NTNU

Norwegian University of
Science and Technology

Tracking a Moving Object with an Industrial Manipulator using a Particle Filter

Håkon Hystad

Master of Science in Mechanical Engineering
Submission date: June 2018
Supervisor: Olav Egeland, MTP

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Acknowledgments

I would like to thank my supervisor Professor Olav Egeland for great insights and inspiration. I consider the opportunities and challenges he has presented me to have had great influence in my educational endeavours, for which I am very grateful. I would also like to thank Research Scientist Torstein Myhre for invaluable help and advise. His exemplary prior work is the sole reason this thesis could be written.

Last but not least a big thank you to my family and friends for their support and motivation, especially my wife for always being there for me.

For my children, Jacob and Signe.

Summary

As industries are getting more automated the demand for new sensor technologies and algorithms increases. Industrial manipulators have for instance been equipped with more sensors to handle increasingly complex tasks to further automate manufacturing. Particularly there is now a desire to handle objects with undetermined movements in real-time.

This thesis will present a way to estimate the position, orientation and velocities of a suspended object. This is accomplished by using a particle filter on image data to get the most likely estimates.

Some issues such as open-loop control and the sheer amount of particles needed for the filter have been tried mitigated through optimization. As the system is heavily dependent on camera calibration a new calibration routine and optimization was proposed and the particle filter was efficiently implemented on a graphical processing unit.

The proposed calibration optimization have been substantiated, through real-world experiments, to be less accurate than a usual form of calibration even though simulations proved it better. Luckily the traditional calibration was seen to have satisfactory accuracy. The filter have no problems running with 500 000 particles between frame captures at 33fps, which points to an efficient implementation.

To test the validity of the approach a case study was planned where the particle filter estimates was to be fed on-line to an industrial manipulator to follow a moving object with a fixed pose relative to the object itself. Tests showed that as long as the object is unique enough in the images it can be tracked with decent accuracy by the particle filter, they also showed that an industrial manipulator could be controlled smoothly by simulated filter output. The complete experiment was on the other hand never conducted due to time limitations and safety concerns.

Since the local coordinate system has been shown to be known and followed it is plausible that work could be done on the swinging and otherwise moving object as if it was at rest. With further improvements applications for this range from picking suspended parts of conveyors to complex tasks performed on the object while it is suspended on an assembly line.

Sammendrag

Behovet for nye algoritmer og sensorteknologier øker stadig ettersom flere og flere arbeidsoppgaver ute i industrien blir automatisert. For eksempel har robotarmer blitt utrustet med flere sensorer for å håndtere mer komplekse oppgaver for videre automatisering. Et mye ønsket bruksområde for roboter er å håndtere fritt bevegelige objekter i sanntid.

Denne avhandlingen presenterer en måte å estimere posisjon, orientering og hastigheter av et objekt med et enkelt opphengspunkt. Et partikkelfilter blir benyttet på bildedata for å finne de mest sannsynlige tilstandene av systemet.

Enkelte hindringer slik som åpen regulering og det store antallet partikler er forsøkt redusert ved hjelp av optimaliseringer. Systemet er svært avhengig av god kalibrering og en automatisk kalibreringsrutine og en ny optimalisering har derfor blitt utarbeidet. I tillegg har partikkelfilteret blitt implementert og optimalisert for kjøring på grafikkort.

Gjennom eksperimenter har den foreslåtte kalibreringen blitt sannsynliggjort til å være dårligere enn en typisk fremgangsmåte tatt i dag, dette til tross for lovende simuleringsresultater. Heldigvis har den tradisjonelle kalibreringen vist seg å være tilstrekkelig god. Det implementerte partikkelfilteret har heller ingen problemer med å bruke 500 000 partikler for estimering mer enn 33 ganger per sekund, noe som viser til en effektiv implementasjon.

For å teste bruksverdien av den gitte fremgangsmåten ble det planlagt et forsøk hvor robotarmen skulle følge et bevegelig objekt med en konstant posisjon og orientering relativt til objektet. Resultatene av testene viste at så lenge objektet kunne gjenkjennes godt i bildet kunne det følges med tilfredsstillende nøyaktighet av partikkelfilteret. Testene viste også at en robotarm kunne bli kontrollert on-line av simulerte filter estimater. Det komplette systemet ble i midlertid aldri testet på grunn av tidsmangel og sikkerhetshensyn.

Siden det lokale koordinatsystemet til objektet kan bli fulgt er det sannsynlig at arbeid kan bli utført som om objektet stod stille til tross for svingende og lineære bevegelser. Ved videre forbedringer kan systemet benyttes til alt fra plukking av opphengte deler til mer komplekse arbeidsoppgaver på et objekt mens det henger på et samlebånd.

Contents

List of Figures

List of Tables

1. Introduction

As technology advances the production in different industries become more automated. Today robotic manipulators is in widespread use in many instances of society and a cornerstone in cost-effective manufacturing for many organizations.

Use of industrial manipulators are steadily increasing and new solutions are needed to accommodate the new found demand. For instance is adding visual sensing to industrial equipment becoming more and more common in applications like robotic welding and inspection. One of the desirable extensions to a manipulator is to handle moving objects which gives nearly endless possibilities for new uses. By adding tracking capabilities to a manipulator it is now possible to deal with moving or complex objects in new ways.

Object tracking in computer vision is a vast field[2] which still see regular breakthroughs. Popular algorithms for robust tracking include the extended Kalman filter[13] and particle filters[24]. Filters allow for the pose as well as velocities of an object to be estimated for each frame of an image sequence. Using these estimates an industrial manipulator should be able to perform work on the object as if it was at rest.

To demonstrate the idea a case study of a hanging conveyor is considered with the purpose of picking suspended parts from a moving production line. The problem at hand is rooted in a real industrial need which could open for more automation in manufacturing applications which are now dominated by manual labor. This master thesis will present a way to track and compensate for movements of an object, suspended from a moving conveyor in a single point, using a particle filter. It is an extension of the work done by Torstein Myhre[20],[28] by adding movement of the pivot in 3D space.

The problem to be solved is first presented together with the proposed solution, later a thorough explanation on how the particle filer works and the steps taken to implement it on a graphical processor is given. Next the focus shifts to camera calibration where an automatic routine as well as a novel optimization technique is proposed. Finally the results of both calibration and tracking are presented by simulations as well as real world experiments.

2. Objective and proposed solution

The objective of this thesis is to accurately estimate the position and orientation (pose) as well as the velocities of an objects local coordinate system, later referred to as the pivot frame, with respect to a fixed world frame. The estimates are then to be used by an industrial manipulator to compensate/follow the motion of the object. An integral part of the system is the estimation by visual sensing.

A tried and true method of using computer vision in robotics is visual servoing[3], either eye-in-hand or eye-to-hand which are closed-loop and open-loop systems respectively. The image information is typically used to minimize the error between a desired pose on the object and the end-effector of the robot. Image based servoing simply minimizes the error to the feature in the image plane and can therefor be robust, yet limiting in its application. Position based servoing, where the pose of the object is estimated by the camera, gives greater freedom as the manipulator can move to any pose relative to the object but often lacks the robustness of closed-loop feedback. Position based servoing often rely on computation on extracted features of the image which can also be prone to noise.

The wider range of applications for a position based approach makes it desirable, and an alternative to feature extraction and calculations are filters. Filters are a probabilistic way of determining the implicit states of a system using explicit measurements. Kalman filters[30] assume a Gaussian distribution of the states to calculate an analytic solution to the most likely estimate given a set of measurements. Particle filters[5] on the other hand takes a more general approach by introducing several hypothesis and weighting them according to the measurements which lets it represent arbitrary distributions. The generality of the particle filter is an attractive but more computationally expensive property, however as computational power increases this issue diminishes.

Even though the filters are capable of estimating the states directly the drawback of an open-loop system is still present and therefore great care must be taken to calibrate the cameras accurately. It is here proposed to calibrate the cameras with respect to the base frame of the manipulator performing the work, this allows for automatic and simultaneous calibration of both cameras. The tracking will also be performed with respect to the same base.

To estimate the pose and velocities a particle filter was chosen for robust and general tracking. Particle filters have the nice property of being widely applicable with no limitations or assumptions such as linearity or uni-modal distributions. The particle filter presented in this thesis will be based on sequential importance sampling (SIR) with resampling and will estimate the most likely state vector

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \alpha & \beta & \gamma & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix} \tag{2.1}$$

where x, y, z are the Cartesian coordinates in a three dimensional space while α, β, γ are the XYZ-euler angles for minimal representation of the orientation. The dot-notation represents the time derivative of the marked entity; linear and rotational velocities respectively. All states are with respect to the base frame of the manipulator performing the work. The estimates are to be calculated in real-time on images taken at 30-60fps and a predictive model used to achieve a suitable refresh rate to update the configuration of a 6DOF manipulator.

3. Particle filter

The premise of a particle filter is the ability to represent a probability density function (pdf) by a set of weighted samples (particles). If each particle holds a hypotheses for the state and can be weighted based on sensor data the represented pdf can give the expected, most likely, value of the states. This stands in contrast to the Kalman filter which assumes Gaussian distribution to calculate a single analytic solution.

The Kalman filter has seen many attempted improvements to handle non-linear problems such as the extended Kalman filter[30] which linearises the problem beforehand. However one inherent limitation of the Kalman filter is that it expects a single bell-curve and can therefore only represent a single hypotheses making it less robust when several states have high probabilities.

Using particles as a pdf representation allows for arbitrary, multi-modal distributions which handles all the limitations of the Kalman filter in a simple manner. This off-course comes at a cost, and a large amount of particles is often needed to represent a high-dimensional problem.

To advance the particles to the next step a motion model, also known as a dynamic model, must be used. The motion model incorporates the known behavior of the system to predict the next state. To evaluate the likelihood/quality of the predicted state the observation model which incorporates sensor data is used to weight each particle. The particle filter described in this thesis is the sequential importance sampling with re-sampling (SIR) scheme:

Begin: with weighted samples from step n-1 $p(x_n|y_{n-1})$

Drift: Apply motion model (no noise) to every particle to predict

Diffuse: Apply noise to spread particles and account for inaccuracy of the motion model or random movement

Measure: Assign weights using the likelihood/observation model on each hypothesized state (particle) $p(y_n|x_n)$.

Resample: Draw samples, with replacement, from the previous set according to these weights (small weighted particles will most likely be discarded)

Finish: Estimate density/expectation and repeat.

3.1. The motion model

Since the conveyors have hangers fixed in one rotating point a natural thing to do is using its center of mass and model it as a spherical pendulum with a moving pivot. Friction in the joint, air resistance and any other force other than gravity is not accounted for and so must be compensated for by adding (Gaussian) noise.

The orientation of the center of mass is estimated beforehand (see section 3.1.4) as angular offsets around the pivot frames x- and y-axis, $\theta_\alpha, \theta_\beta$ respectively. While the length to the pivot origin is estimated as l. Now consider the position and orientation of the pivot frame to be found with respect to a fixed world frame, $\mathcal{W}$. The mass center is then found in $\mathcal{W}$ by applying $\alpha = \alpha_{\text{state}} + \theta_\alpha, \quad \beta = \beta_{\text{state}} + \theta_\beta$ to the Euler angles estimated by the particle filter. The position of the pivot in $\mathcal{W}$ is

$$\boldsymbol{t} = \begin{bmatrix} x_{\text{state}} & y_{\text{state}} & z_{\text{state}} \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{3.1}$$

while the orientation of the mass center is

$$\boldsymbol{R}_X(\alpha)\boldsymbol{R}_Y(\beta)\boldsymbol{R}_Z(\gamma) = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix} = \begin{bmatrix} c_\beta c_\gamma & -c_\beta s_\gamma & s_\beta \\ c_\alpha s_\gamma + s_\alpha s_\beta c_\gamma & c_\alpha c_\gamma - s_\alpha s_\beta s_\gamma & -s_\alpha c_\beta \\ s_\alpha s_\gamma - c_\alpha s_\beta c_\gamma & s_\alpha c_\gamma + c_\alpha s_\beta s_\gamma & c_\alpha c_\beta \end{bmatrix} \tag{3.2}$$

Here the rotation matrix is given as the X-Y-Z Euler angles where s_ϕ and c_ϕ is short for sine and cosine of an arbitrary variable ϕ. The directional cosine $\boldsymbol{r}_3$ gives the orientation of the mass center z-axis which is assumed to point in the same general direction as the fixed z-axis in $\mathcal{W}$ ($\beta \in (-\pi/2, \pi/2)$). The position of the mass center can then be expressed as

$$\boldsymbol{p}_m = \boldsymbol{t} - l \cdot \boldsymbol{r}_3 \tag{3.3}$$

where l is the known, constant, length of the pendulum. Likewise the velocity of the

mass center is

$$\begin{aligned}\dot{\boldsymbol{p}} &= \dot{\boldsymbol{t}} - l \cdot \dot{\boldsymbol{r}}_3 \\ &= \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} - l \begin{bmatrix} \dot{\beta} c_\beta \\ \dot{\beta} s_\alpha s_\beta - \dot{\alpha} c_\alpha c_\beta \\ -(\dot{\alpha} s_\alpha c_\beta + \dot{\beta} c_\alpha s_\beta) \end{bmatrix}\end{aligned} \tag{3.4}$$

Assuming the gravitational force is aligned with the fixed z-axis, i.e $\boldsymbol{g} = -g\boldsymbol{k}$, the potential energy of the pendulum is

$$U = mgh = mg(l - l\boldsymbol{r}_3^T \boldsymbol{k}) = mg(l - lc_\alpha c_\beta) \tag{3.5}$$

While the kinetic energy is

$$\begin{aligned}T &= \frac{1}{2} m v^2 = \frac{1}{2} m \boldsymbol{v}^T \boldsymbol{v} \\ &= \frac{1}{2} m \left(\left(\dot{x} - l(\dot{\beta} c_\beta) \right)^2 + \left(\dot{y} - l(\dot{\alpha} c_\alpha c_\beta - \dot{\beta} s_\alpha s_\beta) \right)^2 + \left(\dot{z} + l(\dot{\alpha} s_\alpha c_\beta + \dot{\beta} c_\alpha s_\beta) \right)^2 \right) \\ &= \frac{1}{2} m \left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2 + 2l \left[\dot{\alpha} c_\beta (\dot{y} c_\alpha + \dot{z} s_\alpha) - \dot{\beta} s_\beta (\dot{y} s_\alpha - \dot{z} c_\alpha) - \dot{\beta} \dot{x} c_\beta \right] + l^2 (\dot{\beta}^2 + \dot{\alpha}^2 c_\beta^2) \right) \\ &= \frac{1}{2} m \left(\dot{\boldsymbol{t}}^T \dot{\boldsymbol{t}} + 2l \left[\dot{\alpha} c_\beta (\dot{y} c_\alpha + \dot{z} s_\alpha) - \dot{\beta} s_\beta (\dot{y} s_\alpha - \dot{z} c_\alpha) - \dot{\beta} \dot{x} c_\beta \right] + l^2 (\dot{\beta}^2 + \dot{\alpha}^2 c_\beta^2) \right)\end{aligned} \tag{3.6}$$

Having the potential and kinetic energy the Lagrangian can be built as

$$\begin{aligned}L &= T - U \\ &= \frac{1}{2} \left(\dot{\boldsymbol{t}}^T \dot{\boldsymbol{t}} + 2l \left[\dot{\alpha} c_\beta (\dot{y} c_\alpha + \dot{z} s_\alpha) - \dot{\beta} s_\beta (\dot{y} s_\alpha - \dot{z} c_\alpha) - \dot{\beta} \dot{x} c_\beta \right] + l^2 (\dot{\beta}^2 + \dot{\alpha}^2 c_\beta^2) \right) + g l c_\alpha c_\beta\end{aligned} \tag{3.7}$$

where the shared constant m and the constant term $-gl$ are removed since they are of no interest in further calculations. The Euler-Lagrange equations of motion

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\phi}} \right) - \frac{\partial L}{\partial \phi} = 0, \quad \phi = \alpha, \beta \tag{3.8}$$

can then be used to find the angular accelerations.

