

## Objektigenkänning med bildbehandling och avståndsmätning

En tillämpning av Sensor Fusion

Mekatronik högskoleingenjör

Fredrik Johansson Tornéus  
Johannes Ohlson



EXAMENSARBETE INOM HÖGSKOLEINGENJÖRSPROGRAMMET  
MEKATRONIK

**Objektigenkänning med bildbehandling och  
avståndsmätning**  
**- En tillämpning av Sensor Fusion**

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

FREDRIK JOHANSSON TORNÉUS  
JOHANNES OHLSON



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

Institutionen för signaler och system  
CHALMERS TEKNISKA HÖGSKOLA  
Göteborg, Sverige 2016

Objektigenkänning med bildbehandling och avståndsmätning  
- En tillämpning av Sensor Fusion  
Examensarbete inom högskoleingenjörsprogrammet Mekatronik  
FREDRIK JOHANSSON TORNÉUS  
JOHANNES OHLSON

© FREDRIK JOHANSSON TORNÉUS  
JOHANNES OHLSON, 2016.

Handledare: Manne Stenberg, Signaler och System  
Examinator: Bertil Thomas, Signaler och System

Examensarbete 2016  
Institutionen för signaler och system  
Chalmers tekniska högskola  
SE-412 96 Göteborg  
Telefonnummer +46 31 772 1000

Förstasida: Riggen som håller kamera, IR-sensor, pan och tilt servos samt Raspberry Pi.

Typsättning L<sup>A</sup>T<sub>E</sub>X  
Göteborg, Sverige 2016

## Förord

Exjobbet är den avslutande kursen på utbildningen. Genom detta har vi fått en smak av verkligheten vilket har inneburit mycket mer googlande än övriga tiden i skolan. Vi vill tacka Broccoli Engineering AB som låtit oss genomföra detta arbete samt försett oss med en engagerad och duktig handledare, Tobias Olsson. På idéstadiet av projektet fick vi mycket hjälp och nödvändig input av Björn Bergholm på Broccoli. Vi vill även tacka vår rutinerade handledare på Chalmers, Manne Stenberg, som tagit oss under sina vingar. Via skolan har vi även 3D-printat den ram som resten av hårdvaran sitter fäst vid. Detta möjliggjordes av Sakib Sisteck som ansvarar för 3D-printern och gav oss tillåtelse att använda den.

Trevlig läsning!

Fredrik Johansson Tornéus & Johannes Ohlson, Göteborg, Juni 2016

# Sammanfattning

Sensor Fusion är ett begrepp som används då två eller flera sensorer arbetar tillsammans för att utvinna mer information än vad de skulle ha gjort enskilt. Eftersom tekniken är aktuell inom olika industrier och inte minst inom fordonsindustrin, används den i detta projekt. Projektets syfte är att med hjälp av sensorer hitta en tennisboll med bildbehandling samt göra avståndsmätningar på bollen för att få ett djup i bilden. Det ska alltså vara möjligt att se att bollen inte bara har en höjd och en bredd, utan även ett djup. I starten av projektet tas en IR-sensor och en kamera fram som lämpliga sensorer för ändamålet. För att styra dessa sensorer används en Raspberry Pi som plattform där all kod skrivs i Python. En hållare för de båda sensorerna tas fram i Autodesk Fusion 360 och skrivs ut med en 3D-printer. För att rikta IR-sensorn och scanna av objektet används två servomotorer. Resultatet är en rigg som tar en bild med kameran, hittar en tennisboll på denna bild och omvandlar bollens koordinater till steg för servomotorerna. Med hjälp av koordinaterna scannas ett kvadratisk område runt bollen med IR-sensorn. På varje nytt steg servomotorerna tar i kvadraten tas en avståndsmätning som sedan används för att rita upp en bild av bollen som scannats.

Nyckelord: Sensor Fusion, avståndsmätning, bildbehandling, IR-sensor, kamera, servo, Raspberry Pi, Python

Object recognition with Image Processing and Distance Measurement  
- An application of Sensor Fusion  
Bachelor's thesis in Mechatronics  
FREDRIK JOHANSSON TORNÉUS  
JOHANNES OHLSON  
Department of Signals and Systems  
CHALMERS UNIVERSITY OF TECHNOLOGY

## **Abstract**

Sensor Fusion is a term used when two or more sensors are working together in order to extract more information than they would have done individually. Because this technology is up-to-date in various industries, and not least in the automotive industry, it is used in this project. The project aims at using sensors to find a tennis ball with the help of image processing and then making distance measurements on the ball to get a depth in the image. It shall be possible to see that the ball does not only have a height and a width, but also a depth. At the start of the project, an infrared sensor and a camera is selected as appropriate sensors for the objective. To control these sensors, a Raspberry Pi is used as a platform with all code written in Python. A carrier for both sensors are designed in Autodesk Fusion 360 and is printed by a 3D printer. To aim the infrared sensor and scan the object, two servo motors are used. The result is a setup that takes a picture with the camera, finds the tennis ball in the picture and converts its coordinates to steps for the servo motors. With help from the coordinates a square area around the ball is scanned with the IR-sensor. On each new step that the servo motors take in the square a distance reading is made. The readings are later used to plot a picture of the ball that has been scanned.

Keywords: Sensor Fusion, Distance Measurement, Image Processing, IR-sensor, Camera, Servo, Raspberry Pi, Python





# Innehåll

|                                                      |             |
|------------------------------------------------------|-------------|
| <b>Beteckningar</b>                                  | <b>xi</b>   |
| <b>Figurer</b>                                       | <b>xiii</b> |
| <b>Tabeller</b>                                      | <b>xv</b>   |
| <b>1 Inledning</b>                                   | <b>1</b>    |
| 1.1 Bakgrund . . . . .                               | 1           |
| 1.2 Syfte . . . . .                                  | 1           |
| 1.3 Avgränsningar . . . . .                          | 1           |
| 1.4 Precisering av arbetsuppgiften . . . . .         | 2           |
| <b>2 Teknisk Bakgrund</b>                            | <b>3</b>    |
| 2.1 SPI . . . . .                                    | 3           |
| 2.2 Färgsystem . . . . .                             | 3           |
| 2.2.1 BGR . . . . .                                  | 3           |
| 2.2.2 HSV . . . . .                                  | 3           |
| 2.3 Hårdvara . . . . .                               | 4           |
| 2.3.1 Raspberry Pi 3 Modell B . . . . .              | 4           |
| 2.3.2 Arduino Due . . . . .                          | 4           |
| 2.3.3 Pi Camera . . . . .                            | 5           |
| 2.3.4 Pixy CMUcam5 . . . . .                         | 5           |
| 2.3.5 Sharp GP2Y0A02YK0F IR-sensor . . . . .         | 5           |
| 2.3.6 HRLV-MaxSonar - EZ0 Ultraljudssensor . . . . . | 5           |
| 2.3.7 LIDAR lite v2 . . . . .                        | 5           |
| 2.3.8 Radar . . . . .                                | 5           |
| 2.3.9 Lynxmotion Pan och Tilt kit . . . . .          | 5           |
| 2.3.10 Adafruit PWM HAT . . . . .                    | 5           |
| 2.3.11 MCP3008 ADC . . . . .                         | 6           |
| 2.4 Mjukvara . . . . .                               | 6           |
| 2.4.1 Python . . . . .                               | 6           |
| 2.4.2 OpenCV . . . . .                               | 6           |
| 2.4.2.1 Funktioner till OpenCV . . . . .             | 6           |
| <b>3 Metod</b>                                       | <b>9</b>    |
| 3.1 Inför projekt . . . . .                          | 9           |
| 3.2 Under projektet . . . . .                        | 9           |

|          |                                               |           |
|----------|-----------------------------------------------|-----------|
| <b>4</b> | <b>Genomförande</b>                           | <b>11</b> |
| 4.1      | Urvalsprocess . . . . .                       | 11        |
| 4.1.1    | Val av plattform . . . . .                    | 11        |
| 4.1.2    | Val av kamera . . . . .                       | 11        |
| 4.1.3    | Val av sensor . . . . .                       | 12        |
| 4.1.4    | Val av sensorbärare . . . . .                 | 13        |
| 4.1.5    | Val av mjukvara . . . . .                     | 13        |
| 4.2      | Setup av kamera samt hitta objekt . . . . .   | 13        |
| 4.3      | Setup av IR . . . . .                         | 16        |
| 4.4      | Förhållande mellan kamera och servo . . . . . | 17        |
| 4.4.1    | Kamerapixlar och synfält . . . . .            | 17        |
| 4.4.2    | Servonas steg och synfält . . . . .           | 17        |
| 4.4.3    | Översättning från pixlar till steg . . . . .  | 19        |
| 4.5      | Skapande av avståndsmatris . . . . .          | 20        |
| 4.6      | Rigg och offsetberäkningar . . . . .          | 21        |
| 4.7      | Programmet körs . . . . .                     | 24        |
| <b>5</b> | <b>Resultat</b>                               | <b>25</b> |
| <b>6</b> | <b>Diskussion</b>                             | <b>27</b> |
|          | <b>Litteraturförteckning</b>                  | <b>29</b> |
| <b>A</b> | <b>Bilagor</b>                                | <b>I</b>  |
| A.1      | Kopplingsschema . . . . .                     | II        |
| A.2      | Programkod . . . . .                          | III       |
| A.2.1    | Definitions_file.py . . . . .                 | III       |
| A.2.2    | MAIN_file.py . . . . .                        | VII       |

## Beteckningar

**ADC** - Analog to Digital Converter

**SPI** - Serial Peripheral Interface

**RPi** - Raspberry Pi

**GPIO** - General Purpose Input/Output

**HSV** - Hue Saturation Value

**FPS** - Frames Per Second

**CSI** - Camera Serial Interface

**BGR** - Blue Green Red

**PWM** - Pulse Width Modulation

**UART** - Universal Asynchronous Receiver/Transmitter

**TWI** - Two Wire Interface

**JTAG** - Joint Test Action Group



# Figurer

|      |                                                                                                                        |    |
|------|------------------------------------------------------------------------------------------------------------------------|----|
| 2.1  | BGR färgmodell . . . . .                                                                                               | 3  |
| 2.2  | Hue färgintervall med värden mellan 0-360 grader . . . . .                                                             | 4  |
| 2.3  | HSV färgskala . . . . .                                                                                                | 4  |
| 4.1  | Boll med tre avståndsmätningar och den form objektet antas ha . . . . .                                                | 12 |
| 4.2  | Pan och tilt rigg med IR-sensor . . . . .                                                                              | 13 |
| 4.3  | BGR och HSV värden för undre och övre gräns på det valda färgintervall<br>tervallet . . . . .                          | 14 |
| 4.4  | Vanlig och suddig bild från kamera . . . . .                                                                           | 14 |
| 4.5  | Ett exempel på hur en svartvit bild med störningar (vänster) eroderas<br>(mitten) och sedan späds ut (höger) . . . . . | 15 |
| 4.6  | Svartvit bild på objekt . . . . .                                                                                      | 15 |
| 4.7  | Hittat objekt med cirkel . . . . .                                                                                     | 16 |
| 4.8  | Koppling mellan IR och ADC (se bilaga för hela kopplingsschemat) . . . . .                                             | 16 |
| 4.9  | Kamerans synfält uttryckt i grader och pixlar . . . . .                                                                | 17 |
| 4.10 | Bollens storlek uttryckt i vinkel . . . . .                                                                            | 18 |
| 4.11 | Servots position beroende på pulsbredd . . . . .                                                                       | 18 |
| 4.12 | Översättning från pixlar till steg tillsammans med scanningsmönster . . . . .                                          | 20 |
| 4.13 | Scannad tennisboll på 100 cm avstånd . . . . .                                                                         | 20 |
| 4.14 | Scannad tennisboll på 50 cm avstånd . . . . .                                                                          | 21 |
| 4.15 | Riggen med IR, kamera och RPi . . . . .                                                                                | 21 |
| 4.16 | Riggen med IR och kamera i profil för att visa de överlappande synfälten . . . . .                                     | 22 |
| 4.17 | Rätvinklig triangel med spets i IR-sensor samt objekt . . . . .                                                        | 23 |
| 4.18 | Programmets flödesschema . . . . .                                                                                     | 24 |
| 5.1  | 3D bild på 100cm avstånd . . . . .                                                                                     | 26 |
| 5.2  | 3D bild på 50cm avstånd . . . . .                                                                                      | 26 |



# Tabeller

|     |                                        |    |
|-----|----------------------------------------|----|
| 4.1 | Pughmatris för val av sensor . . . . . | 12 |
|-----|----------------------------------------|----|





# 1

## Inledning

### 1.1 Bakgrund

Sensor Fusion handlar om att använda fler än en sensor för att få ut mer data än vad sensorerna hade kunnat ge var för sig. Detta arbete handlar om att med hjälp av Sensor Fusion hitta ett objekt med två olika tekniker. Den första tekniken är användning av bildbehandling och den andra tekniken är användning av avståndsmätning. Anledningen till kombineringsen av teknikerna är för att på ett mer säkert sätt kunna avgöra om rätt objekt hittats. Kombineringsen möjliggör skapandet av en bild med mer information än vad de två teknikerna kan göra var för sig. Dessa tekniker appliceras i detta fall på en tennisboll. Broccoli Engineering AB är ett konsultföretag som jobbar mycket mot fordonsindustrin. Dagens bilar innehåller många olika typer av sensorer och några av dessa är bland annat kameror, IR-sensorer, ultraljud, radar, lidar med flera. Detta är en av anledningarna till att några av dessa sensorer används inom examensarbetet. Sensor Fusion är aktuellt inom fordonsindustrin, speciellt när det gäller självkörande bilar. Anledningen till detta är att man vill få ut så mycket som möjligt av de komponenter som redan finns i bilarna.

