.4 Förbättrad analys

3.4.1 Försök till förbättringar

Efter koordinaterna hade räknats ut kvarstod fortfarande problemet med fält som innehöll både objekt och brus. I ett försök att lösa detta försökte vi allt vi kunde komma på. Vi testade allt från att förbättra den gamla algoritmen till att med bättre hårdvara få bättre resultat. Även testade vi att skapa nya algoritmer som analyserade helt andra saker, kontraster, trendlinjer med mera. Dessa försök till algoritmer hade många problem olika miljöer och vissa gav helt och hållet ointressanta resultat.

En del algoritmer som till exempel den så kallade Hough transform[12] är väldigt beräkningsintensiva. Dessa testade vi inte ens då vi insåg hur mycket den behövde analysera och därmed hur långsam den skulle vara. Tiden för datorn att utföra algoritmen ansåg vi också som en viktig del.

För att förbättra hårdvaran testade vi att använda starka lampor för att eliminera bakgrundsbrus men för det första gjorde detta att vi begränsade miljöerna algoritmen funkade i och för det andra så funkade det inte så bra. På samma linje testade vi med att byta till en annan kamera men resultaten var i stort sätt oförändrade. Vi testade också att modifiera objektet genom att slipa ytan och på för att kunna använda ett tätare färgspann. Tidigare hade den blanka ytan skapat ljusa reflektioner som tvingade oss att använda ett större färgspann. Detta minskade problemet och vi valde att använda det tillsvidare.

För att utgå från den gamla algoritmen testade vi att definiera färgspannen på nya sätt, antingen med RGB eller med en blandning av RGB och HSB. Inget av dessa sätt hjälpte dock, vi kunde inte isolera objektets färger bättre för det. Vi hade även idéer om att dela upp fält i mindre fält men det gick inte alls då vi inte kom på något bra sätt att göra detta på.

För att tänka i helt nya banor försökte vi analysera skillnader mellan på varandra följande bildrutor, objektet har jämfört med bruset en relativt konstant position. Bruset ändrar sig helt mellan bildrutorna. Detta gav inget resultat över huvud taget då vi på pixelnivå ej kunde göra denna distinktion. Andra nya idéer inkluderade att analysera differenser i färg vilket dock medförde exakt samma problem som vår ursprungliga algoritm. Vår mest komplicerade algoritmidé gick ut på att knyta ihop kanter av fält i en lång kedja där varje pixel satt ihop med exakt två andra. Denna långa kedja skulle sedan analyseras med tanke på lutning för att hitta cirkulära trendlinjer i kanterna för att på så sätt ignorera brus. Denna algoritm var dock så komplicerad att vi inte lyckades få den att fungera, dessutom hade den en del oklarheter där vi inte visste exakt hur vi skulle göra.

Vi behövde en algoritm som klarar av fyra problem: den ska fungera i alla miljöer, den ska kunna särskilja objekt från brus även om de sitter ihop, den ska kunna genomföra analysera i realtid och den ska ge resultat med bra noggrannhet I detta läge verkade det inte som om allt det var möjligt.

3.4.2 Den slutgiltiga algoritmen[13]

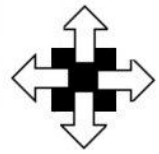
Efter att gjort många försök som till synes inte gjorde någon nytta gick vi tillbaka till vår kontakt från Uppsala Universitet, Ewert Bengtsson. Vi diskuterade problemen vi hade samt vad vi hade åstadkommit så här långt. Vi fick några tips om vad vi kunde göra men avfärdade vissa då vi redan undersökt dessa och upptäckt varför de inte funkade. Efter en stunds diskussion utformades dock idéen som vi sedan skulle komma att skapa vår nya algoritm från.

---

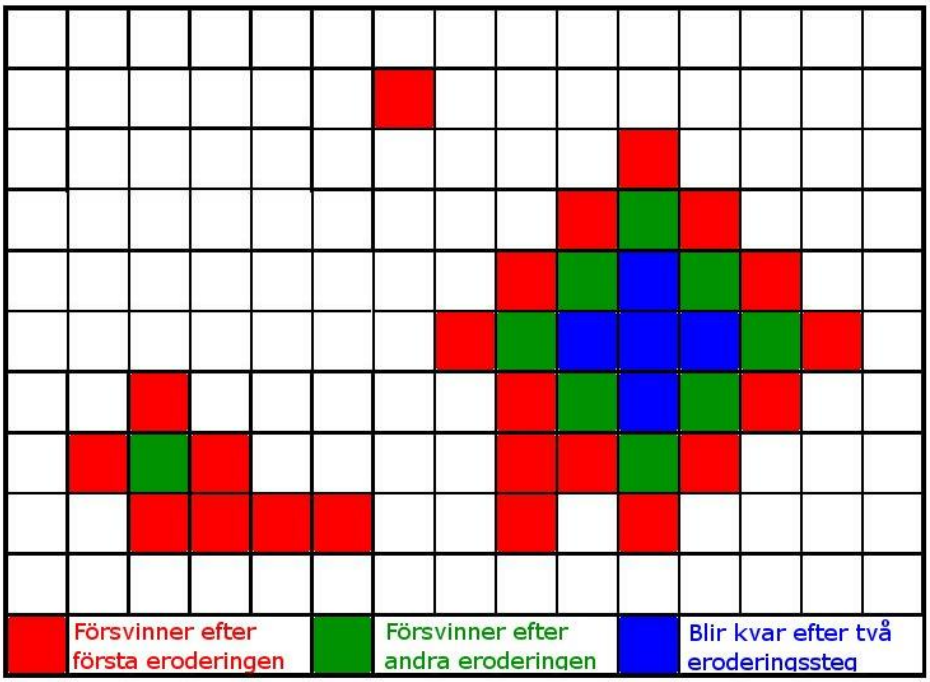
[12] http://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf

[13] Algoritmen är beskriven i detalj med ett flödesdiagram i bilaga 1 och bilder i bilaga 3

Den nya algoritmen var en blandning av våra gamla algoritmer men innehöll även nya delar. Först testas vilka pixlar vars färger ligger inom det valda HSB färgspannet, dessa pixlar kallas hädanefter för träffar. Alla pixlar som vid det här skedet var träffar kallas hädanefter även för orginalträffar. Sedan plockas alla träffar bort som inte har fyra av fyra träffar runt sig, bilden eroderas. Detta återupprepas ett bestämt antal gånger tills att allt av bruset försvinner. På samma gång försviner mycket av objektets kant. [14]



Varje godkänd pixel kontrolleras i fyra riktningar. Fyra godkända grannar krävs för att fortsätta vara godkänd.