3.1.1. Pendulum acceleration

Using (3.8) on (3.7) with respect to $\alpha, \dot{\alpha}$ yields

$$\begin{aligned}
\frac{\partial L}{\partial \alpha} &= l\left(\dot{\alpha}c_\beta(\dot{z}c_\alpha - \dot{y}s_\alpha) - \dot{\beta}s_\beta(\dot{y}c_\alpha + \dot{z}s_\alpha) - gs_\alpha c_\beta\right) \\
\frac{\partial L}{\partial \dot{\alpha}} &= lc_\beta\left(\dot{y}c_\alpha + \dot{z}s_\alpha + \dot{\alpha}lc_\beta\right) \\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}}\right) &= \frac{\partial L}{\partial \alpha} + gls_\alpha c_\beta + lc_\beta\left(\ddot{\alpha}lc_\beta + \ddot{y}c_\alpha + \ddot{z}s_\alpha - \dot{\alpha}\dot{\beta}\cdot 2ls_\beta\right) \\
&\Rightarrow lc_\beta\left(\ddot{\alpha}lc_\beta + \ddot{y}c_\alpha + \ddot{z}s_\alpha - \dot{\alpha}\dot{\beta}\cdot 2ls_\beta + gs_\alpha\right) = 0 \\
&\Leftrightarrow \ddot{\alpha} = 2\dot{\alpha}\dot{\beta}\tan\beta - \frac{\ddot{y}c_\alpha + \ddot{z}s_\alpha + gs_\alpha}{lc_\beta}
\end{aligned} \tag{3.9}$$

Doing the same for $\beta, \dot{\beta}$ gives

$$\begin{aligned}
\frac{\partial L}{\partial \beta} &= l\left(\dot{\beta}\dot{x}s_\beta - \dot{\alpha}s_\beta(\dot{y}c_\alpha + \dot{z}s_\alpha) - \dot{\beta}c_\beta(\dot{y}s_\alpha - \dot{z}c_\alpha) - \dot{\alpha}^2 ls_\beta c_\beta + gc_\alpha s_\beta\right) \\
\frac{\partial L}{\partial \dot{\beta}} &= l\left(\dot{\beta}l - \dot{x}c_\beta - s_\beta(\dot{y}s_\alpha - \dot{z}c_\alpha)\right) \\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\beta}}\right) &= \frac{\partial L}{\partial \beta} + \dot{\alpha}^2 l^2 s_\beta c_\beta + glc_\alpha s_\beta + l\left(\ddot{z}c_\alpha s_\beta - \ddot{x}c_\beta - \ddot{y}s_\alpha s_\beta + \ddot{\beta}l\right) \\
&\Rightarrow l\left(\dot{\alpha}^2 ls_\beta c_\beta + gc_\alpha s_\beta + \ddot{z}c_\alpha s_\beta - \ddot{x}c_\beta - \ddot{y}s_\alpha s_\beta + \ddot{\beta}l\right) = 0 \\
&\Leftrightarrow \ddot{\beta} = \frac{\ddot{x}c_\beta + (\ddot{y}s_\alpha - (\ddot{z}+g)c_\alpha)\,s_\beta}{l} - \frac{1}{2}\dot{\alpha}^2\sin(2\beta)
\end{aligned} \tag{3.10}$$

3.1.2. Pivot acceleration

Since the pivot is fixed on a conveyor it is assumed that motion of the mass-center will not affect the pivot. Further more the conveyor has close to constant speed; the only acceleration comes from gradual changes in direction compensated for as noise only. From the angular acceleration expressions it is seen that this simplifies to the case of a static pivot as expected.

3.1.3. Velocities and pose

To get the new states based on the prior ones, both pose and velocities, numerical integration is applied to the acceleration expressions outlined above. Since the time interval to integrate over is known to be small (ideally 1/60s for 60 frames per second) a small step size can be chosen while keeping a manageable amount of iterations. The

need for higher order integration techniques for accuracy is then reduced. Couple this with the fact that the solution is not directly applied as the final state estimate it is proposed to only use a simple numerical integration scheme for speed purposes. One simple, first order, form of numerical integration is the forward Euler method which runs with a complexity $O(N)$. It uses the finite difference approximation to the derivative:

$$g'(t) \approx \frac{g(t+\Delta t) - g(t)}{\Delta t} \Leftrightarrow g(t+\Delta t) \approx g(t) + \Delta t g'(t)$$

where Δt is the time step. The linear accelerations are all assumed zero which gives the trivial results

$$\dot{x}_{k+1} = \dot{x}_0 \tag{3.11}$$
$$\dot{y}_{k+1} = \dot{y}_0 \tag{3.12}$$
$$\dot{z}_{k+1} = \dot{z}_0 \tag{3.13}$$

at step $k+1$. At the same time the position coordinates are

$$x_{k+1} = x_k + \Delta t \dot{x}_0 = x_0 + (k+1)\Delta t \dot{x}_0 \tag{3.14}$$
$$y_{k+1} = y_k + \Delta t \dot{y}_0 = y_0 + (k+1)\Delta t \dot{y}_0 \tag{3.15}$$
$$z_{k+1} = z_k + \Delta t \dot{z}_0 = z_0 + (k+1)\Delta t \dot{z}_0 \tag{3.16}$$

The more interesting case is the angular velocities and orientation:

$$\dot{\alpha}_{k+1} = \dot{\alpha}_k + \Delta t \ddot{\alpha} = \dot{\alpha}_k + \Delta t \left(2\dot{\alpha}_k \dot{\beta}_k \sin\beta_k - \frac{g}{l} \sin\alpha_k \right) \frac{1}{\cos\beta_k} \tag{3.17}$$
$$\dot{\beta}_{k+1} = \dot{\beta}_k + \Delta t \ddot{\beta} = \dot{\beta}_k + \Delta t \left(\frac{g}{l} \cos\alpha_k - \dot{\alpha}_k^2 \cos\beta_k \right) \sin\beta_k \tag{3.18}$$
$$\dot{\gamma}_{k+1} = \dot{\gamma}_k = \dot{\gamma}_0 \tag{3.19}$$

With the expressions from (3.9) and (3.10) substituted for the angular accelerations. The Euler angles are

$$\alpha_{k+1} = \alpha_k + \Delta t \dot{\alpha}_k \tag{3.20}$$
$$\beta_{k+1} = \beta_k + \Delta t \dot{\beta}_k \tag{3.21}$$
$$\gamma_{k+1} = \gamma_0 + (k+1)\Delta t \dot{\gamma}_0 \tag{3.22}$$

The choice of step size depends on the desired accuracy, but it is also important to note that the round-off error of the floating point numbers will contribute to the inaccuracy of the result, its error is approximately given as $\frac{\varepsilon}{\sqrt{\Delta t}}$ where ε is the machine precision (for a single precision this is around 10^{-8}). This means that too small step sizes will not

have the increase in accuracy as intended. The effect of step size can be seen in Fig. 3.1. While the much used 4'th order Runge-Kutta method converges close to the analytic solution already at a step size of 1 the forward Euler method follow suit when the step size is reduced.

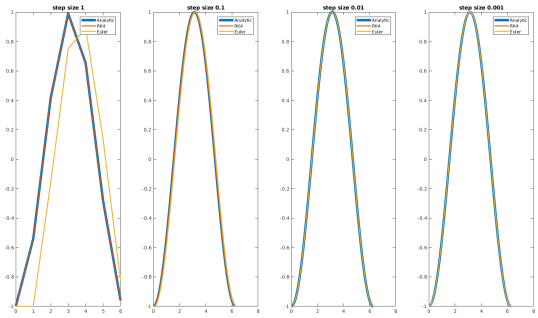

Figure 3.1.: Numerical integration of $\sin(t), t \in [0, 2\pi]$ using 4'th order Runge-Kutta and the forward Euler methods.

It is seen (and quantitatively verified) that there is almost no difference between a step size of 10^{-2} and 10^{-3}, but this is problem specific. Doing the same comparison with the expressions for $\alpha, \beta, \dot{\alpha}, \dot{\beta}$ using the Runge-Kutta 4th order method with a small step size as ground truth reveals pretty much the same convergence. This can be seen in Fig. 3.2; the Euler method only starts behaving badly as time progresses way past the expected interval.

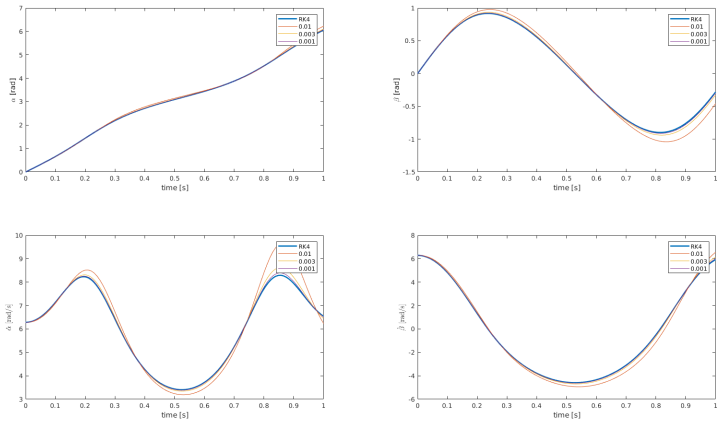

Figure 3.2.: Numerical integration with different step sizes for the Euler method at 1Hz.

A step size between 10^{-2} and 10^{-3} seems reasonable, therefore a step size of $\Delta t = 3 \cdot 10^{-3}$

is chosen and the number of iteration in the ideal case becomes

$$\frac{1/60}{\Delta t} = 5.56$$

Six to twelve iterations is therefore the expected outcome, which is pretty fast. In contrast choosing the step size only slightly smaller, say 10^{-3}, results in 3 times as many iterations. The effects of the different step sizes over a 33fps interval are shown in Fig. 3.3.

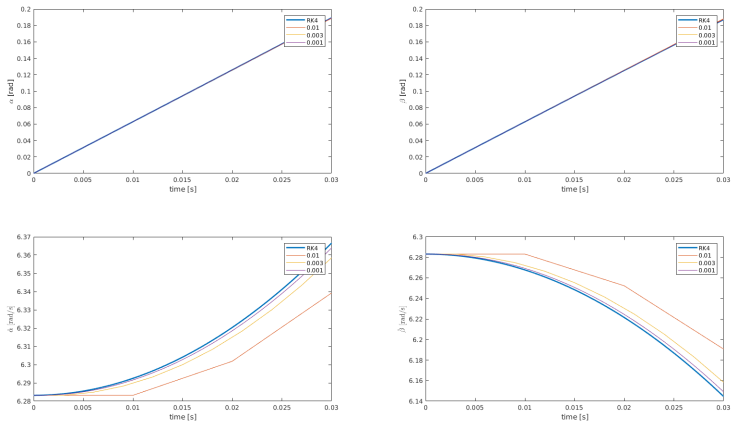

Figure 3.3.: Close up of numerical integration using different step sizes for the Euler method at 1Hz.

It is seen that the Euler angles are spot on while the angular velocities have an inaccuracy well below a degree per second for $\Delta t = 0.003$, the worst case scenario. This is more than acceptable for a probabilistic measure. It is noted that the constant velocities let 8 of the 12 state parameters be found without integration as

$$\dot{\phi}' = \dot{\phi}$$
$$\phi = \phi + t_{12}\dot{\phi}$$

Where $\dot{\phi}'$ and ϕ' denote the new value of an arbitrary variable $\dot{\phi}$ and ϕ respectively while t_{12} is the time interval between state estimates. Finally the state parameters are updated to reflect the orientation of the pivot instead of the center of mass using

$$\alpha_{\text{state}} = \alpha - \theta_\alpha$$
$$\beta_{\text{state}} = \beta - \theta_\beta$$

3.1.4. Parameter estimation

To properly use the motion model the static parameters $\theta_\alpha, \theta_\beta$ and l describing the position of the center of mass in the pivot frame must be known with some certainty. There are numerous ways of estimating the center of mass of an object in the literature, [18] and [19] even describes how a particle filter can be utilized on a dynamical system closely related to that of this thesis.

As tempting as it might be to add the three parameters to the state vector of the particle filter it is not recommended. For one the added length of the already large state vector is undesirable and will definitively add considerably to the amount of particles needed. The main problem however is that the parameters are static; no dynamics/update equation exists. Off course a random walk can and has been used in such instances. On-line methods for static parameter estimation typically use expectation maximization of the smoothed filter output such as in [26] or stochastic gradient search methods as in [23] according to [15],[20].

Since the experiments in this thesis where performed on a simple hanger with constant parameters it was opted to use predetermined parameters found by the simplest means possible to reduce the scope of the work.

Angular offsets

The proposed particle filter when finished already provides a method to determine $\theta_\alpha, \theta_\beta$. When the object is kept at rest and the filter converges on it the offset in α_{state} and β_{state} from 0 is simply θ_α and θ_β respectively. One can argue that utilizing the particle filter this way produces the best parameters possible as it is more important to get good compliance with the filter rather than the actual values.

Length

To get information on the arm between pivot and center of mass the object must the moving. Since the system is modelled as a spherical pendulum it is sufficient to do a simple analyses on the period of oscillation to determine the length of the arm. To gain the most simple form the movement of the pendulum can be restricted to a single axis resulting in 1 degree of freedom. The small-angle approximation can then be used to get Huygens's law of the period:

$$T_0 = 2\pi\sqrt{\frac{l}{g}}$$

where T_0 is the initial period of the oscillating motion. The aforementioned equation yields the approximation

$$l \approx \frac{T_0^2}{4} \tag{3.23}$$

As $\frac{g}{\pi^2} \approx 1$. Thus a simple measurement of the period is sufficient to determine the length of the pendulum to an acceptable accuracy.

Further exploration of parameter estimation is not provided in this thesis but it should be noted that in a changing system, such as loading and unloading of hanging conveyors, a proper way of performing estimation on-line will most likely be needed to ensure reliable tracking.

3.2. Observation model

When the particles have been moved based on the prediction of the motion model and noise has been added each particle can be evaluated against the measured data. There are two types of observation models: discriminate and generative. The discriminate model will use features and classifiers to find the object in the image while the generative one will look for the most similar region to a model/template. For simplicity the generative model is chosen here.

The task is then to build the object on the image plane in the predicted pose and check if it matches up with the image information. There are several ways of doing this, for instance the original CONDENSATION paper [14] finds an initial contour in the image and performs affine transformations on it using the states of each particle. Another way is to project the geometry of the object onto the image according to the camera parameters and the pose contained in each particle. The availability of the (3D) geometry to the tracking object and being the more direct route makes the last approach the most attractive.

3.2.1. Projecting a 3D object

Taking an image is the process of projecting a world point onto the image plane parallel, and with a distance f, to the xy-plane of the camera frame. f is often referred to as the

focal length. This can be modeled as

$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{I} \mid \boldsymbol{0}]\boldsymbol{X}^c = \boldsymbol{P}^c\boldsymbol{X}^c, \quad \boldsymbol{K} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here $\boldsymbol{K}$ is the upper triangular camera calibration matrix dependent on intrinsic values of the camera found from camera calibration. f_u and f_v is the focal length expressed in the units of pixel width and height respectively. The effect of non-square pixels is a different scaling in the two directions. u_0 and v_0 is the translation from the principal point (where the z-axis of the camera frame intersect the image plane) to the image frame, also they are given in pixel units.

$\boldsymbol{P}^c = \boldsymbol{K}[\boldsymbol{I}|\boldsymbol{0}]$ is a 3×4 matrix that will project the homogeneous 3D point $\boldsymbol{X}^c$ given with respect to the camera frame. The projection of a world point, $\boldsymbol{X}$, given in a world coordinate system, w, is therefore a Euclidean transformation away:

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{P}^c\boldsymbol{T}_w^c\boldsymbol{X} = \boldsymbol{P}^c \begin{bmatrix} \boldsymbol{R}_w^c & -\boldsymbol{R}_w^c\boldsymbol{t}_{wc}^w \\ \boldsymbol{0}^T & 1 \end{bmatrix} \boldsymbol{X} = \boldsymbol{P}\boldsymbol{X} \\ &\Rightarrow \boldsymbol{P} = \boldsymbol{K}\boldsymbol{R}[\boldsymbol{I}| - \boldsymbol{t}] \end{aligned} \tag{3.24}$$

Where $\boldsymbol{R} = \boldsymbol{R}_w^c$ and $\boldsymbol{t} = \boldsymbol{t}_{wc}^w$ gives the pose of the camera in the world frame which can be found through camera calibration. Each particle will speculate over a pose of the pivot frame which the object can be placed in by known measurements, it is then only a question of measuring discrete points of the image where the object is hypothesized to be.