### 1.2 Syfte

Syftet är att få två lättillgängliga och billiga sensorer med teknikerna bildbehandling och avståndsmätning att arbeta tillsammans för att hitta en tennisboll. Denna ska behandlas med båda teknikerna för att få ut dels en bild av bollen som beskriver dess färg och konturer och dels en bild som beskriver bollens fysiska form. Dessa bilder tillsammans kommer ge en bättre bild av objektet än vad de två bilderna var för sig kan ge.

### 1.3 Avgränsningar

Tennisbollen ska kunna hittas på en meters avstånd från sensorerna i en kontrollerad miljö. Detta betyder att ljuset samt färger i omgivningen kan komma att väljas så att tennisbollen kan hittas på ett enkelt sätt. Endast hårdvara som anses vara konsumentvänlig används i arbetet. Fokuset i arbetet ligger i att hitta och läsa av tennisbollen. Därför kommer befintlig mjuk- och hårdvara att användas där det är möjligt så att större delen av tiden kan läggas på att få allt att arbeta tillsammans. På grund av begränsad tidsbudget kommer inte olika typer av optimering att

prioriteras.

### 1.4 Precisering av arbetsuppgiften

- Vilka sensorer är lämpliga för projektet, samt lättillgängliga och billiga?
- Vad för slags rigg behöver byggas?
- Vad för precision måste servomotorerna minst ha för att kunna rikta sig tillräckligt precist för att mäta objektet?
- Hur ska de olika delarna arbeta tillsammans?
- Är detta ett lämpligt tillvägagångssätt för att få ut mer data?
- Hur ska det avlästa objektet presenteras?

# 2

## Teknisk Bakgrund

### 2.1 SPI

Står för Serial Peripheral Interface och är en buss för seriell kommunikation på korta avstånd. Det används främst i inbyggda system för kommunikation mellan enheter. Den har fyra logiska signaler:

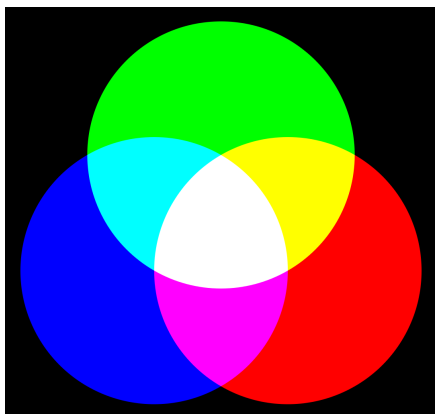
- SCLK: Serial Clock - En klocksignal som genereras av masterenheten.
- MOSI: Master Output, Slave Input
- MISO: Master Input, Slave Output
- SS: Slave Select

[1].

### 2.2 Färgsystem

#### 2.2.1 BGR

Är en färgskala som står för 'Blue Green Red'. Det är samma sak som den vanliga RGB färgskalan fast i en annan ordning. Färgerna sätts till ett värde från 0-255 och då dessa tre färgintervall blandas fås en färg.



Figur 2.1: BGR färgmodell

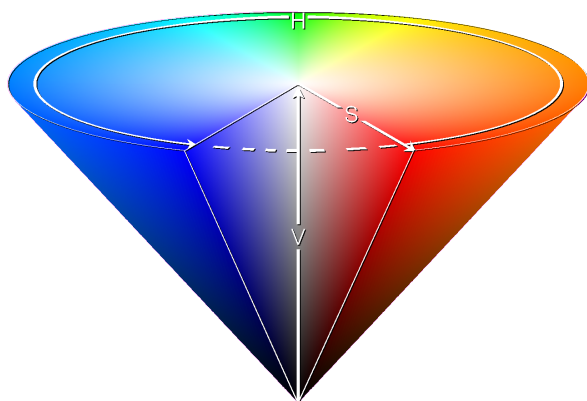
#### 2.2.2 HSV

Är en färgskala som står för 'Hue Saturation Value'. I denna skala sätter 'Hue' vilken typ av färg som ska användas på ett värde mellan 0-360 grader. 'Saturation'

och 'Value' sätts från 0-100% och säger hur stor mättnadsgrad samt vilken ljusstyrka färgen ska ha.



**Figur 2.2:** Hue färgintervall med värden mellan 0-360 grader



**Figur 2.3:** HSV färgskala

## 2.3 Hårdvara

### 2.3.1 Raspberry Pi 3 Modell B

Är en enkortsdator i storlek av ett kreditkort. Den används vanligtvis i projekt då man vill koppla sig till annan hårdvara som till exempel en kamera eller olika sensorer. Detta görs lätt då denna Raspberry Pi är utrustad med kamera interface (CSI) och 40 GPIO pins. Förutom detta är denna modell den kraftigaste i Raspberry Pi-serien då den är uppgraderad till en 1.2GHz 64-bit quad-core ARMv8 CPU. Den har även en HDMI port för skärm, ethernet port för internetkoppling och 4 USB portar för tangentbord och mus[2].

### 2.3.2 Arduino Due

Är en microcontroller som används inom projekt då man vill utföra snabba beräkningar och koppla sig till diverse hårdvara, såsom sensorer. Den är baserad på Atmel SAM3X8E ARM Cortex-M3 CPU med 32-bits kärna, klockfrekvens på 84 MHz, 96 kBytes SRAM, 512 kBytes flashminne. Den har 54 stycken in- och utsignalspinnar med möjlighet att använda PWM som utsginal på 12 av pinnarna. Det finns 4 stycken UART portar, 2 stycken DAC, 2 stycken TWI samt kontakter för SPI och JTAG[3].

### **2.3.3 Pi Camera**

Pi Camera är en kameramodul som är byggd för Raspberry Pi. Den kan filma i 1080p med 30 FPS (bilder per sekund) och kan ta stillbilder på 2592 x 1944 pixlar. Dess horisontala respektive vertikala synfält ligger på 53.50 samt 41.41 grader[4].

### **2.3.4 Pixy CMUcam5**

En kameramodul för Raspberry Pi, Arduino, BeagleBone och liknande enheter. Den är speciellt utformad för objektigenkänning och med ett knapptryck kan man ställa in vad för objekt den ska leta efter. Detta objekts koordinater kan skickas direkt till enheten som kameran är kopplad till[5].

### **2.3.5 Sharp GP2Y0A02YK0F IR-sensor**

En avståndssensor som använder sig av IR ljus och triangulering för att bestämma ett avstånd till objekt som är inom ett smalt område framför sensorn. Den har en längdupplösning på 1mm och kan mäta inom intervallet 20-150cm[6].

### **2.3.6 HRLV-MaxSonar - EZ0 Ultraljudssensor**

En avståndssensor som använder sig av ultraljud för att bestämma avståndet till ett objekt som är framför sensorn. Den har en längdupplösning på 1mm och kan mäta inom intervallet 30-500cm[7].

### **2.3.7 LIDAR lite v2**

En avståndssensor som använder sig av laser för att mäta avståndet till ett objekt som befinner sig inom ett smalt område framför sensorn. Den har en längdupplösning på 1cm och kan mäta upp till 4000cm[8].

### **2.3.8 Radar**

En teknik för att hitta objekt med hjälp av radiovågor. Används främst för att hitta flygplan, båtar, missiler och andra stora objekt som rör på sig[9].

### **2.3.9 Lynxmotion Pan och Tilt kit**

Två servon monterade på varandra för att göra det möjligt att vrida en yta i både vertikalt och horisontellt led. Servomotorerna som används till detta är två stycken Hitec HS-422[10].

### **2.3.10 Adafruit PWM HAT**

En påbyggnadsmodul, även kallad sköld, till Raspberry Pi som möjliggör styrning av upp till 16 stycken servomotorer[11].

### 2.3.11 MCP3008 ADC

Är en analog till digital signalomvandlare. Den har 8 kanaler och omvandlar en insignal som kan variera mellan spänningen 0 V och  $V_{ref}$ , där  $V_{ref}$  är matningsspänningen till ADC. Insignalen omvandlas till en digital utsignal på 10 bitar vilket motsvarar 1024 värden. Komponenten drivs av en spänning på mellan 2.7 till 5.5 V. Kommunikation med enheten sker med ett enkelt seriellt interface som är kompatibelt med SPI protokollet[12].

## 2.4 Mjukvara

### 2.4.1 Python

Är ett vanligt förekommande otypat högnivåspråk inom programmering. Koden skrivs på ett sätt som anses rent och ska vara lätt att förstå och läsas. Det är ett allmänt språk som är gjort för att kunna programmera många olika typer av program. Till Python finns ett stort standardbibliotek och stöd till objektorienterad och funktionell programmering[13].

### 2.4.2 OpenCV

Står för Open Source Computer Vision och är ett bibliotek av funktioner som riktar sig mot computer vision i realtid, alltså analys av bilder. OpenCV används bland annat för att hitta objekt, titta på rörelser och känna igen ansikten. Språket är från början skrivet i C men har på senare tid översatts till bland annat C++, C#, Java och Python[14].

#### 2.4.2.1 Funktioner till OpenCV

**cv2.GaussianBlur()** - Skapar oskärpa i bilden genom att vara ett lågpasfilter som filtrerar bort höga frekvenser, alltså brus och skarpa kanter[15].

**cv2.cvtColor()** - Konverterar bildens färgskala[16].

**cv2.inRange()** - Gör om bilden till svartvit där färger som ligger i ett bestämt intervall blir vitt och resten svart[17].

**cv2.erode()** - Analyserar pixlarna på en binär bild. De angränsande pixlarna till en nolla sätts till noll (svart). Detta används för att ta bort brus[18].

**cv2.dilate()** - Analyserar pixlarna på en binär bild. De angränsande pixlarna till en etta sätts till ett (vit)[18].

**cv2.findContours()** - Hittar konturen i en svartvit binär bild, där objektets kontur är ett vitt område mot en svart bakgrund[19].

**cv2.contourArea()** - Analyserar arean på den hittade konturen[19].

**cv2.minEnclosingCircle()** - Hittar minsta omslutande cirkel runt konturen[19].

**cv2.moments()** - Används för att räkna ut centrumpunkt, area med mera[19].

**cv2.circle()** - Ritar en cirkel med hjälp av koordinaterna för centrumpunkt samt radie[20].

**cv2.imshow()** - Visar bild





# 3

## Metod

### 3.1 Inför projekt

För att få en bra start på projektet görs först en förstudie på de olika komponenterna. Hård- och mjukvara studeras med hjälp av internet. En planeringsrapport skrivs och en tidsplan i form av ett gantt-schema skapas för att underlätta starten och genomförandet av projektet så mycket som möjligt. Sensorer väljs genom en urvalsprocess för att möjliggöra beställning av komponenter så de finns tillgängliga när projektet startar.

