ISSN 0280-5316
ISRN LUTFD2/TFRT--5828--SE

Objektföljning med mobil robot baserad på inbyggt Linux

Peter Jönsson

Institutionen för Reglerteknik
Lunds Universitet
December 2008

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2008

Document Number
ISRN LUTFD2/TFRT--5828--SE

Author(s)
Peter Jönsson

Supervisor
Karl Mårtensson Automatic Control
Pablo Cases and Ola Palm ÅF Combra, Lund
Karl-Erik Årzén Automatic Control (Examiner)

Sponsoring organization

Title and subtitle
Objektföljning med mobil robot baserad på inbyggt Linux(Object Tracking with a Mobile Robot using Embedded Linux)

Abstract
This thesis describes the development of a object tracking mobile robot. The robot is built on an Olympus IV development board, with an Linux operating system. The camera, used to track a moving object, is an Axis 215 PTZ with a built-in motor. The suitability of the Olympus IV development board and Linux operating system in this context are evaluated. The thesis describes the computer vision and the control system developed for the object tracking. Tracking of the object is done by both rotating the camera and moving the mobile platform.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
Swedish

Number of pages
46

Recipient's notes

Security classification

Förord

Jag skulle vilja börja med att tacka ÅF-Combra för möjligheten att utföra mitt examensarbete hos er. Vill också tacka Pablo Cases, Ola Palm och Karl Mårtensson för er kontinuerliga handledning under hela projektet. Axis Communications för lånet av kamera. Petter Cederholm för hjälp med strömförsörjning. Tobias Hultén och Joakim Remne för bra tidigare arbete samt vägledning under första delen av projektet. Karl Åström för vägledning med bildanalysen.

Lund, Oktober 2008
Peter Jönsson

Innehåll

Kapitel 1

Bakgrund

1.1 Introduktion

Intelligenta robotar med inbyggda kameror är idag ett stort och aktivt forskningsområde. En av anledningar är att framsteg inom bildanalys och elektriska komponenter har gjort design av små, billiga konstruktioner möjlig. Applikationer som tidigare behövde skrymmande beräknings- och styrdatorer kan nu exekveras direkt i det inbyggda systemet. För att inte behöva implementera ett specifikt operativsystem i dessa inbyggda system, används ofta det fria operativsystemet Linux. Eftersom Linux bygger på öppen källkod är det lätt att anpassa och lägga till funktionalitet för att möjliggöra den specifika applikationen.

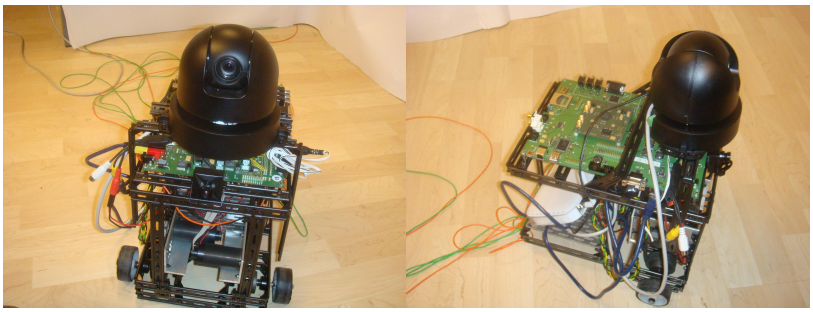
Denna rapport beskriver utvecklingen av de objektföljande delarna till en mobil robot baserad på Linux. Arbetet är en fortsättning på tidigare examensarbete där en fjärrstyrd mobil robot konstruerats [18]. Rapporten behandlar framtagning av en lämplig kamera, utvecklingen av bildanalytiska algoritmer samt utvecklingen av kontrollsystemet för objektföljningen. Arbetet innefattar också en utvärdering av utvecklingskortet Olympus IV samt lämpligheten att använda Linux för en objektföljande robot.

Målet med examensarbetet har varit att utveckla en mobil robot som med hjälp av en kamera kan identifiera och följa ett fördefinierat objekt. En bild på roboten i slutskedet av detta examensarbete kan ses i figur 1.1. Kretskortet är av modellen Olympus IV och är ett utvecklingskort från företaget Logipard AB [16]. Två elmotorer av märket Faulhaber driver roboten via två motorstyrningskretsar som kopplas till en seriell port på kretskortet. Plattformen har en http-server och kan fjärrstyras av en användare från en webbläsare. I webbläsaren visas också en bild från en kamera som är kopplad till en videoingång på kretskortet.

1.2 Syfte

Syftet med examensarbetet var att vidareutveckla den mobila roboten så att den med hjälp av en kamera kan identifiera och följa ett föremål. Under arbetets gång skulle följande två frågeställningar besvaras:

- Är utvecklingskortet Olympus IV och kretsen Argus IV lämpligt för bildanalys/reglersystem?



Figur 1.1: Roboten framifrån och från sidan i slutet av detta examensarbetet

- Är Linux ett lämpligt operativsystem i en mobil robot?

1.3 Delmål

Projektet innehöll fyra delmål.

- Ta fram en konstruktion för en rörlig kamera som monteras på den befintliga plattformen.
- Implementera bildanalytiska algoritmer för att identifiera ett fördefinierat objekt mot en homogen bakgrund.
- Reglera kamerarörelsen för att följa ett rörligt objekt med plattformen i stationärt läge.
- Reglera kamera och plattform för att följa ett rörligt objekt på ett fördefinierat avstånd.

1.4 Begränsningar

För att klara av alla delmål inom tidsramen för projektet infördes några begränsningar. Eftersom tyngdpunkten av examensarbetet var reglerteknik och inte främst bildanalys sattes de största begränsningarna på den senare.

- Objektet valdes till att vara en blå boll med diameter 4 cm.
- Den blåa bollen skulle identifieras mot en vit och ljus bakgrund.
- Objektföljningen skulle endast ske i bildplanet och inte i rumskoordinater.
- Avståndet till bollen skulle vara mindre än 3 m.

1.5 Översikt

Kapitel 2 ger en överblick av robotens uppbyggnad samt en beskrivning av dess hårdvara.

Kapitel 3 innehåller några grundläggande definitioner som används i rapporten.

Kapitel 4 beskriver några tidigare arbeten på området.

Kapitel 5 ger en beskrivning av bildanalysen.

Kapitel 6 och 7 innehåller beskrivning av regleringen för kameran och roboten.

Kapitel 8 ger en översikt av de implementerade programmen.

Kapitel 9 innehåller en avslutande diskussion och förslag på vidareutveckling.

Kapitel 2

Hårdvara och program

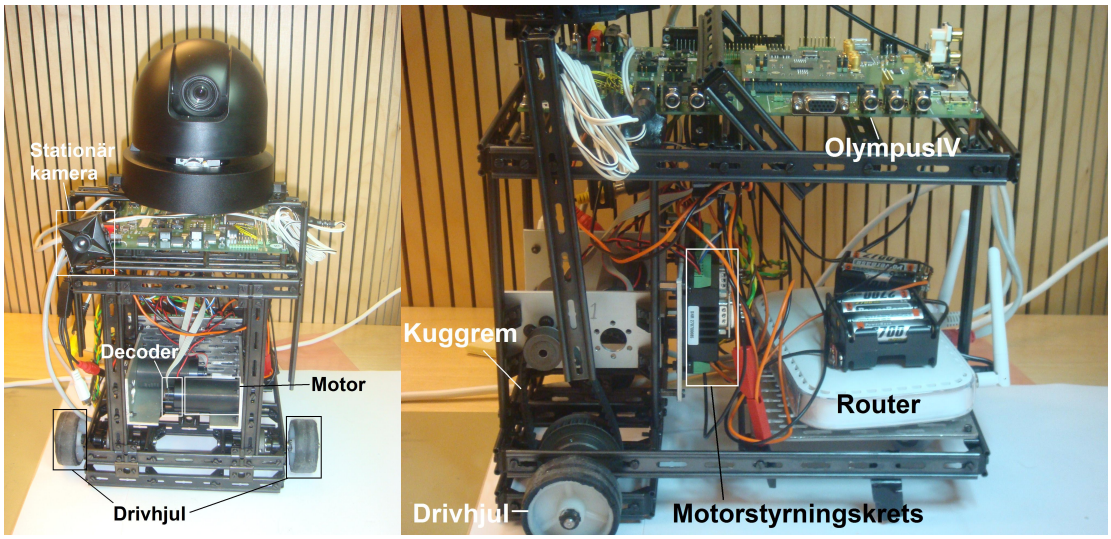
2.1 Robot

Roboten är uppbyggd av universalbyggsatsen FAC [22] och kan ses i figur 2.1. Fram på roboten sitter två hjul som driver roboten. Roboten styrs genom att rotera hjulen med olika hastigheter. Exempelvis för att svänga roboten åt vänster roteras höger hjul snabbare än vänster hjul. För att rotera roboten roteras hjulen med samma hastighet men i motsatt riktning. Bak på roboten sitter ett länkhjul utan drivning. Mer om robotens konstruktion kan läsas i [18]. Kommunikationen med roboten sker via en trådlös router, se figur 2.2. Roboten har en http-server och användaren kan styra roboten från en vanlig dator. Routern sköter också kommunikationen mellan Olympus IV och kameran.

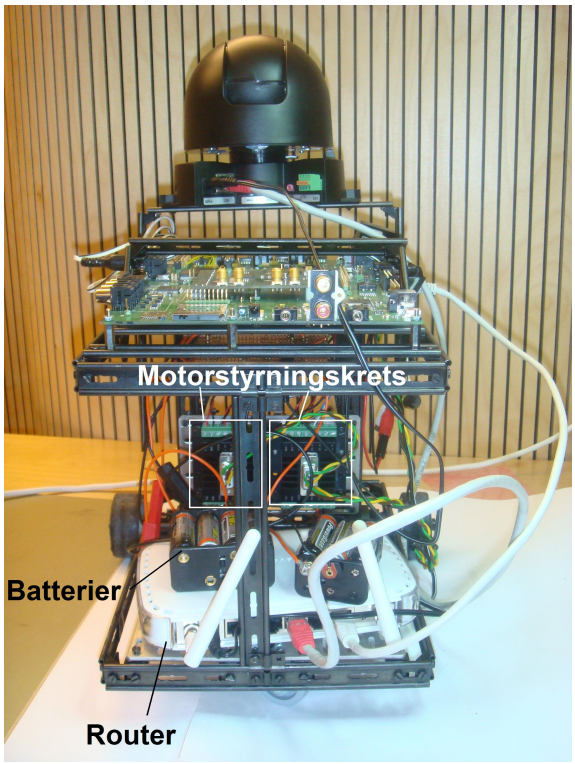
Strömmen till motorer, utvecklingskort, kamera och router tas från olika spänningsomvandlare som är placerade på roboten. Motorerna, kameran och routern går på 12V medan utvecklingskortet körs på 10V. Det finns fyra olika spänningsomvandlare, två används av kameran och routern. De båda motorerna använder en och utvecklingskortet har en egen 10V spänningsomvandlare. Strömkällan består av 14 stycken uppladdningsbara seriekopplade 1,2 volts, 2700 mAh NiMh batterier. När både kameran och plattformen står stilla använder plattformen cirka 1.5A ström. När både kameran och motorerna körs används cirka 2.5A ström.