3.2.2. Sampling the object

To be able to discretize the object N samples must be drawn from it. An object described by straight line segments is considered from here on. Line segment $i = 1, \ldots, n$ is given by the two homogeneous endpoints in the pivot frame

$$\boldsymbol{A}_i, \boldsymbol{B}_i \in \mathbb{P}^3(\mathbb{R}^3), \quad i = 1, 2, \ldots, n \tag{3.25}$$

Going via the world frame the points can be found in the image plane as the homogeneous coordinates

$$\tilde{\boldsymbol{a}}_i = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}^T = \boldsymbol{P}\boldsymbol{T}_p^w\boldsymbol{A}_i, \quad \tilde{\boldsymbol{b}}_i = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}^T = \boldsymbol{P}\boldsymbol{T}_p^w\boldsymbol{B}_i$$

where $\boldsymbol{T}_p^w$ takes the points from the local pivot frame to the fixed world frame, this comes from the state estimate of the particle. In non-homogeneous pixel coordinates this equates to

$$\boldsymbol{a}_i = \begin{bmatrix} a_1/a_3 \\ a_2/a_3 \end{bmatrix}, \quad \boldsymbol{b}_i = \begin{bmatrix} b_1/b_3 \\ b_2/b_3 \end{bmatrix} \tag{3.26}$$

assuming that the points are not at infinity, i.e $a_3 \neq 0$. A step vector for uniform sampling can then be constructed as

$$\left(1 - \frac{j}{m_i + 1}\right) \boldsymbol{a}_i + \frac{j}{m_i + 1} \boldsymbol{b}_i, \quad j = 1, 2, \ldots, m_i \tag{3.27}$$

Where m_i is the total number of samples along the line and j is the sample number. m_i, can be predetermined and chosen explicitly for each line segment.

In the case of rotations, edges farther from the axis of rotation moves a greater distance and gives a larger source of information to the edge detectors than those close to it and should therefore count more. To account for this either the edge response could be weighted or the number of samples for each edge, m_i, could be chosen accordingly. Although the latter option seems more desirable to reduce the computations it may not be the best option for parallel execution due to the GPU architecture. A number of samples of each edge as a multiple of warp size (section 4.2) is the best option in that case.

One simple weighting algorithm is to use the mean distance, i.e the mid-point on the line, to the pivot so that the weights are determined at compile-time/startup. The weight on edge i is

$$\hat{l}_i = \left\| \frac{1}{2} \left(\boldsymbol{B}_i + \boldsymbol{A}_i \right) \right\|$$

When normalized by

$$l_i = \frac{\hat{l}_i}{\sum_k \hat{l}_k}$$

If a variable number of samples is desired it can be used to give each line segment a integer fraction of the (nominal) total number of samples N_{samples} using

$$m_i = round(l_i \cdot N_{\text{samples}}) \tag{3.28}$$

The normal vector of line segment in the image can be found as $(\boldsymbol{b}_i - \boldsymbol{a}_i)^T \boldsymbol{n}_i = 0$ giving

$$\boldsymbol{n}_i = \begin{bmatrix} \frac{a_2}{a_3} - \frac{b_2}{b_3} \\ \frac{b_1}{b_3} - \frac{a_1}{a_3} \end{bmatrix} \tag{3.29}$$

Which of course can be normalized by $\boldsymbol{n}_i / \|\boldsymbol{n}_i\|$ and will become useful in the next section.

3.2.3. Detecting an object

There are several ways of describing if the object is at hypothesized position in the image, some examples include keypoints, colors, templates and edges. A simple robust descriptor is an edge which is more invariant to changes in lighting and texture than many other methods. A very common way to perform edge detection is the Sobel filter. A Sobel filter works by applying a kernel, $\boldsymbol{M}$, and its transpose on points centered around $\boldsymbol{x}$

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$\boldsymbol{M}$ and $\boldsymbol{M}^T$ will weight and sum the intensities and it is seen that a mostly uniform intensity area results in small responses, e_v, e_h. Normally $\boldsymbol{M}$ is applied to find vertical peaks while $\boldsymbol{M}^T$ mark the horizontal ones, the real edge is then characterized by the Pythagorean theorem, i.e $\sqrt{e_v^2 + e_h^2}$. For an oriented kernel a single kernel turned θ degrees is sufficient: $\hat{\boldsymbol{M}} = \boldsymbol{M}^T \cos\theta + \boldsymbol{M} \sin\theta$. Recognizing that the coefficients is the elements of the unit normal of an edge $\boldsymbol{n} = [n_1, n_2]^T$ a simple oriented kernel can be constructed as

$$\begin{aligned} \hat{\boldsymbol{M}} &= \boldsymbol{M}^T \cdot n_1 + \boldsymbol{M} \cdot n_2 \\ &= \begin{bmatrix} n_1 & 2n_1 & n_1 \\ 0 & 0 & 0 \\ -n_1 & -2n_1 & -n_1 \end{bmatrix} + \begin{bmatrix} n_2 & 0 & -n_2 \\ 2n_2 & 0 & -2n_2 \\ n_2 & 0 & -n_2 \end{bmatrix} = \begin{bmatrix} n_1 + n_2 & 2n_1 & n_1 - n_2 \\ 2n_2 & 0 & -2n_2 \\ -(n_1 - n_2) & -2n_1 & -(n_1 + n_2) \end{bmatrix} \end{aligned}$$

A less computationally expensive way was proposed by Myhre[28]; Measure the image intensity where the edge point is believed to be, and ca λ pixels away normal to the edge. If the difference in intensity is large it is more likely to be an edge. This way a

cost function can be formulated for m_i samples of edge $i = 1, \ldots, N$

$$c_i = w_i \sum_{j=1}^{m_i} \left(\mathcal{I}(\boldsymbol{p}_j) - \mathcal{I}(\boldsymbol{p}_i - \lambda \boldsymbol{n}_j)\right)^2$$

where $\mathcal{I}(\boldsymbol{p})$ represents the intensity at position $\boldsymbol{p}$, found along the step vector (3.27) and w_i is the edge weight (3.28). Myhre reports $\lambda = 5$ to be a reasonable value. The cost function lets all discrete measuring points quickly rate the particle, it should be noted that the cost function is bias to contrast rich edges, especially because of the squaring term. It can also be sensitive to noise in the image since single pixels are compared, hitting white noise would for instance contribute heavily to the cost in most circumstances. The hope is that taking enough measurements will compensate for single miss-represented edge points.

An arguably better variant closer to the Sobel filter would be

$$c_i = w_i \sum_{j=1}^{N} \left(\mathcal{I}(\boldsymbol{p}_j + \lambda \boldsymbol{n}_i) - \mathcal{I}(\boldsymbol{p}_j - \lambda \boldsymbol{n}_i)\right)^2 \tag{3.30}$$

Which makes a measurement on each side of the edge.

3.2.4. Projective ambiguity

Even for a calibrated camera the properties in the image can only be determined up to a similarity transform for a single image[10]. Since the geometry of the object is known there is enough extra information to reproduce the image in the correct scale as well, in theory. There is however a problem with accuracy in the single camera approach.

Consider for instance the case when a planar object is rotated around the cameras x- and y-axis. For moderate rotations the protective property (making parallel lines cross) is rather small and the object will still appear to be an orthogonal projection only slightly smaller. This reintroduces the scaling issue some what as particles hypothesizing the object to be farther from the camera instead of rotated will also produce a likely observation.

A small ambiguity in scale can pose convergence problems for the filter making it less efficient and robust. A good solution to minimize this issue is utilizing more cameras from different viewpoints to reduce the observation response of bad particles that appear good from a single point of view. An added bonus of using more viewpoints is the possibility of increasing the frame-rate.

3.3. Hidden Markov Models (HMM) and Bayesian inference

Given the same motion -and observation model for each particle there is now a need of a mathematical framework to take advantage of this information. The bootstrap/SIR particle filter framework was first introduced in [8] and a tutorial is given in [5]. Say a large number of particles all share the same motion model, in the application case it uses the equations of motions which is only dependent on the previous state making it a Markov process. This is described by a random variable and the realization of the previous state as

$$X_n|(X_{n-1} = x_{n-1}) \sim f(x_n|x_{n-1}) \tag{3.31}$$

Distributed by the state transition function/pdf $f(x_n|x_{n-1})$. The problem is that this distribution is only implicitly measured through the realization of the observation model

$$Y_n|(X_n = x_n) \sim g(y_n|x_n) \tag{3.32}$$

Distributed according to $g(y_n|x_n)$. This is therefore called a Hidden Markov model.

The system has an initial state X_1 distributed according to a probability density function $\mu(x_1)$. As it stands (3.31) defines the prior distribution of the current state X_n with

$$p(x_{1:n}) = \mu(x_1) \prod_{k=2}^{n} f(x_k|x_{k-1}) \tag{3.33}$$

While (3.32) define the likelihood function

$$p(y_{1:n}|x_{1:n}) = \prod_{k=1}^{n} g(y_k|x_k) \tag{3.34}$$

To merge the two expressions the Bayesian formula can be used to calculate the posterior distribution as

$$p(x_{1:n}|y_{1:n}) = \frac{p(x_{1:n})p(y_{1:n}|x_{1:n})}{p(y_{1:n})} \tag{3.35}$$

where $p(y_{1:n})$ is the normalizing factor $\int p(x_{1:n})p(y_{1:n}|x_{1:n})\, dx_{1:n}$.

The Bayesian gives a probability of all states up to the current time step given the observations at each step. Substituting (3.33) and (3.34) into (3.35) the posterior distribution

can be expressed recursively

$$p(x_{1:n}|y_{1:n}) = p(x_{1:n-1}|y_{1:n-1})\frac{f(x_n|x_{n-1})g(y_n|x_n)}{p(y_n|y_{1:n-1})} \tag{3.36}$$

with $p(y_n|y_{1:n-1}) = \int p(x_{n-1}|y_{1:n-1})f(x_n|x_{n-1})g(y_n|x_n)\,dx_{n-1:n}$. It can be seen that

$$p(x_n|y_{1:n}) = \int p(x_{1:n}|y_{1:n})\,dx_{1:n-1} = \int p(x_{n-1}|y_{1:n-1})f(x_n|x_{n-1})\,dx_{n-1}\frac{g(y_n|x_n)}{p(y_n|y_{1:n-1})}$$

where the integral is recognized as $p(x_n|y_{1:n-1})$, the prediction carried out by the motion model, which is updated by the observation $g(y_n|x_n)$ and normalization factor $\frac{1}{p(y_n|y_{1:n-1})}$. The target distribution, also called the posterior, $\pi_n = p(x_n|y_{1:n})$ can be described by

$$\pi_n = \frac{\gamma_n}{Z_n}, \quad \gamma_n = p(x_n|y_{1:n-1})g(y_n|x_n), \quad Z_n = p(y_n|y_{1:n-1}) \tag{3.37}$$

where mainly the normalizing constant Z_n is unknown and γ_n can be approximated by the particles (samples) and their weights as seen next.

3.4. Monte Carlo Methods and importance sampling

3.4.1. Monte Carlo

Monte Carlo methods are a way to approximate a distribution and following expected values. $\pi_n(x_n)$ can be Monte Carlo approximated using N samples from $\pi_n(x_n)$ as

$$\hat{\pi}_n(x_n) = \frac{1}{N}\sum_{i=1}^{N}\delta_{X_n^i}(x_n) \tag{3.38}$$

where $\delta_{X_n^i}$ is the Dirac-Delta function at sample i, effectively counting the relative frequencies of the particular realizations x_n. In turn the expectation of a function φ_n can be approximated directly as

$$E_n(\varphi_n(x_n)) \approx \int \varphi_n(x_n)\hat{\pi}_n(x_n)\,dx_n = \frac{1}{N}\sum_{i=1}^{N}\varphi_n(X_n^i), \quad X_n^i \sim \pi_n(x_n)$$

In a particle filter one often seeks the expected value of the state and so the value of Monte Carlo methods are obvious, however $\pi_n(x_n)$ is not known or complex to sample from. This is where importance sampling comes in.

3.4.2. Importance sampling

It is seen that

$$\pi_n(x_n) = \frac{\gamma_n(x_n)}{Z_n} = \frac{\gamma_n(x_n)}{Z_n q_n(x_n)} q_n(x_n) = \frac{w_n(x_n) q_n(x_n)}{Z_n}$$

With $w_n(x_n) = \frac{\gamma(x_n)}{q_n(x_n)}$ being the un-normalized weight function and $q_n(x_n)$ a chosen importance distribution with the only requirement of being positive wherever $\pi_n(x_n) > 0$. The normalizing constant is

$$Z_n = \int \gamma_n(x_n)\, dx_n = \int \frac{\gamma_n(x_n)}{q_n(x_n)} q_n(x_n)\, dx_n = \int w_n(x_n) q_n(x_n)\, dx_n$$

Which means that it can be Monte Carlo approximated by drawing samples from $q_n(x_n)$:

$$Z_n \approx \frac{1}{N} \sum_{i=1}^{N} w_n(X_n^i), \quad X_n^i \sim q_n(x_n)$$

Then the expectation of a function $\varphi_n(x_n)$ using the target distribution become

$$E_n(\varphi_n(x_n)) = \frac{1}{Z_n} \int \varphi_n(x_n) w(x_n) q_n(x_n)\, dx_n \approx \frac{N}{\sum_{i=1}^{N} w_n(X_n^i)} \int \varphi_n(x_n) w_n(x_n) q_n(x_n)\, dx_n$$

which off-course can be approximated further using a second Monte Carlo simulation

$$\begin{aligned} E_n(\varphi_n(x_n)) &\approx \frac{N}{\sum_{i=1}^{N} w_n(X_n^i)} \frac{1}{N} \sum_{i=1}^{N} \varphi(X_n^i) w_n(X_n^i), \quad X_n^i \sim q_n(x_n) \\ &\approx \sum_{i=1}^{N} \varphi(X_n^i) W_n(X_n^i), \quad W_n(X_n^i) = \frac{w_n(X_n^i)}{\sum_{i=1}^{N} w_n(X_n^i)} \end{aligned} \tag{3.39}$$

The only remaining question is how $q_n(x_n)$ is chosen. There might be choices which will minimize the variance for a specific $\varphi(x_n)$ but to keep a constant computational complexity the sequential importance sampling was introduced.

3.4.3. Sequential Importance Sampling

Generally whenever a new observation y_{n+1} is done one must recompute the weights based on all the previous steps. However by letting the importance density be dependent of the density of the previous state the density as well as the weights can be calculated

recursively. That is

$$q_n(x_n) = q_{n-1}(x_{n-1})q_n(x_n|x_{n-1}) \tag{3.40}$$

The first importance density can for instance be a uniform distribution while the rest are determined recursively dependent on the previous sample. The only choice is then $q_n(x_n|x_{n-1})$, a popular choice is using the transitional density $f(x_n|x_{n-1})$ such that the recursive weights simply become

$$\begin{aligned} w_n(x_n) &= \frac{\gamma_n(x_n)}{q_n(x_n)} \\ &= \frac{\gamma_{n-1}(x_{n-1})}{q_{n-1}(x_{n-1})} \frac{\gamma_n(x_n)}{\gamma_{n-1}(x_{n-1})q_n(x_n|x_{n-1})} \\ &= w_{n-1}(x_{n-1}) \cdot \frac{\gamma_n(x_n)}{\gamma_{n-1}(x_{n-1})f(x_n|x_{n-1})} \\ &= w_{n-1}(x_{n-1}) \cdot g(x_n|y_n) \end{aligned} \tag{3.41}$$

In practise this means applying the motion model to the states drawn (with replacement) from the the previous step to get N new samples, and then updating the weight of each particle by its likelihood from the observation model.

3.4.4. Resampling

It can be shown that as long as the importance density $q_n \neq \pi_n$ the variance of the weights will increase with time. The result of this is that the weight of a single particle will tend to one while the others will be negligible. As a consequence a large number of useless particles will be carried forward in the calculations while only a small fraction of them actually matter after some time. This is known as degeneracy.

A measure of degeneracy is the number of effective particles:

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (W_n^i)^2} \tag{3.42}$$

Effectively measuring the variance of the weights, N_{eff} will be small if some particles holds significantly more weight than others. When degeneracy is detected for a set threshold $N_{\text{eff}} < N_{\text{threshold}}$ so-called resampling can be performed to correct it. The threshold could for instance be set to half the number of particles. A simulated test showed that resampling was needed 11% of the time on average over 400 frames and that the ratio increased as the particles converged using this threshold.

Another option is to perform resampling every iteration, that way a threshold is not needed while the execution remains more deterministic. The expected states can then be calculated by summation and a single division by N_p, the number of particles, since all weights are the same. This stands in contrast to the matrix-vector product of $N_{\text{states}}N_p^2$ multiplications additionally needed otherwise. The matrix operation can admittedly be performed quite fast using an optimized linear algebra library such as `BLAS`.