### 3.2 Under projektet

Till en början monteras hårdvaran ihop och installeras mot RPi med hjälp av guider från internet. Det undersöks vilket språk som lämpar sig för att styra den hårdvara som finns utifrån hur användarvänligt språket är. En rigg skapas med CAD programmet Autodesk Fusion 360 som sensorerna kan monteras på så dessa har en fast utgångspunkt. Metod för hur kameran ska kunna hitta bollen med OpenCV tas fram genom att läsa tidigare arbeten där målet varit att hitta en boll. Metod för hur sensorn ska kunna mäta flera olika punkter på bollen tas fram. Program skrivs i Python. Ett program skrivs för att översätta kamerans koordinater till pan och tilt koordinater. Ett annat skrivs för att scanna av bollen med sensorn i ett bestämt mönster. Värdena som fås av sensorn sparas i en matris som sedan plottas med hjälp av en plotfunktion i Python. Datan presenteras så man ser att de båda olika teknikerna hittar en boll.



# 4

## Genomförande

### 4.1 Urvalsprocess

Här bestäms vilken hårdvara och mjukvara som är mest lämpad för projektet. Flera alternativ av kamera, sensor och plattform tas fram för att sedan utvärderas vidare. Från denna utvärderingsprocess bestäms det vilken kamera, sensor och plattform som är lämplig för projektet.

#### 4.1.1 Val av plattform

Plattformen behöver kunna hantera bildbehandling, uträkningar, och presentation av datan på ett bra sätt. Som alternativ finns Arduino Uno, Raspberry Pi samt en PC. Efter undersökning anses RPi mest lämpad för uppgiften eftersom den har goda möjligheter till inkoppling av moduler, såsom en kamera och sensorer. Förutom detta kan den enkelt kopplas till en extern skärm, tangentbord och mus vilket gör att enheten lätt kan styras.

Arduino Due har goda möjligheter till inkoppling av extern hårdvara precis som RPi. Förutom digital kommunikation kan Arduino Due även hantera analoga in- och utsignaler, vilket inte RPi kan. Arduino Due programmeras genom att den kopplas till en dator och programmet förs över till enheten. Den är inte gjord för att stödja direkt inkoppling av tangentbord, mus och skärm på samma sätt som RPi då den saknar både USB- och bildportar. Detta samt att RPi har större beräkningskapacitet leder till att Arduino Due inte används.

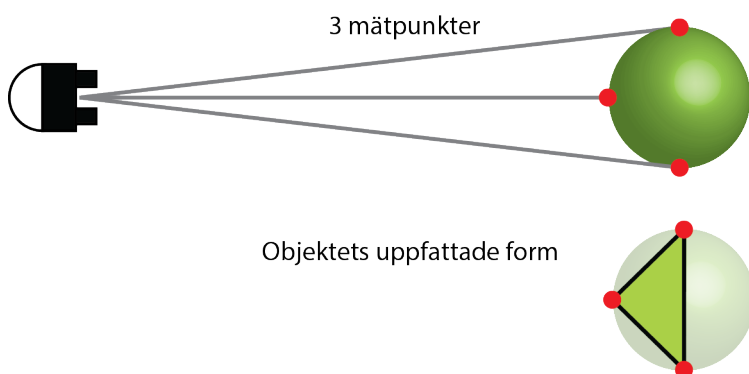
En PC är lika lätt att styra som en RPi med tangentbord, skärm och mus. Den har mer beräkningskapacitet än en RPi men saknar att lätt kunna kopplas till annan hårdvara. Därför väljs PC bort.

#### 4.1.2 Val av kamera

Kameran ska kunna identifiera en tennisboll på en meters avstånd samt vara konsumentvänlig. Som kandidater finns Pixy CMUcam v5 samt Raspberry Pi Camera. Kameran för RPi valdes eftersom den klarar kriterierna samt att Pixy CMUcam v5 ansågs vara för välutvecklad för ändamålet och skulle ta bort en del av utmaningen i att realisera detta system.

### 4.1.3 Val av sensor

Sensorn ska kunna göra minst tre olika avståndsmätningar längs centrumlinjen på en tennisboll på en meters avstånd. Detta för att minst kunna få en väldigt grov bild av att objektet har ett djup.



**Figur 4.1:** Boll med tre avståndsmätningar och den form objektet antas ha

De typer av sensorer som undersöks är ultraljud, IR samt LIDAR. Här används en pughmatrix eftersom många kriterier spelar in i valet.

**Tabell 4.1:** Pughmatrix för val av sensor

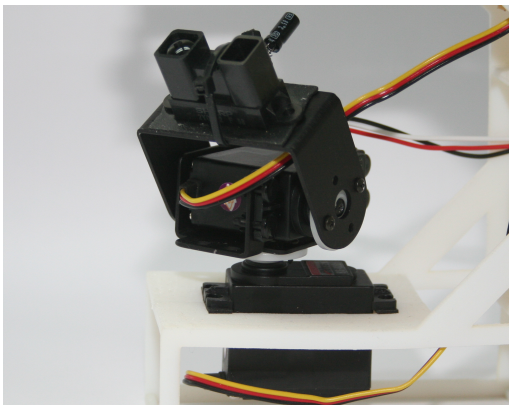
| Kriterier                     | IR-sensor Sharp<br>GP2Y0A02YK0F<br>(Ref) | LIDAR lite<br>v2 | Ultraljud<br>HRLV-<br>MaxSonar<br>- EZ0 |
|-------------------------------|------------------------------------------|------------------|-----------------------------------------|
| Avståndsupplösning            | 0                                        | -                | 0                                       |
| Mätkonvinkel                  | 0                                        | -                | -                                       |
| Mätavstånd                    | 0                                        | -                | 0                                       |
| Sample rate                   | 0                                        | -                | -                                       |
| Minsta hittbara objektstorlek | 0                                        | +                | +                                       |
| Användarvänlighet             | 0                                        | 0                | -                                       |
| Kostnad                       | 0                                        | -                | -                                       |
| Lämplighet för projektet      | 0                                        | 0                | 0                                       |
| Antal +                       |                                          | 1                | 1                                       |
| Antal 0                       |                                          | 2                | 4                                       |
| Antal -                       |                                          | 4                | 3                                       |
| Nettovärde                    | 0                                        | -3               | -2                                      |
| Vidareutveckling              | JÄ                                       | NEJ              | NEJ                                     |

Av dessa valdes IR på grund av att den är lättillgänglig, konsumentvänlig och har ett relativt litet mätområde jämfört med de övriga testade sensorerna. Den har även relativt noggrann avståndsmätning med en upplösning på 1mm. Ultraljudet valdes

bort eftersom mätområdet är för stort. LIDAR valdes bort på grund av en ohållbart hög avståndsupplösning på 25mm. Den tennisboll som används har en radie på 32.5 mm vilket betyder att avståndsmätningarna lätt skulle kunna bli för otydliga för att urskilja objektets verkliga form. Radar undersöktes också men lämpar sig bättre för avståndsmätning på längre avstånd och i miljöer utan hinder. Ingen radarmodul hittades till ett konsumentvänligt pris. Den verkar även lämpa sig bättre för användning i utomhusmiljö då radiovågorna lätt kan studsas mot väggar och på så sätt skapa störningar i en inomhusmiljö. Därför är radar inte ens med i pughmatrisen.

#### 4.1.4 Val av sensorbärare

För att kunna ta flera mätningar på tennisbollen måste sensorn kunna riktas mot olika punkter på den. Detta görs genom att sensorn monteras på en pan och tilt rigg. Riggen består av två servon monterade på varandra. Dessa servon måste kunna rikta sig så att sensorn minst kan ta tre olika mätningar längs med tennisbollens centrumlinje på en meters avstånd. Efter undersökning hittades till slut Lynxmotion Pan and Tilt kit med två stycken Hitec HS-422 servomotorer. Riggen uppfyller kraven och valdes därför. För att kunna styra servomotorerna används helst en extern drivare. En Adafruit PWM HAT visade sig vara lämpad att styra servomotorerna och valdes därför.



Figur 4.2: Pan och tilt rigg med IR-sensor

#### 4.1.5 Val av mjukvara

Projektet kräver mjukvara för att styra kamera, sensor och sensorbärare. Det visade sig att alla tre delar kunde styras på ett bra sätt med färdiga moduler skrivna i Python, därför valdes det som programmeringsspråk.



## 4.2 Setup av kamera samt hitta objekt


Som kamera väljs RPi kameramodul som direkt kopplas in i ett CSI på RPi. Denna kan filma i 1080p men ställs in med upplösningen 640\*480 då RPi hanterar det bättre. För att hitta objekt med kameran väljs programmet OpenCV. Programmet är både lättillgängligt och bygger på öppna bibliotek som innehåller all nödvändig kod

## 4. Genomförande

för ändamålet. Genom att använda denna kod behöver hjulet inte uppfinnas igen.

För att hitta det objekt som önskas, i detta fallet är en grön tennisboll, används färgkoder för att märka ut rätt objekt samt skala bort ointressanta objekt. I OpenCV matas ett övre och undre färgvärde in i BGR färgskalan. Detta översätts till HSV färgskalan som programmet sedan använder. Det är dessa färgvärden som bestämmer vilket färgintervall programmet letar efter. All färg som inte tillhör färgintervallet skalas bort från bilden.

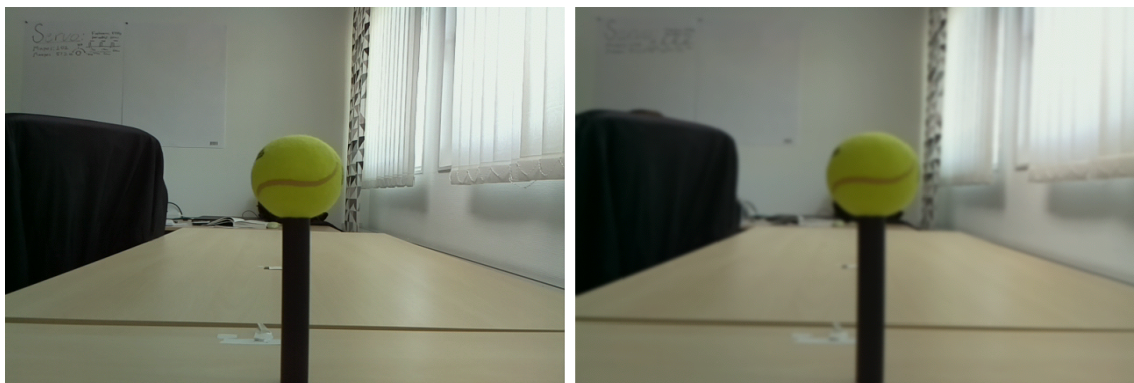
|                        |                                                                                   |                        |                                                                                    |
|------------------------|-----------------------------------------------------------------------------------|------------------------|------------------------------------------------------------------------------------|
| Enter red color (R):   | <input type="text" value="40"/>                                                   | Enter red color (R):   | <input type="text" value="240"/>                                                   |
| Enter green color (G): | <input type="text" value="100"/>                                                  | Enter green color (G): | <input type="text" value="255"/>                                                   |
| Enter blue color (B):  | <input type="text" value="29"/>                                                   | Enter blue color (B):  | <input type="text" value="64"/>                                                    |
| Hue (H):               | <input type="text" value="111"/> °                                                | Hue (H):               | <input type="text" value="65"/> °                                                  |
| Saturation (S):        | <input type="text" value="71.0"/> %                                               | Saturation (S):        | <input type="text" value="74.9"/> %                                                |
| Value (V):             | <input type="text" value="39.2"/> %                                               | Value (V):             | <input type="text" value="100.0"/> %                                               |
| Color preview:         |  | Color preview:         |  |



**Figur 4.3:** BGR och HSV värden för undre och övre gräns på det valda färgintervallet [21].

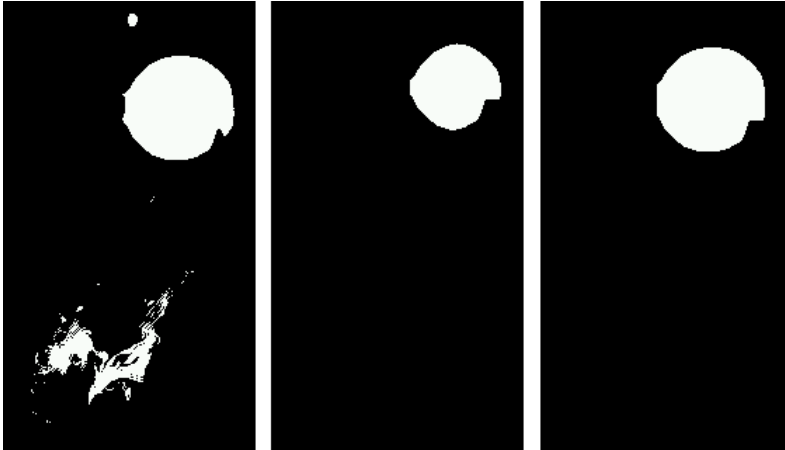
Färgintervallet är ganska stort eftersom ljuset i rummet kan variera vilket har som följd att bollens färg också varierar.