2.1.1 Drivlina

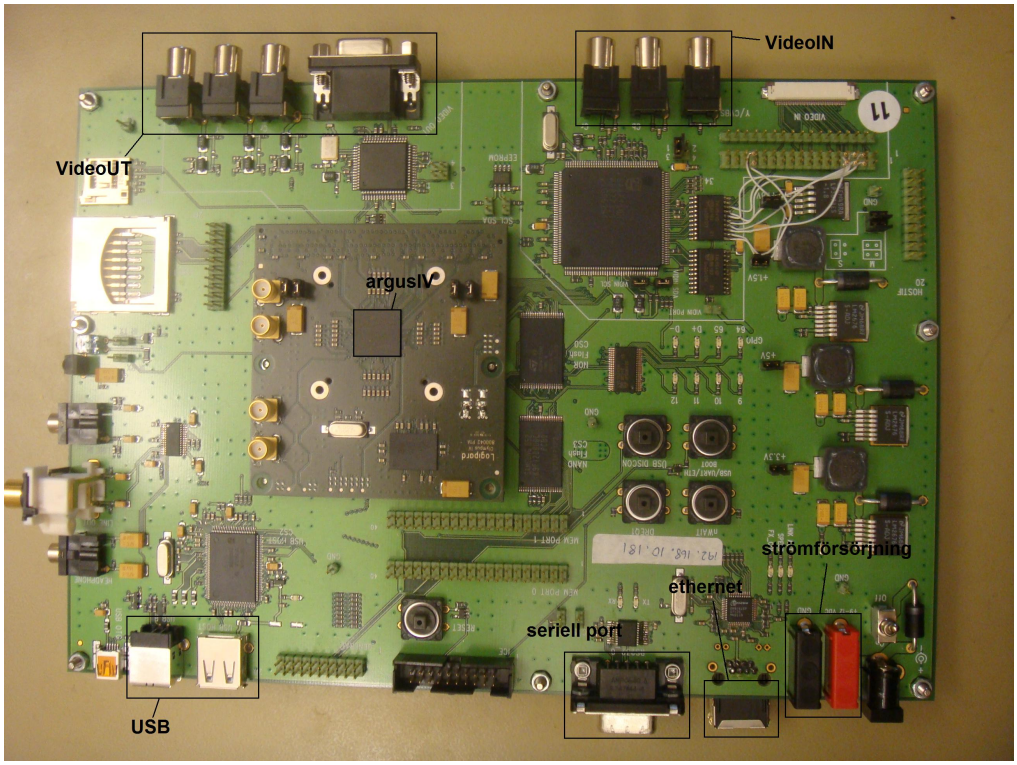
Roboten drivs med hjälp av två stycken Faulhaber 3257G0112CR motorer med tillhörande IE-512 encoder, se figur 1.1. Som motorstyrningskrets används Faulhabers egna MCDC 3006S som styrs gemensamt via en seriell port [20], se figur 2.2.



Figur 2.1: Roboten framifrån och från sidan



Figur 2.2: Roboten bakifrån, bland annat ses motorstyrningskretsar, router samt batterier



Figur 2.3: Utvecklingskortet Olympus IV, kretsen argus IV är placerad i mitten av kortet

2.1.2 Olympus IV

Olympus IV är ett utvecklingskort konstruerat av Lundaföretaget Logipard AB, se figur 2.3. Kortet är byggt för att testa och utveckla produkter baserade på kretsen Argus IV. Kortet har många video in- och utgångar samt kommunikationsanslutningar, så som Ethernet, USB och seriellport [16].

2.1.3 Argus IV

Argus IV kretsen består av en ARM9 processor på 225MHz samt en ISP(Image Signal Processor) som kan avkoda enligt standarderna MPEG-4, 4CIF och SDTV [16]. ISPn klarar av att korrigera för döda pixlar, göra färgkorrigeringar, avkoda bilder mm. Bildbehandling använder inte ARM9 processorn nämnvärt, vilket gör att den kan användas till andra uppgifter [16].

2.1.4 Inbyggt Linux

Den stora skillnaden mellan Linux i en PC eller server och inbyggt Linux, är att utvecklaren i det senare alternativet måste lägga större fokus vid begränsningar i bl.a. primär- och sekundärminne samt beräkningskomplexitet. För att anpassa Linux så att det kan köras i ett inbyggt system görs olika typer av förändringar.

Figur 2.4: Axis-kameran när den är placerad på roboten

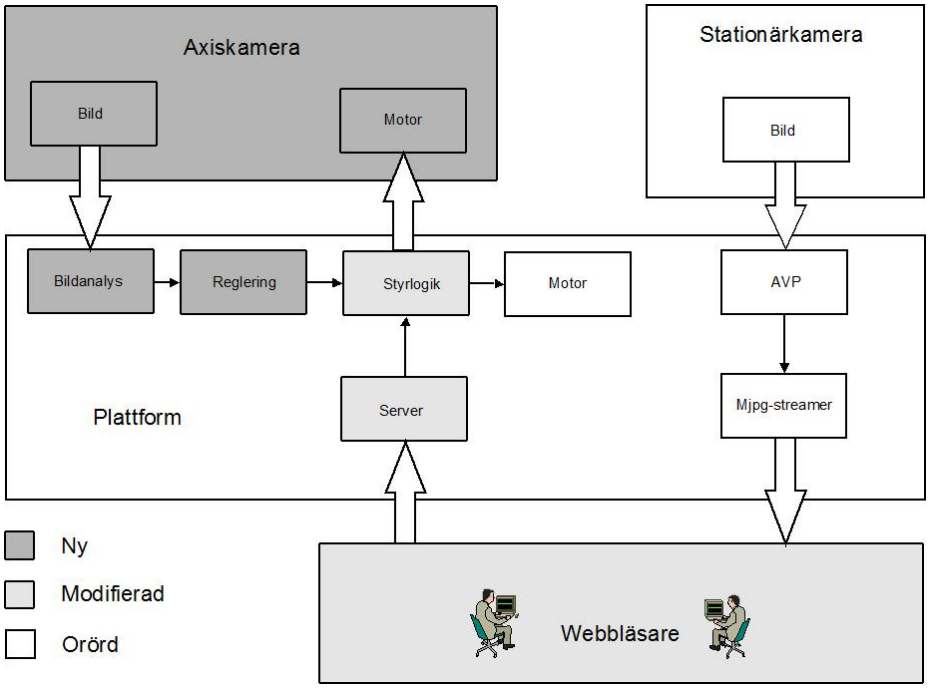
En typisk förändring är att man tar bort drivrutiner och program som inte behövs för den givna applikationen. Dessutom används ofta resursoptimerade verktyg som Busybox istället för GNU Core Utilities [14] [15]. Linuxportningen till Olympus IV kommer från Logipard själva och använder Linux v2.6.16 samt verktyget Busybox v1.00.

2.2 Kamera

Det första delmålet var att hitta en fungerande lösning för en rörlig kamera. Valet stod mellan två olika alternativ:

- Egen motordriven lösning som roterade den befintliga kameran.
- En färdig kamera med inbyggd motor.

Eftersom Axis Communications, genom institutionen för reglerteknik, var villiga att låna ut en kamera med inbyggd motor valdes denna lösning, då detta antogs vara den mest tidseffektiva lösningen. Den befintliga kameran behölls och placerades som en förarkamera längst fram på roboten. Den rörliga kameran är en nätverkskamera av märket Axis 215PTZ (Pan Tilt Zoom) [19]. Kommunikationen med kameran sker med protokollet http v1 över ethernet. En bild på den valda Axis-kameran kan ses i figur 2.4.



Figur 2.5: De mörkgråa delarna är nya i detta examensarbete. De ljusgråa är modifierade, medan de ofärgade inte är modifierade i detta examensarbete

2.3 Systemöversikt

En överblick av systemet kan ses i figur 2.5. De nya delar som utvecklats är förutom Axis-kameran även bildbehandlingen och regleringen. Samtidigt har servern, styrlogiken och GUIt i webbläsaren vidareutvecklats. En mer utförlig beskrivning av programmen ges i kapitel 8.

Kapitel 3

Definitioner

3.1 Digitalbild

En digital bild är en transformering från det diskreta bildplanet $\Omega \subset \mathbf{Z}^2$ till ett diskret färgrum. Varje punkt, som noteras $I(\mathbf{x})$, kallas en pixel med koordinaterna $\mathbf{x} = (x, y) \in \Omega$. Två olika sorters bilder används i rapporten.

1. Binär bild
   $I_{bin} : \Omega \rightarrow \{0, 1\}, \Omega \subset \mathbf{Z}^2$
2. 24 bitars RGB bild
   $I_{RGB} : \Omega \rightarrow \mathbf{Z}^3, \Omega \subset \mathbf{Z}^2$
   $I(\mathbf{x}) \in \{(I_R, I_G, I_B)\,;\, I_R, I_G, I_B \in \mathbf{Z}, 0 \leq I_R, I_G, I_B \leq 255\}$
   Där I_R, I_G och I_B beskriver intensiteten av färgerna rött, grönt respektive blått.

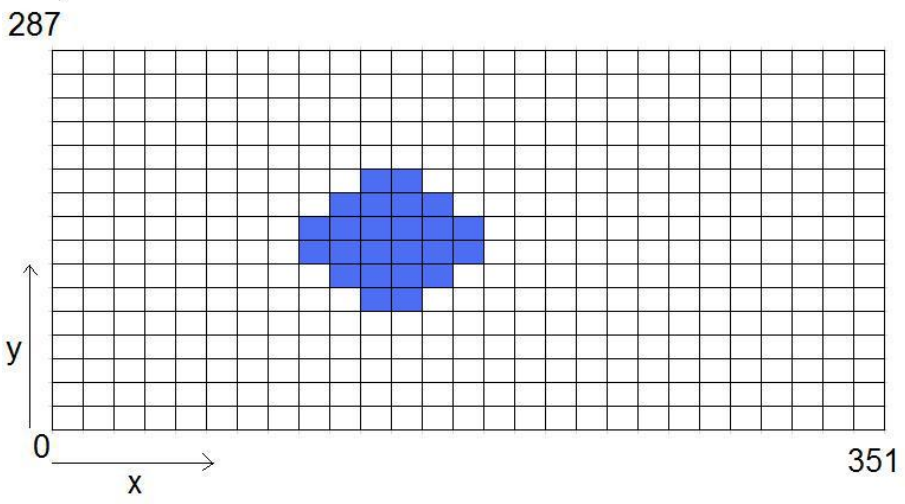
En binär bild är en bild som bara består av ettor och nollor. I rapporten betecknas en nolla vara objekt och en etta vara bakgrund. I en RGB bild beskrivs varje pixel av tre siffror. En för rött, grönt respektive blått. En 24 bitars RGB bild betyder att varje pixel beskrivs av 24 bitar, det vill säga 8 bitar per färg och pixel.

3.2 Koordinatsystem

Systemet använder tre koordinat rummet, bildplanet och kamerans.