När detta har genomförts är det tanken att det bara finns ett fält kvar per objektet. Mittpunkten av varje fält räknas sedan ut. Utifrån mittpunkten dras sedan radier utifrån mittpunkten, en radie tar slut när programmet stöter på en pixel som ej är en orginalträff. Typvärdet av dessa radier är objektets radie och objektets koordinater kan nu räknas ut på samma sätt som tidigare[15].

3.5 Tidsoptimering

Vi var vid det här laget väldigt nöjda med våra resultat men algoritmen var relativt långsam. Vi började undersöka hur beräkningsintensiv den var och för att göra detta lät vi programmet skriva ut tidskonsumtionen. På våra testdatorer fick vi i bästa fall ut fem resultat per sekund vilket inte var för dåligt men om det gick så skulle det vara fördelaktigt att snabba på processen.

Vi förbättrade koden lite här och var där vi såg att det gick att ta genvägar men i slutändan gjorde vi det mer systematiskt. Vi räknade ut hur lång tid de olika delarna av algoritmen tog. Detta gjorde att vi kunde fokusera på de delar som tog lång tid och försöka

14 Se även bilaga 3.3
15 Se sektion 3.3.3

snabba på just dessa delar. I och med dessa ändringar förbättrades det hela och resultat kunde oftast fås på en tiondels sekund.

3.6 Precisionstester

För att testa hur noggranna våra resultat var satte vi upp ett experiment. En fiskelina monterades så att den efterliknade Y-axeln, från kameralinsen till en vägg där den fästes. Sedan placerades en boll fastsatt på en ställning framför kameran. Med hjälp av måttband och vattenpass mättes sedan avståndet till kameran ut i alla tre dimensioner. Fiskelinan användes som referenspunkt. När alla måtten var tagna fick datorn räkna ut 100 stycken avstånd och ta ett medelvärde av dem. De uppmätta värdena kunde nu jämföras med datorns uträknade värden.[16]

4. Diskussion och Slutsatser

4.1 Precision

Hur exakt värdena på koordinaterna blir beror på många olika parametrar. Om det i varje steg skiljer sig något från de verkliga värdena så kommer den totala skillnaden ge stort utslag.

4.1.1 Upplösning och halva pixlar

För det första handlar det om själva bilderna som vi haft till vårt förfogande, de har en begränsad upplösning. De simpla kamerorna som vi har jobbat med har haft en upplösning på 640 gånger 480 pixlar. Det betyder att detaljer som är tillräckligt små inte kommer att synas i bilden. En pixel kan endast ha en färg vilket medför att en pixel innehåller ett visst objekt eller så gör den det inte.

För högre precision så skulle vi kunnat analysera till hur stor grad vissa pixlar har rätt färg, vi skulle kunnat få att ett objektets diameter är 35,6 pixlar om vi har 34 pixlar med full träff och två pixlar som endast stämmer till 80%. Detta är dock svårt att genomföra då problemet med bakgrundsbrus samt problem med färgvariationer i bilden ökar.[17]

Vi har funnit att räkna på delar av träffvärden troligen skulle kunna ge oss bättre resultat men har valt att inte gå vidare med detta. När vi ser nyttan mindre än mödan. När det gäller upplösningen så kan denna säkerligen ge jämnare resultat men redan i tidigt stadium var en av våra begränsningar tillgången av kamera därför har vi nöjt oss med det vi har, dessutom medför högre upplösning längre körtid.[18]

---

16 Se bilaga 2 för resultaten.
17 Se sektion 4.1.2
18 Se sektion 4.2

En annan del av upplösningen är också att felvärdena ökar med avståndet. Skillnaden mellan 1000 och 1001 pixlar är betydligt mindre än skillnaden mellan 100 och 101 pixlar. Detta ger att på större avstånd får varje pixel större innebörd för resultaten då föremålet ser mindre ut för kameran. I den här frågan ser vi ingen lösning även med en extremt högupplöst kamera får man detta problem om än i en mindre utsträckning

4.1.2 Bakgrundsbrus

När det gäller att få bra och exakta värden så gäller det att kunna få så klara träffar som möjligt skilda från bakgrundsbrus. Det finns två vägar att gå, antingen att minska bakgrundsbruset genom att minska godkända träffvärden. Vi utformade programmet så att det är lätt att justera färgspannet, vi har ett grafiskt gränssnitt där vi enkelt kan justera för olika omgivningar.[19] Det andra sättet är att med hjälp av en algoritm särskilja brus från intressanta träffar.

Vår första algoritm särskilde brus från träffar genom att kontrollera former och storlekar. Detta gav bra resultat när brusträffarna inte att ihop med de önskade. Satt de däremot ihop med varandra gav de inga resultat alls. Vilket ofta skedde.

Med den senare algoritmen klarar programmet att jobba med bilder med väldigt mycket bakgrundsbrus. Resultat fås oftare då de tillåts sitta ihop med bakgrundsbrus. Problemet med att den klarar av så mycket bakgrundsbrus blir dock att bakgrundsbruset i viss grad påverkar resultatet. Detta hände inte i den gamla algoritmen. När den gamla algoritmen gav resultat satt de nämligen inte ihop med bakgrundsbrus. I dessa fall påverkas även den nya algoritmen inte av bruset. De den nya algoritmen gör är alltså att den ger oss fler resultat även om de nya resultaten är till viss mån osäkrare.