Kamerabilden på bollen behandlas i olika steg för att komma fram till en svartvit bild där allting är svart förutom färgerna som tillhör intervallet, dessa blir vita. Det första steget i behandlingen är att göra bilden suddig för att få bort högfrekvent brus och få mjukare kontraster.

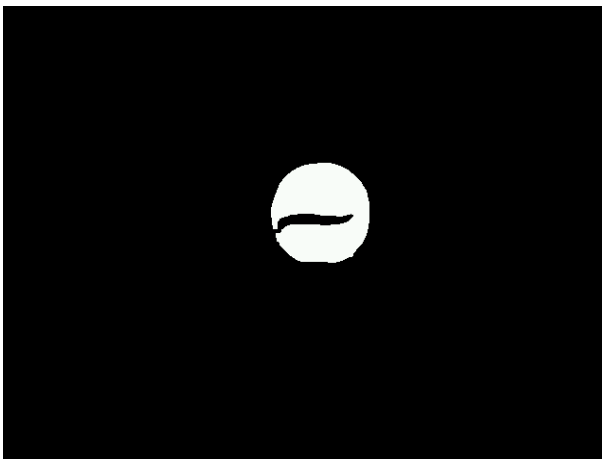


**Figur 4.4:** Vanlig och suddig bild från kamera

Därefter jämförs färgintervallet med resten av färgerna i bilden för att bestämma vad som ska vara vitt och vad som ska vara svart. Bilden görs svartvit, och för att få bort ytterligare störningar eroderas bilden. Detta gör att alla vita områden krymper i storlek, vilket betyder att de minsta områdena försvinner helt. Efter detta späds bilden ut för att utvidga de vita områdena som finns kvar till dess ursprungliga storlek. Ett av dessa vita områden symboliserar bollen. Då bollen eroderar krymper området, när bollen sedan späds ut växer området till sin ursprungliga storlek.



**Figur 4.5:** Ett exempel på hur en svartvit bild med störningar (vänster) eroderas (mitten) och sedan späds ut (höger)



**Figur 4.6:** Svartvit bild på objekt

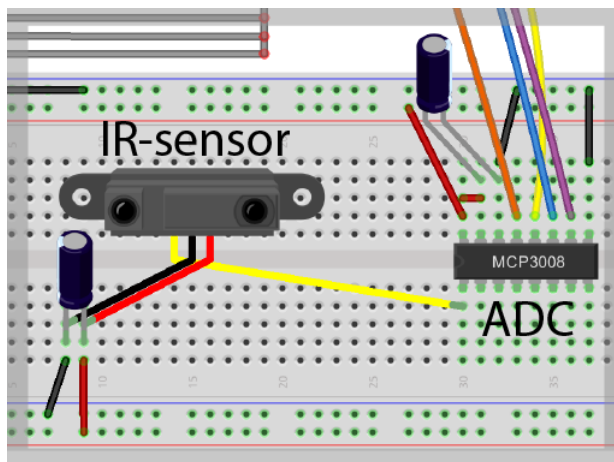
En ny bild med upplösningen 640\*480 pixlar analyseras 32 gånger per sekund. Först kontrolleras det att minst ett vitt område har hittats. Om detta är sant letas det nu efter största området på bilden. En centrumpunkt prickas ut samt en cirkel ritas runt området för att tydligt visa i den vanliga bilden vad som har hittats.



**Figur 4.7:** Hittat objekt med cirkel

### 4.3 Setup av IR

IR-sensorn som används är en Sharp GP2Y0A02YK0F. Den sitter fast i pan och tilt riggen. Sensorn kräver tre kablar, en jord, en matning på 4.5 till 5.5 V samt en signalkabel som skickar en spänning på mellan 0.4 till 2.8 V där 0.4 V motsvarar ett objektavstånd på ca 150 cm och 2.8 V motsvarar ca 20 cm. Denna signal kopplas till en ADC eftersom RPi endast kan behandla digitala signaler.



**Figur 4.8:** Koppling mellan IR och ADC (se bilaga för hela kopplingschemat)

Den ADC som används är en MCP3008. Vilken port som ska läsas från ADC bestäms av en modul på RPi. Samma modul ger en utsignal som säger på vilket avstånd i millimeter som IR-sensorn känner av ett objekt. Modulen som sköter detta heter readadc och kan hittas i kodbilagan.



## 4.4 Förhållande mellan kamera och servo

I detta avsnitt skapas ett samband mellan kamerans pixlar/grad samt servons grader/steg. Detta används sedan för att översätta kamerans pixlar till servons steg.

### 4.4.1 Kamerapixlar och synfält

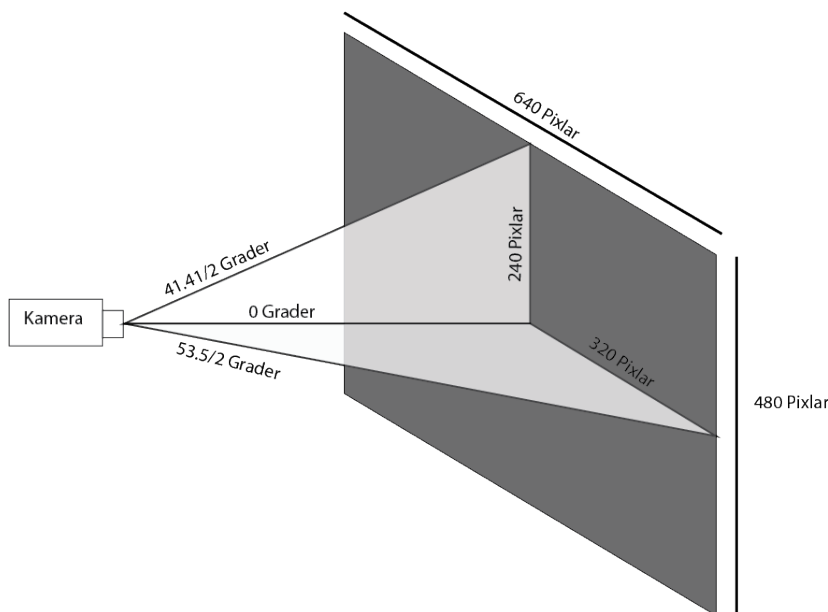
Kameran filmar i 640\*480 pixlar i horisontal respektive vertikalled. Synfältet är på 53.50 grader horisontellt och 41.41 grader vertikalt. Detta betyder att antal pixlar per grad i horisontalled blir

$$\frac{640[\text{pixlar}]}{53.5[\text{grader}]} = 11.96 \frac{[\text{pixlar}]}{[\text{grad}]} \quad (4.1)$$

I vertikalled blir det

$$\frac{480[\text{pixlar}]}{41.41[\text{grader}]} = 11.59 \frac{[\text{pixlar}]}{[\text{grad}]} \quad (4.2)$$

I programmet är noll grader i mitten av kamerans bild vilket gör att kameran kan titta  $\pm 41.41/2$  grader i vertikalled och  $\pm 53.5/2$  grader i horisontalled.



**Figur 4.9:** Kamerans synfält uttryckt i grader och pixlar

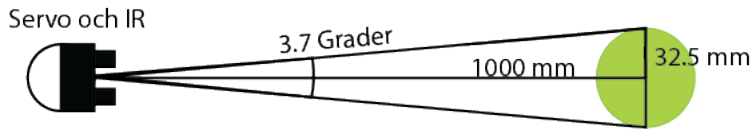
### 4.4.2 Servonas steg och synfält

Ett av kraven är att servomotorerna minst ska kunna klara av tre mätningar längs tennisbollens centrumlinje på en meters avstånd. Tennisbollen har en radie på 32.5 mm och avståndet till den är 1000 mm.

$$\tan \frac{x[\text{grader}]}{2} = \frac{32.5[\text{mm}]}{1000[\text{mm}]} \quad (4.3)$$

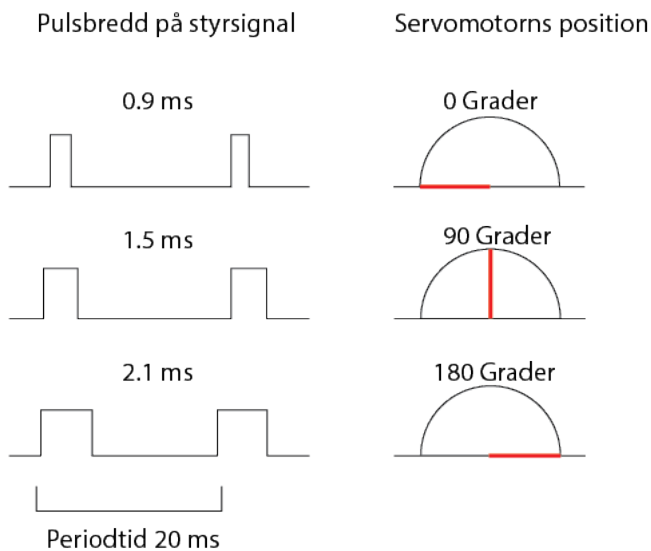
$$x[\text{grader}] = 2 \tan^{-1} \frac{32.5[\text{mm}]}{1000[\text{mm}]} = 3.72[\text{grader}] \quad (4.4)$$

Detta ger att bollen tar upp 3.72 grader av servots synfält.



**Figur 4.10:** Bollens storlek uttryckt i vinkel

Servona kan vrida sig 180 grader. Vridningen bestäms av längden på en positiv puls som skickas till servot. Om längden på pulsen är 0.9 ms vrids servot till 0° och om pulsen är 2.1 ms vrids servot till 180° [22].



**Figur 4.11:** Servots position beroende på pulsbredd

Om minimumkravet med 3 steg ska uppnås behöver servot ha en steglängd som är max

$$\frac{3.72[\text{grader}]}{3[\text{steg}]} = 1.24 \frac{[\text{grader}]}{[\text{steg}]} \quad (4.5)$$

Detta leder till att servot på 180 grader minst ska kunna ta

$$\frac{180[\text{grader}]}{1.24 \frac{[\text{grader}]}{[\text{steg}]}} = 145[\text{steg}] \quad (4.6)$$

Antalet steg servot verkligen tar på 180 grader togs fram genom att vrida servot till dess nollposition med kommandot `pwm.setPWM`.

```

1  from Adafruit_PWM_Servo_Driver import PWM
2  import time
3
4  # Initialize the PWM device using the default address
5  pwm = PWM(0x40)
6
7  pwm_freqvens = 50
8  pwm_kanal = 1
9  pwm_start = 0
10 pwm_stop0 = 102
11 pwm_stop1 = 527
12
13 pwm.setPWMFreq(pwm_freqvens)
14
15 time.sleep(2)
16 pwm.setPWM(pwm_kanal, pwm_start, pwm_stop0) #0 grader
17 time.sleep(2)
18 pwm.setPWM(pwm_kanal, pwm_start, pwm_stop1) #180 grader

```

Servots stegintervall går från 102 till 527, alltså 425 steg. Detta ger att varje steg motsvarar

$$\frac{180[\text{grader}]}{425[\text{steg}]} = 0.42 \frac{[\text{grader}]}{[\text{steg}]} \quad (4.7)$$

Antalet mätningar som kan tas längs bollens centrumlinje på 1000 mm avstånd blir

$$\frac{3.72[\text{grader}]}{0.42 \frac{[\text{grader}]}{[\text{steg}]}} = 8.86[\text{steg}] \quad (4.8)$$

Vilket är fler än antalet minimisteg. Servomotorerna klarar alltså kraven.