3.2.1 Bildplanet

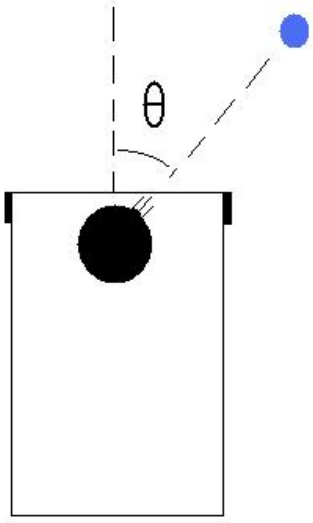
Bollens position ges som en (x, y) koordinat i bildplanet, se figur 3.1. Upplösningen på bilden är 352x288, bollen utgör normalt 300 till 2500 pixlar. Masscentrum på bollen ges som en x-koordinat, som är ett heltal mellan 0 och 351, och en y-koordinat, som ett heltal mellan 0 och 287.



Figur 3.1: Bildplanets koordinatsystem med en blå boll. Positionen ges som masscentrum på det identifierade objektet

3.2.2 Kamera

Det andra koordinatsystemet har origo i kameran, se figur 3.2. Vinklen, θ, är vinklen mellan plattformen och kameran. Att θ är lika med noll betyder att plattformen och kameran pekar i samma riktning.



Figur 3.2: Kamerans koordinatsystem, vinklen θ är vinkeln mellan plattformen och kameran

Kapitel 4

Tidigare arbete

Det finns många artiklar, böcker och rapporter som behandlar objektföljning. Många behandlar problemet med att följa en specifik person med hjälp av en mobil robot. Andra arbeten behandlar problemet att identifiera och följa personer i övervakningssyfte. Det finns många skiljda sätt hur detta görs men algoritmerna ser ofta väldigt lika ut. Strukturen på algoritmen kan sammanfattas i tre steg.

1. **Translatera bilden från en färgbild till en binär bild**
   Translationen görs ofta genom en statistisk analys av färgbilden. Ofta används kunskap om objektets färginnehåll och form [4]. Andra metoder initieras genom att visa objektet för kameran samtidigt som mjukvaran bygger en statistisk modell av objektet [3]. En annan metod som används är att identifiera rörelse i bilden [6]. Metoden bygger på att objektet som ska följas rör sig medan bakgrunden är stilla, se avsnitt 4.2.

   I avsnitt 4.1 ges några olika metoder för translatera en färgbild till en binärbild. Idén med en binärbild är att sätta pixlar som anses tillhöra objektet till 0 och pixlar som anses tillhöra bakgrunden till 1.

2. **Identifiera objektet i den binära bilden**
   Om translationen inte sker perfekt kommer bilden innehålla flera område med möjliga objekt. De flesta translationsalgoritmer resulterar i en bild med många små områden, mellan 1 till 10 i storlek, med falska objekt. Dessa kan enkelt sorteras bort genom filtrering av bilden, som beskrivs i avsnitt 4.3.

3. **Flytta kameran och den mobila roboten**
   Många applikationer använder enkla PD regulatorer för att följa objektet [4], [7]. Ett annat sätt är att efter identifieringen flytta kameran så att objektet centreras innan en ny identifiering sker [3]. Denna metod är enkel, men kan lätt ge upphov till en hackig rörelse.

4.1 Translation

4.1.1 Mean shift

Metoden utnyttjar kunskap om objektets färginnehåll och beskrivs i [4]. En kärna förs över bilden och letar upp området som bäst stämmer med det givna objektet. Kärnan är ofta en rektangel med storleken $n \times m$ pixel, som täcker en del av bilden. Kärnan flyttas sedan i steg över hela bilden. När kärnan rör sig över bilden räknas ett medel på färginnehållet ut och jämförs med objektet. Om kärnans innehåll ligger tillräckligt nära objektets sägs kärnans innehåll vara en kandidat till objektet. Andra områden antas tillhöra bakgrunden. En utökning av metoden är att efter varje iteration räkna ut en ny beskrivning av objektet. Detta leder till metoden lyckas fånga upp små stegvisa förändringar av objektets färginnehåll. Dock leder detta till ökad exekveringstid.

4.1.2 Tröskling

Metoden går ut på att gå igenom bilden pixel för pixel [12]. Varje pixel testas mot en tröskelnivå. Om pixeln klara nivån antas den tillhöra objektet annars antas den tillhöra bakgrunden. Mer om metoden kan läsas i avsnitt 5.1.

4.2 Rörelsedetektering

Ett sätt att angripa objektföljning är att identifiera och följa rörelse. Fördelen med rörelsedetektering är att algoritmen inte behöver tidigare kunskap om objektet. Algoritmen tar inte hänsyn till form eller färg utan bara rörelse i förhållande till bakgrunden. Metoden kan användas på två olika sätt.

Stegvis detektering Identifierar rörelse när kameran och roboten står stilla och flyttar sedan robot och kamera med rörelsedetekteringen avstängd, innan en ny identifikation sker [4]. Metoden leder till att robot och kamera får en hackig rörelse, se avsnitt 4.2.1.

Egenrörelsekompensering Skiljer sig från den stegvisa metoden genom att kompensera för rörelsen av plattform och kamera [6], [2]. Den stora fördelen är att roboten och kameran kan vara i rörelse samtidigt som rörelsedetekteringen sker, se avsnitt 4.2.2. Ett nytt problem som då uppstår är om egenrörelsekompenseringen inte är perfekt.

4.2.1 Stegvis detektering

Algoritmen förutsätter att kameran tar n bilder, I_t, $I_{t-1} \ldots$, I_{t-n-1} (tidssteg =1), med samma bakgrund. Genom att jämföra bilderna kan man detektera vilken del av bilden som förändrats och därför är i rörelse. Efter detekteringen flyttas sedan kameran och roboten innan en ny detektering sker.

Differensen på två följande bilder defineras som

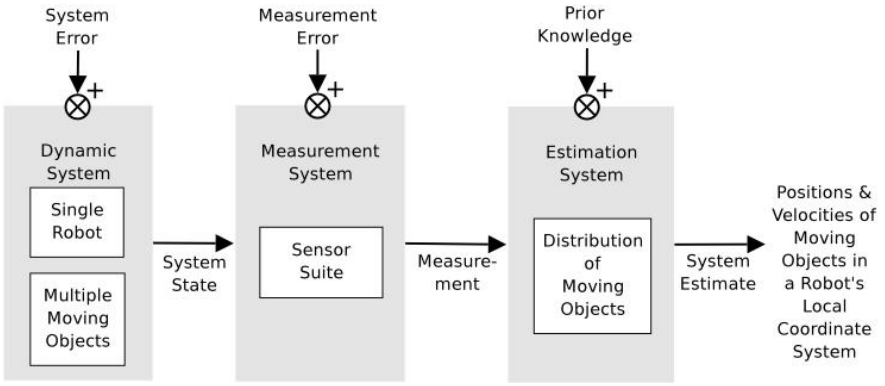
$$I_t^d = |I_t(\mathbf{x}) - I_{t-1}(\mathbf{x})|.$$

Från följden (I_0^d, I_1^d, I_2^d, $\ldots$ I_{t-1}^d, I_t^d) detekteras sedan rörelse i bilden. När algoritmen implementeras måste en avvägning ske mellan hur mycket tid som läggs

på rörelse och hur mycket tid som läggs på rörelsedetektering. I [1] används metoden med bra resultat. En PTZ(Pan Tilt Zoom) kamera används för att följa personer i en inomhusmiljö. Eftersom PTZ kameran är väldigt snabb samtidigt som personerna flyttar sig relativt långsamt gör att algoritmen fungerar bra.

4.2.2 Egenrörelsekompensering

Problemet med stegvis detektering är att detektering och rörelse inte tillåts ske samtidigt. Med egenrörelsekompensering löses detta genom att kompensera för egenrörelse. En ny begränsning är att om inte rörelsekompenseringen är väldigt bra kommer detta vara en stor felkälla. I [6] diskuteras problemet med objektföljning baserat på rörelsedetektering. I figur 4.1 ges en sammanfattning om olika felkällor.



Figur 4.1: Objektföljning från en mobilrobot: Problemet är att estimera positionen och hastigheten av objektet i robotens koordinatsystem

Egenrörelsekompensering

Nedan följer en kort beskrivning av egenrörelsekompensering. En mer utförlig beskrivning kan läsas i [6]. Vid tidpunkten t befinner sig roboten i (x, y, z) och registrerar bilden I_t. Sedan flyttas roboten till $(x + \Delta x, y + \Delta y, z + \Delta z)$ och vid tiden $t + \Delta t$ registreras bilden $I_{t+\Delta t}$. Bilderna I_t och $I_{t+\Delta t}$ kan inte jämföras direkt eftersom de är tagna i olika koordinatsystem.

För att lösa problemet transformeras I_t så att det verkar som om bilden är tagen i $(x + \Delta x, y + \Delta y, z + \Delta z)$. Det finns två angreppssätt att hitta transformeringen T. Det ena är att genom sensorer räkna ut $(\Delta x, \Delta y, \Delta z)$ och sedan genom geometriska beräkningar få fram transformeringen T. Det andra sättet är att genom detektering av fasta föremål i bilden. Genom att se hur dessa ändras mellan I_t och $I_{t+\Delta t}$ kan sedan en transformering T tas fram, mer kan läsas i [8].

Rörelsedetektering

För att identifiera ett föremål i rörelse jämförs på varandra följande bilder. Bilden I_{t-1} $(\Delta t = 1)$ transformeras till I_c genom

$$I_c(\mathbf{x}) = T\, I_{t-1}(\mathbf{x})$$

där T är kompensationen för egenrörelsen från förra avsnittet. Differensen på två följande bilder blir då

$$I_t^d = |I_c(\mathbf{x}) - I_t(\mathbf{x})|.$$

Algoritmen genererar en följd av bilder (I_0^d, I_1^d, I_2^d, ... I_{t-1}^d, I_t^d), vars pixelvärde representerar rörelsen i positionen. För att analysera bilderna behöver man ta hänsyn till två olika störningar. Dels att kompensationen för egenrörelsen inte är perfekt. Men också störningar i själva bilden från tex dåligt ljus och ändrad form på objekten.

4.3 Identifiering

Efter translationen analyseras den binära bilden. Första steget är att filtrera bort falska objekt. Ett enkelt sätt att göra detta är att stegvis föra en kärna på över bilden. Ofta är kärna 3 x 3 pixlar stor och om inte minst n av 9 pixlar är identifierat som objekt sätt alla pixlar till att tillhöra bakgrunden. Storleken och formen på kärnan samt hur många pixlar n som måste vara identifierat som objekt är beroende av den givna applikationen.

Om det återstår flera objekt i bilden, måste en mer noggrann undersökning av objekten göras. I generella applikationer är det vanligt att identifiera flera objekt. Exempelvis i övervakningsmiljöer, som i [5], kan flera personer identifieras och följas samtidigt. Om den givna applikationen inte ska identifiera flera objekt måste en analys göras vilket identifierat objekt som är mest likt det fördefinierade objektet, som behandlas i [3].