Another small aspect of consistent resampling is that (3.41) reduces to $g(x_n|x_{n-1})$ since the previous weight is constant and removed in the normalizing step. The result is one less memory load and multiplication per particle.

The net cost will most certainly be higher when performing resampling each time, but the particle filter has to meet the deadlines even when performing resampling anyway making it less important.

Systematic resampling

One of the most used resampling strategies is *systematic resampling*[1]:

1. Uniformly sample $U_1 \in [0, 1/N]$.
2. For particle $i = 2, \ldots, N$ define $U_i = U_1 + \frac{i-1}{N}$
3. U_i will land between two values of the cumulative sum of the (normalized weights).
4. The new sample should then be particle j where j is the upper bound index of this interval.

This is very closely related to sampling from the particles cumulative distribution using uniform sampling other than using a single random number as a seed instead of separate samples. Resampling will focus the computations on regions of high probability while avoiding degeneracy, the weights are all reset and the particle distribution is now represented by the number of occurrences of a particle instead of weights. Since the cumulative sum is pre-sorted a binary search can be performed for the correct interval in logarithmic time, making it quite efficient. Even better the lowest index possible will always be bounded by the previous j narrowing the search space for each iteration i performed.

Rejection sampling

Systematic resampling relies on a cumulative distribution as well as a binary search, both of which are not GPU friendly. The binary search can cause excessive warp divergence (see 4.2) since it relies on branching and making a cumulative array is a highly sequential operation.

One such alternative sampling strategy was proposed in [17] noting that rejection sampling is only dependent on pairwise comparisons of weights. Rejection sampling is performed for particle $i = 1, \ldots, N$ by

1. Initialize a candidate particle $j = i$.

2. Draw a uniform sample $U \in [0, 1]$.

3. While $u > \frac{w_j}{w_{\max}}$ pick a new candidate $j \in [1, N]$ as well as a new sample $U \in [0, 1]$ uniformly.

4. The first candidate j which satisfy $U \leq \frac{w_j}{w_{\max}}$ is to replace particle i.

The distribution of the particle weights are scaled by $\frac{1}{w_{\max}}$ such that the proposal distribution (simply 1) is greater than or equal to the weight distribution for all particles. A particle, j, is drawn at random and a sample u is taken between zero and the proposal probability at j, i.e 1. If the sample is less than the (scaled) particle weight, j is accepted as being a sample from the weight distribution. The probability of accepting is higher whenever the weights are high, which is the goal of resampling; to carry on the good particles and still represent the weight distribution.

Note that there is no need to normalize the weights but the largest weight needs to be found which can be done quite effectively on a GPU. There is also the non-deterministic number of iterations until a sample is accepted and due to the GPU architecture 32 threads/particles at a time will have the same duration as the longest one of that set.

4. Particle filter implementation

There are countless ways of implementing a particle filter, given the short deadlines of the application some effort must be used to identify and improve time consuming areas of the code. Most importantly the parallel structure and control must be establish. To find the fastest solutions iterative improvement was performed on the slowest parts of the code until satisfactory run-times where met. Since the particle filter has a fair amount of states to estimate a goal to handle 50000 particles in 60fps (16.67ms per iteration) was established.

4.1. CPU optimization

As a start the particle filter was implemented in a usual object oriented (OO) way in C++ on a single threaded CPU only (Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz). Each particle had its own object accessed through a member vector in the filter class with its own set of states and weights. Every step of the filter where performed sequentially for each particle and the results where gathered in the filter class. The execution times stays fairly consistent and so single measurements are assumed to be representative from there on. The run-time results for 50000 particles with and without the compiler optimization flag `-O3` is shown in tab 4.1.

Dataset	**OO-filter [ms]**	**OO-filter -O3 [ms]**	**Speed-up**
Motion	50.90	33.89	1.5
Observation	211.29	131.52	1.6
Resample	29.74	2.74	10.8
Mean	28.33	2.78	10.2
Total	320.26	**170.93**	**1.9**

Table 4.1.: Execution time for each step of the filter with 50000 particles using particle objects.

This implementation was kind of convoluted and the separated memory of the states as well as the weights had its drawbacks on cache utilization and performance. Therefore the particle class was discarded altogether and the states of each particle was stored continuously in a member array of the filter class, the same was done for the weights.

The results of 4.2 shows significant improvements over the prior implementation.

Dataset	**Memory-filter [ms]**	**Memory-filter -O3 [ms]**	**Speed-up**
Motion	48.49	33.03	1.5x
Observation	102.00	64.31	1.6x
Resample	19.61	3.13	6.27x
Mean	1.50	0.30	5.0x
Total	**171.58**	100.77	**1.7**

Table 4.2.: Execution time for each step of the filter with 50000 particles using continuous memory.

Already before compiler optimization it is seen that the new implementation runs as fast as the optimized object oriented one, a witness of better performing code. This may also be seen by the lower speed-up gained by compiler optimization on the second version hinting at better code to begin with. Although the performance was improved by 319% using better memory access and compiler optimization there is still a need for a 6x speed-up to reach the required deadline even before other processing is considered, this can only be achieved by parallelization.

4.2. The CUDA architecture

CUDA is a platform for parallel computing created by the graphics processing units designer and supplier NVIDIA[1]. It allows for GPU programming using C/C++ like syntax and supplies a compiler for full integration with the C++ language. To effectively program on the GPU it is imperative to know its architecture to understand the limitations and how to circumvent them.

4.2.1. Parallel execution

A subroutine executed on the device (GPU) is called a kernel. The purpose of a graphical processor is to run a large number of threads of the same kernel at a time, for instance if the body of a `for-loop` is independent of the previous iteration it can just as well be executed concurrently with one thread for each iteration.

At the hart of a graphics card there are many parallel processors grouped into several streaming multiprocessors known as *SMs*, an SM has several cores which each executes the same instructions (in lock-step) 32 threads at a time, this is called a *warp*.

[1] `https://developer.nvidia.com/cuda-zone`

Since a warp executes all its instructions in lock-step all the 32 threads will be performing every branch of the program regardless if the individual threads needs it or not. An example of this is conditional statements such as `if-else` statements where each thread will perform both the `if` and `else` body even though only a single thread evaluates the condition differently. This is known as divergence and can lead to severe performance hits.

A typical SM can execute 32 warps at a time, stemming from several *block*s. A block is a gathering of warps and is a logical division of the threads chosen by the programmer. Blocks are needed to ensure all the warps/threads grouped by the programmer are run on the same SM, which let the threads of a block share the same (fast) memory on the SM and cooperate on computations.

To launch a kernel the number of threads and blocks has to be specified, for max utilization of the SM the number of threads in each block should be a multiple of the warp size. A sub-optimal launch configuration could for instance be 385 ($3 \cdot 32 + 1$ threads per block; in the worst case resulting in $127 \cdot 32$ vacant threads on the SM since 4 warps would be needed to compute each block.

For good performance the ratio of active threads/warps, also called occupancy, on the SM should be high ($> 60\%$).

4.2.2. Memory model

Ignoring cache patterns there is still considerations to be done to get effective memory operations on the GPU.

A GPU typically has a considerable amount of DRAM called global memory because it is reachable by both the CPU and GPU. Although fast a global memory load can take several hundred cycles and be a significant bottleneck in execution speed.

It is practically impossible to eliminate the use of global memory as it is the only thing the CPU can see but it should be reduced as much as possible. Another key insight is that a load can be of a fixed byte size (e.g 32) in a SIMD fashion, even though a single 4 byte float was needed by a specific thread. It is done this way to utilize the whole bandwidth and benefit the surrounding threads which also needs data for the same instruction.

To perform as few operations as possible the data used by each thread should be close to each other to take advantage of the SIMD/SIMT architecture. This is known as

memory coalescing. If this is not done the superfluous data read is simply discarded and the bandwidth sees no benefit of the multiple data approach.

Another form of memory on the GPU is that integrated in the SMs which makes for much quicker access than the global case. Every thread of a block can utilize the same allocated SM memory and it is therefor named shared memory. Shared memory can for instance be used to reduce the global memory loads if many threads needs the same data.

In addition to shared memory each SM also has a number of registers available to each thread. Both shared memory and registers are a spare resources and could limit the number of warps that fits in the SM at the same time. As an example consider an SM with 64K registers, if 2048 threads uses 33 registers each the register requirement is exceeded and the SM can only process 30 warps instead of 32.

The last form of memory on a GPU is textured memory. Initially made for texturing polygons it is optimized for random access with a dedicated cache. For many applications having random access patterns the use of texture memory could give good performance boosts, it is however read only which limits its usage.

4.3. Exposing parallelism

In order to run as fast as possible parallel computation must be utilized where appropriate. The criteria for good performance is a nice balance between parallel computation and the related bottlenecks and overhead. Examples of limitations when using GPUs are the relative slow memory transfer and also the overhead of launching parallel kernels.

The GPU and CPU have different physical memory which means that for one processing unit to give input/output to the other the data must be transferred over the PCIe-bus limiting the datarate to a theoretical maximum of 8-16GB/s dependent on the version used. Although fast this is several orders of magnitude slower than the modern day processing units making it a bottleneck in the pipeline. Memory transfer should therefore be held to a minimum.

Another issue is the overhead related to launching new threads on the GPU; a small number of procedures of moderate size might not see any performance benefit of being run in parallel because the overhead of launching the kernels outweighs the speed-up seen from concurrent computation. This is especially true for nested/dynamic parallelism on the GPU where memory transfer is not the limiting factor.

These limitations especially effects the way the program should be written; a good parallelization should be able to run without much CPU interaction and warrant the use of overhead. A top-down approach was used to find routines that could be parallelized and each found case was considered based on this criteria.

Particles

Since the particle filter consists of a large number of independent particles it is only natural that routines performed for every particle run in its own GPU thread. This is where the largest performance savings will be seen and the focus should be on making it as CPU independent as possible.

The motion model relies on numerical integration which is inherently sequential and so each particle thread performs the dynamic calculation for the motion model.

The observation model is also performed on each particle's states and in addition performs a number of measurements in the image that does not depend on each other. There is in other words more potential for concurrent execution by separating the measurements in stead of performing them sequentially for each particle.

Resampling and mean

The last steps of the filter is tricky; as seen by the CPU tests they have little impact on the execution time, but needs all the hypothesized states and weights to be performed. That is: a large memory transfer has to be done (2.4MB for 12 states and 50000 particles using single precision) to perform the sub-routines on the CPU. In the numerical example this equates to 3ms on a fully utilized PCIe.v2-bus which can quickly become more, that is more than the sub-routines take to execute in and of themselves on the CPU in 4.1 and nearly 20% of the deadline. Ideally all the data needs to stay in the GPU until the expected states (mean) can be transferred back.

4.4. GPU optimization

To optimize the parallel parts of the code the Assess, Parallelize, Optimize, Deploy(APOD) design cycle was used. That is; the limiting factor (bandwidth, compute throughput or latency) was identified and improved until the deadline was met with good margin. The optimization was performed with the help of `NVIDA Visual Profiler` where all kernel measurements also stems from, the profiler is a nice visual tool that display occupancy, memory usage, concurrency and the limiting factors of the kernels.

The measurements where taken from the code running on a low-end graphics card, namely Nvidia GeForce 940MX.

4.4.1. Motion model

To get good memory coalescing the states where placed in a one-dimensional array sectioned as a $N_{\text{states}} \times N_{\text{particles}}$ matrix. The states of each particle was first read into thread memory and only copied back to global memory after all calculations where completed to minimize memory transfer. By using dedicated CUDA math operations, e.g `sincosf`, the hardware computations of the operations could be used. The compiler flag `--usefastmath` trades a little accuracy for extra speed by not refining the hardware result further as is usually done.

In the end the duration of the motion kernel was reduced to 0.71ms from the original 33.03ms on the CPU at 50000 particles, a 46x speed-up. It should be noted that the speed-up is dependent on the number of particles and can be changed almost arbitrarily by just changing the sample size. The only thing to take away from the speed-up is that running a particle filter on the GPU is highly effective compared to the CPU for a large number of particles.

4.4.2. Observation model

A large limiting factor of the observation kernel, and GPU execution in general, is the memory operations. Other than using the principles of the motion model the constant variables such as the world points describing the object and the camera matrix where placed in shared memory. Each thread read a piece of the constant variables and placed it in shared memory for all the threads of a block to use.

A problem of the observation model is the many memory reads done on the image. In fact removing the image reads makes the kernel execute an order of magnitude faster. Early tests showed a good performance boost by reading the image through the texture cache of the GPU designed for random access.

In its first version the observation model was given a thread per particle, allowing the motion and observation kernel to be merged which lets the observation model take advantage of the motion model's results directly. The disadvantage to this approach is the sequential measurements of the image which halt the execution of the kernel for each reading.

To remedy the sequential reads an attempt of using dynamic parallelism was tried next.

The parent thread representing the particle was to launch its own kernels in a different GPU stream to hide the image read latencies. As tests actually showed worse performance the idea was discarded. Dynamic parallelism can be challenging to get right and programming errors can admittedly be the reason for bad performance. Other than that the restrictive nature of a nested kernel execution called for several kernel launches which might impose an overhead penalty.

The second version of the observation model launched a block for each particle and let each thread be a single image measurement. This let the threads take advantage of the variables shared by each particle including states, world points, camera matrix and projected image points while hiding the latency by performing context switching on inactive warps. After tweaking the observation kernel had a duration[2] of about 6.5ms, nearly a 10x speed-up compared to the CPU version.

4.4.3. Resampling and mean

Systematic resampling is best suited for the CPU with its sequential nature and branching binary search. To use this resampling method a rather large memory transfer must be performed, in the very least the weights of each particle. For this reason other resampling techniques was also tested and measured against the CPU alternative.

An alternative resampling method tested as rejection sampling, however in a simple test of a typical filter run, the reprojection resampling required on average 22 iterations per particle, including the initial read and fetching of the pseudo random state. That is 24 memory reads from global memory *per thread*.

Resampling using rejection sampling on the GPU took 1.7x as long as copying the weights to the CPU and using systematic resampling. It does not scale better than systematic resampling either; doubling the amount of particles results in a 2.8x increase for rejection sampling while systematic resampling on the CPU only increases by 2.3x. The multiprocessors are already at satisfactory occupancy so the performance is mainly dependent of memory latency. The large memory dependency of parallel rejection sampling excludes it from being viable in the current application.

As nearly 70% of the resampling time is spent on performing binary searches sequentially the question arise if parallel searches would be beneficial despite the many memory reads it introduces. A non-branching binary search can be implemented as

[2]Worst case scenario with all measurements within the image bounds

```
int findInterval( float val, const float *arr, int arrSZ )
{
    int idx = 0;

    // find closest power of 2 rounded down
    int n = floor_pow_2( arrSZ );

    // compensate for non-power of two list
    idx += ( arr[ n ] <= val )*( arrSZ - n );

    // branchless binary search
    for (int i = n/2; i>1; i/=2)
        idx += (arr[ idx+i ]<= val)*i;

    // returns index of the last element smaller than val
    return idx;
}
```

Even though it increases the complexity to a constant $\log_2(N)$ trials, which is worst case for a binary search, it is beneficial performance wise as warp divergence is eliminated. Care must be taken to ensure the operations are compiled to non-branching instructions, for instance will `idx = (arr[ 4 ]<= val ) ? 4 : 0;` compile to 4.1

```
/*00e8*/                   LD.E R2, [R2], P0;                /* 0x8090000000070202 */
/*00f0*/                   ISETP.LE.AND P0, PT, R2, R0, PT;  /* 0x5b67038000070207 */
/*00f8*/                   PSETP.AND.AND P0, PT, !P0, PT, PT;  /* 0x50900380e0078007 */
                                                             /* 0x007fbc03fde01fef */
/*0108*/               @P0 BRA 0x140;                        /* 0xe24000000300000f */
/*0110*/                   BRA 0x118;                        /* 0xe24000000007000f */
/*0118*/                   MOV32I R0, 0x4;                   /* 0x01000000047f000 */
```

Figure 4.1.: NVCC uses branching instead of conditional move.

by the `NVCC` compiler, this is a conditional jump instead of the desired move. The correct option is therefore to use `idx = (arr[ 4 ]<= val )*4;` as shown in 4.2

```
/*00e8*/                   LD.E R2, [R2], P0;                /* 0x8090000000070202 */
/*00f0*/                   ISETP.LE.AND P0, PT, R2, R0, PT;  /* 0x5b67038000070207 */
/*00f8*/                   SEL R0, RZ, 0x1, !P0;             /* 0x38a004000017ff00 */
                                                             /* 0x00643c03fde0190f */
/*0108*/                   IMUL32I R0, R0, 0x4;              /* 0x1fc0000000470000 */
```

Figure 4.2.: A multiplication replaces the jump.