Koordinaterna algoritmen ger påverkas av mittpunkten samt fältets storlek. Bruset kan påverka båda delarna. För det första påverkas bildens erodering av bakgrundsbrus, om det finns brus i kanten av ett objekt kan det få eroderingen att gå långsammare i kanten med brus. Om ett fält inte eroderas lika snabbt från alla håll leder det till en felställd mittpunkt då hela fältet blir lite förskjutet.

Brus kring kanten kan även ge en felaktig uträkning av radier från mittpunkten, men då det är typvärdet av radiernas längd som räknas påverkar detta inte resultatet i de flesta fall. Fast om brus påverkar tillräckligt stor del kan typvärdet slå fel och objektets diameter blir fel.

Om mittpunkt och/eller diameter blir fel påverkas det slutgiltiga resultatet då de matematiska formlerna får in fel värden och även om analysen av bilden är stabil så är det nog här som en del av avsaknaden i precision uppstår. Dessa problem hade vi inte i den första algoritmen men som tidigare skrivit så är de endast de fallen där den förra algoritmen misslyckas totalt med att ge resultat som den nya ger en aning osäkrare resultat. Frekvensen av resultat värdesattes mer än hög noggrannhet i detta fall.

---

19 Se bilaga 3

4.1.3 Autokorrigering

En annan typ av bakgrundsbrus eller snarare en uppkomst till detta är färgändringar som uppstår av autokorrigeringar i bilden. Många billiga kameror försöker korrigera bilden till vad den tror är bäst. Detta för att du ska kunna få bra tydliga bilder när du videochattar, vilket är kamerornas huvudsakliga syfte. När det kommer till vår bildanalys vill vi inte att kameran ska korrigera bilden efter vad den tror är bäst. Kameran ändrar ljusstyrkan och färgbalansen i bilden för att passa omgivningen. Detta leder för vår del till att samma objekt i samma ljus kan få olika värden beroende på området runt den. För billiga kameror vilket vi arbetade med är det ofta svårt att stänga av denna process. Färgerna ändras plötsligt vilket leder till felvärden då antingen mer brus skapas eller att ett objekt inte längre tillhör färgspannet.

Detta är ett problem som vi fått leva med. Istället har fokus lagts på att handskas med bakgrundsbrus på ett bättre sätt. Att ha en kamera som inte autokorrigerar färger skulle kunna ge oss möjligheten att ta smalare färgspann. Då bakgrundbrus påverkar precisionen[20] skulle en bättre kamera kunna ge bättre resultat. Dessa kameror är ofta dyra och var inget vi hade tillgång till. Därför valde vi att hantera autokorrigeringen med en starkare algoritm som hanterar ett större färgspann.

4.1.4 Matematikens noggrannhet

Formlerna bakom uträknandet av koordinaterna bygger på många parametrar. Vissa som fås ur bilden[21] genom vår algoritm medan vissa är konstanta[22]. Som tidigare nämnt kan värdena som fås ur bilden blir fel eller lite oprecisa. Detta leder i sin tur till att koordinater blir fel då de utgår från värden som inte stämmer. På samma sätt kan ett feluppmätt konstant värde leda till likande fel. Om objektets verkliga storlek skiljer sig från det datorn tror är objektets verkliga storlek eller om vidvinkeln på kameran inte är uppmätt helt korrekt så leder det givetvis till en inexakthet. Vi har efter bästa förmåga mätt upp dessa värden för att få ökad precision.

4.2 Optimering

Ett mål i projektet har hela tiden varit att kunna göra analysen i realtid. Detta har lett till att vi under arbetets gång förkastat många tidskrävande algoritmer. Tiden det tar för en dator är ofta direkt kopplat till antalet uträkningar den måste göra. Tiden det verkligen tar i sekunder skiljer sig dock från dator till dator. Detta handlar om datorns processorkraft. Så vårt mål har varit att kunna köra programmet på våra skoldatorer vilka har en ganska blygsam processorkraft. Detta genom att hålla nere antalet uträkningar.

---

20 Se sektion 4.1.2

21 Objektets position och storlek i bilden

22 Objektets diameter, kamerans vidvinkel och kamerans upplösning. Dessa är uppmätta sedan tidigare.

Vår algoritms beräkningsantal är starkt sammankopplat med bildens upplösning, en högupplöst bild har många pixlar således krävs många beräkningar medan en lågupplöst bild inte alls kräver lika många. Även om färre pixlar leder till färre beräkningar leder det dock till sämre precision[23], så det hela blir en balansgång.

Första steget i vår algoritm är i de flesta fall den mest tidskrävande, det går ut på att gå igenom varje pixels färg och jämföra det mot det givna färgspannet. Detta blir väldigt många beräkningar och som tidigare nämnt skulle det bli betydligt fler för fler pixlar. När första steget är klart är dock största delen av algoritmen gjord rent beräkningsmässigt, detta för att ett jämförande mellan en pixels färg och ett givet spann kräver många beräkningar för varje pixel medan senare steg endast är beräkningsintensiva för utvalda pixlar.

Steg två, eroderingen, är trots detta även den ganska beräkningsrik av den simpla anledningen att den körs väldigt många gånger. För varje gång den körs blir dock pixlarna som måste analyseras färre och således även beräkningarna färre vilket leder till att det är de första iterationerna av detta steg som faktiskt kräver många beräkningar. Tillsammans tar alla iterationer av steg två vanligtvis ungefär lika lång tid som steg ett, men det beror på bilden som analyseras.

Resten av algoritmen utgör endast en bråkdel av vad de två första stegen gör då pixlarna som behöver analyseras inte är så många och för varje pixel behövs inte så många beräkningar vilket i sin tur betyder att det totala antalet beräkningssteg inte blir så många. I de flesta fallen går inte tidskonsumtionen för de resterande delarna av algoritmen att räkna ut på de datorer vi har testat med[24].