Kapitel 5

Bildbehandling

5.1 Objektidentifiering

Från Axis-kameran hämtas en bitmapbild i RGB format med 24 bitar per pixel som definieras i kapitel 3. Algoritmen för att hitta bollen bygger på att transformera bilden från RGB format till en binär bild. En binär bild har endast värdena 1 eller 0. Pixlar som anses tillhöra objektet sätts till 0 och pixlar som inte anses tillhöra objektet får värdet 1. Transformeringsmetoden som används kallas tröskling och går ut på att hitta tröskelvärde i varje färg som särskiljer bollen från omgivningen [12]. Tröskelvärdena bestämdes experimentellt och fungerar bra under förutsättning att bakgrunden är av en homogen färg. Ett exempel på tröskling ses i figur 5.1 där en bild i en viss given färg tex rött, R, transformeras till en binär bild. Varje pixel i bilden som uppfyller tröskelvärdet $R > 100$ sägs vara en del av objektet. I bilden kan man urskilja två föremål ett rundare och ett fyrkantigt.

84	92	80	80	82	82	93	82	96	89	88	43	82	83	79
82	74	89	86	92	92	84	92	92	80	89	83	92	92	80
92	57	74	51	83	82	88	83	83	78	74	92	78	85	85
83	79	91	85	83	92	84	92	92	80	91	39	74	89	79
89	67	79	74	66	83	88	83	78	79	74	80	74	80	80
89	79	57	87	156	144	88	74	67	187	120	117	91	91	83
82	80	144	134	122	145	94	62	74	173	123	140	90	74	80
92	83	134	155	177	168	176	89	80	134	150	145	91	91	83
83	78	96	164	133	124	49	89	78	103	116	117	74	80	80
80	49	93	55	122	79	80	80	55	93	82	96	91	91	83
83	80	55	49	67	80	93	82	96	84	92	80	93	82	96
84	92	80	80	80	80	55	80	55	88	83	78	91	91	83

(a) Röd bild, värden på R

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	1	1	1	1	0	0	0	1	1	1
1	1	0	0	0	0	1	1	1	0	0	0	1	1	1
1	1	0	0	0	0	0	1	1	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(b) Binär bild, trösklade värde $R > 100$

Figur 5.1: Ett exempel på tröskling, där tröskelvärdet $R > 100$ använts

Identifikation av blå boll

För att avgöra om en pixel är en del av den blå bollen användes

$$\{0 \leq I_R \leq 255\} \wedge \{200 < I_G \leq 255\} \wedge \{200 < I_B \leq 255\}. \tag{5.1}$$

En ny binär bild B bildas där varje pixel som enligt 5.1 kan tillhöra bollen sätts till 0 annars till 1. Eller mer formellt

$$B = \begin{cases} 0 & \text{om 5.1} \\ 1 & \text{annars.} \end{cases} \tag{5.2}$$

Eftersom målet var att identifiera en rund boll krävdes en algoritm för att identifiera runda föremål. För att göra detta implementerades funktioner för att räkna ut omkrets och area av de identifierade föremålen. För en cirkel är förhållandet mellan omkretsen i kvadrat och arean konstant lika med $k_{ideal} = 4\pi$, vilket också är det minsta möjliga förhållandet för ett geometriskt objekt,

$$k_{ideal} = \frac{(2r\pi)^2}{r^2\pi} = \frac{4\pi^2 r^2}{r^2\pi} = 4\pi.$$

En acceptabel övre gräns på $k = 5\pi$, togs fram experimentiellt. För att ytterligare testa objektet sattes min och max gränser för area

$$300 \leq A_{boll} \leq 2500.$$

Arean av föremålet användes även för att styra robotens rörelse och få den att hålla ett visst avstånd till det observerade objektet.

5.2 Implementation

För att identifiera objektet i den binära bilden söks bilden igenom rad för rad. När en nolla upptäcks söker algoritmen efter sammanhängande nollor, samtidigt räknas area, maximal bredd och höjd samt omkrets ut. Positionen ges sedan som x, y koordinater i bilden. För att minimera programmets exekveringstid avslutas sökningen när det identifierat ett objekt som uppfyller kraven enligt stycke 5.1. Detta kan i sig vara en stor felkälla, men eftersom bildanalysen förutsätter en homogen bakgrund och att inga falska objekt är synliga fungerar algoritmen för denna tillämpning. Ett annat alternativ skulle vara att analysera hela bilden och sedan välja ut det objekt som bäst matchar ovanstående grundkrav. Nedan följer kod hur identifikationen av objektet i den binärabilden gjordes.

```
/*
Den binärabilden ligger i blue[][]
*/
analyzeImage(){
for (varje rad){
     for (varje kolumn){
           if (blue[kolumn][rad] ==0){
                kolla raden för sammanhängande nollor
                uppdatera arean

                while{
                     1. Öka raden med ett (rad+1)
                     2. Leta efter nollor sammanhängande med rad-1
                     3. uppdatera arean, bredd och omkrets
                     4. om inga nollor hittas bryt. Annars börja om med 1.
                }
           }
           testa det givna objektet om det uppfyller givna villkor
           Om objektet hittat bryt. Annars börja om
      }
}
}
```

En stor fördel med den valda algoritmen var att implementationstiden blev relativt kort. Minnesåtgången för programmet blev dock relativt stor eftersom en minneskrävande rådatabild måste hämtas och lagras vid varje avläsning.

5.2.1 Begränsningar

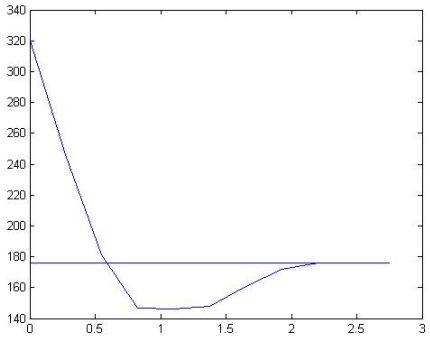
Den stora begränsningen är dock att man inte kan välja en godtycklig samplingstid för värde på bollpositionen. Detta beror dels på exekveringstiden för algoritmen men även på att processorn är hårt belastad. Den långa exekveringstiden för algoritmen beror främst på att det tar lång tid att hämta bilden från kameran. Efter implementationen blir den kortaste möjliga samplingstiden 0.25 sekund. Ytterligare en begränsning är att objektet måste identifieras på en homogen bakgrund. Mer om programstrukturen kan läsas i kapitel 8.

Kapitel 6

Reglering - prediktion

Som beskrevs i avsnitt 5.2.1 kunde samplingstiden i programmet inte sättas till ett valfritt värde. Den undre gränsen för samplingstiden var 0.25 sekunder. Detta gjorde det väldigt svårt att konstruera en bra regulator till kameran. I figur 6.1 ses ett exempel där bollen befinner sig i ytterkant av bilden och en P-regulator centrerar bollen. Detta resulterar i en översläng i regleringen vilket normalt inte inträffar med en P-regulator. Denna översläng beror på den långa samplingstiden.

På grund av att samplingstiden inte kunde minskas till lämplig längd konstruerades ett kalmanfilter för att prediktera värden mellan inläsningarna av bollpositionen. På detta sätt gavs möjlighet att ge styrsignal till kameramotorn med en högre frekvens. Filtret används endast till att prediktera bollens rörelse i x-led i bildplanet, det vill säga horisontell rörelse. Rörelsen i y-led, kommer inte att predikteras.



Figur 6.1: Experiment med en P-regulator som försöker centrera bollen i bilden till referensvärdet 176

6.1 Kalmanfilter

Nedan ges en kort sammanfattning av teorin bakom ett Kalmanfilter, en mer omfattande beskrivning kan läsas i [10]. Processen beskrivs av

$$\begin{aligned} x(kh+h) &= \Phi\, x(kh) + \Gamma\, u(kh) + v(kh) \\ y(kh) &= C\, x(kh) + e(kh) \end{aligned} \tag{6.1}$$

Med

$$\begin{aligned} E[v(kh)v^T(kh)] &= R_1 \\ E[v(kh)e^T(kh)] &= R_{12} \\ E[e(kh)e^T(kh)] &= R_2 \end{aligned} \tag{6.2}$$

$$\tag{6.3}$$

I denna applikationen, antas varianserna vara tidsoberoende och konstanta. Dessutom antas att R_{12} är lika med noll.

Kalmanfiltret ges då av

$$\hat{x}(kh+h|kh) = \Phi\hat{x}(kh|kh-h) + \Gamma u(kh) + K(y(kh) - C\,\hat{x}(kh|kh-h). \tag{6.4}$$

K ges av

$$K = (\Phi P C^T)(CPC^T + R_2)^{-1} \tag{6.5}$$

där P ges av lösningen av matrisekvationen

$$P = \Phi P \Phi^T + R_1 - (\Phi P C^T)(CPC^T + R_2)^{-1} CP\Phi^T. \tag{6.6}$$

6.2 Process beskrivning

Eftersom kalmanfiltret endast används till att prediktera bollens rörelser i sidled (x -led), blir tillståndsvektorn

$$x = (x_{pos} \quad \dot{x}_{pos}). \tag{6.7}$$

I ekvation 6.1 blir

$$\Phi = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix} \tag{6.8}$$

och

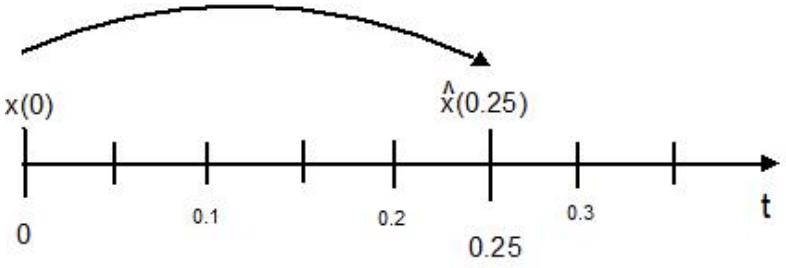
$$C = \begin{pmatrix} 1 & 0 \end{pmatrix}. \tag{6.9}$$

Styrsignalen till kameran är ett värde mellan -100 och 100 som anger kamerans hastighet. Det vill säga inte ett direkt värde på hur bollens position ändras. Detta innebär att Γ -matrisen är svår att bestämma och beror på avståndet till objektet. I kalmanfiltret användes istället Γ som en designparameter.