### 4.4.3 Översättning från pixlar till steg

Med ekvation 4.1, 4.2 samt 4.7 fås dessa samband:

I horisontalld:

$$11.96 \frac{[\text{pixlar}]}{[\text{grad}]} 0.42 \frac{[\text{grader}]}{[\text{steg}]} = 5.02 \frac{[\text{pixlar}]}{[\text{steg}]} \quad (4.9)$$

I vertikalld:

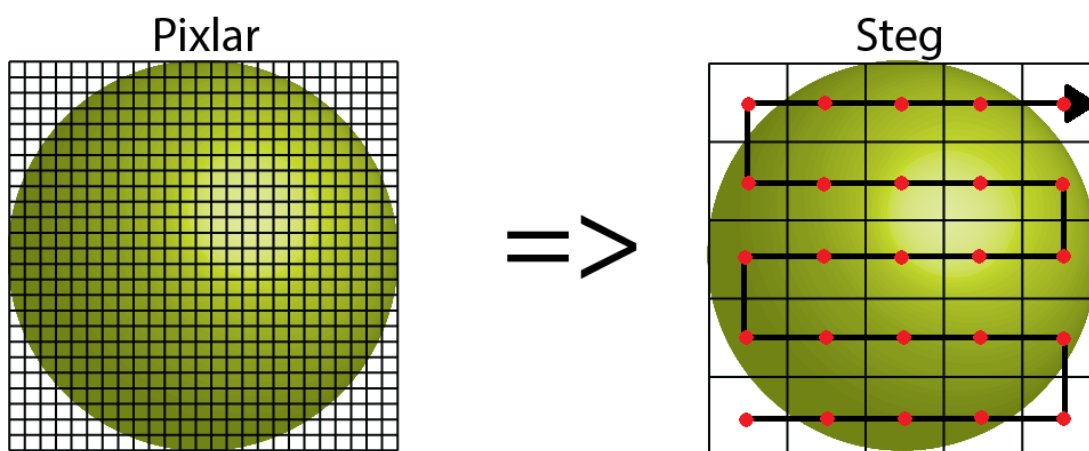
$$11.59 \frac{[\text{pixlar}]}{[\text{grad}]} 0.42 \frac{[\text{grader}]}{[\text{steg}]} = 4.87 \frac{[\text{pixlar}]}{[\text{steg}]} \quad (4.10)$$

Dessa samband används i programmet för att översätta bollens position från kamerans pixlar till servomotorernas steg.

## 4.5 Skapande av avståndsmatrix

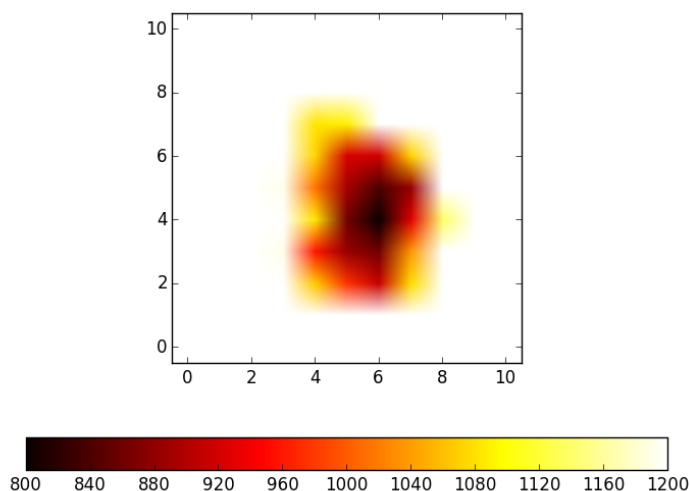
För att måla upp en bild av tennisbollen skapas en avståndsmatrix. I matrisen används avståndsmätningar från IR-sensorn där den mäter avståndet på olika punkter på tennisbollen.

För att veta hur stort område som ska mätas används radien från den cirkel som ringar in objektet på kamerans bild. Radien multipliceras med två, detta mått blir sidolängden hos en kvadrat. Denna kvadrats storlek omvandlas från kamerans pixlar till servomotorernas steg och görs till en matris där varje element är  $1 \times 1$  steg stort.

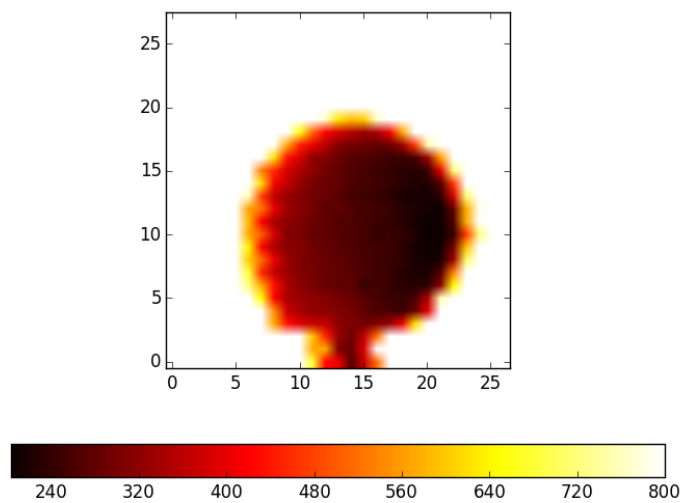


**Figur 4.12:** Översättning från pixlar till steg tillsammans med scanningsmönster

Genom att ta en avståndsmätning i varje element enligt mönstret som presenteras i figur 4.12) fylls avståndsmatrisen med värden. Matrisen plottas och visar en konvex vta som överensstämmer med formen på tennisbollen.



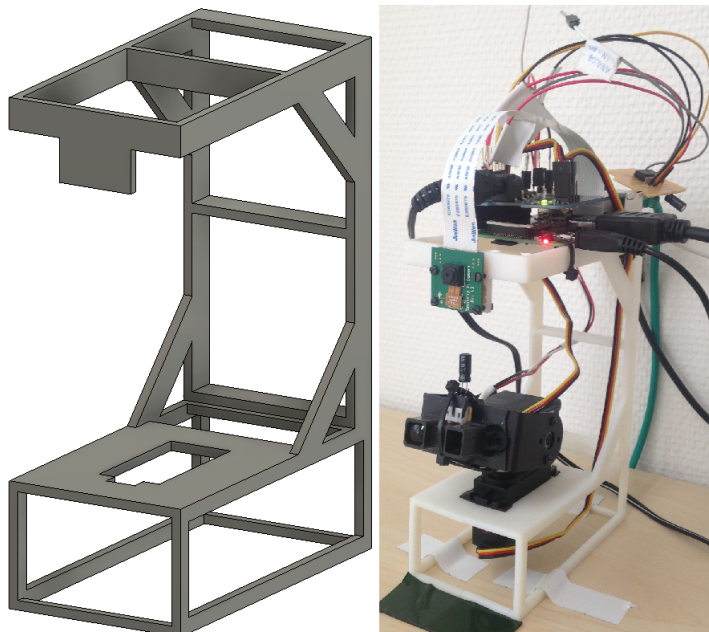
**Figur 4.13:** Scannad tennisboll på 100 cm avstånd



Figur 4.14: Scannad tennisboll på 50 cm avstånd

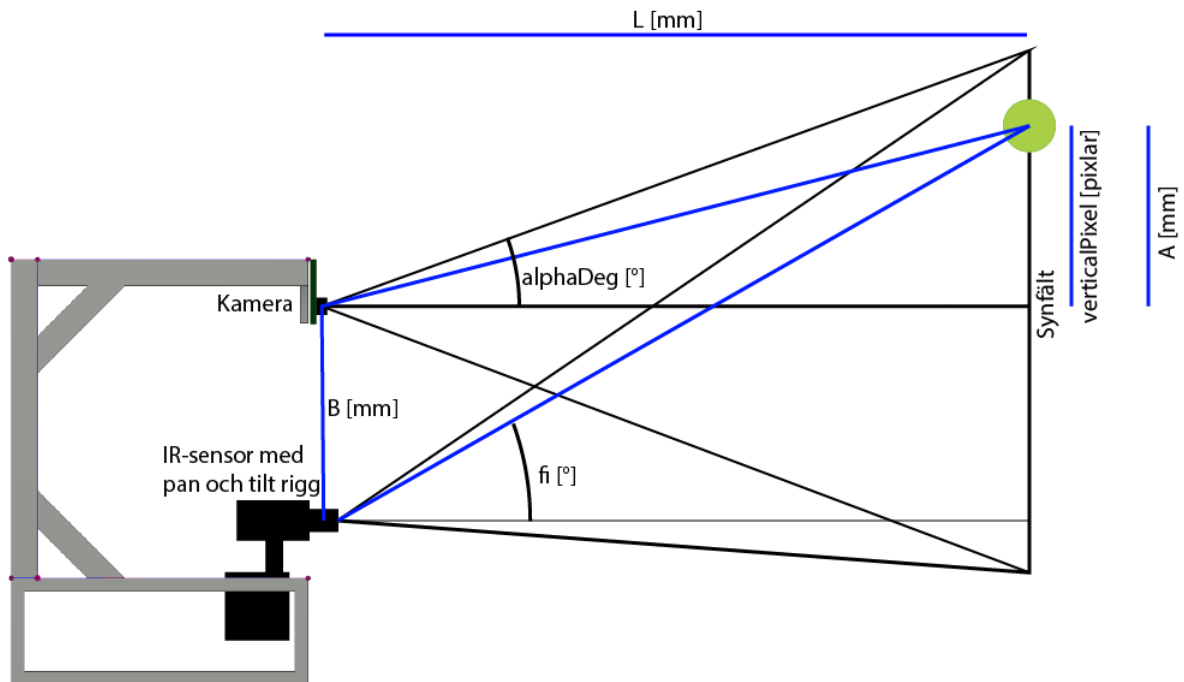
## 4.6 Rigg och offsetberäkningar

För att hålla kamera samt IR-sensor riktade åt samma håll samt ha en stationär utgångspunkt monteras dessa på en rigg. Denna ritas i Autodesk Fusion 360 och 3D-printas sedan ut på Chalmers Lindholmen.



Figur 4.15: Riggen med IR, kamera och RPi

Riggen håller kameran och IR-sensorn på ett avstånd från varandra på 82mm. Denna offset gör att IR-sensorn och kameran kommer se objektet från olika perspektiv. För att kompensera för detta användes trigonometri.



**Figur 4.16:** Riggen med IR och kamera i profil för att visa de överlappande synfälten

Trigonometriuträkningarna tar de vertikala antalet pixlar till objektet sett från kameran och omvandlar denna till antal vertikala steg som servona ska vrida sig för att vara riktad mot objektet.