Då det är de två första stegen av algoritmen som tar tid så skulle en optimering av de andra vara helt meningslös då intjänad tid där skulle vara en droppe i havet. Istället kan man titta på de första stegen vilket var vad vi gjorde. Genom att snabba upp det sätt vi kollade om en pixels färg tillhörde det givna spannet samt att snabba upp eroderingen av pixlar lyckades vi fyrdubbla hastigheten för hela algoritmen. Efter att de två första stegen optimerades så gott vi kunde så fanns det inte så mycket mer vi kunde göra för att drastiskt skära ned på tidsåtgången.

4.3 Resultat

När vi gick in i projektet hade vi ingen aning om vilken typ av precision vi kunde förväntas få. Eller huruvida vi skulle få några resultat överhuvudtaget. Med den bakgrunden måste vi trots allt säga att våra resultat är riktigt bra. Vi har under bra förutsättningar en noggrannhet på några få centimeter.[25]

Huruvida resultaten är rättvisande eller ej är dock lite svårare att säga. Det är svårt att exakt mäta upp ett objekts position i rummet. Speciellt svårt har de varit att kunna mäta i rätta vinklar rakt ut i luften. Vi har på många sätt försökt förbättra våra mätningar man kan dock inte

23 Se sektion 4.1.1

24 Datorerna vi testade med kunde mäta tider med 10 ms noggrannhet

25 Se bilaga 2.

räkna med större noggrannhet än 1-2 centimeter. Våra uppmäta resultat har därför troligtvis större felvärden vid större avstånd. I vårt precisionstest var det på stora avstånd som vi fick stora differenser Detta är troligtvis en kombination av både osäkra uppmäta värden och att varje pixel spelar större roll på längre avstånd. [26]

I de mätningarna vi genomförde såg vi en svag tendens hos värdena i X-led. De visade ofta ett värde närmre bildens mitt än det uppmätta. Vi har inte gått in djupare på det men vi misstänker att det kan bero på att vi ha mätt upp kamerans horisontella vidvinkel fel. Vilket skulle kunna resultera i sådana resultat. Vi fann det mycket svårt att hitta bra mätmetoder för att mäta vidvinklar med exakthet. Vi gjorde så gott vi kunde.

4.4 Källor

De källor som vi använt under vår undersökning har vi ganska bra anledning att lita på. Då de få källor vi använt handlar om hur man gör diverse uträkningar kan vi personligen styrka dem på två sätt. För det första så märker vi helt enkelt att uträkningarna ger bra resultat och för det andra kan vi bekräfta dem matematiskt eller logiskt med de kunskaper vi kunnat sedan tidigare från skolan eller annat håll.

4.5 Tillämpningar

När vi hade fått fram bra resultat började vi fundera på hur vi kan presentera våra resultat på till exempel en mässa. Vi funderade på om man kan tillämpa analysen på något roligt sätt. Vi har lagd till nya delar i programmet så att man kan använda den som en spelmotor. Ett utomstående program kan enkelt få ut både kartesiska och sfäriska koordinater. Efter det är det bara fantasin som stoppar en från vad för program man kan använda det till.

Vi byggde till att börja med ett simpelt datorspel där man använder objektet för att “skjuta ner” måltavlor som flyger runt på skärmen. Även ett ritprogram där man styr penseln med bollen skapades.

4.6 Slutsats

Vi har med det här projektet undersökt möjligheterna med bildanalys i tre dimensioner. Vi har visat att det är möjligt att med hjälp av enbart en dator och en vanlig webkamera bestämma ett objekts position i rummet både relativt snabbt och exakt. Vi har konstruerat en algoritm som effektivt särskiljer en sfär från de flesta bakgrunder under varierande ljusförhållanden.