6.3 Prediktion

Samplingstiden på bollpositionen var 0.25 sekund. Eftersom detta var för långt var idén att uppskatta värden, på bollens position, mellan de uppmätta värdena. För att prediktera bollens position behövdes en uppskattning på bollens hastighet. För att göra detta implementerades ett kalmanfilter. Vid tidpunkten t_0 predikteras hastigheten i tidpunkten $t_0 + 0.25$, se figur 6.2. Φ-matrisen i ekvation 6.8 blir

$$\Phi = \begin{pmatrix} 1 & 0.25 \\ 0 & 1 \end{pmatrix}. \tag{6.10}$$



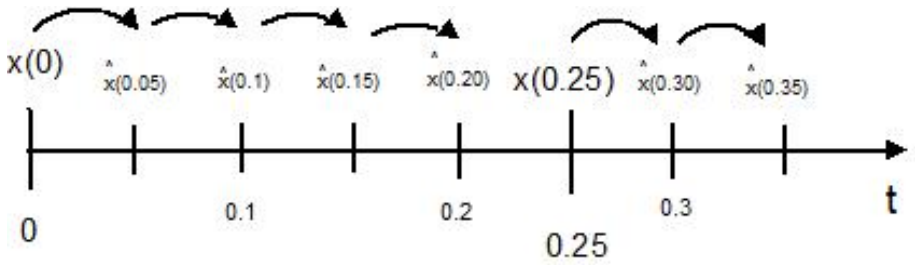
Figur 6.2: Tidslinje över kalmanfiltret som predikterar bollens hastighet

Den uppskattade hastigheten används för att prediktera bollens position mellan tidsstegen, se figur 6.3. Tidssteget på prediktionen var 0.05 sekund, det vill säga frekvensen ökade med en faktor fem. För prediktionen användes

$$\hat{x}(kh + h|kh) \;=\; \Phi\, \hat{x}(kh|kh - h) + \Gamma\, u(kh) \tag{6.11}$$

med

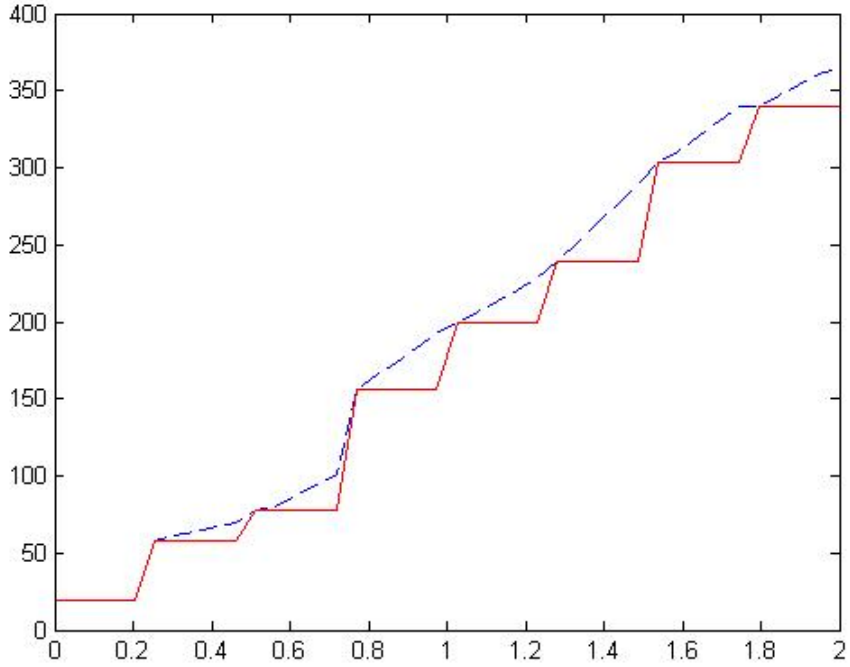
$$\Phi = \begin{pmatrix} 1 & 0.05 \\ 0 & 1 \end{pmatrix}. \tag{6.12}$$



Figur 6.3: Tidslinje över prediktionen av bollens position

6.3.1 Resultat

Ett testresultat av prediktionen kan ses i figur 6.4. Bollen fördes framför kameran från vänster till höger, det vill säga från låga till höga pixelvärden. Den streckade linjen i figuren visar de predikterade värdena, medan den heldragna är de uppmätta värdena. Som ses följer prediktionen de uppmätta värdena bra. Notera att under experimentet var både kamera och plattform stilla.



Figur 6.4: Testresultat av prediktionen, som visar att prediktionen fungerar bra för ett stationärt system

Kapitel 7

Reglering av systemet

Regleringen implementerades i några olika steg.

- Reglering av kameran med stationär robotplattform.
- Reglering av kameran med användarstyrd robotplattform.
- Reglering av kameran och roboten för att automatiskt centrera och följa bollen.

Utsignalerna från reglersystemet är signaler för styrning av robotens motorer samt styrning av kameran. Kameran styrs genom att sända värden på hastigheten i x- respektive y-led. Negativa värden för vänster/ner och positiva för höger/upp. På samma sätt styrs roboten genom att sända positiva och negativa hastigheter till var och en av motorerna.
Om man antar att bildanalysen ligger utanför systemet finns det tre insignaler. Direkt från kameran får man en position i grader. Som kameran är monterad betyder $\theta = 0°$ att kameran pekar rakt fram. Från kameran hämtas också en bitmapbild i upplösningen 352x288. Via bildanalys fås sedan ett positionsvärde i x-led och y-led samt ett mått på storleken på objektet. Värdena på bollpositionen ges i pixlar, dvs mellan 0-351 i x-led och 0-287 i y-led. Typiska värden på storleken är 300-4000 pixlar. Referensvärdet på positionen är mitten alltså 176 och i storlek 1200, vilket motsvarar ca 0.5 meter.

7.1 Tidsdiskret PID regulator

För att reglera objekt följningen användes tidsdiskreta PID regulatorer. Nedan ges en kort beskrivning av en generell tidsdiskret PID regulator, läsaren antas ha en viss kunskap i området, en mer utförlig förklaring kan hittas i [10].

PID regulatorn ges i kontinuerlig tid av

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int^t e(s)ds + T_d\frac{de(t)}{dt}\right). \tag{7.1}$$

Eftersom en ren derivata del skulle ge väldigt stora förstärkningar av mätbrus approximerar man överföringsfunktionen sT_d med

$$sT_d \approx \frac{sT_d}{1 + sT_d/N}.$$

Detta gör att derivataförstärkningen blir som högst N vid höga förstärkningar. En annan ändring från 7.1 är att man ofta inte låter P-delen verka på hela referenssignalen u_c. Genom Laplace transform av u_c, y och u ges nu PID regulatorn av

$$U(s) = K\left(\underbrace{bU_c(s) - Y(s)}_{P} + \underbrace{\frac{1}{sT_i}\left(U_c(s) - Y(s)\right)}_{I} - \underbrace{\frac{sT_d}{1 + \frac{sT_d}{N}}Y(s)}_{D}\right) \tag{7.2}$$

För att diskretisera regulatorn kan man använda många diskretiseringsmetoder, nedan används framåt differens för I-delen och bakåt differens för D-delen.

Den diskretiserade regulatorn blir då

$$\begin{aligned} P(kh) &= K(bu_c - y(kh)) \\ I(kh+h) &= I(kh) + \frac{Kh}{T_i}e(kh) \\ D(kh) &= \frac{T_d}{T_d + Nh}D(kh-h) + \frac{KT_dN}{T_d + Nh}(y(kh) - y(kh-h)) \end{aligned} \tag{7.3}$$

styrsignalen u ges nu av

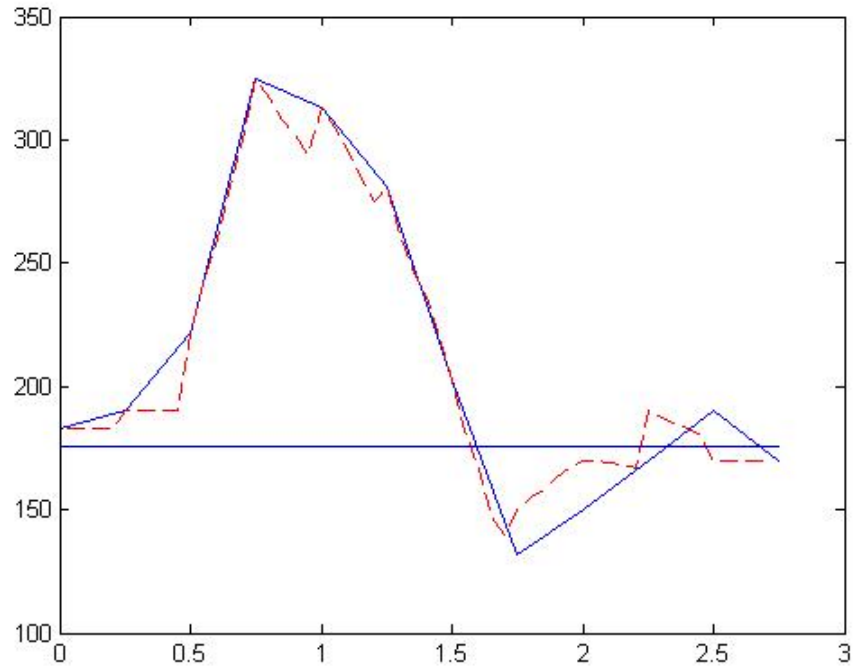
$$u(kh) = P(kh) + I(kh) + D(kh). \tag{7.4}$$

7.2 Stationär robot

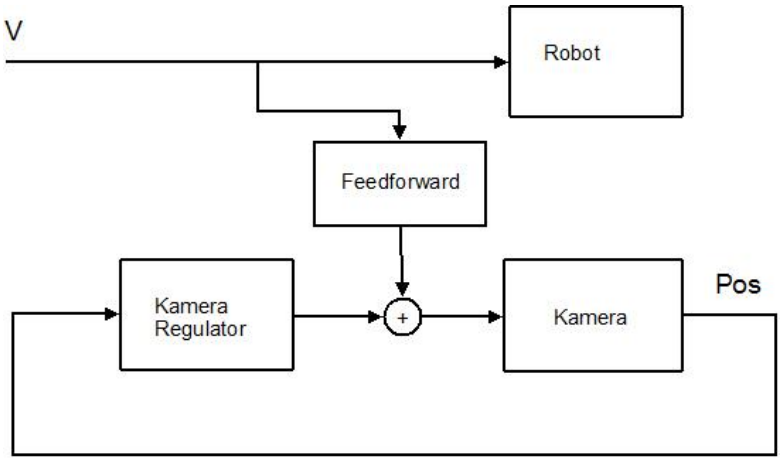
I ett första steg designades en reglering för kameran med en stationär robot, som regulator användes en PID regulator som beskrivs i avsnitt 7.1. Regulatorns uppgift är att hålla bollen centrerad i kamerabilden, vilket betyder positionsvärdet 176 i x-led. Regulatorn använde de predikterade värdena, från kalmanfiltret, på positionen för att räkna fram styrvärde till kameramotorn. I figur 7.1 visas ett test där bollen rört sig i sidled framför kameran och sedan stannat samtidigt som regulatorn försökt hålla bollen centrerad. Den heldragna linjen visar bollens position i bilden och den streckade de predikterade värdena. Parametrarna för regulatorn i ekvation 7.3 ställdes in experimentellt och blev

$$T_d = 0.01,\ K = 0.5,\ N = 10,\ b = 1,\ Ti = 15.$$

Kameramotorn i höjdled är inte lika snabb som i sidled. Detta tillsammans med att bollen inte antas förflytta sig mycket i höjdled gjorde att endast en enkel P-regulator implementerades. Förstärkningen i regulatorn sattes till $K = 0.5$.



Figur 7.1: Heldragen linje är uppmätta positionsvärde och streckad linje predikterade värde. Referensnivå på 176.