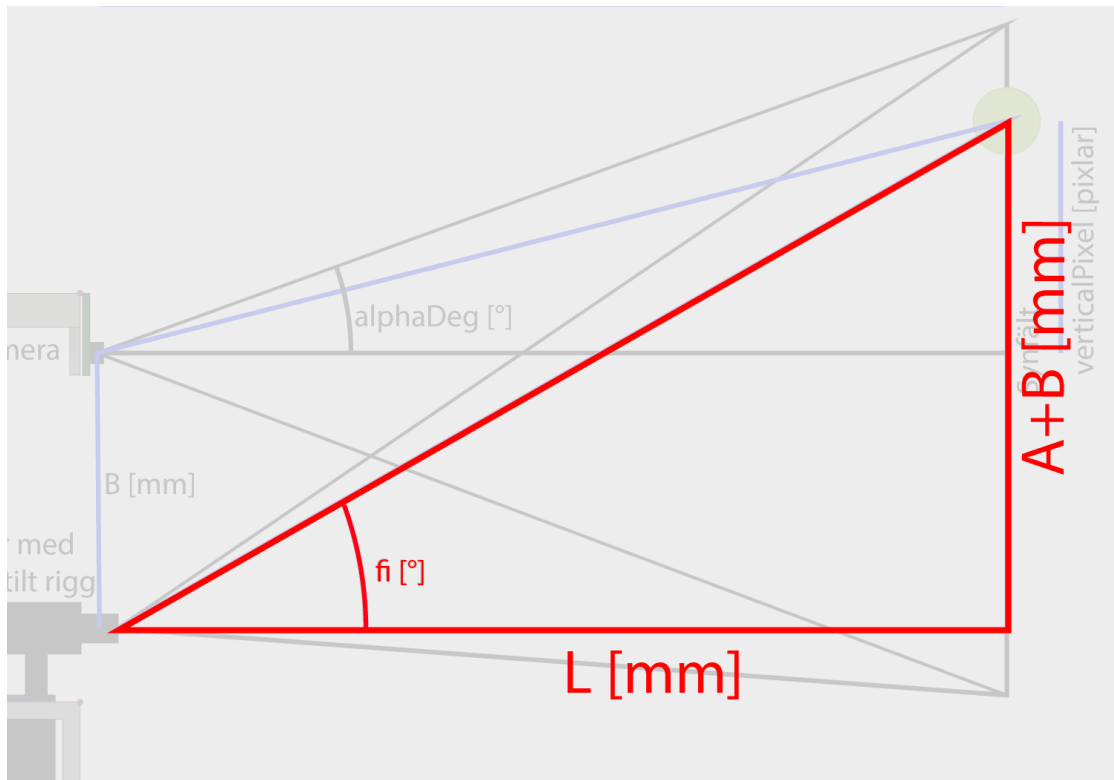
För detta används följande variabler och konstanter.

- $L$ : Längden mellan kamera och objekt.
- $verticalPixel$ : Vertikalt centrum hos objektet sett i kamerans pixlar.
- $alphaDeg$ : Halva kamerans vertikala synvinkel,  $20.7^\circ$
- $alpha$ :  $alphaDeg$  omräknat i radianer.
- $B$ : Avståndet mellan kamera och IR-sensor.
- $A$ : Vertikal offset till objektet sett från kameran.
- $fi$ : Vertikal vinkeln till objektet sett från IR-sensorn.
- $verticalDegreesTriangle$ :  $fi$  omräknat i radianer.
- $verticalTick$ : Hur många vertikala steg pan och tilt riggen ska ta.

En rätvinklig triangel skapas med spetsiga vinkeln  $fi$ . Dess närliggande sida är  $L$ , längden till objektet. Motstående sidan är  $A + B$ , objektets vertikala offset från kameran samt avståndet mellan kamera och IR-sensor.  $A$  är motstående sidan i triangeln som bildas med  $L$ ,  $alpha$  och objektets position i bilden.

$$A = L \tan alpha \frac{verticalPixel - 240}{240} \quad (4.11)$$

$$fi = \tan^{-1} \frac{A + B}{L} \quad (4.12)$$



**Figur 4.17:** Rätvinklig triangel med spets i IR-sensor samt objekt

För att veta hur många steg servot ska vrida sig för att hitta objektet används vinkeln  $f_i$  multiplicerat med servots stegintervall dividerat med servots vinkelintervall. På detta adderas hälften av servots vertikala stegintervall som krävs för att täcka hela kamerans synfält. Detta eftersom servots utgångspunkt är då den tittar i mitten av kamerans synfält, och inte i botten av kamerans synfält.

$$verticalTick = f_i[rad] \frac{180[^\circ]}{\pi[rad]} \frac{servots\ stegintervall[steg]}{servots\ vinkelintervall[^\circ]} + \frac{servots\ vertikala\ stegintervall[^\circ]}{2} \quad (4.13)$$

Servots vinkelintervall är  $180^\circ$ . Servots vertikala stegintervall bestäms utifrån att först rikta servot mot den övre gränsen av kamerans synfält, och sedan mot den nedre gränsen. Detta ger ett intervall på  $verticalTick$  som är 97.77 steg. Detta intervall är samma sak som servots vertikala stegintervall.

När hänsyn tas till att kameran och IR-sensorn sitter på olika positioner går det nu att rikta IR-sensorn till samma koordinater som kameran tittar på och därigenom bättre hitta objektet som ska avståndsbestämmas.

425 steg togs fram som stegintervall från  $0^\circ$  till  $180^\circ$ . Med detta kan ovanstående ekvation skrivas om till:

$$verticalTick = f_i[rad] \frac{425[steg]}{\pi[rad]} + \frac{97.77[steg]}{2} \quad (4.14)$$

Eftersom det inte är någon offset i horisontalled är det lättare att räkna ut antalet steg i den riktningen. För detta krävs några variabler:

- `horizontalTick`: Hur många horisontala steg pan och tilt riggen ska ta.
- `horizontalPixel`: Horisontellt centrum hos objektet sett i kamerans pixlar.

$$\text{horizontalTick} = \text{horizontalPixel}[\text{pixlar}] \frac{53.5[^\circ]425[\text{steg}]}{180[^\circ]640[\text{pixlar}]} \quad (4.15)$$

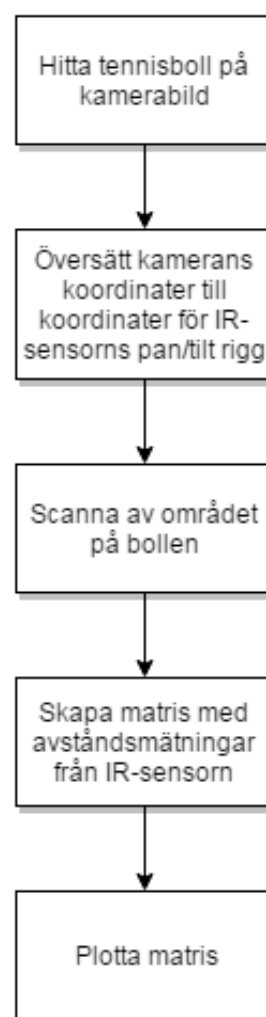
Där 53.5 är det horisontala synfältet i grader och 640 är kamerans horisontala pixlar.

## 4.7 Programmet körs

När programmet körs är det första som händer att kameran initieras och börjar leta efter ett objekt som innehåller en färg inom det specificerade färgintervall. Det färgintervall som används är den gröna färgen av en tennisboll och färger i närheten av det. Anledningen till att det är ett färgintervall och inte endast en färg är eftersom små ändringar i ljus eller läge på bollen lätt kan ändra den uppfattade färgen.

När kameran hittat ett objekt med rätt färg ritas en punkt i centrum av det samt en cirkel runt objektet. Här är cirkelns radie är densamma som objektets. Detta ritas för att tydligt visa vilket objekt som hittats. Efter ett knapptryck från användaren fryser kamerabilden och koordinaterna tas på punkten i mitten av objektet samt radien på cirkeln och skickas vidare till en modul som sköter omvandlingen av kamerans pixelkoordinater till IR-riggens stegkoordinater. En matris ritas upp runt objektets koordinater utifrån radien på cirkeln och scanningen börjar i nedre vänstra hörnet av objektet och likaså matrisen. Efter en kort tid har matriselementet fått ett avståndsvärde från IR-sensorn. Då tas ett steg åt höger och samma procedur upprepas. Efter att matrisens sista element i nedersta raden fyllts tas ett steg upp till nästa rad och proceduren fortsätter, fast denna gång åt vänster.

När hela matrisen fyllts med avståndsvärden plottas matrisen för att visa den fysiska formen av objektet som scannats.



**Figur 4.18:** Programets flödesschema



# 5

## Resultat

Användandet av teknikerna bildbehandling och avståndsmätning med lättillgänglig och konsumentvänlig hård- och mjukvara är gjord. Då sensorerna arbetar tillsammans kan de komplettera varandra och en bättre avläsning av objektet kan göras. Här under besvaras preciseringen av arbetsuppgiften:

### **Vilka sensorer är lämpliga för projektet, samt lättillgängliga och billiga?**

I början av genomförandet görs en urvalsprocess där en IR-sensor och kamera väljs som lämpliga sensorer. Kameran för RPi väljs då den både är billig och lätt att komma igång med. IR-sensorn Sharp GP2Y0A02YK0F är också billig men dock inte lika lätt att komma igång med då den ger ut analoga signaler. Då RPi bara har digitala ingångar får signalen från IR-sensorn gå genom en ADC. Upplösningen för kameran och IR-sensorns räckvidd är tillräcklig då avgränsningen är att bollen ska vara på en meters avstånd.

### **Vad för slags rigg behöver byggas?**

Riggen behöver hålla fast två sensorer så att de inte plötsligt ändrar läge i förhållande till varandra. Att med hjälp av en 3D-skrivare göra en egendesignad rigg och sedan montera fast sensorerna underlättar därför.

### **Vad för precision måste servomotorerna minst ha för att kunna rikta sig tillräckligt precist för att mäta objektet?**

Kravet för servomotorerna är att mäta minst 3 punkter på bollens centrumlinje. Detta ger en precision på minst  $1.24 \frac{[^\circ]}{[steg]}$ . Då servomotorerna har steg som motsvarar  $0.42 \frac{[^\circ]}{[steg]}$  motsvarar detta 8.86 steg längs med bollens centrumlinje vilket är betydligt bättre än minimikravet. Detta gäller då objektet är på en meters avstånd.

### **Hur ska de olika delarna arbeta tillsammans?**

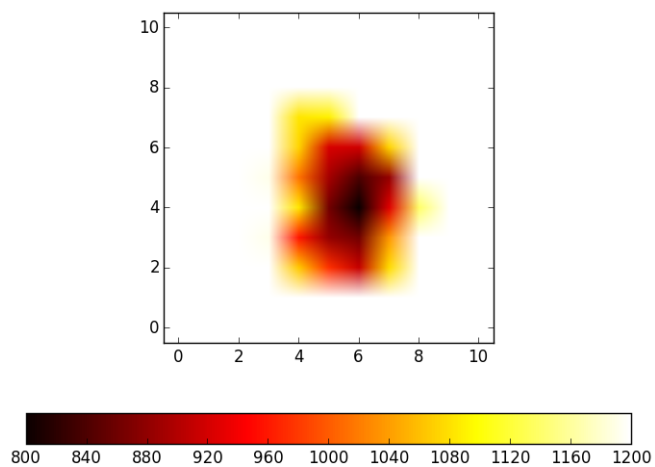
Delarna arbetar tillsammans genom huvudprogrammet. Detta tar en position på objektet från kameran och bestämmer utifrån det hur pan och tilt riggen ska stegas. Mellan varje steg tas avståndsmätningar från IR-sensorn och läggs i en matris.

### **Är detta ett lämpligt tillvägagångssätt för att få ut mer data?**

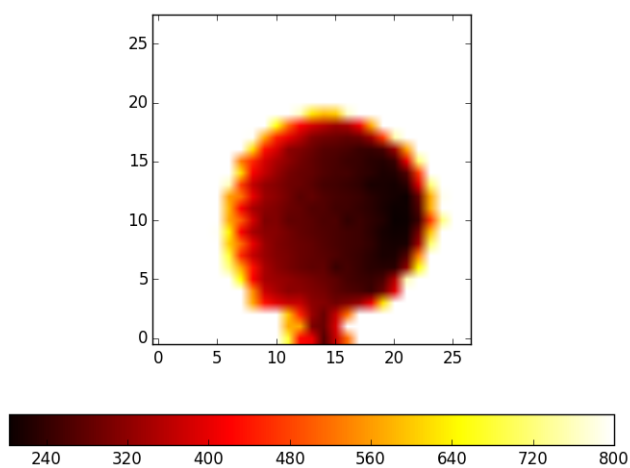
Ja. Att hitta ett objekt med kamera och sedan använda en sensor för att mäta avstånd och därmed ge en till dimension på objektet är ett bra sätt att få mer data.