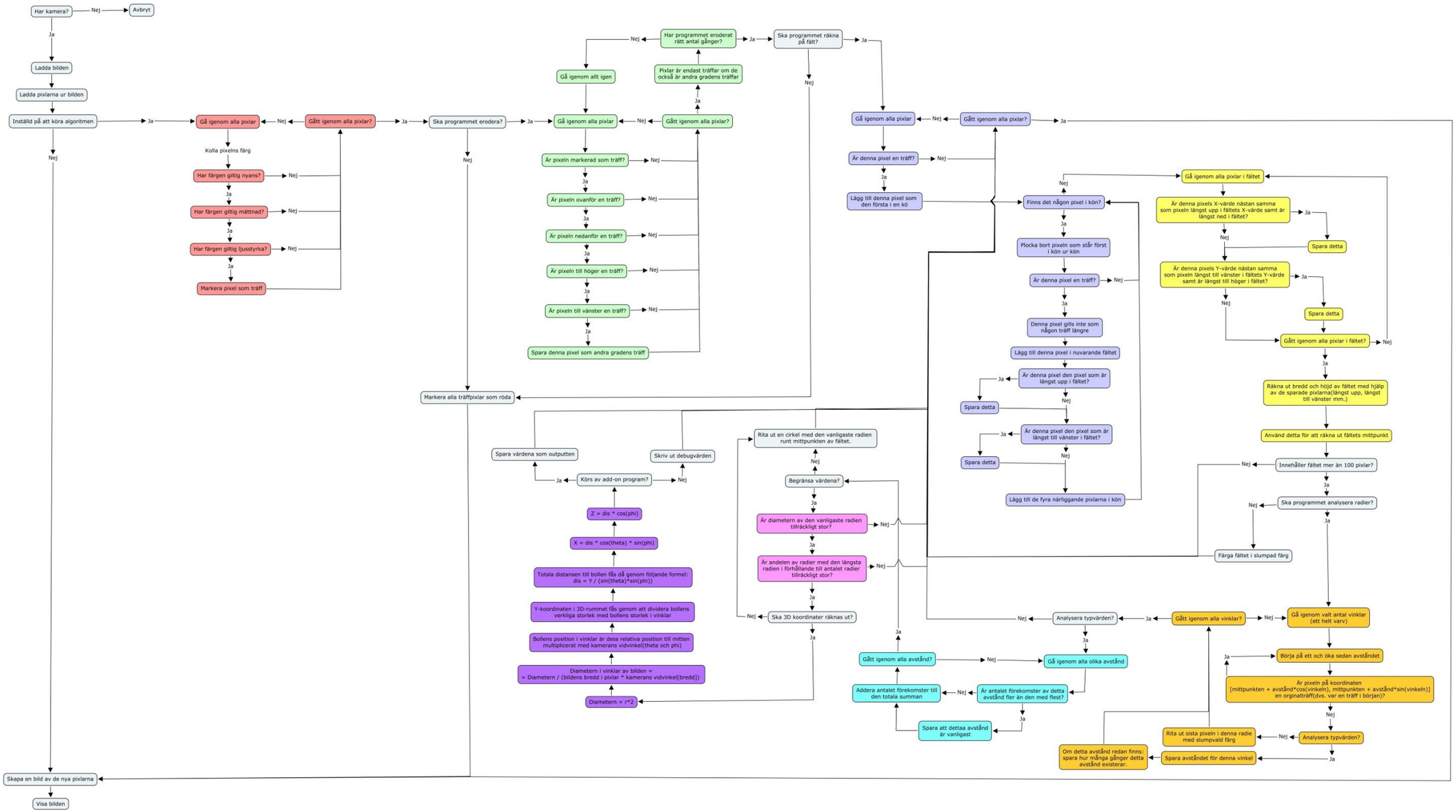
---

26 Se 4.1.1

Källförteckning

http://www.ne.se/lang/datorseende 19/2 2012
http://www.examiner.com/video-game-in-knoxville/sony-reveals-how-the-playstation-move-works 20/2 2012
http://www.youtube.com/watch?v=ETAKfSkec6A 20/2 2012
seattlepi.com/e3-2009-microsoft-at-e3-several-metric-tons-of-press-release 20/2 2012
http://www.cfa.harvard.edu/webscope/activities/pdfs/measureSize.PDF 22/11 2011
http://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf 17/1 2012

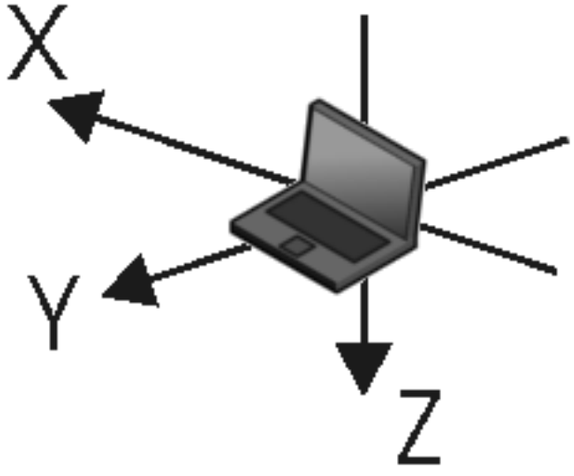
Bilaga 1


Har kamera?
Nej
Avbryt
Ja
Ladda bilden
Ladda pixlarna ur bilden
Inställd på att köra algoritmen
Ja
Nej
Gå igenom alla pixlar
Nej
Gått igenom alla pixlar?
Ja
Kolla pixelns färg
Har färgen giltig nyans?
Nej
Ja
Har färgen giltig mättnad?
Nej
Ja
Har färgen giltig ljusstyrka?
Nej
Ja
Markera pixel som träff
Ska programmet erodera?
Ja
Nej
Gå igenom alla pixlar
Nej
Gått igenom alla pixlar?
Ja
Är pixeln markerad som träff?
Nej
Ja
Är pixeln ovanför en träff?
Nej
Ja
Är pixeln nedanför en träff?
Nej
Ja
Är pixeln till höger en träff?
Nej
Ja
Är pixeln till vänster en träff?
Nej
Ja
Spara denna pixel som andra gradens träff
Pixlar är endast träffar om de också är andra gradens träffar
Ja
Har programmet eroderat rätt antal gånger?
Ja
Nej
Gå igenom allt igen
Ska programmet räkna på fält?
Ja
Nej
Gå igenom alla pixlar
Nej
Gått igenom alla pixlar?
Ja
Är denna pixel en träff?
Nej
Ja
Lägg till denna pixel som den första i en kö
Finns det någon pixel i kön?
Ja
Nej
Plocka bort pixeln som står först i kön ur kön
Är denna pixel en träff?
Nej
Ja
Denna pixel gills inte som någon träff längre
Lägg till denna pixel i nuvarande fältet
Är denna pixel den pixel som är längst upp i fältet?
Ja
Nej
Spara detta
Är denna pixel den pixel som är längst till vänster i fältet?
Ja
Nej
Spara detta
Lägg till de fyra närliggande pixlarna i kön
Gå igenom alla pixlar i fältet
Är denna pixels X-värde nästan samma som pixeln längst upp i fältets X-värde samt är längst ned i fältet?
Ja
Nej
Spara detta
Är denna pixels Y-värde nästan samma som pixeln längst till vänster i fältets Y-värde samt är längst till höger i fältet?
Ja
Nej
Spara detta
Gått igenom alla pixlar i fältet?
Nej
Ja
Räkna ut bredd och höjd av fältet med hjälp av de sparade pixlarna(längst upp, längst till vänster mm.)
Använd detta för att räkna ut fältets mittpunkt
Innehåller fältet mer än 100 pixlar?
Nej
Ja
Ska programmet analysera radier?
Nej
Ja
Färga fältet i slumpad färg
Markera alla träffpixlar som röda
Rita ut en cirkel med den vanligaste radien runt mittpunkten av fältet.
Nej
Begränsa värdena?
Ja
Är diametern av den vanligaste radien tillräckligt stor?
Nej
Ja
Är andelen av radier med den längsta radien i förhållande till antalet radier tillräckligt stor?
Nej
Ja
Ska 3D koordinater räknas ut?
Nej
Ja
Diametern = r*2
Diametern i vinklar av bilden =
= Diametern / (bildens bredd i pixlar * kamerans vidvinkel[bredd])
Bollens position i vinklar är dess relativa position till mitten multiplicerat med kamerans vidvinkel(theta och phi)
Y-koordinaten i 3D-rummet fås genom att dividera bollens verkliga storlek med bollens storlek i vinklar
Totala distansen till bollen fås då genom följande formel:
dis = Y / (sin(theta)*sin(phi))
X = dis * cos(theta) * sin(phi)
Z = dis * cos(phi)
Körs av add-on program?
Ja
Nej
Spara värdena som outputten
Skriv ut debugvärden
Analysera typvärden?
Ja
Nej
Gått igenom alla vinklar?
Ja
Nej
Gå igenom valt antal vinklar (ett helt varv)
Börja på ett och öka sedan avståndet
Är pixeln på koordinaten
[mittpunkten + avstånd*cos(vinkeln), mittpunkten + avstånd*sin(vinkeln)]
en orginaltråff(dvs. var en träff i början)?
Nej
Analysera typvärden?
Ja
Nej
Rita ut sista pixeln i denna radie med slumpvald färg
Spara avståndet för denna vinkel
Om detta avstånd redan finns: spara hur många gånger detta avstånd existerar.
Gå igenom alla olika avstånd
Är antalet förekomster av detta avstånd fler än den med flest?
Ja
Nej
Spara att dettaa avstånd är vanligast
Addera antalet förekomster till den totala summan
Gått igenom alla avstånd?
Ja
Nej
Skapa en bild av de nya pixlarna
Visa bilden