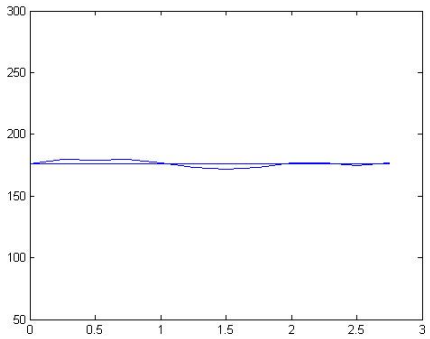


Figur 7.2: Blockdiagram över reglering när användarstyrningen används

7.3 Användarstyrning

Regleringen under användarstyrning har som mål att hålla objektet centrerat i bilden samtidigt som användaren styr roboten. En feedforward koppling, som

visas i figur 7.2, infördes. Syftet med kopplingen är att rotera kameran i samma hastighet men i motsatt riktning som roboten svänger. De små störningar som sedan uppstår klarar PID regulatorn enkelt av. Resultatet av att användaren svänger roboten med konstant hastighet ses i figur 7.3. Som figuren visar ligger positionsvärdena nära referensvärdet.



Figur 7.3: Den uppmätta positionen när användaren svänger roboten med konstant hatighet, referensnivå på 176.

7.4 Reglering av robot

För att roboten skulle kunna följa bollen implementerades två stycken regulatorer. En riktningsregulator som ser till att roboten är riktad mot bollen och en avståndsregulator som håller ett definierat avstånd till bollen. Variabeln som regleras av riktningsregulatorn är kamerans vinkel i förhållande till plattformen. Avståndsregulatorn kör roboten framåt eller bakåt beroende på bollens position. Eftersom kameraregleringen centrerar bollen i bilden, pekar kameran mot bollen. För att avståndsregulatorn ska ha möjlighet att reglera avståndet blev ett bra referensvärde för ritningsregulatorn noll grader, det vill säga att kameran pekar rakt fram. Detta innebär att roboten precis som kameran alltid försöker vara riktad mot bollen. Riktningsregulatorn implementerades som en PID regulator. Parametrarna, i ekvation 7.3, ställdes in experimentellt och blev

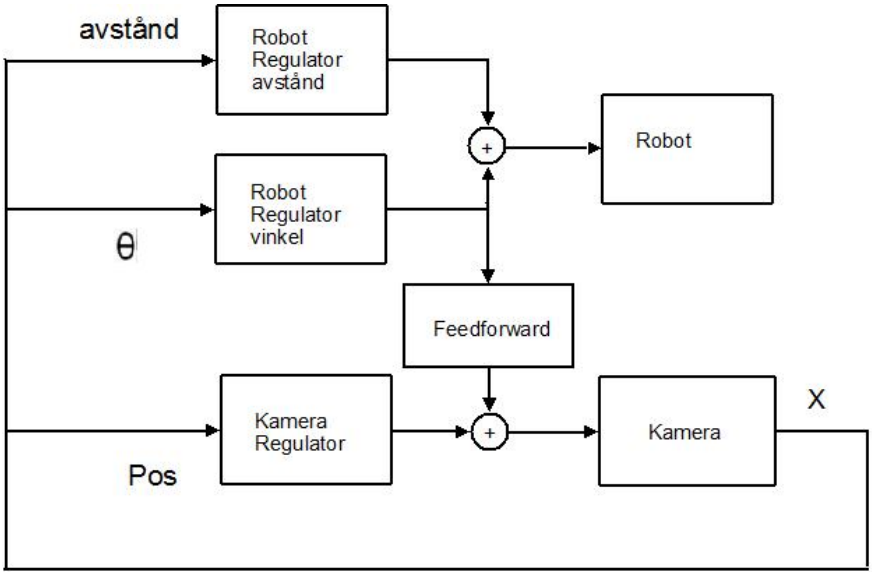
$$T_d = 0.01,\ K = 1.5,\ N = 3,\ b = 1,\ Ti = 10.$$

För att hålla avståndet implementerades en P-regulator med den experimentellt bestämda förstärkningen 1.5. Regulatorn använder arean av objektet räknat i pixlar för att uppskatta avståndet, referensvärdet är 1200 pixlar. Det vill säga om arean är större än 1200 upptar objektet en stor del av bilden och antas därför vara nära kameran. Regulatorn ger då utsignalen att backa roboten. Om arean är mindre än 1200 antas objektet vara för långt ifrån och roboten kör framåt. Avståndsregulatorn bygger på att man vet storleken på objektet. Om storleken på objektet ändras bör även referensvärdet 1200 ändras. Detta leder annars till att det bestämda avståndet till objektet ändras. Värdet 1200 är experimentellt framtaget och betyder för den givna blå bollen ett avstånd på ca 0.5 meter.

Styrsignalen till motorerna fås genom att summera styrsignalerna från de båda regulatorerna. Det vill säga om riktningsregulatorn ville svänga vänster och avståndsregulatorn köra framåt körde roboten framåt vänster.

7.5 Reglering av komplett system

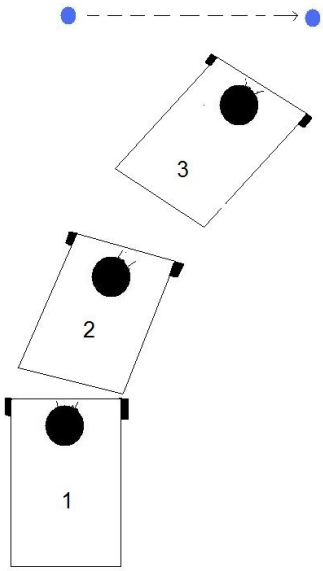
I ett sista steg kopplas alla regulatorerna ovan ihop, blockschemat kan ses i figur 7.4.



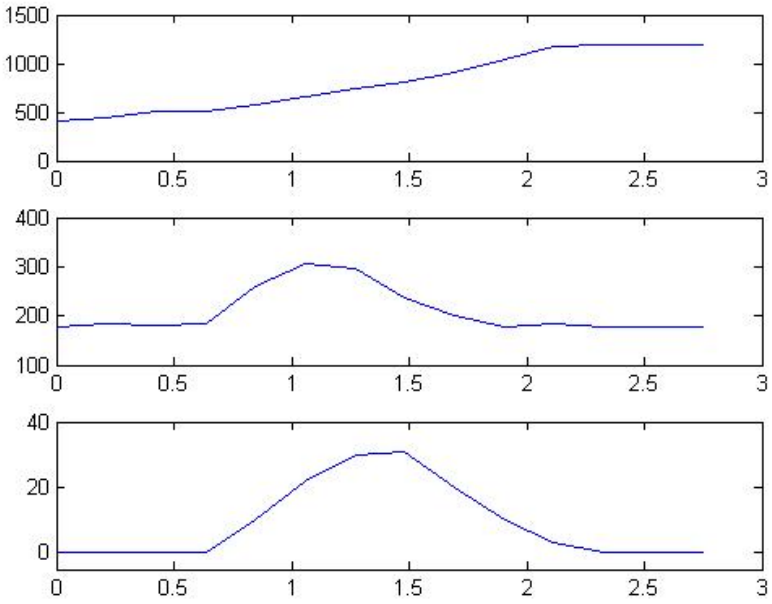
Figur 7.4: Blockdiagram av reglering när det kompletta systemet används

I figuren ses att utsignalen från robotens regulatorer summeras. Utsignalen som styr riktningen på roboten kopplas också till en feedforward koppling. Detta gör att kameraregulatorn inte påverkas av att roboten svänger. Figur 7.5 visar hur roboten rör sig när bollen placeras på stort avstånd från roboten och rör sig i sidled innan den stannar. Kameran ställer in sig snabbt och har centrerat bollen redan i läge 2. Plattformen tar någon längre tid och är på rätt avstånd och läge först i läge 3.

I figur 7.6 visas avståndet, bollens position i bilden samt kamerans vinkel relativt plattformen. Avståndet till bollen ändras under en ganska lång tid till referensvärdet 1200. Efter cirka 1 sekund rör sig bollen i sidled som visas i figur 7.5. Positionen i bilden ändras innan den regleras till referensvärdet 176. Kameravinkeln påverkas av två saker. Kameraregulatorn reglerar kameran så att bollen centreras i bilden samtidigt svänger plattformen för att köra mot bollen. Som ses i figuren centreras bollen i kameran relativt snabbt, medan riktningen på roboten och avståndet regleras relativt långsamt.



Figur 7.5: Bollen rör sig åt höger och plattformen följer efter och stannar på lämpligt avstånd



Figur 7.6: Värden för avstånd, position i bilden och kameravinkeln när bollen rör sig som i figur 7.5

Kapitel 8

Programstruktur

8.1 Användare

Som användare av programmet kan man köra i tre olika lägen.

- Användaren styr både kameran och plattformen från webbläsaren.
- Användaren styr plattformen samtidigt som programmet centrerar bollen i kameravyn.
- Programmet styr både kameran och plattformen för att följa bollen.

Användaren surfar in på plattformens hemsida och får via en indexsida upp en java-applet, se figur 8.1.

Figur 8.1: Användaren styr roboten från webbläsaren i datorn

8.2 Main

För kommunikation med användaren, motorer, kamera och reglering används ett program med fyra olika trådar. Två trådar är för de olika regulatorerna, en tråd för bildanalys och hämtning av bild, samt en tråd för kommunikation med användaren. Nedan beskrivs de olika trådarna mer i detalj. I figur 8.3 ges en schematisk översiktsbild av programmet och i 8.4 ses ett sekvensdiagram för ett tidssteg (0.25 s).

8.2.1 angBotObjectIdent

Här hämtas bitmapbilden från kameran och en objektidentifiering görs. Positionen på objektet skickas till angBotCameraControl och avståndet till angBotPlattformControl. Tråden har en samplingstid på 0.25 sekund. Här ligger också kalmanfiltret som ger en prediktion på hastigheten som angBotCameraControl behöver. Nedan beskrivs vad som görs under varje iteration.

```
getImage();      //Bilden hämtas, se nedan
analyzeImage();  //Bilden analyseras, Beskrivs i bildanalys kapitlet
calculateSpeed();//Predikterar hastigheten, se nedan

\*
Bilddelarna hamnar i recv_camera och bytes_read indikerar
hur många bytes som fåtts vid hämtningen. Hela bilden kan alltså
inte hämtas på en gång utan genom flera anrop till recv(...).
Trösklingen sker direkt och den binära bilden
lagras i blue[][]
*\
getImage(){
length=351;//bredden på bilden
while( (bytes_read = recv(sockfd, recv_camera, sizeof(recv_camera),0))>0){
    //sockfd är en tidigare öppnad socket

    for(i=rest; i<bytes_read;i+=3){ //Hoppar en pixel i taget
        if(recv_camera[i] >200 && recv_camera[i-1]>200 ){ //Tröskling
            blue[col][row]=0;              //blue är den binärabilden
        }else{
            blue[col][row]=1;
        }

        col++;
        if(col==length){ //Eftersom bilden hämtas som en vektor måste
            col=0;       //det tas hänsyn till att skifta rad när man
            row++;       //vill ha bilden som en matris.
        }
    }
    rest=i-bytes_read; //Om bytes_read inte är delbart med 3

}
```

```
calculateSpeed(){
    e=pos-xhat1;
    xhat1=xhat1+K1*e+dt*xhat2;
    xhat2=xhat2+K2*e; //speed=xhat2
}
```