To make the sampling process fully parallelizable systematic resampling which starts with a single seed was replaced with a true resampling drawing a random index for each particle. Eliminating the data transfers and running the binary searches in parallel resulted in a 1.17x speedup over CPU resampling. Not a huge improvement but at least now the whole particle filter is contained on the GPU which frees up the CPU for asynchronous processing until the results are copied back.

4.4.4. Hiding latency

As already mentioned the bottleneck of many GPU implementations are the memory transfers between host and device. One such instance for this thesis' application is the image transfer which first reaches the CPU and must be copied to the GPU for the observation model to do its measurements.

There is little that can be done to improve the transfer speed in itself, but it just so happens that before the observation model needs the image the motion model must be performed in its entirety. Since CUDA allows for asynchronous memory transfers the image can be copied while the motion model is executing. This is shown in the Gantt chart of figure 4.3 where most of the latency due to image transfer is eliminated.

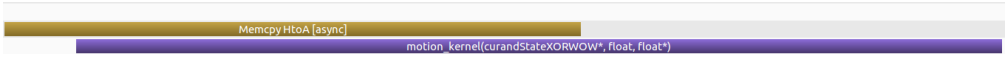

Figure 4.3.: The image transfer (top) is being performed at the same time as the motion model (bottom).

Unfortunately the rest of the particle filter is strictly sequential which make further concurrent kernel executions impossible.

4.4.5. Finishing words on the GPU implementation

Figure 4.4 shows the execution scheduling for each filter module on the GPU in a Gantt chart. It is without a doubt the observation model which impact the execution time the most. It also has a wide range of possible implementations and the best one is not obvious.

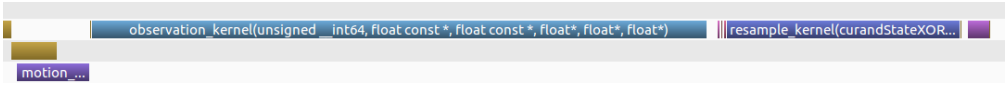

Figure 4.4.: Gantt chart of the filter modules executing on the GPU.

Most of the duration of the observation model is spent on memory reads of the image

which makes it hard to improve much further at this point. The extensive experimentation on the observation model in combination with the various other improvements resulted in a typical execution time of 10ms per image, well within the (soft) real-time requirement of 16.67ms.

Table 4.3 details the improvements made to the best CPU implementation. The finished GPU filter has a 10x speed-up from the best CPU implementation, a considerable improvement.

Dataset	**Filter CPU [ms]**	**Filter GPU [ms]**	**Speed-up**
Motion	33.03	0.71	46.5x
Observation	64.31	6.55	9.8x
Resample	3.13	2.48	1.3x
Mean	0.30	0.23	1.3x
Total	100.77	10.41	**9.7x**

Table 4.3.: Execution time for each step of the filter with 50000 particles on the CPU vs. the GPU.

Note that there is a 0.44ms discrepancy between the sum of module times and the measured total of the GPU implementation due to miscellaneous overhead.

Since the CUDA code where measured and optimized based on the measurements from a low-end graphics card (GTX 940MX) it comes as no surprise that the results where improved significantly by deploying on better hardware. The actual experiments where performed on a Titan X with 10 times as many cores and 5 times the memory clock rate which resulted in a further 10x speed-up with run-times below 1ms at the same amount of particles.

5. Camera calibration

Since the proposed solution can be regarded as an open-loop regulator it is of utmost importance that the transformations from the base-frame to the cameras as well as the intrinsic camera parameters have good estimates. The lack of feedback is one of the most severe criticisms of the system as a wrongful estimate of the constant parameters never will be detected nor compensated for by the routine.

5.1. Intrinsic and extrinsic calibration

A traditional way of calibrating a single camera is using a planar pattern of known dimensions in several poses to estimate both intrinsic and extrinsic parameters[12]. The intrinsic parameters are calculated first through linear equations using the pin-hole model of a camera. The reprojection error of all the parameters is then minimized through a non-linear optimization[31] usually involving the iterative methods Levenberg-Marquardt or Newton-Gauss[4].

Recently Gai et al.[6] shows improvement on the extrinsic parameter estimation by initializing with Zhang's method[31] and optimizing for 3D reconstruction. Accurate estimates of the extrinsic parameters, that is the 3D poses of the planar pattern, plays a crucial part in estimating the camera poses as will be evident in section 5.2. The methods of both Gai[6], Zhang[31] was tested with respect to the particle filter application. A third, more novel alternative was also constructed in the hunt for accurate parameters.

5.1.1. Initial linear calibration

All the tested methods where initialized using Zhang's approach. The projection, $\boldsymbol{x}$, of a homogeneous point $\boldsymbol{X}$ is given in (3.24) as

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}]\boldsymbol{X}$$

For a planar point - with z-coordinate 0 - the projection can be seen as a transformation from plane to plane by

$$\boldsymbol{H} = \boldsymbol{K}\begin{bmatrix}\boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t}\end{bmatrix} = \begin{bmatrix}\boldsymbol{K}\boldsymbol{r}_1 & \boldsymbol{K}\boldsymbol{r}_2 & \boldsymbol{K}\boldsymbol{t}\end{bmatrix} = \begin{bmatrix}\boldsymbol{h}_1 & \boldsymbol{h}_2 & \boldsymbol{h}_3\end{bmatrix}$$

where $\boldsymbol{r}_i, i = 1\ldots,3$ denotes the column vectors of $\boldsymbol{R}$. $\boldsymbol{H}$ can be least-squares estimated by the direct linear transform[10, section 4.1]. The orthogonality of $\boldsymbol{R}$ gives two constraint in the form of

$$\boldsymbol{r}_1^T\boldsymbol{r}_2 = 0 \Leftrightarrow \boldsymbol{h}_1^T\boldsymbol{K}^{-T}\boldsymbol{K}^{-1}\boldsymbol{h}_2 \tag{5.1}$$

$$\boldsymbol{r}_1^T\boldsymbol{r}_1 = \boldsymbol{r}_2^T\boldsymbol{r}_2 \Leftrightarrow \boldsymbol{h}_1^T\boldsymbol{K}^{-T}\boldsymbol{K}^{-1}\boldsymbol{h}_1 = \boldsymbol{h}_2^T\boldsymbol{K}^{-T}\boldsymbol{K}^{-1}\boldsymbol{h}_2 \tag{5.2}$$

Noting that $\boldsymbol{B} = \boldsymbol{K}^{-T}\boldsymbol{K}^{-1}$ is a symmetric matrix since $\boldsymbol{K}$ is upper triangular reveals 6 degrees of freedom. As each homography supplies 2 constraints a minimum of 3 different configurations of the calibration pattern is needed to determine $\boldsymbol{K}$.

Using the unique elements of $\boldsymbol{B}$ in a six-dimensional vector $\boldsymbol{b}$ lets the constraints of configuration j be reformulated as $\boldsymbol{a}_{1,j}^T\boldsymbol{b} = 0, \boldsymbol{a}_{2,j}^T\boldsymbol{b} = 0$ where

$$\boldsymbol{a}_{1,j} = \begin{bmatrix} h_{12}h_{11} \\ h_{12}h_{21} + h_{11}h_{22} \\ h_{22}h_{21} \\ h_{12}h_{31} + h_{32}h_{11} \\ h_{22}h_{31} + h_{32}h_{21} \\ h_{32}h_{31} \end{bmatrix} \qquad \boldsymbol{a}_{2,j} = \begin{bmatrix} h_{11}^2 - h_{12}^2 \\ 2(h_{11}h_{21} - h_{12}h_{22}) \\ h_{21}^2 - h_{22}^2 \\ 2(h_{11}h_{31} - h_{12}h_{32}) \\ 2(h_{21}h_{31} - h_{22}h_{32}) \\ h_{31}^2 - h_{32}^2 \end{bmatrix}$$

For

$$\boldsymbol{B} = \begin{bmatrix} b_1 & b_2 & b_4 \\ b_2 & b_3 & b_5 \\ b_4 & b_5 & b_6 \end{bmatrix}$$

The constraints of the n configurations can then be gathered in

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_{1,1}^T \\ \boldsymbol{a}_{2,1}^T \\ \vdots \\ \boldsymbol{a}_{1,n}^T \\ \boldsymbol{a}_{2,n}^T \end{bmatrix} \tag{5.3}$$

To solve $\boldsymbol{A}\boldsymbol{b} = \boldsymbol{0}$ by least-squares using the singular value decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ where $\boldsymbol{b}$ is the last column of $\boldsymbol{V}$. Having found $\boldsymbol{B}$, $\boldsymbol{K}^{-1}$ can be determined by Cholesky decomposition which gives $\boldsymbol{K}$ through inversion, $\boldsymbol{K}$ is scaled such that its last element, k_{33} is 1.

Once $\boldsymbol{K}$ is known the extrinsic parameters is easily calculated since $\boldsymbol{r}_1 = \lambda\boldsymbol{K}^{-1}\boldsymbol{h}_1, \boldsymbol{r}_2 =$

$\lambda \boldsymbol{K}^{-1}\boldsymbol{h}_2, \boldsymbol{t} = \lambda \boldsymbol{K}^{-1}\boldsymbol{h}_3$ and $\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2$ where $\lambda = \|\boldsymbol{K}^{-1}\boldsymbol{h}_1\|$ scales the homography so that $\|\boldsymbol{r}_i\| = 1$. Note that the following rotation matrix must be corrected to ensure $\det \boldsymbol{R} = 1$, this is done by setting $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T$ where $\boldsymbol{U}, \boldsymbol{V}$ stems from the singular value decomposition of the initial matrix estimated from $\boldsymbol{r}_i, i = 1, 2$ [1].

5.1.2. Initial transformation between cameras

Let the j'th (homogeneous) point of the calibration pattern be denoted $\boldsymbol{X}_j^o$ in its object frame, the same point is then

$$\boldsymbol{X}_{i,j}^l = \boldsymbol{T}_o^{l,i}\boldsymbol{X}_j^o, \quad \boldsymbol{X}_{i,j}^r = \boldsymbol{T}_o^{r,i}\boldsymbol{X}_j^o$$

In the left and right camera respectively where $\boldsymbol{T}_o^{l,i}$ represent the i'th pose of the object frame in the left camera frame and likewise for the right. It follows that all the points of the left camera can be placed in the right reference frame through

$$\boldsymbol{X}_{i,j}^r = \boldsymbol{T}_o^{r,i}\left(\boldsymbol{T}_o^{l,i}\right)^{-1}\boldsymbol{X}_{i,j}^l \tag{5.4}$$

The point sets $\{\boldsymbol{X}_{i,j}^l, i = 1, \ldots, n, j = 1, \ldots, m\}, \quad \{\boldsymbol{X}_{i,j}^r, i = 1, \ldots, n, j = 1, \ldots, m\}$ can thus be used to find a transformation between the two camera coordinate systems, $\boldsymbol{T}_r^l$. First the point-clouds are translated to have their centroids as origin, then there is only a rotation separating them. The optimal rotation matrix is the one which minimizes

$$\min_{\boldsymbol{R}} C = \sum_{i=1}^{n}\sum_{j=1}^{m}\left(\left\|\boldsymbol{R}\hat{X}_{i,j}^r - \hat{X}_{i,j}^l\right\|\right) \tag{5.5}$$

where $\hat{X}$ denotes a non-homogeneous point translated to have origin at the centroid. This is the orthogonal Procrustes problem. An alternative expression of the cost C can be obtained by utilizing the Frobenious norm

$$\min_{\boldsymbol{R}} C = \left\|\boldsymbol{R}\hat{\boldsymbol{X}}_r - \hat{\boldsymbol{X}}_l\right\|_F^2 = \operatorname{trace}(\hat{\boldsymbol{X}}_r\hat{\boldsymbol{X}}_r^T) + \operatorname{trace}(\hat{\boldsymbol{X}}_l\hat{\boldsymbol{X}}_l^T) - 2\operatorname{trace}(\boldsymbol{R}\hat{\boldsymbol{X}}_r\hat{\boldsymbol{X}}_l^T) \tag{5.6}$$

where $\hat{\boldsymbol{X}}_l = \begin{bmatrix}\hat{X}_l^1 & \ldots & \hat{X}_l^k\end{bmatrix}$ and $\hat{\boldsymbol{X}}_r = \begin{bmatrix}\hat{X}_r^1 & \ldots & \hat{X}_r^k\end{bmatrix}$ contains the coordinates of the left and right point sets. It is seen that minimizing the cost is equivalent to maximizing $\operatorname{trace}(\boldsymbol{R}\hat{\boldsymbol{X}}_r\hat{\boldsymbol{X}}_l^T) = \operatorname{trace}(\boldsymbol{R}^T\boldsymbol{M})$.

As Golub[7, 12.4.1] points out; if $\boldsymbol{Z} = \boldsymbol{V}^T\boldsymbol{R}^T\boldsymbol{U}, \quad \boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ then $\boldsymbol{Z}$ is orthogonal

[1] see (5.7) and (5.9) for why this is.

and it follows that $\boldsymbol{R} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T$ which yields

$$\operatorname{trace}(\boldsymbol{R}\boldsymbol{M}) = \operatorname{trace}\left(\boldsymbol{V}\boldsymbol{Z}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\right) = \operatorname{trace}(\boldsymbol{Z}\boldsymbol{\Sigma}) = \sum_{i=1}^{3} z_{ii}\sigma_i \leq \sum_{i=1}^{3}\sigma_i$$

where the upper bound is found when $\boldsymbol{R} = \boldsymbol{V}\boldsymbol{U}^T$ such that $\boldsymbol{Z} = \boldsymbol{I}$. Note however that this only requires $\boldsymbol{R}_X$ to be orthogonal, not $SO(3)$ with $\det \boldsymbol{R}_X = 1$. Umeyama [29] proposes a solution to ensure an optimal rotation matrix. Consider

$$\boldsymbol{R} = \boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T$$

where $\boldsymbol{S}$ is a diagonal matrix with elements of absolute value 1, then

$$\det \boldsymbol{R} = \det \boldsymbol{S} \det\left(\boldsymbol{U}\boldsymbol{V}^T\right)$$

Constructing $\boldsymbol{S}$ so that its determinant has the same sign as $\det\left(\boldsymbol{U}\boldsymbol{V}^T\right)$ ensures $\det \boldsymbol{R} = 1$. To make the solution optimal, that is maximize $\operatorname{trace}(\boldsymbol{R}\boldsymbol{M})$ under the new restriction, one has to maximize $\operatorname{trace}(\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T) = \operatorname{trace}(\boldsymbol{S}\boldsymbol{\Sigma}) = \sum_i^m s_i\sigma_i$ when changing the sign of an element s_i. Since the singular values are sorted as $\sigma_1 \geq \sigma_2, \ldots, \sigma_m \geq 0$ the least effect is made if the last element of $\boldsymbol{S}$, s_3, changes sign.

The optimal *rotation* matrix is therefore

$$\boldsymbol{R} = \boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T, \quad \boldsymbol{S} = \operatorname{diag}\left(1, 1, \det(\boldsymbol{U}\boldsymbol{V}^T)\right) \tag{5.7}$$

where $\boldsymbol{M} = \hat{\boldsymbol{X}}_r\hat{\boldsymbol{X}}_l^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. After determining the rotation matrix finding the translation vector from left to right is then the simple case of using the computed centroids

$$\boldsymbol{t} = \bar{\boldsymbol{X}}^r - \boldsymbol{R}\bar{\boldsymbol{X}}^l \tag{5.8}$$

Recognizing that $\boldsymbol{R} = \boldsymbol{R}_r\boldsymbol{R}_l^T$ and $\boldsymbol{t} = \boldsymbol{t}_r - \boldsymbol{R}_r\boldsymbol{R}_l^T\boldsymbol{t}_l$ from (5.4) an equivalent alternative circumventing the world points is to perform an averaging of these parameters for every extrinsic pair. To see this consider minimizing

$$\sum_{i=1}^{n} d_c(\boldsymbol{R}_i, \boldsymbol{R})^2 = \sum_{i=1}^{n} \operatorname{trace}(2\boldsymbol{I} - 2\boldsymbol{R}\boldsymbol{R}_i^T) = 2n \cdot 3 - 2\operatorname{trace}(\boldsymbol{R}\sum_{i=1}^{n}\boldsymbol{R}_i^T) \tag{5.9}$$

where $d_c(\boldsymbol{R}_i, \boldsymbol{R}) = \|\boldsymbol{R}_i, \boldsymbol{R}\|_F$ is the chordal distance[9] between the calculated rotation matrix between the cameras at the i'th configuration and the optimal rotation matrix. Substituting $\boldsymbol{M} = \sum_{i=1}^{n}\boldsymbol{R}_i^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ results in the same Procrustes solution as (5.7).