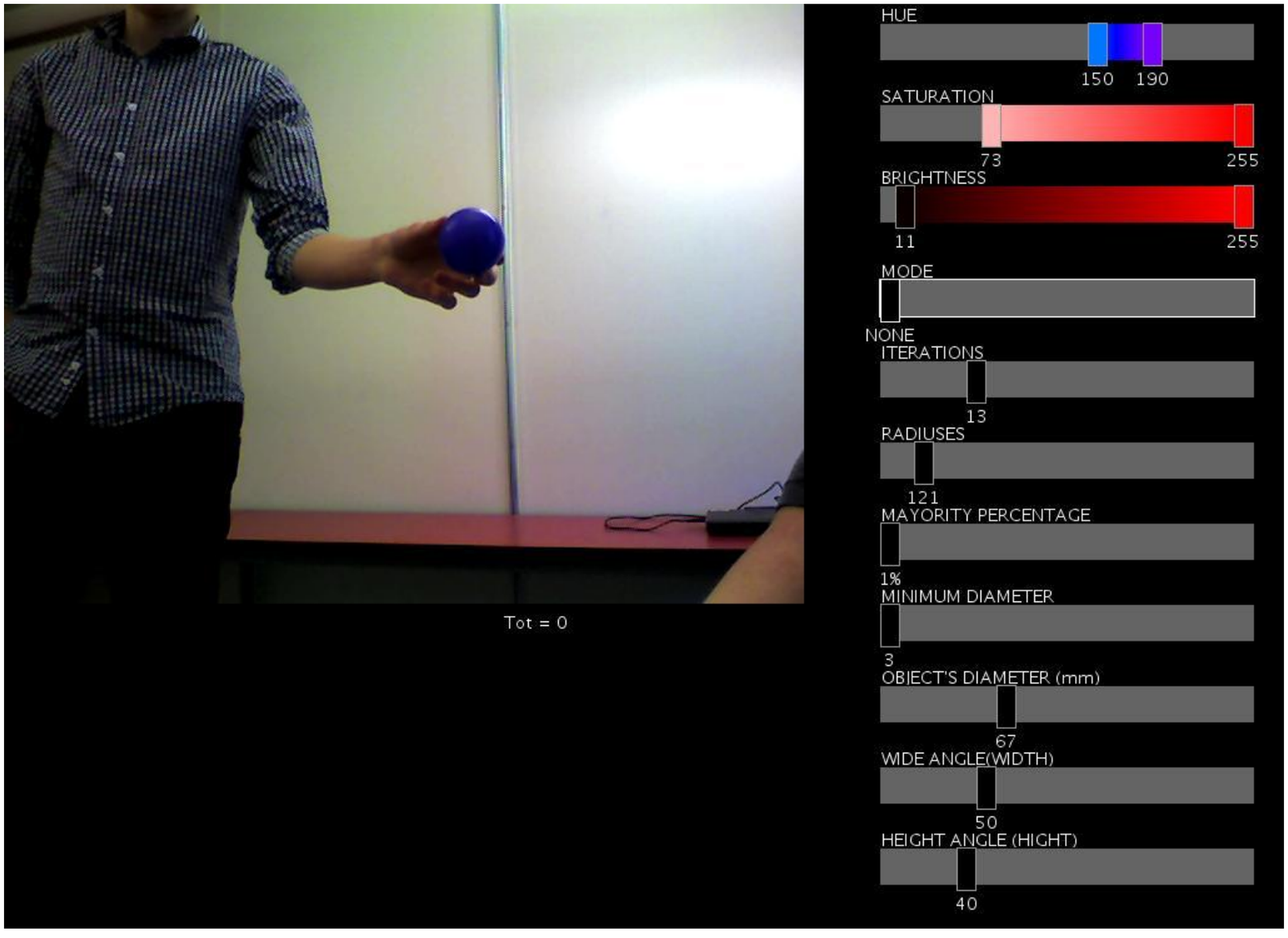
Bilaga 2

Experiement 14/2 2012

Uppmätt(cm)												
	X	0	-17,5	74,5	31	67,5	0	-0,5	-52	-106	2	6
	Y	171	69	230	82	215,5	197	114	120	240	236	26
	Z	12,65	-12,5	28	25	-24,5	7	5,5	5,5	74	74	1
Uträknat(cm)												
	X	0	-17	67	29	59	0	0	-53	-96	2	5
	Y	176	70	223	78	205	197	116	126	230	205	26
	Z	13	-13	29	25	-21	8	6	7	78	66	1
Differens(cm)												
	X	0	-0,5	7,5	2	8,5	0	-0,5	1	-10	0	1
	Y	-5	-1	7	4	10,5	0	-2	-6	10	31	0
	Z	-0,35	0,5	-1	0	-3,5	-1	-0,5	-1,5	-4	8	0