8.2.2 angBotCameraControl

Tråden angBotCameraControl har till uppgift att reglera centreringen av objektet. Den hämtar positionen i x- och y-led samt den uppskattade hastigheten i x-led, från angBotObjIdent. Samplingstiden är 0.05 sekund vilket innebär att den kör 5 gånger oftare än angBotObjectIdent. Vid varje iteration predikteras positionen i x-led. Med de predikterade värdena i x-led och de hämtade värdena i y-led räknas styrsignaler till kameran ut och skickas till angBotControll.

```
if (t==4){
    pos_x= ObjectIdent.getPosX();
    pos_y= ObjectIdent.getPosY();
    speed= ObjectIdent.getSpeed();
    t=0
}else{
    pos=predict(pos_x,speed);
    t++;
}
calculateOutput(pos_x,pos_y); //Räknar ut utsignalerna mha
                              //två regulatorer



calculateOutput(pos_x, pos_y){
    e=176-pos_x;
    P=K*e;
    D=ad*D-bd*(pos_x-pos_x_old); //ad och bd konstanter

    P_y=Ky*(144-pos_y);
    control.moveCamera(P+I+D,Py,1); //sista ettan förklaras i
                                        //avsnittet om angBotControll

    I=I+ai*e;                    //ai konstant
    pos_x_old=pos_x;
}
```

8.2.3 angBotPlattformControl

Tråden angBotPlattformControl ser till att plattformen pekar i samma riktning som kameran och att roboten följer objektet på ett fördefinierat avstånd. Från kameran hämtas vinkeln och från angBotObjectIdent fås arean av det identifierade objektet. Arean mäts i antalet pixlar som objektet utgör i bilden. Det

vill säga om objektet är nära roboten kommer arean vara stor och om objektet är långt ifrån blir arean liten. En P regulator använder arean till att hålla plattformen på ett visst avstånd till objektet. Referensvärdet på arean är 1200 pixlar. Samplingstiden är 0.5 sekund.

```
range=ObjectIdent.getRange();
cameraPos=getCameraPos(); //Vinkeln som kameran står i
                          //förhållande till roboten,
                          //hämtas från kameran
calculateOutput();


calculateOutput{

//Riktningsregulatorn
    e=cameraPos; //0 grader betyder att kameran pekar rakt fram
    D=ad*D-bd*(cameraPos-old_pos);//ad och bd konstanter
    P=e*K;

    U=P+I+D;

    control.movePlat(U,1);//1 en flagga
    control.moveCamera(-(U*0.58),0,2);//Framkopplingen

    old_pos=cameraPos;
    I=I+ai*e;//ai konstant

//Avståndsregulatorn
    e_range=1200-range; //referensvärde på arean=1200
    P=K_range*e_range;
    control.movePlat(P,2);//2 en flagga
}
```

8.2.4 angBotController

Objektet angBotController är ingen egen tråd utan motsvarar plustecknena och feedforward i blockdiagramen. Innan värdena skickas till motorerna och kameran passerar de här. Objektet ser också till att rätt saker sker beroende på vilket läge användaren ställt in. Om användaren själv styr roboten ignoreras värdena från regulatorerna osv. Nedan ses C-kod för metoderna som styr kamera och plattform.

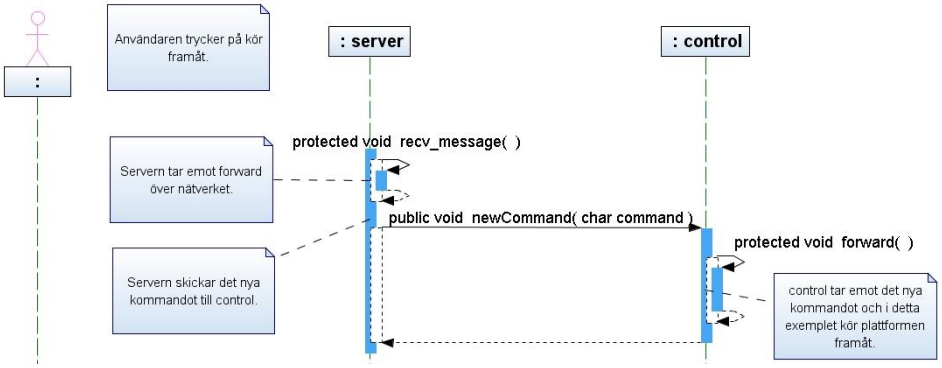
```
boolean driverAll;     //sann = Användaren styr hela roboten
boolean driverMode;    //sann = Användaren styr plattformen men inte kameran
boolean plattformMode; //sann = Plattformen helt autonom

.
.
.
/*
I metoden moveCamera läggs olika utsignaler till kameran ihop.
```

```
Flaggan är till för att bestämma varifrån signalen kommer.
u_x betyder kamerans rörelse i x-led medan u_y betyder
kamerans rörelse i y-led. Metoden move() innehåller kod
för att skicka signalerna till kameran
*/
moveCamera(int u_x,int u_y, int flag){
    if(!driverAll){
        if(flag==0){            //Flag =0 användaren (feedforward i driverMode)
            UplattformUser=u_x;

        }else if(flag==1){       //Flag ==1 kameraregleringen
            Ucamera=u;
            Uy=u_y;

        }else if(flag==2){       //Flag=2 plattformsreglering
            Uplattform=u;//(feedforward i plattformMode)
        }
    }
    if(driverMode){
        Uplattform=0; //Tar bort feedforward från plattformen
                      //om den inte är i plattformMode.
    }
    move(UplattformUser+Uplattform+Ucamera,Uy);
}


/*
movePlat används för att reglera roboten när den kör autonomt
*/

movePlat(int Uplat,int flag){


    if(flag==2){           //Sätter utsignalen framåt/bakåt
        Uforward=Uplat;

    }else if(flag==1){     //Sätter utsignalen vänster/höger
        Uside=Uplat;   //Uside <0 betyder att roboten ska svänga vänster
    }

    if(Uforward==0){       //Avståndet är rätt och plattformen roterar
        motVelL= Uside;
        motVelR= -Uside;

    }else if (Uforward>0){ //Plattformen kör framåt
        if(Uside<0){
                motVelL=Uforward;
                motVelR=Uforward-2*Uside; //höger hjul snabbare=svänger vänster
        }else{
                motVelL=Uforward+2*Uside;
```

```
                motVelR=Uforward;
        }
    }else{              //Plattformen backar
        if(Uside<0){
                motVelL=Uforward+2*Uside;
                motVelR=Uforward;
        }else{
                motVelL=Uforward;
                motVelR=Uforward-2*Uside;
        }
    }
}

sendData();         //Skickar data till motorerna baserat på motVelL och motVelR
```

8.2.5 angBotServer

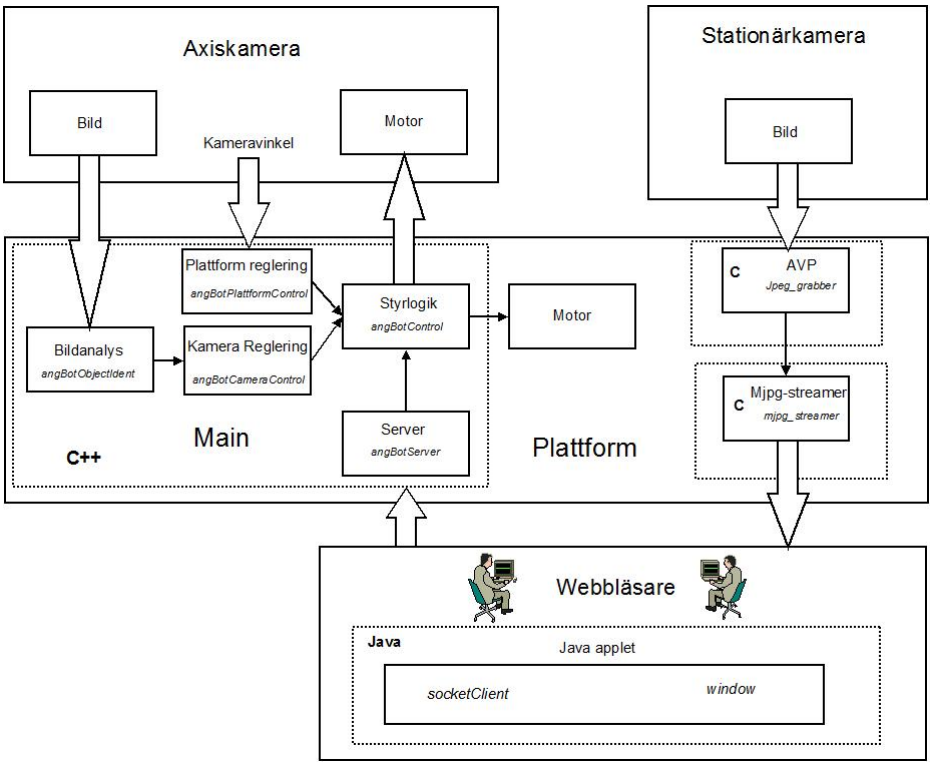
Tråden angBotServer sköter kommunikationen med användaren på datorn skickar värdena till angBotControl som bestämmer vad som görs beroende på läge. Samplingstiden är 0.5 sekund. I figur 8.2 visas vad som händer när användaren vill köra roboten framåt. Användaren trycker på framåt i webbläsaren och meddelandet tas emot i servern. Servern skickar sedan vidare meddelandet till angBotControl som skickar meddelande till motorerna. Notera att om användaren inte ställt in roboten i användarstyrningsläge kommer inget anrop till forward() göras.