### **Hur ska det avlästa objektet presenteras?**

IR-sensorn läser av objektet och placerar avståndsmätningarna i en matris. Matrisen presenteras därefter genom att plottas upp som en typ av 3D-bild där mätningarna visas i olika färger beroende på mätningens värde.



**Figur 5.1:** 3D bild på 100cm avstånd



**Figur 5.2:** 3D bild på 50cm avstånd

Avläsningarna gjordes i en kontrollerad miljö vilket gjorde det lättare att ha ett bra ljus och lagom med reflektion på bollen. Detta underlättade omständigheterna både för IR-sensorn och kameran då de kan vara känsliga mot detta.

# 6

## Diskussion

Vi kunde ha gjort en mer ordentlig analys av vilken sensor vi skulle använda redan från början. Innan projektet började, samt den första veckan var vi inställda på att använda en ultraljudssensor. Den borde vi släppt tidigare och insett att det finns bättre alternativ.

Planeringen vi gjorde följdes ungefär en vecka. Sedan hade arbetet gått så fort att vi var för långt fram i schemat. Att lära känna kameran och sensorerna, att bygga rigg och att kunna plotta de två olika bilderna har gått mycket fortare än vi tänkte på grund av att mycket av det vi ville göra redan var gjort och skrivet till guider med den kod som behövdes. Som med annan programmering har det gått bra att googla på lösningar till de problem som uppstått.

De bilder vi kunnat plotta med IR-sensorn kan ge en uppfattning om objektets fysiska form, men mätningarna skiljer sig en del från varandra även om sensorn tittar på samma punkt. Detta ger en osäkerhet i mätningarna, men den är försumbart liten i jämförelse med hur mycket det skiljer sig i avstånd beroende på var på bollen IR-sensorn tittar. Som exempel i figur 5.1 under resultat visas en bild på bollen på 100 cm avstånd. De mätningar som är i mitten av bilden visar ett avstånd på 800 mm, medan de mätningar som görs i utkanten av bollen visar ett avstånd på ungefär 1120 mm. Med dessa mätningar ser det ut som att bollen har en radie på 320 mm, och inte 32.5 mm som den egentligen har. Eftersom vi är mer ute efter objektets form, och inte koncentrerat oss på objektets mått anser vi att trots detta kunna få nödvändig information från mätningarna. Anledningen till dessa konstiga avstånd kan ha att göra med bollens luddiga yta. Eftersom IR-sensorn mäter avståndet med reflekterat ljus spelar det in i mätvärdet vilket material man mäter på.

Vi kan se i figur 5.2 i resultat att den högra sidan av tennisbollen är mörkare än den vänstra sidan. Orsaken till detta är okänd, men vi misstänker att det antingen har att göra med störande ljus eller hur IR-sensorn ser ut fysiskt. Den har en lampa på sin vänstra sida och mottagare på sin högra sida vilket gör att ljuset får olika infallsvinklar på de olika sidorna av bollen. Detta fenomen märks inte i vertikalled, vilket kan stödja teorin om osymmetrin hos sensorn.

IR-sensorn ser med en ljuskon, om vi hade kunnat göra den konen mindre, eller göra den till en rak stråle borde det gå att ta mer exakta mätningar.

Vi har haft problem med att ta reda på hur många steg servomotorerna ska kunna

ta. Efter att ha mailat tillverkarna fick vi svar att de ska kunna ta 512 steg på 180 grader, men med vårt program har vi endast lyckats ta ca 425 steg på 180 grader. Utifrån databladet för servona räknade vi ut med hjälp av dödbandbredden vilket är den tid som styrsignalen till servot ska behöva ändras förrän servot byter position. Denna ska vara på 8 microsekunder för våra servos, och styrsignalen ska variera mellan 0.9 till 2.1 millisekunder. Detta leder till  $2100/8 = 262.5$  steg, vilket ser fint ut i teorin, men inte stämmer i praktiken.

För att scanna objekt som rör på sig hade man kunnat låta kameran köra hela tiden och inte göra som det är i nuläget att kameran fryser då scanning pågår. Detta skulle möjliggöra att man kan spåra objektet och kompensera för dess förflyttning så man fortfarande kan scanna i ett bestämt mönster.

För att möjliggöra snabbare scanning av objektet skulle man kunna ha en array av sensorer då detta tillåter att ta flera avståndsmätningar samtidigt eller i kort följd efter varandra. En dyrare sensor skulle även kunna ha högre mätfrekvens vilket skulle underlätta för en snabbare scanning.

För att förbättra scanningen med den nuvarande IR-sensorn skulle man kunna justera matningsspänningen till ADC för att på så sätt minska det avlästa avståndsintervall. Man skulle även kunna kontrollera att både IR-sensor och ADC matas med konstant spänning för att undvika störningar. Den karakteristiska stora horisontella avståndsskillnaden på tennisbollen skulle kunna elimineras genom att vrida IR-sensorn 90 grader efter första scanningen för att få fler perspektiv på den. Genom att jämföra de två avståndsbilderna skulle man kunna filtrera bort osanna, eller åtminstone jämna ut osann mätdata och på så sätt få en mer korrekt bild.

# Litteraturförteckning

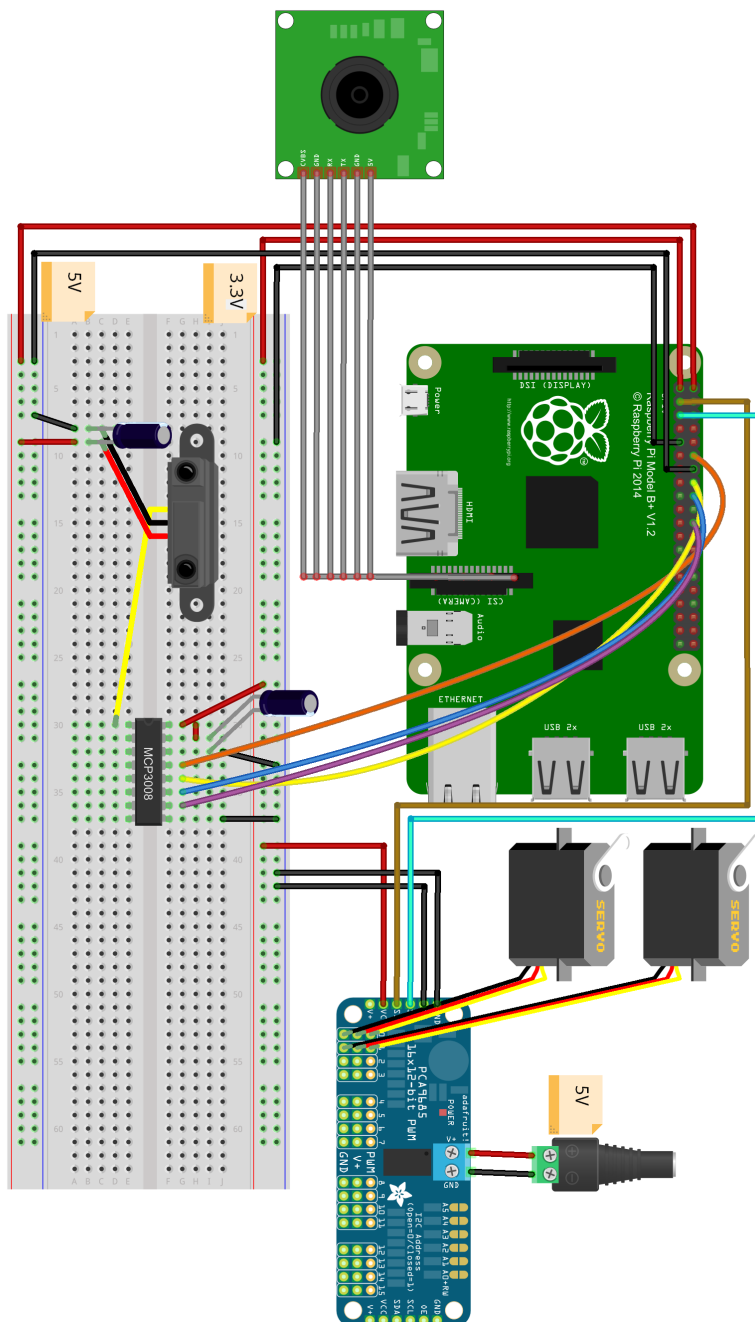
- [1] **SPI:**  
[https://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface\\_Bus](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)
- [2] **Raspberry Pi 3 Modell B:**  
<https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>
- [3] **Arduino Due:**  
<https://www.arduino.cc/en/Main/ArduinoBoardDue>
- [4] **Pi Camera:**  
<https://www.raspberrypi.org/documentation/hardware/camera.md>
- [5] **Pixy CMUcam5:**  
<http://www.cmucam.org/projects/cmucam5>
- [6] **IR-sensor Sharp GP2Y0A02YK0F:**  
<https://www.sparkfun.com/products/8958>
- [7] **HRLV MaxSonar - EZ0:**  
[http://www.maxbotix.com/Ultrasonic\\_Sensors/MB1003.htm](http://www.maxbotix.com/Ultrasonic_Sensors/MB1003.htm)
- [8] **LIDAR lite v2:**  
<http://pulsedlight3d.com/>
- [9] **Radar:**  
<https://en.wikipedia.org/wiki/Radar>
- [10] **Lynxmotion Pan and Tilt kit:**  
<http://www.robotshop.com/en/lynxmotion-pan-and-tilt-kit-aluminium2.html>
- [11] **Adafruit PWM HAT:**  
<http://www.kjell.com/se/sortiment/dator-natverk/enkortsdator/raspberry-pi/adafruit-servokontroller-hat-for-raspberry-pi-p87991#ProductDetailedInformation>
- [12] **MCP3008 ADC:**  
<https://www.adafruit.com/datasheets/MCP3008.pdf>
- [13] **Python:**  
[https://en.wikipedia.org/wiki/Python\\_%28programming\\_language%29](https://en.wikipedia.org/wiki/Python_%28programming_language%29)
- [14] **OpenCV:**  
<https://en.wikipedia.org/wiki/OpenCV>
- [15] **cv2.GaussianBlur:**  
[http://docs.opencv.org/3.1.0/d4/d13/tutorial\\_py\\_filtering.html#gsc.tab=0](http://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html#gsc.tab=0)
- [16] **cv2.cvtColor:**  
[http://docs.opencv.org/3.1.0/df/d9d/tutorial\\_py\\_colorspaces.html#gsc.tab=0](http://docs.opencv.org/3.1.0/df/d9d/tutorial_py_colorspaces.html#gsc.tab=0)
- [17] **cv2.inRange:**  
[http://docs.opencv.org/2.4/modules/core/doc/operations\\_on\\_arrays.html](http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html)

- [18] **cv2.erode och cv2.dilate:**  
[http://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_morphological\\_ops/py\\_morphological\\_ops.html](http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html)
- [19] **cv2.findContours, cv2.contourArea, cv2.minEnclosingCircle, cv2.moments:**  
[http://docs.opencv.org/3.1.0/d4/d73/tutorial\\_py\\_contours\\_begin.html#gsc.tab=0](http://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html#gsc.tab=0)
- [20] **cv2.circle:**  
[http://docs.opencv.org/2.4/modules/core/doc/drawing\\_functions.html](http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html)
- [21] **BGR till HSV konverterare:**  
<http://www.rapidtables.com/convert/color/rgb-to-hsv.htm>
- [22] **Servosteg:**  
<http://www.robotshop.com/media/files/pdf/servomanual-31422s.pdf>  
<http://www.robotshop.com/media/files/pdf/hs422-31422s.pdf>
- [23] **Adafruit Github:**  
<https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git>
- [24] **Jeremy Blythe IR-avstånd:**  
<http://jeremyblythe.blogspot.se/2012/09/raspberry-pi-distance-measuring-sensor.html>
- [25] **Adrian Rosebrock RPi kamera med OpenCV:**  
<http://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>
- [26] **Adrian Rosebrock Objektigenkänning:**  
<http://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/>



# A Bilagor

## A.1 Kopplingschema



fritzing



## A.2 Programkod

Utöver koden som finns här används modulen PWM i filen Adafruit\_PWM\_Servo\_Driver. Denna går att ladda ned från Adafruits github sida[23].