5.1.3. Zhang's cost function

As [31] was meant for a single camera its cost function must be tweaked to accomodate for a dual camera setup. In its original format the cost function is simply the square sum of the reprojection error:

$$C_1 = \sum_{i=1}^{n}\sum_{j=1}^{m} \|\boldsymbol{x}_{i,j} - \hat{\boldsymbol{x}}_{i,j}\|^2 \tag{5.10}$$

where $\hat{\boldsymbol{x}}_{i,j}$ is the projection of the j'th calibration point using the optimized calibration parameters of image i. The two camera extension simply involves adding the reprojection error of the second image:

$$C_2 = \sum_{i=1}^{n}\sum_{j=1}^{m} \left(\left\|\boldsymbol{x}^l_{i,j} - \hat{\boldsymbol{x}}^l_{i,j}\right\|^2 + \left\|\boldsymbol{x}^r_{i,j} - \hat{\boldsymbol{x}}^r_{i,j}\right\|^2 \right) \tag{5.11}$$

Which can be viewed as minimizing the mean reprojection error. It is seen that despite the information added by using two cameras the extension to Zhang's method does not really take advantage of it.

5.1.4. Reconstruction

Once the relation between the camera reference frames has been established the epipolar geometry of the system is essentially known. The fundamental matrix, mapping image points to lines in the other image plane, can be calculated as

$$\boldsymbol{F} = \boldsymbol{K}_r^{-T}[\boldsymbol{t}]_\times \boldsymbol{R}\boldsymbol{K}_l^{-1} \tag{5.12}$$

Where $[.]_\times$ denotes the skew symmetric form of a vector. The known camera matrices can be used to triangulate and reconstruct the 3D world points in the left reference frame. The problem is already determined up to the unknown scale factors, $\rho_l \neq 0$ and $\rho_r \neq 0$:

$$\begin{aligned}\rho_l \boldsymbol{s}^l_{i,j} &= \boldsymbol{K}_l^{-1}\boldsymbol{x}^l_{i,j} = [\boldsymbol{I}|\boldsymbol{0}]\boldsymbol{X}^l_{i,j} = \boldsymbol{P}_l\boldsymbol{X}^l_{i,j} \\ \rho_r \boldsymbol{s}^r_{i,j} &= \boldsymbol{K}_r^{-1}\boldsymbol{x}^r_{i,j} = [\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{X}^l_{i,j} = \boldsymbol{P}_r\boldsymbol{X}^l_{i,j}\end{aligned}$$

where $\boldsymbol{s} = \begin{bmatrix} s_1 & s_2 & 1 \end{bmatrix}^T$ is the homogeneous normalized image coordinates. Recognizing that $\boldsymbol{s}^l \times \boldsymbol{p}^l = \boldsymbol{0}$ and $\boldsymbol{s}^r \times \boldsymbol{p}^r = \boldsymbol{0}$ gives rise to $2 \cdot 3$ sets of equations of which $2 \cdot 2$ are linearly independent. Choosing 4 linearly independent equations a linear problem can

be set up as

$$\begin{bmatrix} s_1^l \boldsymbol{p}_{l,3}^T - \boldsymbol{p}_{l,1}^T \\ s_2^l \boldsymbol{p}_{l,3}^T - \boldsymbol{p}_{l,2}^T \\ s_1^r \boldsymbol{p}_{r,3}^T - \boldsymbol{p}_{r,1}^T \\ s_2^r \boldsymbol{p}_{r,3}^T - \boldsymbol{p}_{r,2}^T \end{bmatrix} \boldsymbol{X}_{i,j}^l = \boldsymbol{A}\boldsymbol{X}_{i,j}^l = \boldsymbol{0} \tag{5.13}$$

where $\boldsymbol{p}_{l,i}^T$ denote the i'th row of $\boldsymbol{P}_l$ and likewise for $\boldsymbol{P}_r$. This can for instance be solved using the direct linear transform (DLT)[10, section 12.2] where the singular value decomposition of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ gives the solution as

$$\boldsymbol{X}_{i,j}^l = \boldsymbol{v}_3, \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \end{bmatrix} \tag{5.14}$$

Optimal triangulation can be obtained by first correcting the point correspondences, $\boldsymbol{x}_{i,j}^l \leftrightarrow \boldsymbol{x}_{i,j}^r$, so that their rays stemming from the camera origins intersect in the three-dimensional space. [10, Algorithm 12.1], [11] shows how this can be achieved by solving a six-degree polynomial once the fundamental matrix is known. Note also that [4] undistort the image points beforehand since the measured points $\hat{x}$ might be affected by the lens geometry typically modeled by [12]:

$$\hat{\boldsymbol{x}}_{i,j}^l = \left(1 + k_1^l d(\boldsymbol{x}_{i,j}^l)^2 + k_2^l d(\boldsymbol{x}_{i,j}^l)^4\right) \boldsymbol{x}_{i,j}^l \tag{5.15}$$

$$\hat{\boldsymbol{x}}_{i,j}^r = \left(1 + k_1^r d(\boldsymbol{x}_{i,j}^r)^2 + k_2^r d(\boldsymbol{x}_{i,j}^r)^4\right) \boldsymbol{x}_{i,j}^r \tag{5.16}$$

where $d(.)$ is the distance to the focal point (u_0, v_0) given by the intrinsic parameters and $k_1^l, k_2^l, k_1^r, k_2^r$ are the radial distortion coefficients of the left and right camera respectively.

5.1.5. Gai's Cost function

[4] chooses to assume that the calibration pattern transformed by the extrinsic parameters $\boldsymbol{T}_o^{l,i}$ are good enough even for the corrected points. The pattern is then transformed to the left reference frame to make the cost function. [6] on the other hand re-calculates

the transformation analogous to before, only now with known intrinsic parameters:

$$\begin{aligned} \boldsymbol{X}_{i,j}^{l} &= \begin{bmatrix} \boldsymbol{R}_{w}^{l,i} & \boldsymbol{t}_{lw,i} \\ \boldsymbol{0}^{T} & 1 \end{bmatrix} \begin{bmatrix} X_w^j \\ Y_w^j \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1^{l,i} & \boldsymbol{r}_2^{l,i} & \boldsymbol{t}_{lw,i} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w^j \\ Y_w^j \\ 1 \end{bmatrix} \\ &\Leftrightarrow \\ \hat{\boldsymbol{x}}_{i,j}^{l} &= \begin{bmatrix} \boldsymbol{r}_1^{l,i} & \boldsymbol{r}_2^{l,i} & \boldsymbol{t}_{lw,i} \end{bmatrix} \begin{bmatrix} X_w^j \\ Y_w^j \\ 1 \end{bmatrix} = \boldsymbol{H}\boldsymbol{x}_j^w \end{aligned} \tag{5.17}$$

where $\boldsymbol{r}_k$ is the k'th column of a rotation matrix $\boldsymbol{R}$, $\hat{\boldsymbol{x}}_{i,j}^{l}$ is the non-homogeneous version of $\boldsymbol{X}_{i,j}^{l}$ which can be viewed as the normalized image coordinate with known scale $\rho_l = 1$. All the points of image pair i can then be used to find a least-squares solution to $\boldsymbol{H}$. Since a rotation matrix $\boldsymbol{R}$ is orthogonal, that is $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$, it follows that $\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2$ and the complete transformation $\boldsymbol{T}_w^{l,i}$ can be retrieved.

The triangulated points are then placed in the object frame by the calculated extrinsic parameters to make the cost function

$$C = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\left\| \boldsymbol{X}_j^o - (\bar{\boldsymbol{T}}_o^{l,i})^{-1} \boldsymbol{X}_{i,j}^{l} \right\|^2 \right) \tag{5.18}$$

The parameters to optimize is then the rotation and translation between the cameras $\boldsymbol{R}, \boldsymbol{t}$, the left extrinsic parameters $\boldsymbol{R}_i, \boldsymbol{t}_i, i = 1, \ldots, n$ and the intrinsic parameters $\boldsymbol{K}_l, \boldsymbol{K}_r, \boldsymbol{k}_l, \boldsymbol{k}_r$ giving a total of $6+6 \cdot n+2(4+2) = 6(3+n)$ variables. The right intrinsic parameters are determined as

$$\boldsymbol{R}_r = \boldsymbol{R}\boldsymbol{R}_l, \quad \boldsymbol{t}_r = \boldsymbol{t} + \boldsymbol{R}_r \boldsymbol{R}_l^T \boldsymbol{t}_l$$

Which ensures compatibility with the epipolar geometry.

5.2. Finding the camera poses in the world frame

The robotic manipulator can be used to find the camera poses with respect to its base (world) frame. To do so a chess board pattern with known geometry is mounted on its end-effector which is detected by the cameras. Assuming the intrinsic parameters of the camera is already known the transformation from the camera to the object frame holding the calibration pattern, $\boldsymbol{T}_o^c$, can be calculated. Since the end-effector pose with respect to the base frame, $\boldsymbol{T}_e^w$, is known through forward kinematics there is only a single unknown,

but constant, transformation between end-effector and calibration pattern, $\boldsymbol{T}_o^e$. Finding this means finding the transformation between base frame and camera:

$$\boldsymbol{T}_c^w = \boldsymbol{T}_e^w \boldsymbol{T}_o^e (\boldsymbol{T}_o^c)^{-1} \tag{5.19}$$

Since $\boldsymbol{T}_c^w$ is unchanged, two different poses, 1 and 2, of the chess board relate through

$$\boldsymbol{T}_{e1}^w \boldsymbol{T}_o^e \boldsymbol{T}_c^{o1} = \boldsymbol{T}_{e2}^w \boldsymbol{T}_o^e \boldsymbol{T}_c^{o2}$$

Or re-written the same relation is expressed as

$$\boldsymbol{T}_w^{e2} \boldsymbol{T}_{e1}^w \boldsymbol{T}_o^e = \boldsymbol{T}_o^e \boldsymbol{T}_c^{o2} \boldsymbol{T}_{o1}^c \tag{5.20}$$

Which is the transformation from end-effector pose 2 to pose 1 and pattern pose 2 to 1 transformed by $\boldsymbol{T}_o^e$. For ease of notation the following is hereby used:

$$\boldsymbol{X} = \boldsymbol{T}_o^e \tag{5.21}$$
$$\boldsymbol{A}_i = \boldsymbol{T}_w^{e2,i} \boldsymbol{T}_{e1,i}^w \tag{5.22}$$
$$\boldsymbol{B}_i = \boldsymbol{T}_c^{o2,i} \boldsymbol{T}_{o1,i}^c \tag{5.23}$$

A solution to $\boldsymbol{X}$ is given in [22] when $n \geq 2$ for $\boldsymbol{A}_i, \boldsymbol{B}_i, i = 1, \ldots, n$. That is; if at least two *pairs* of images have been taken of the chess board at different locations the unknown transformation $\boldsymbol{T}_o^e$ can be found. Since the solution can use a least-squares approach it is favorable to use more to reduce the effects of noise.

Consider the arbitrary homogenous transformations $\boldsymbol{A}$ and $\boldsymbol{B}$, 5.20 can then be written

$$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{B}$$
$$\begin{bmatrix} \boldsymbol{R}_A & \boldsymbol{t}_A \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_X & \boldsymbol{t}_X \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_X & \boldsymbol{t}_X \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_B & \boldsymbol{t}_B \\ \boldsymbol{0} & 1 \end{bmatrix}$$

which shows that

$$\boldsymbol{R}_A \boldsymbol{R}_X = \boldsymbol{R}_X \boldsymbol{R}_B \tag{5.24}$$
$$\boldsymbol{R}_A \boldsymbol{t}_X + \boldsymbol{t}_A = \boldsymbol{R}_X \boldsymbol{t}_B + \boldsymbol{t}_X \tag{5.25}$$

First it is seen that finding $\boldsymbol{R}_X$ first will yield a solution to $\boldsymbol{t}_X$. Fortunately it is recognized that $\boldsymbol{R}_A = \boldsymbol{R}_X \boldsymbol{R}_B \boldsymbol{R}_X^T$ which is simply a transformation of the rotation matrix $\boldsymbol{R}_B$ to the coordinate system of $\boldsymbol{X}$. This is the same as transforming the axis of

rotation in B by $\boldsymbol{R}_X$:

$$\theta_A \boldsymbol{k}_A = \theta_B \boldsymbol{R}_X \boldsymbol{k}_B \tag{5.26}$$

The best fitting $\boldsymbol{R}_X$ is therefore the one which minimizes

$$\sum_{i=1}^{n} \|\theta_{Ai}\boldsymbol{k}_{Ai} - \theta_{Bi}\boldsymbol{R}_X\boldsymbol{k}_{Bi}\|^2 = \text{trace}\left(\boldsymbol{K}_A\boldsymbol{K}_A^T + \boldsymbol{K}_B\boldsymbol{K}_B^T - 2\boldsymbol{R}_X\boldsymbol{K}_B\boldsymbol{K}_A^T\right) \tag{5.27}$$

Or equivalently maximizes trace $\left(\boldsymbol{R}_X\boldsymbol{K}_B\boldsymbol{K}_A^T\right)$ where $\boldsymbol{K}_A = \begin{bmatrix}\theta_{A1}\boldsymbol{k}_{A1} & \dots & \theta_{An}\boldsymbol{k}_{An}\end{bmatrix}$ and $\boldsymbol{K}_B = \begin{bmatrix}\theta_{B1}\boldsymbol{k}_{B1} & \dots & \theta_{Bn}\boldsymbol{k}_{Bn}\end{bmatrix}$. This is again the orthogonal Procrustes problem for $\boldsymbol{M} = \boldsymbol{K}_B\boldsymbol{K}_A^T$ solved by (5.7).

Having found the rotation matrix the translation can be expressed as

$$(\boldsymbol{R}_A - \boldsymbol{I})\,\boldsymbol{t}_X = \boldsymbol{R}_X\boldsymbol{t}_B - \boldsymbol{t}_A \quad \Leftrightarrow \quad \boldsymbol{C}\boldsymbol{t}_x = \boldsymbol{d}$$

giving the normal equation for a least-squares solution

$$\boldsymbol{t}_X = (\boldsymbol{C}^T\boldsymbol{C})^{-1}\boldsymbol{C}^T\boldsymbol{d}, \quad \boldsymbol{C} = \begin{bmatrix}\boldsymbol{R}_{A1} - \boldsymbol{I} \\ \vdots \\ \boldsymbol{R}_{An} - \boldsymbol{I}\end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix}\boldsymbol{R}_X\boldsymbol{t}_{B1} - \boldsymbol{t}_{A1} \\ \vdots \\ \boldsymbol{R}_X\boldsymbol{t}_{Bn} - \boldsymbol{t}_{An}\end{bmatrix} \tag{5.28}$$

The linear solution to $\boldsymbol{X}$ may be used as initialization for an iterative optimization technique to reduce the effect of noise by minimizing

$$\sum_{i=1}^{n} \|\boldsymbol{A}_i\boldsymbol{X} - \boldsymbol{X}\boldsymbol{B}_i\|_F^2 \tag{5.29}$$

Once $\boldsymbol{X} = \boldsymbol{T}_o^e$ is found 5.19 can be used to average (5.9) the results of all the configurations of $\boldsymbol{T}_e^w, \boldsymbol{T}_o^c$ to retrieve the transformation from base to camera.

Note that $\boldsymbol{T}_c^w$ also could have been found directly[21] by rearranging (5.19) to the form

$$\boldsymbol{T}_o^e = \boldsymbol{T}_w^e\boldsymbol{T}_c^w\boldsymbol{T}_o^c \tag{5.30}$$

which is also constant, meaning $\boldsymbol{X} = \boldsymbol{T}_c^w$ can be found by taking the same steps as before with

$$\boldsymbol{A}_i = \boldsymbol{T}_{e2,i}^w\boldsymbol{T}_w^{e1,i} \qquad\qquad \boldsymbol{B}_i = \boldsymbol{T}_{o2,i}^c\boldsymbol{T}_c^{o1,i}$$

Then the optimization by (5.29) can be performed on $\boldsymbol{T}_c^w$ directly to reduce error prop-

agation.