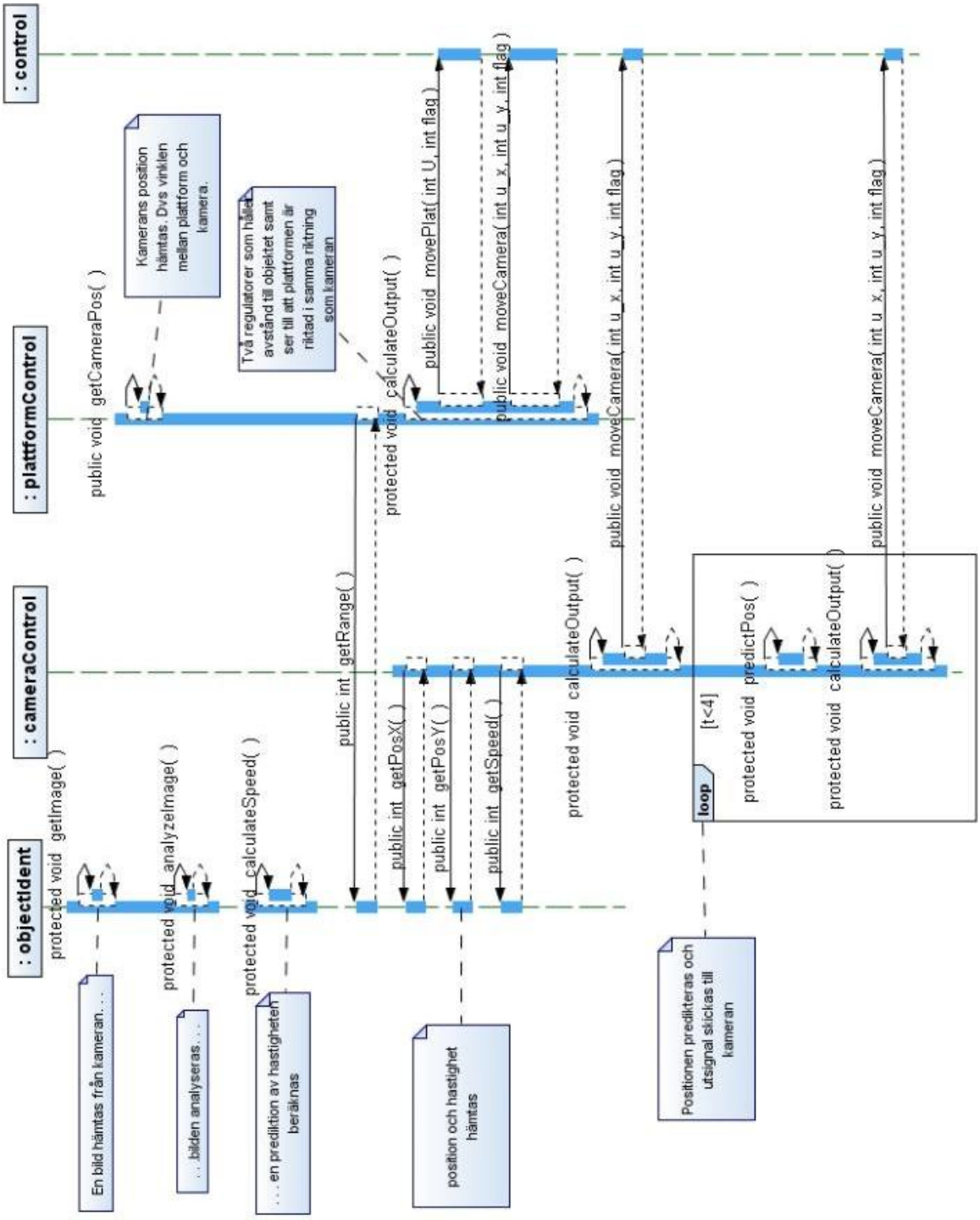
Figur 8.2: Ett sekvensdiagram som visar när användaren vill köra roboten framåt

8.2.6 Programöversikt

En översikt av programmet ses i två figurer nedan. I figur 8.3 ses bland annat hur de olika trådarna används i programmet. I figur 8.4 ses ett sekvensdiagram över vad som hände under ett tidsteg. Det börjar med att bilden hämtas från kameran och slutar med att utsignaler skickas till motorerna och kameran.



Figur 8.3: Programöversikt som visar hur de olika trådarna och objekten används och samverkar



Figur 8.4: Ett sekvensdiagram som visar vad som sker mellan två bilder

8.2.7 Implementation

För att göra main programmet multitrådat användes Posixtrådar vilka ofta kallas pthreads. Posixtrådar är ett standard interface för att skapa och manipulera trådar. Under implementationen uppkom en del problem med pthreads och operativsystemet. Vid regleringen vill man gärna ha ett deterministiskt system, dvs man vill köra de olika regulatorerna med fasta samplingstider. Ett stort problem som uppstod var att tiden från det att en tråd vill börja tills den börjar exekveras var stor trots att tråden hade hög prioritet. Tråden angBotObjectIdent hade högst prioritet och en samplingstid på 0.25 sekund. Mätningar visade att denna tråd fick vänta upp till 0.03 sekund på processor tid. Tidsfördröjningarna gjorde också att samplingstiden på prediktionen inte kunde sättas lägre än 0.05 sekund.

8.3 Kommunikationsprotokoll

I programmet används TCP/IP i två olika syfte, dels kommunikation med kameran och dels kommunikation med användaren. För kommunikation med servern på kameran används protokollet http 1.0. Servern (angBotServer) på plattformen, som kommunicerar med användaren, använder ett egenimplementerat protokoll.

8.4 Stationärkamera

8.4.1 Jpeg grabber

Jpeg grabber är uppbyggt av drivare och hjälpklasser från AVPn. Bilder hämtas i YUV420 format och omvandlas till jpeg. För att överföra bilder från Jpeg grabber till Mjpg streamer användes en Jpeg device.

8.4.2 Mjpg streamer

Mjpg streamer är ett öppen källkodsprogram som omvandlar Jpeg bilder till en Mjpg ström. Källkoden finns på projektets hemsida [17]. Programmet innehåller även en http-server som sköter angBots hemsida.

8.5 Verktygskedja

För utveckling av mjukvara användes en verktygskedja från Logipard. Verktygskedjan består av många olika program bla Linux, Busybox, standard bibliotek för C och länkningsverktyg osv. Det behövs även en korskompilator eftersom program utvecklade på utvecklingsdatorn(en vanlig laptop) även ska kunna köras på Argus IV. En korskompilator gör det allstå möjligt att kompilera program på en laptop och sedan köra dem på Olympus IVs ARM processsor.

För att utveckla program på Olympus IV installeras ett utvecklingsverktyg på utvecklingsdatorn. Med hjälp av utvecklingsverktyget kan man kompilera hela vertygskedjan. Efter kompilering kan man ladda över det färdiga programmet

till Olympus IV över nätverket. Egna program kompileras med hjälp av en korskompilator på utvecklingsdatorn och läggs sedan i filsystemet i verktygskedjan innan denna laddas till Olympus IV.

För att slippa bygga om filsystem och ladda om varje gång ett nytt program skulle testas användes Samba. En Sambamapp skapades på utvecklingsdatorn och kopplades till hemmamappen på plattformen. Ett program som låg på värddatorn kunde då köras i Olympus IV.

Kapitel 9

Avslutning

9.1 Slutsatser

I inledningen gavs två syften med examensarbetet, dels att testa Olympus IV för bildanalys och reglering samt att testa Linux lämplighet att användas som implementationsplattform för ett reglersystem. Slutsatsen är att Olympus IV med tillhörande implementerade funktioner, AVP, inte kan sägas vara lämplig för bildanalys. AVPn innehåller en del funktioner för att omvandla mellan olika bildformat, men inga direkta funktioner för att göra en bildanalys.

Under arbetets gång har en del problem uppstått med Linux. För att göra regleringen önskades att programmet skulle köra deterministiskt. Det vill säga att de olika regulatorerna skulle köra med fasta samplingstider i C++ programmet. För att göra detta användes så kallade Posix trådar. Programmet fick dock aldrig riktigt önskat resultat. Det största problemet låg i att få högprioriterade trådar att köra när de var redo. Ofta blev det en tidsfördröjning på mellan $0.01 - 0.04$ sekunder. En idé är att byta operativsystem till realtidslinux. I [21] undersöks tidsfördröjningar för olika Linux versioner. Tidsfördröjningarna avser den tid det tar från det kontrollalgoritmen vill börja tills att den börjar exekveras. Undersökningen visar att den maximala tidsfördröjningen kan minskas med mer än en faktor tio vid byte till realtidslinux. Slutsatsen är att ett byte till ett realtidsoperativsystem skulle kunna ge ett betydligt bättre resultat.

En annan idé för att förbättra systemet vore att implementera bildanalysen i kameran istället för på Olympus IV. Då skulle tiden för att hämta bilden försvinna och bara små mängder data hade behövts skickas mellan plattformen och kameran.

9.2 Förslag på vidareutveckling

- Vidareutveckla bildanalysen för att klara identifikation och följning i svårare miljöer.
- Implementera bildanalysen på kameran. Detta skulle leda till att man blev av med exekveringstiden för att hämta rådata bilden över nätverket. Det man skulle behöva skicka mellan plattformen och kameran skulle bara vara små variabler. Eftersom kameran är ett liknande system skulle det

inte vara speciellt komplext att implementera en liknande bildanalys på kameran.

- Installera en andra kamera för stereoseende. Med stereoseende skulle det vara lättare att identifiera objektet i en 3D miljö.
- Genom att lägga till en robotarm skulle man kunna få roboten att interagera med omgivningen. Exempel på användning vore att få roboten att städa ett rum.
- Konstruera en andra robot och få dem att kommunicera.
- Installera ett lämpligare realtidsoperativsystem. Med ett realtidsoperativsystem skulle det vara lättare att utveckla reglersystem med större krav på realtidsegenskaper.
- Byta Olympus IV mot ett lämpligare kort. Eftersom ISPn inte används speciellt finns det ingen anledning till att använda ett till ytan så stort kort. Det finns flera liknande kort med samma prestanda men som är betydligt mindre till ytan.

Litteraturförteckning

[1] S. Kang, J. Paik, A. Koschan, B. Abidi, M. A. Abidi *Real-time video tracking using PTZ cameras* Department of Electrical & Computer Engineering, University of Tennessee

[2] Sang Wook Lee, Kwangyoen Wohn *Tracking moving objects by a mobile camera*, University of Pennsylvania

[3] C. Schlegel, J.Illmann, H. Jaberg, M. Schuster, R. Wurz *Vision based person tracking with a mobile robot* Institute for applied knowledge processing, Ulm Germany

[4] Aksel Ethembabaoglu *Active target tracking using a mobile robot int the USARSim* Faculteit der natuurwetenschappen, Wiskunde en informatica

[5] Juhyun Lee and Peter Stone *Person tracking on a Mobile Robot with Heterogeneous Inter-Characteristic Feedback* The University of Texas at Austin

[6] Boyoon Jung Gaurav S. Sukhatme *Real Time Motion Tracking from a Mobile Robot* University of southern California

[7] A. Biswas, P. Guha, A. Mukerjee, K.S Venkatesh *Intrusion detection and tracking with pan-tilt cameras* Dept of Computer Sc. and Engg., IIT Kampus, India

[8] C. Tomasi och T. Kanade *Detection and tracking of points features* Caregie Mellon University, Pittsburgh, april 1991

[9] Stanley B. Lippman and Josee Lajoie and Barbara E.Moo. *C++ primer fourth edition.* Addison-Wesley, 2007.

[10] Åström K. J. and Wittenmark B. *Computer Controlled System.* Prentice Hall,

[11] P. Harbison III Samuel and Steele Jr. Guy. *C A Refrence Manual.* Prentice Hall

[12] Forsyth David and Ponce Jean. *Computer Vision a modern approach.* Prentice Hall, 2003

[13] Årzén Karl-Erik. *Real-Time Control Systems.* Department of Automatic Control, Lund Institute of Technology, Lund 2003

[14] *Linux,* http://sv.wikipedia.org/wiki/Linux, 2008-09-11

[15] *Linux* http://www.kernel.org, 2008-10-07

[16] Logipard *Developer's Manual* Revision 2, 2007

[17] *Mjpg streamer* http://sourceforge.net/projects/mjpg-streamer, 2008-09-12

[18] Tobias Hultén och Joakim Remne *Interaktiv mobil robot basrad på Olympus IV med inbyggt Linux* Combra och Instutitionen för Datavetenskap Lunds Tekniska högskola.

[19] *Datasheet Axis 215 PTZ network camera* http://www.axis.com/products/axis215/index.htm 2008-10-07

[20] *Faulhaber* http://www.faulhaber-group.com, 2008-09-22

[21] *Latency comparision* http://linuxdevices.com/articles/AT3479098230.html, 2008-10-14

[22] *FAC* http://www.facsystem.se, 2008-10-13