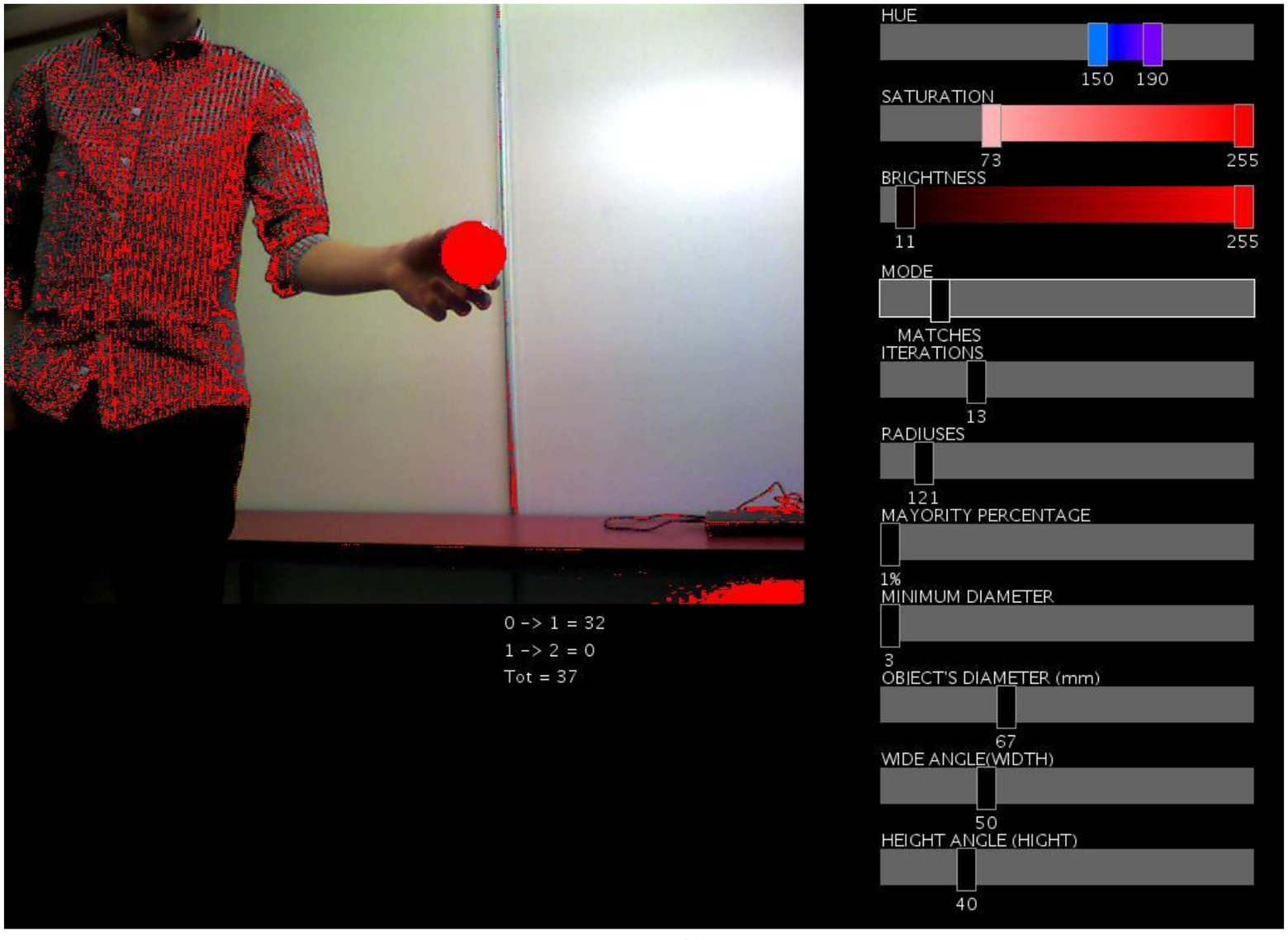


Bilaga 3.1



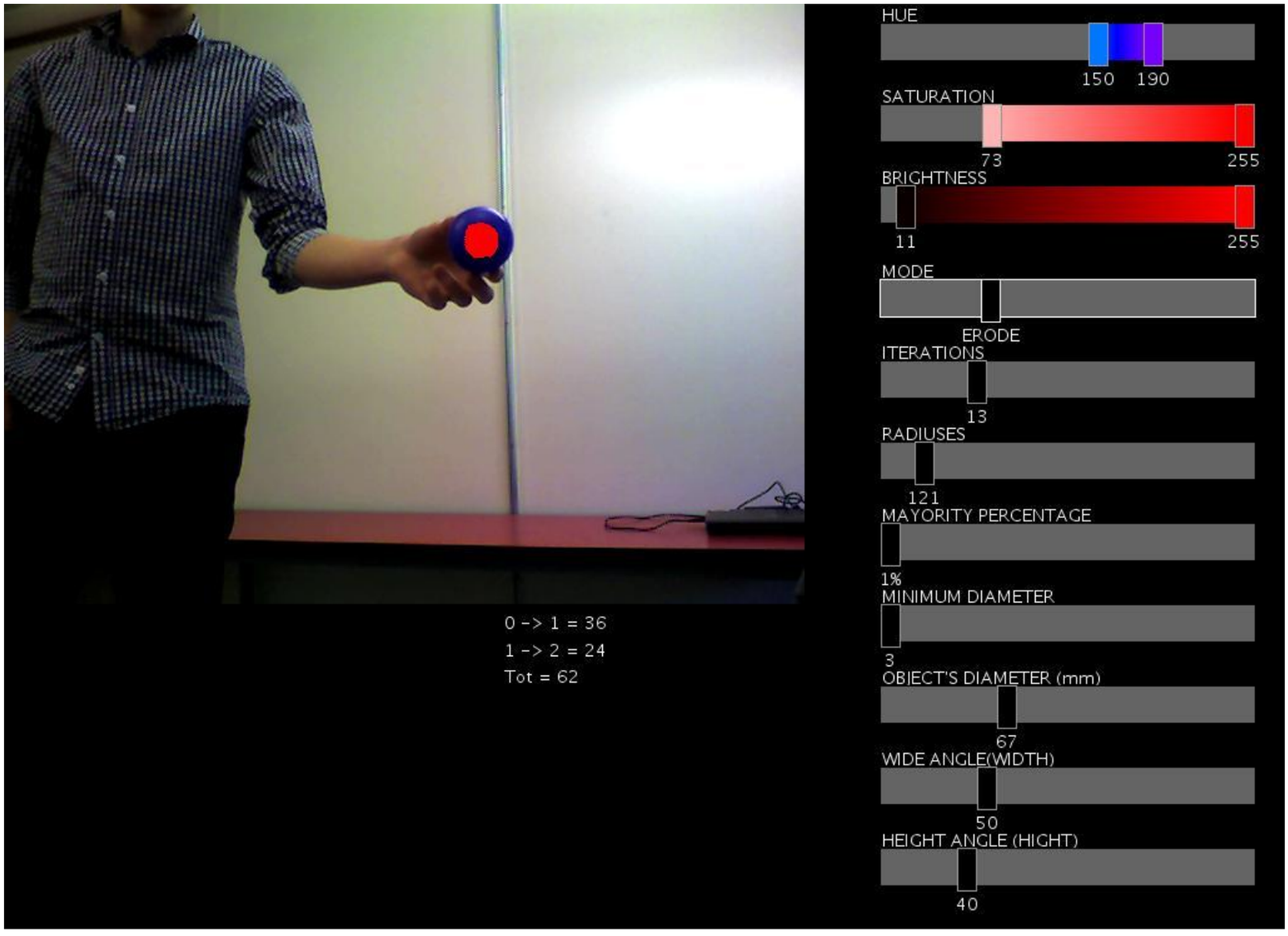
Visar en ej analyserad bild