5.2.1. Minimizing error for a dual camera setup

As previously seen; two cameras share certain relationships which can be found in every image pair. It is therefore important to retain the so called epipolar compatibility when calculating the poses of each camera in the world frame. The transformation between two cameras, given by $\boldsymbol{R}, \boldsymbol{t}$, can be passed by the world frame as

$$\boldsymbol{T}_r^w = \boldsymbol{T}_l^w \boldsymbol{T}_r^l$$

To ensure correct epipolar geometry and transitively consistent matrices it was opted to use the cost function

$$\sum_{i=1}^{n} \left(\|\boldsymbol{A}_{l,i}\boldsymbol{X}_l - \boldsymbol{X}_l\boldsymbol{B}_{l,i}\|_F + \|\boldsymbol{A}_{r,i}\boldsymbol{X}_r - \boldsymbol{X}_r\boldsymbol{B}_{r,i}\|_F \right)^2 \tag{5.31}$$

Where $\boldsymbol{X}_r = \boldsymbol{X}_l\boldsymbol{T}_r^l$ with optimization parameters of $\boldsymbol{X}_l$ and $\boldsymbol{R}, \boldsymbol{t}$. This is analogous to the 1 camera instance of (5.29).

5.3. Calibration via the world frame

As the particle filter is dependent on projection and not 3D measurements it might seem as though optimizing for reprojection error is the more correct method for the observation model. However Gai's method shows more accurate extrinsic results by optimizing for reconstruction. The choice of optimization measure is not straight forward. For good results it might be even be necessary to make an application specific calibration method. Next one such method is introduced.

To keep the focus on good reprojection it is here proposed to optimize the transformation between the calibration pattern and camera going via the world frame instead of the extrinsic parameters directly like Zhang[31]. In fact, looking at the observation model it is clear that the particle filter is only dependent on consistently good (re)projection via the transformation from world frame to the respective cameras, not the accuracy of the transformation itself.

There are three transformations between a camera and the calibration pattern; camera to world frame, world frame to end-effector and lastly end-effector to calibration pattern. The manipulator is assumed to produce accurate measurements of the end-effector pose and so the real uncertainties lay in the two transformations estimated using the camera

measurements.

First the transformations, camera to calibration frame, $\boldsymbol{T}_o^c$, and camera to world frame, $\boldsymbol{T}_w^c$ are calculated using the approach presented in 5.2 and parameters estimated by Zhang's method. This includes non-linear minimization of (5.31). The proposed cost function to be minimized can then be formulated:

$$C = \sum_{i=1}^{n}\sum_{j=1}^{m}\left(\left\|\boldsymbol{x}_{i,j} - \boldsymbol{P}_1\boldsymbol{X}_j^o\right\|^2 + \left\|\boldsymbol{x}'_{i,j} - \boldsymbol{P}_2\boldsymbol{X}_j^o\right\|^2\right) \tag{5.32}$$

Where $\boldsymbol{P}_k = \boldsymbol{K}_k[\boldsymbol{I} \mid \boldsymbol{0}]\boldsymbol{T}_{c,k}^o, k = 1, 2$ is the k'th camera matrix constructed from the optimized parameters $\boldsymbol{K}_k, \boldsymbol{T}_{c,1}^w, \boldsymbol{T}_o^e, \boldsymbol{R}, \boldsymbol{t}$ and the constant transformation $\boldsymbol{T}_e^w$ to form $\boldsymbol{T}_{c,k}^o = \boldsymbol{T}_e^o\boldsymbol{T}_w^e\boldsymbol{T}_{c,k}^w$. Note that $\boldsymbol{T}_{c,2}^w$ is calculated from $\boldsymbol{R}, \boldsymbol{t}$ to still ensure the intra-camera dependencies.

The proposed optimization has the nice effect of optimizing a constant amount of parameters making it perfectly scalable for increased amounts of images/configurations. In contrast to Gai's method which optimizes $6(3 + n)$ variables where n is the number of image pairs the new method uses only $6(3 + 2) = 36$. The proposed method will in other words always optimize on less parameters as $n \geq 3$.

The better scalability and computationally cheaper cost function results in a significant speed-up compared to Gai's method, especially at a high number of image pairs.

6. Experimental setup

To perform the experiments a robotics cell containing two large scale robotic manipulators where rigged with a couple of industrial ethernet cameras as described in 6.1.

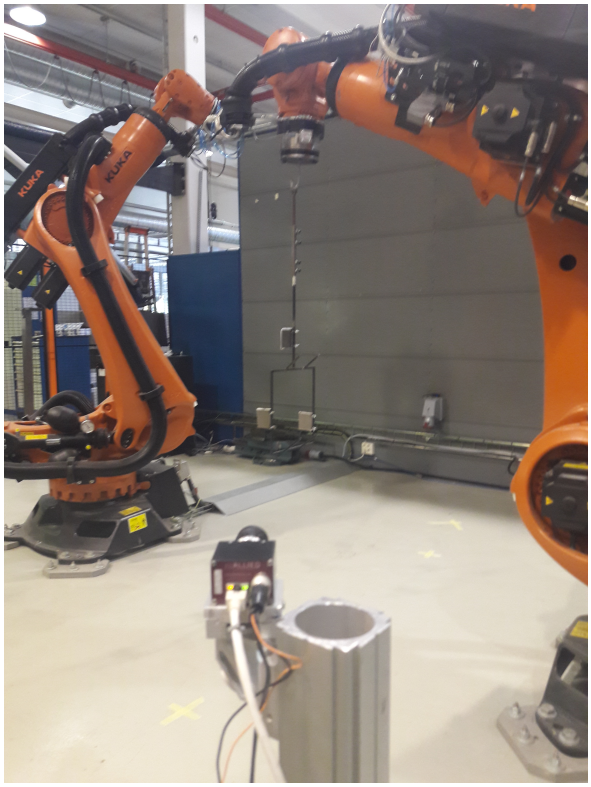

Figure 6.1.: The physical setup of the experiments.

6.1. Camera setup

Another aspect with the calibration dependence of the filter is that a slight change in setup calls for complete re-calibration (except the intrinsic camera values). It is therefore essential to have good, *automatic* calibration routines in place.

6.1.1. Environment

The cameras where placed along the y-axis of the robots base frame and a calculated distance from the trajectory to ensure that the hanger was within the image bounds at all times. Fig. 6.2 shows the scene seen from above where l is the predetermined length of the trajectory (red).

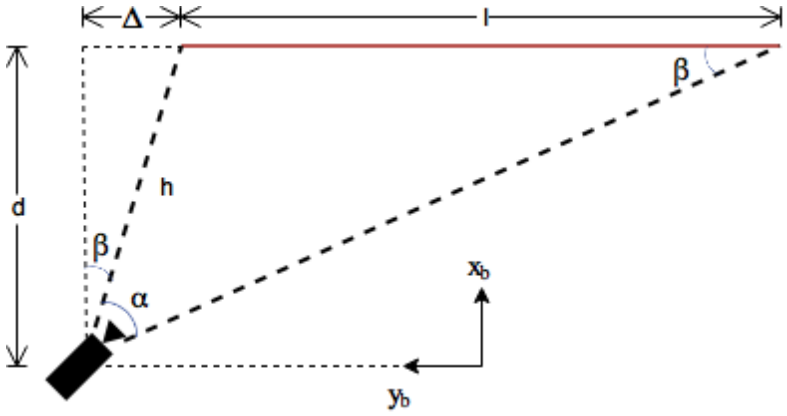

Figure 6.2.: The camera setup seen from above

α denotes the horizontal field of view (fov) dependent on the camera and image size. It can be found using the intrinsic camera parameters where the focal length, f_w, is expressed in pixel units giving half the horizontal fov as

$$\frac{\alpha}{2} = \arctan\left(\frac{\frac{W}{2}}{f_w}\right) \tag{6.1}$$

where W is the image width in pixels. The cameras where rotated 45° about the base z-axis which makes β:

$$\beta = \frac{\pi}{4} - \frac{\alpha}{2} \tag{6.2}$$

Having two angles and a distance of the fov triangle the law of sines can be used to find h, the hypotenuse of the right triangle d, Δ, h.

$$h = l \cdot \frac{\sin\beta}{\sin\alpha} \tag{6.3}$$

Which determines the distance to the trajectory d

$$\begin{aligned} d &= h\cos\beta = l\cdot\frac{\sin\beta\cos\beta}{\sin\alpha} = \frac{l}{2}\cdot\frac{\sin(2\beta)}{\sin\alpha} \\ &= \frac{l}{2}\cdot\frac{\sin\left(\frac{\pi}{2}-\alpha\right)}{\sin\alpha} = \frac{l}{2}\frac{1}{\tan\alpha} \\ &= \frac{l}{2}\cdot\frac{1-\left(\frac{W}{2f_w}\right)^2}{2\frac{W}{2f_w}} = \frac{l}{2}\cdot\left(\frac{f_w}{W}-\frac{W}{4f_w}\right) = \frac{l}{2}\cdot(\gamma-\frac{1}{4\gamma}) \end{aligned} \tag{6.4}$$

and the placement of the camera from the trajectory start, Δ

$$\begin{aligned} \Delta &= d\tan\beta = \frac{l}{2}\cdot\left(\frac{f_w}{W}-\frac{W}{4f_w}\right)\frac{1-\frac{W}{2f_w}}{1+\frac{W}{2f_w}} \\ &= \frac{l}{2}\cdot\frac{4f_w^2-W^2}{4f_wW}\frac{(2f_2-W)^2}{4f_w^2-W^2} = \frac{l}{2}\cdot\frac{(2f_2-W)^2}{4f_wW} \\ &= \frac{l}{2}\cdot\left(\frac{f_w}{W}+\frac{W}{4f_w}-1\right) = \frac{l}{2}\cdot(\gamma+\frac{1}{4\gamma}-1) \end{aligned} \tag{6.5}$$

Choosing the distance to the trajectory this way gives a fixed vertical field of view with a height of

$$d_z = d\cdot\frac{H}{f_h} \tag{6.6}$$

Where H is the height of the image in pixels and f_h is the focal length in (vertical) pixels.

6.1.2. Calibration routine

Since the focal length needs to remain the same after calibration, that is have a fixed focus, the calibration has to be performed in the operating space of the particle filter.

For a good calibration the chessboard used should exhibit a variety of orientations all over the image. The proposed way to achieve this is to distribute the chessboard positions on a spherical cap and aim the board at a chosen point as seen in Fig. 6.3.

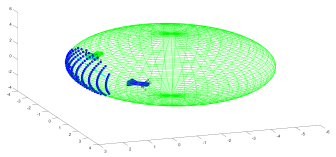

Figure 6.3.: Positions distributed on a spherical cap shown in blue.

The dimensions of the spherical cap was determined by the hanger trajectory where the base radius of the cap was set to the trajectory length. Its

height was chosen such that the angle hypotenuse
and base radius was about 10 degrees to restrict the
angle to the camera plane which already is 45 degrees to the horizontal. The cap was also restricted in the z-direction to stay within image bounds. A set of board positions in the calibration environment is shown in Fig. 6.4.

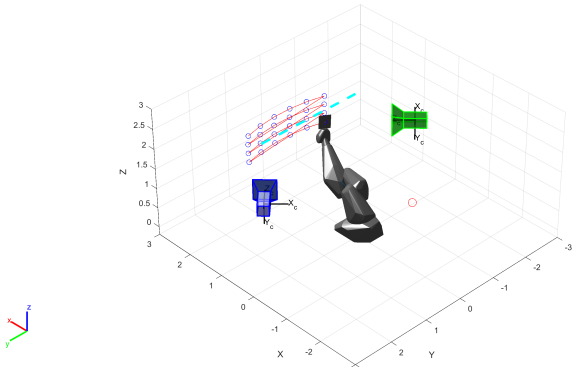

Figure 6.4.: A set of chessboard positions for calibration.

Initially the calibration patterns reference system has its z-axis aimed at the center of the sphere making up the spherical cap, the orientation is then given a random offset restricted such that the angle to the cameras still are acceptable. This is to ensure a wide range of orientation for calibration. The restriction was simply chosen to have a linear relationship to the y-coordinate:

$$o(y) = 30 - \frac{25}{y_{\max}}|y|$$

To allow greater angles when the image planes where more aligned with the calibration board (5° on the boundary, 30° in the center).

When each position and orientation of the board was calculated the joint configuration of the robotic manipulator could be found using inverse kinematics. Some restrictions where put in place to keep the manipulator out of the line of sight of the cameras. Collision detection was also implemented to guarantee a clear view of the chessboard for both cameras in each configuration, Fig. 6.5 demonstrates a collision free line of sight as well as an occlusion by the manipulator.

The routine is very configurable with respect to the amount of calibration images and guarantees a varied set of chessboard poses regardless. By using carefully calculated joint configurations it also guarantees a free line of sight to the board at all times which can be a challenge when using a robotic manipulator.

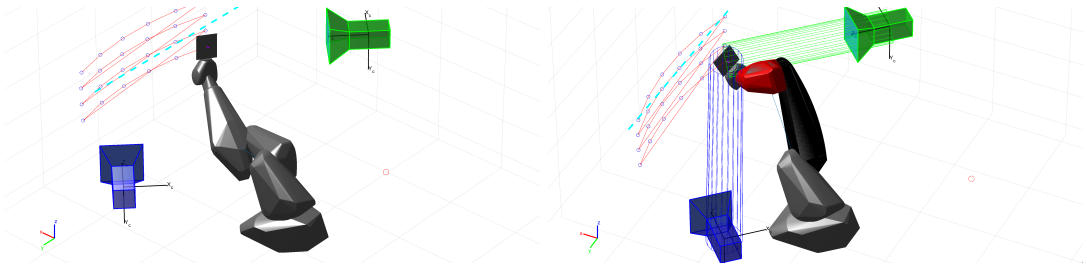

Figure 6.5.: Collision free configuration (left) as well as an occlusion by the manipulator (right) highlighted in red.

Fig. 6.6 shows a typical set of calibration poses of a chessboard seen from both camera reference frames and in the images.

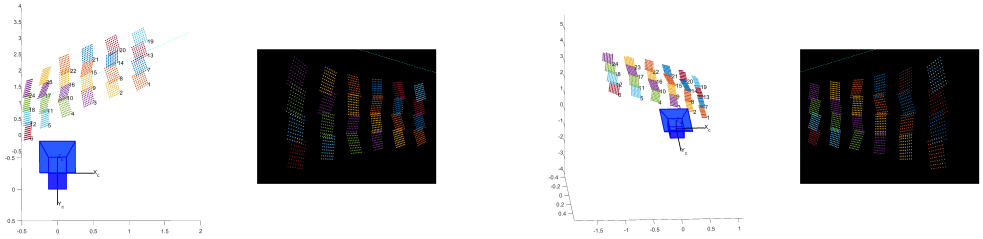

Figure 6.6.: A set of calibration poses seen from the camera perspectives and resulting images.

6.2. Cameras

The cameras where of the type Proscilica GC1020; a small packet GigE camera delivering 1024×768 grayscale images at ~ 33 fps. To interface the cameras the producers C++ SDK named `Vimba` was utilized.

The SDK, Vimba, handles the underlying transport layer and offer a straight forward way of configuring and using the cameras over ethernet. To achieve synchronized time stamps (for integration in the motion model) the time precision time protocol (PTP, IEEE 1588-2008) where enabled to synchronize the 1GHz clocks of the cameras.

Further more to attain a high frame rate the cameras where put in continuous capture mode running asynchronously to the main thread. This involves notifying a handler when a new frame has arrived and deal with it accordingly. Despite best efforts it was not accomplished to trigger the two cameras out of phase to each other through code.

A consequence of two cameras in sync is a total frame rate not much greater than what

a single camera produces despite the theoretical possibility of over 60 fps. According to the simulated results of 8.1 this will effect the accuracy of the filter negatively. In further research timed hardware triggers may be the better option.

6.3. Manipulator

The industrial manipulators used in this thesis where the 120kg Kuka variant, KR 120 R2500. As they are of the previous generation the controllers of the manipulators has limited on-line capabilities. Two ways of interfacing the manipulators where tested to determine the best option.

6.3.1. KUKAVARPROXY (KVP)

KUKAVARPROXY[25] is a third party application which let one set internal variables of the Kuka controllers using the TCP/IP protocol. In combination with a KRL (Kuka robot language) program running on the controller using the same variables for movement the manipulator can be driven to desired poses.

The advantage of KVP is the simplicity of sending a pose with location and ZYX-Euler angles and letting the controller handle all safeties and inverse kinematics. After a few tests however it became apparent that the end-effector of the manipulator must reach the given pose before any new commands are handled, any poses set in between are simply not used. In a constantly updating system as described in these experiments this is detrimental to the ability to follow the object smoothly.