Koden nedan bygger delvis på färdigskrivna kod som sedan skrivits om för att passa syftet med projektet.

För läsning av avstånd från IR-sensorn har denna kod använts[24].

För att börja använda RPi kameran med OpenCV har denna kod använts[25].

För att spåra ett objekt har denna kod använts[26].

### A.2.1 Definitions\_file.py

```

1 import math
2 from Adafruit_PWM_Servo_Driver import PWM
3 import time
4 import RPi.GPIO as GPIO
5
6 def offset_calculation(verticalPixel, horizontalPixel):
7
8     #Length to object
9     L = 1000.0
10    #Half the vertical view of the camera in degrees
11    alphaDeg = 20.7
12    #in radians
13    alpha = alphaDeg*(math.pi/180)
14    #Distance between the IR sensor and the camera
15    B = 82.0
16    #Vertical offset to the object from the camera
17    A = L * math.tan(alpha) * ((verticalPixel-240)/240.0)
18
19    #Vertical angle from zero angle of IR sensor to where object is.
20    fi = math.atan((A+B)/L)
21    verticalDegreesTriangle = fi * 180/math.pi
22
23    #Upper triangle
24    if verticalPixel >= 240:
25        #From zero angle as starting position for the IR
26        #Position of the object seen from the IR
27        verticalTick = int(round(verticalDegreesTriangle * 425/180 + 97.77/2))
28        #97.77 is the vertical tick interval, this divided by 2 is added because
29        #we want to start counting from the middle of the interval.
30
31    #Lower triangle
32    if verticalPixel < 240:
33        #From zero angle as starting position for the IR

```

```
34     #Position of the object seen from the IR
35     verticalTick = int(round(verticalDegreesTriangle * 425/180 + 97.77/2))
36
37     #Position of the object seen from the IR
38     horizontalTick = int(round(horizontalPixel*(425*53.5/180)/640))
39     #425 ticks corresponds to 180 degrees. 53.5 degrees horizontal
40     #view over 640 pixels gives a tick interval of 126.
41
42     return (verticalTick, horizontalTick)
43
44
45 def ir_scan(radius, ball_pos_ir_y, ball_pos_ir_x):
46
47     # Initialize the PWM device using the default address
48     pwm = PWM(0x40)
49
50     #Ignore distance readings that is less or more then the limits
51     max_distance = 1400
52     min_distance = 800
53
54     # Translate camera position to IR and create the area that will be scanned
55     ticksminx=231.84
56     ticksminy=271.1
57     radius=radius+5
58     xmin = int( ball_pos_ir_x+(-radius)*126.32/640 +ticksminx)
59     xmax = int( ball_pos_ir_x+(radius)*126.32/640 +ticksminx)
60     ymin = int( ball_pos_ir_y+(-radius)*97.8/480 +ticksminy)
61     ymax = int( ball_pos_ir_y+(radius)*97.8/480 +ticksminy)
62
63     # Scanning begin in the upper left corner
64     xir = xmin
65     yir = ymin
66     z = 1
67
68     # Create a matrix containing the position and distance
69     #data from the IR sensor
70     # Creates a list containing 5 lists initialized to 0
71     Matrix = [[0 for x in range(xmax-xmin+1)] for x in range(ymax-ymin+1)]
72
73     while yir <= ymax:
74
75         pwm.setPWM(0, 0, yir) #Vertical
76         pwm.setPWM(1, 0, xir) #Horizontal
77         time.sleep(0.2)
78
79         distance_reading = distance_from_ir()
```

```

80
81     if distance_reading > max_distance:
82         distance_reading = max_distance
83     if distance_reading < min_distance:
84         distance_reading = min_distance
85
86     Matrix[yir-ymin][xir-xmin] = distance_reading
87     print "x,y,distance: {}, {}, {}mm. distance from ir:{}".format(xir-xmin,
88     yir-ymin,distance_reading,distance_from_ir())
89
90     xir = xir + z
91
92     if xir > xmax or xir < xmin: # At end of horizontal scan line
93         yir = yir + 1           # take a step upward
94         z = - z                 # Change scan direction
95         xir = xir + z
96
97     # standard direction
98     time.sleep(0.2)
99     pwm.setPWM(0, 0, 320) #Vertical
100    pwm.setPWM(1, 0, 295) #Horizontal
101
102    return Matrix
103
104
105    # read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
106    def readadc(adcnum, clockpin, mosipin, misopin, cspin):
107        if ((adcnum > 7) or (adcnum < 0)):
108            return -1
109        GPIO.output(cspin, True)
110
111        GPIO.output(clockpin, False) # start clock low
112        GPIO.output(cspin, False)    # bring CS low
113
114        commandout = adcnum
115        commandout |= 0x18 # start bit + single-ended bit
116        commandout <= 3   # we only need to send 5 bits here
117        for i in range(5):
118            if (commandout & 0x80):
119                GPIO.output(mosipin, True)
120            else:
121                GPIO.output(mosipin, False)
122            commandout <= 1
123            GPIO.output(clockpin, True)
124            GPIO.output(clockpin, False)
125

```

```
126     adcout = 0
127     # read in one empty bit, one null bit and 10 ADC bits
128     for i in range(12):
129         GPIO.output(clockpin, True)
130         GPIO.output(clockpin, False)
131         adcout <<= 1
132         if (GPIO.input(misopin)):
133             adcout |= 0x1
134
135     GPIO.output(cspin, True)
136
137     adcout >>= 1      # first bit is 'null' so drop it
138     return adcout
139
140
141 def distance_from_ir():
142
143     # change these as desired - they're the pins connected from the
144     # SPI port on the ADC to the Cobbler
145     SPICLK = 18
146     SPIMISO = 23
147     SPIMOSI = 24
148     SPICS = 25
149
150     adc = 0
151
152     # read the analog pin
153     ir_value = readadc(adc, SPICLK, SPIMOSI, SPIMISO, SPICS)
154
155     # take ten readings and find the average (a) - this smooths
156     #things out a little.
157     r=[]
158     for i in range (0,5):
159         r.append(ir_value)
160         a = sum(r)/float(5)
161
162         #waiting for the new ir-sensor value (50Hz in time, 1/50=0.02)
163         time.sleep(0.02)
164
165     # convert analog pin to voltage (v)
166     v = (ir_value/1023.0)*3.3
167
168     # use the magic formula to get the distance (d)
169     distance = 16.2537 * v**4 - 129.893 * v**3 + 382.268 * v**2
170     - 512.611 * v + 306.439
171
```

```

172     # distance from cm to mm
173     distance = distance*10/2
174     distance = int(distance)
175
176     return distance

```

## A.2.2 MAIN\_file.py

```

1  # import the necessary packages
2  from picamera.array import PiRGBArray
3  from picamera import PiCamera
4  import time
5  import cv2
6  import RPi.GPIO as GPIO
7  import matplotlib.pyplot as plt
8  from Adafruit_PWM_Servo_Driver import PWM
9  from Definitions_file import offset_calculation
10 from Definitions_file import ir_scan
11
12 #-----
13 #INITIALIZE:
14
15 GPIO.setmode(GPIO.BCM)
16
17 # change these as desired - they're the pins connected from the
18 # SPI port on the ADC to the Cobbler
19 SPICLK = 18
20 SPIMISO = 23
21 SPIMOSI = 24
22 SPICS = 25
23
24 # set up the SPI interface pins
25 GPIO.setup(SPIMOSI, GPIO.OUT)
26 GPIO.setup(SPIMISO, GPIO.IN)
27 GPIO.setup(SPICLK, GPIO.OUT)
28 GPIO.setup(SPICS, GPIO.OUT)
29
30 # Initialize the PWM device using the default address
31 pwm = PWM(0x40)
32 # Set frequency to 50 Hz
33 pwm.setPWMFreq(50)
34
35 # define the lower and upper boundaries of the "green" in BGR
36 # converts later to the HSV color space
37 greenLower = (29, 100, 40)
38 greenUpper = (64, 255, 240)

```

```
39
40 # initialize the camera and grab a reference to the raw camera capture
41 camera = PiCamera()
42 camera.resolution = (640, 480)
43 camera.framerate = 32
44 rawCapture = PiRGBArray(camera, size=(640, 480))
45
46 # allow the camera to warmup
47 time.sleep(0.1)
48
49 #-----
50 #MAIN:
51
52 # capture frames from the camera
53 for frame in camera.capture_continuous(rawCapture, format="bgr",
54 use_video_port=True):
55     # grab the raw NumPy array representing the image,
56     #then initialize the timestamp and occupied/unoccupied text
57     frame = frame.array
58
59     # blur the frame to reduce high frequency noise
60     frame_blur = cv2.GaussianBlur(frame, (11, 11), 0)
61
62     # convert the blurred frame from BGR to HSV color space
63     hsv = cv2.cvtColor(frame_blur, cv2.COLOR_BGR2HSV)
64
65     # construct a black-and-white mask for the color "green",
66     # then perform a series of dilations and erosions to
67     # remove any small blobs left in the mask
68     mask = cv2.inRange(hsv, greenLower, greenUpper)
69     mask = cv2.erode(mask, None, iterations=2)
70     mask = cv2.dilate(mask, None, iterations=2)
71
72     # Bitwise-AND mask and original image
73     res = cv2.bitwise_and(frame,frame, mask= mask)
74
75     # find contours in the mask and initialize the current
76     # (x, y) center of the ball
77     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
78 cv2.CHAIN_APPROX_SIMPLE)[-2]
79     center = None
80
81     # only proceed if at least one contour was found
82     if len(cnts) > 0:
83         # find the largest contour in the mask, then use
84         # it to compute the minimum enclosing circle and centroid
```

```

85     c = max(cnts, key=cv2.contourArea)
86     ((xk, yk), radius) = cv2.minEnclosingCircle(c)
87     M = cv2.moments(c)
88     center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
89
90     ball_pos_y = 480 - yk
91     ball_pos_x = xk
92
93     # only proceed if the radius meets a minimum size
94     if radius > 10:
95         # draw the circle and centroid on the frame,
96         cv2.circle(frame, (int(xk), int(yk)), int(radius), (0, 0, 255), 2)
97         cv2.circle(frame, center, 5, (0, 0, 255), -1)
98         ball_pos_ir_y = offset_calculation(ball_pos_y, ball_pos_x)[0]
99         ball_pos_ir_x = offset_calculation(ball_pos_y, ball_pos_x)[1]
100        print "center, radius: {}, {}, {}, IR position: {}, {}".format(
101            int(ball_pos_x), int(ball_pos_y), int(radius),
102            int(ball_pos_ir_x), int(ball_pos_ir_y) )
103
104    # show the frames
105    camera.hflip=True
106    camera.vflip=True
107    cv2.imshow("Frame", frame)
108    cv2.imshow("blur", frame_blur)
109    cv2.imshow("mask", mask)
110    cv2.imshow("res", res)
111    key = cv2.waitKey(1) & 0xFF
112
113    # scanning and plot
114    if key == ord("a"):
115        Matrix = ir_scan(radius, ball_pos_ir_y, ball_pos_ir_x)
116
117        print "matix {}".format(Matrix)
118
119        # plot matrix
120        im = plt.imshow(Matrix, origin='lower', cmap = 'hot')
121        plt.colorbar(im, orientation = 'horizontal')
122        plt.show()
123
124    # clear the stream in preparation for the next frame
125    rawCapture.truncate(0)
126
127    # if the 'q' key was pressed, break from the loop
128    if key == ord("q"):
129        break

```