Bilaga 3.2



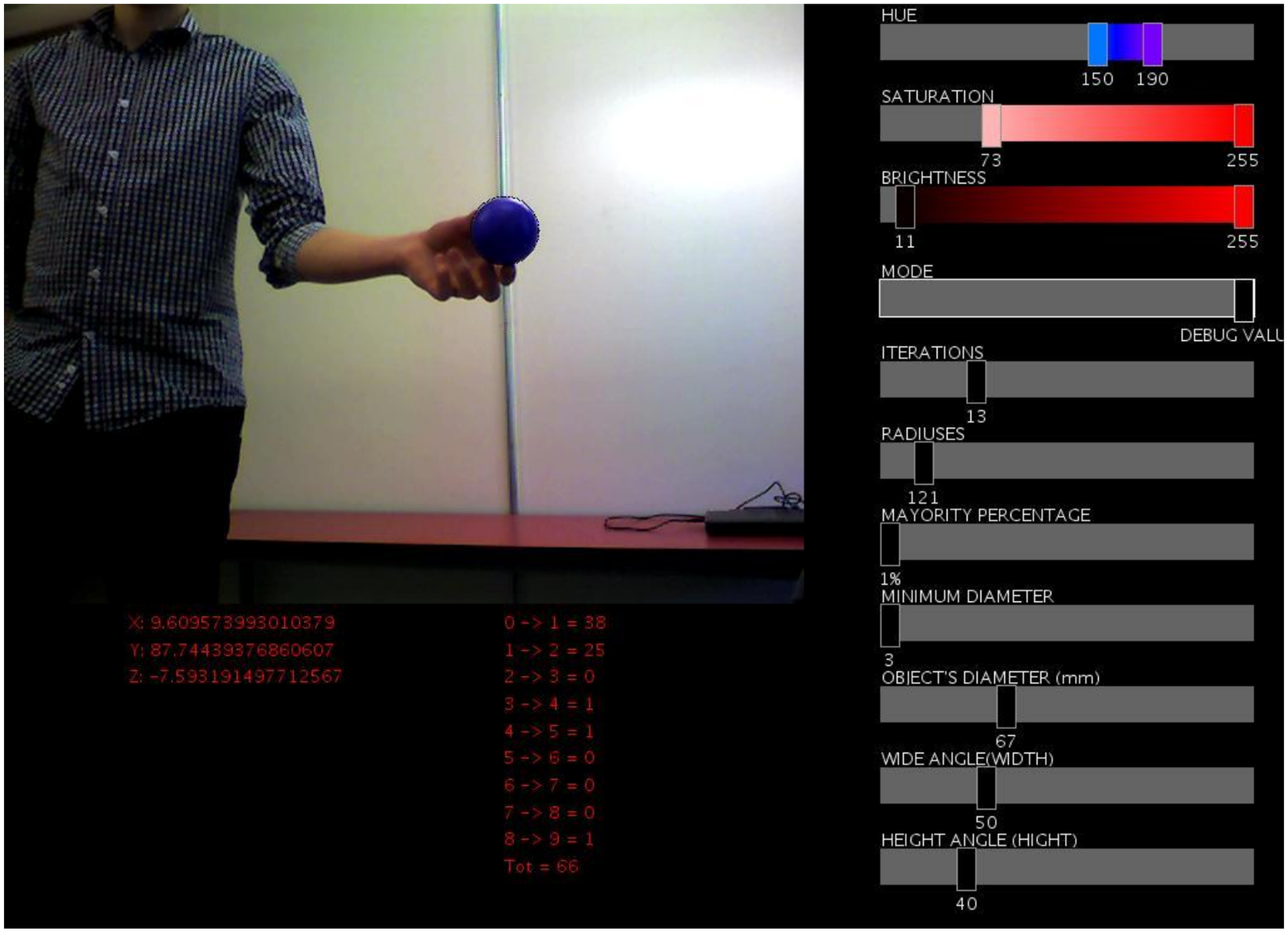
Visar träffar

Bilaga 3.3



Visar erodering

Bilaga 3.4



Visar resultat