The use of KVP proved ideal when calibrating the cameras as setting poses and reading the manipulator measurements through code was made considerably easier than other options.

6.3.2. Robot sensor interface (RSI)

The robot sensor interface[16] is a much lower level form of interfacing the manipulators developed by Kuka. The protocol depends on xml packets sent over UDP controlling the manipulator through given joint angles or relative position updates. This means much more dependence on the host computer which may handle everything from trajectory planning to inverse kinematics. In addition RSI demands a constant update rate of 250Hz.

Since the particle filter delivers sporadic updates limited to the frame rate of the cameras

(max 60Hz) the required RSI refresh rate means prediction has to be applied in between filter iterations. Fortunately the dynamic model of the system have already been derived in 3.1 and the module handling the RSI updates can use numerical integration in a separate thread to give intermediate poses until a new filter result is available.

A general header library for RSI communication was implemented in C++ using the Orocos kdl library[27] for inverse kinematics. The RSI library allows for simple updating of newly found poses while feeding the RSI line with poses in a manner defined by the programmer between actual updates.

Figure 6.7 shows the deployment diagram of the finished system. Each module runs in its own thread to allow for asynchronous updates from both cameras (vimba), particle filter and RSI (robot).

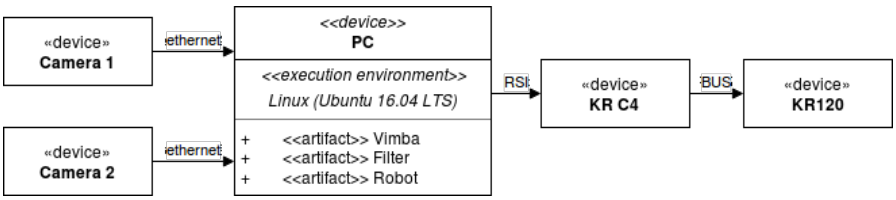

Figure 6.7.: Deployment diagram of the system.

6.3.3. Regulating the manipulator input

To avoid jumps in acceleration when first starting out or a correction of the movement is given it was decided to add a simple regulator in the joint space of the manipulator, this is described by the block diagram of Fig. 6.8.

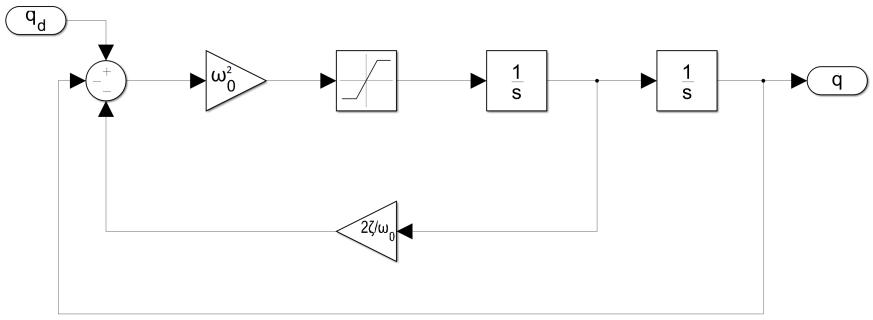

Figure 6.8.: Control block diagram of the regulator, courtesy of Prof. Olav Egeland.

The saturation ensures that no joint acceleration exceeds the limits of the manipulator

while the rest of the diagram equates to the transfer function

$$\frac{q}{q_d}(s) = \frac{1}{1 + 2\zeta\frac{s}{\omega_0} + \frac{s^2}{\omega_0^2}}$$

which gives the (non-saturated) acceleration

$$\ddot{q} = \left(q_d - (q + \frac{2\zeta}{\omega_0}\dot{q})\right)\omega_0^2$$

The joint angle is then found through integration of $\ddot{q}$.

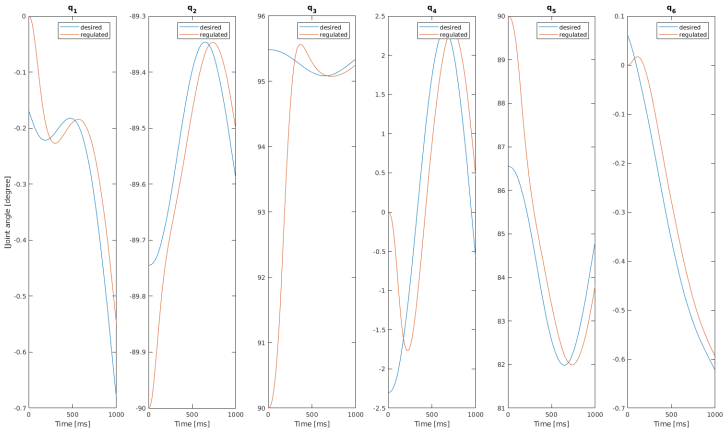

Figure 6.9.: The desired and regulated angle for each joint.

Fig. 6.9 shows desired and sent joint angles for a typical movement of a suspended object. It is seen that the conservatively tuned regulator quickly caught up to the desired joint angles for each separate joint. The regulator in fact proved very useful as the manipulator showed tendencies to jerk and shut down easily for joint control over RSI.

7. Calibration results

To aid in the choice of calibration method some simple tests where performed. The three methods: Zhang[31], Gai[6] and the novel approach of section 5.3 each represent a different objective. Zhang uses reprojection based on the intrinic and extrinsic parameters in its optimization to get an all-round good calibration. Gai utilizes 3D reconstruction in stead to use the constant camera relationship to better the extrinsic parameters. The method developed in this thesis is even more application specific and tries to minimize reprojection error via the world/base-frame to each camera.

As this thesis focuses on the particle filter application the testing criteria will reflect this. Important to the observation model of the filter is the intrinsic parameters and the camera to world frame transformation as seen in (3.24).

7.1. Simulation

To get simulated data a checkerboard pattern was generated with 3cm squares and moved to 24 predefined poses[1] with respect to each camera. The pattern was projected at each configuration using typical camera parameters and the found image points was perturbed by Gaussian noise with zero mean.

Each test was performed at different noise levels at intervals of 0.2 and to investigate consistency the data was collected several times at each level to get an arithmetic mean.

7.1.1. Intrinsic results

In stead of opting for a geometric measure it was chosen to use a simple algebraic difference between the estimated and ground truth calibration matrix to get a single parameter for comparison. This only reveals a level of difference but will give grounds for some comparison between the calibration routines. To achieve this the Frobenius norm was used on each difference matrix to produce the metric

$$d = \left\| \hat{\boldsymbol{K}}_1 - \boldsymbol{K}_1 \right\|_F + \left\| \hat{\boldsymbol{K}}_2 - \boldsymbol{K}_2 \right\|_F \tag{7.1}$$

[1] see chapter 6.1

Where $\hat{\boldsymbol{K}}$ is the estimated calibration matrix and $\boldsymbol{K}$ the ground truth. The results of 10 iterations on each noise level are shown in Fig. 7.1

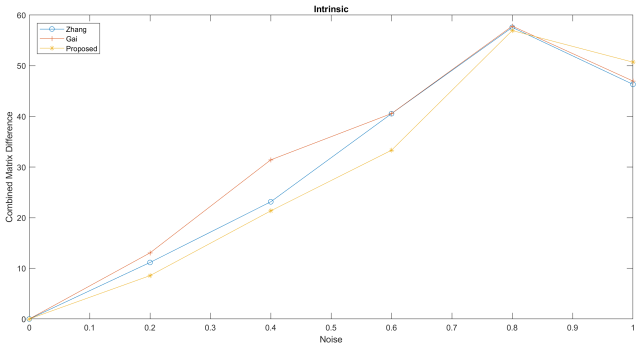

Figure 7.1.: Combined difference of the two intrinsic calibration matrices.

7.1.2. Epipolar relationship

While not important to the particle filter it is still interesting to see how the different calibration methods affect the estimated camera relationship $\boldsymbol{R}, \boldsymbol{t}$. To measure translation difference to the ground truth the euclidean distance was used:

$$d_t = \left\| \boldsymbol{t} - \hat{\boldsymbol{t}} \right\| \tag{7.2}$$

where $\hat{\boldsymbol{t}}$ is the estimated translation. For the orientation measure it was opted to use the angular distance; the smallest angle between two orientations. The difference between to rotation matrices is

$$\boldsymbol{R}_e = \boldsymbol{R}\hat{\boldsymbol{R}}^T$$

The angular distance is then defined as

$$d_R(\boldsymbol{R}, \hat{\boldsymbol{R}}) = d_R(\boldsymbol{R}_e, \boldsymbol{I}) = \theta_e \in [0, \pi] \tag{7.3}$$

where θ_e is the rotational element of the angle axis representation in radians. The average error of the transformation between the cameras can be seen in Fig.7.2

A clear trend is seen for the intrinsic accuracy as noise increases. It looks as though the proposed method performs roughly equal to the other methods at low noise levels but has a significant advantage for noisy measurements. The non-liner optimization of the transformations calculated by Zhang's method and used in the new proposed one can arguably be seen as reducing noise.

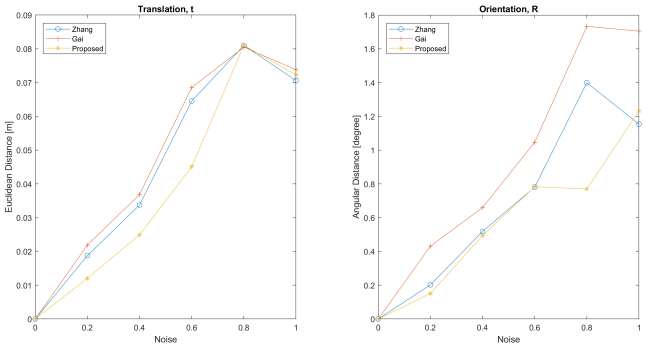

Figure 7.2.: Average error of the camera to camera transformation.

It is then important to note that the initialization for Gai's method differed slightly from the proposed method. Gai used the initial estimate from Zhang's algorithm while the proposed technique utilizes the camera to camera transformation found from optimizing the camera to world transformations as seen in (5.31).

7.1.3. Camera to world frame transformation

As a measure of the correctness of the Euclidean transformation between the cameras and the world frame the two same distance metrics as the previous section where used; euclidean distance of the translation difference and angular distance between the orientations. Both $\boldsymbol{T}_{c,1}^{w}$ and $\boldsymbol{T}_{c,2}^{w}$ where tested against the ground truth and their errors where combined such that

$$d_t = \left\| \boldsymbol{t}_{wc1} - \hat{\boldsymbol{t}}_{wc1} \right\| + \left\| \boldsymbol{t}_{wc2} - \hat{\boldsymbol{t}}_{wc2} \right\| \tag{7.4}$$

and

$$d_R = d_R(\boldsymbol{R}_{c1}^{w}, \hat{\boldsymbol{R}}_{c1}^{w}) + d_R(\boldsymbol{R}_{c2}^{w}, \hat{\boldsymbol{R}}_{c2}^{w}) \tag{7.5}$$

Figure 7.3 shows the results of averaging 10 runs of calibration at each noise level.

Again the proposed method performs better as the noise increases compared to the other two. Gai's method also does not showcase the promised results as seen in [6]. The calibration distances in the original experiments were much closer to the cameras (albeit the baseline was also much shorter) which might have an effect as triangulation generally become less accurate with increased depth.

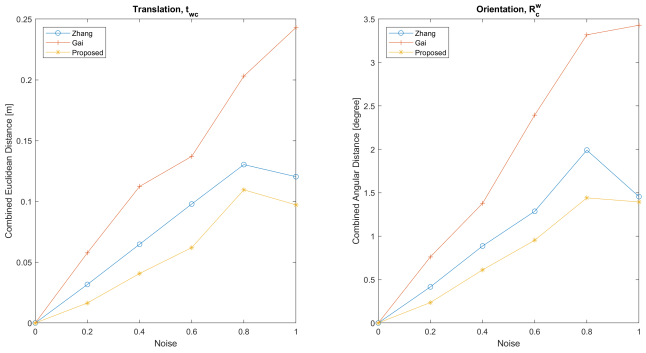

Figure 7.3.: Combined error of the camera to world transformations.

7.2. Experiment

After the cameras where placed according to the calibration routine of section 6.1 a total of 25 images where taken of the chessboard in different poses which can be seen in Fig. 7.4. Out of the 25 images the calibration routine rejected 5 due to various reasons.

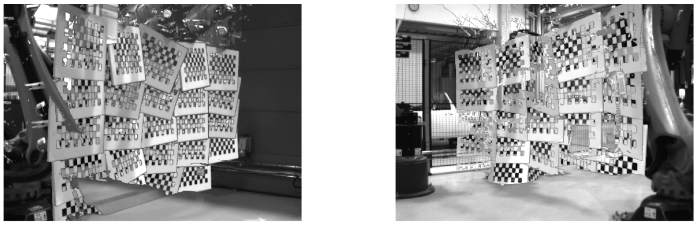

Figure 7.4.: A composite of the calibration images taken with the left and right camera respectively.

Fig. 7.5 shows the results of the estimated extrinsics and reveals a close resemblance to the set calibration poses of the routine.

Both the results of Zhang's method and the proposed refinement where tested against each other. A quick comparison of the output showed near identical calibration matrices, then the only difference that remains is the placement of the cameras in the world frame. Even this difference was rather small, about 2mm in each direction for the translation of camera 1 and at most 10mm for that of camera 2.

The differences are so insignificant that in the typical work area of the filter the difference are at most a couple of pixels when reprojecting. Such a small variation makes it difficult to get any quantitative data in a straight forward manner. It was therefor opted to simply

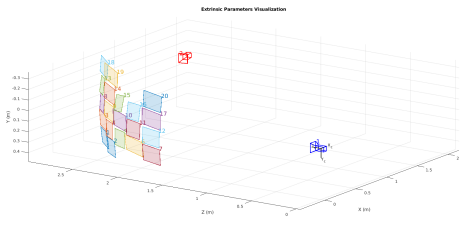

Figure 7.5.: The estimated poses of the chessboard and camera 2 relative to camera 1.

point at camera 2 (which showed the most difference) with the manipulator whose base the camera was calibrated for. This gives a simple indication for which method is more accurate with the stipulation that a 10mm deviation should be qualitatively noticeable.

Fig. 7.6 shows the results of pointing at camera 2 (with an +25mm offset in z) with Zhang's result and the proposed refinement. It is seen that Zhangs metod is closer to the center of the lens where the focal point is assumed to be.

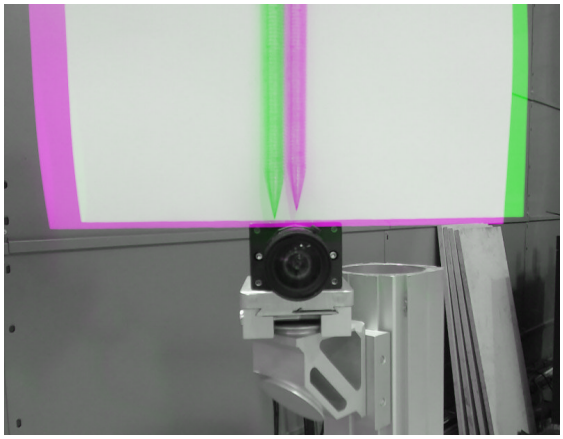

Figure 7.6.: The camera placement according to Zhang (purple) and the proposed method (green).

The same difference is not really noticeable for the first camera because of the small variations. Even though simulations pointed at the proposed method being better for pretty much every noise level the same effect was not seen in the experiment.

It is not clear as to why the simulation and experiment disagree but the proposed method does rely on accurate measurements of the end-effector pose, some noise in these measurements coupled with low noise levels on the image measurements might

help explain the inconsistency.

The difference shown in Fig. 7.6 was taken from by far the most telling angle. From other directions, as well as for the other camera as a whole, it was much harder to decide on the best result. As the other results where inconclusive it was decided to leave the proposed method in favor of the tried and trusted one, but either way would probably work due to relatively small differences.

8. Filter results

8.1. Simulations

To test the particle filter under controlled conditions a sequence of simulated images made from ground truth states was constructed. A fifteen second sequence of images at 60 fps was made by careful numerical integration of the dynamic model and an initial set of states.

The initial states where selected as to be realistic for an industrial scenario with orientations around the x- and y-axis maxing out at about 15 degrees, the rotation around the z-axis, γ, was kept to a minimum.

Fig. 8.1 shows the hanger (red) given in pivot frame (green) coordinates projected onto the image plane. The ground truth states represents the pose and velocities of the pivot frame and the hanger is depicted accordingly.

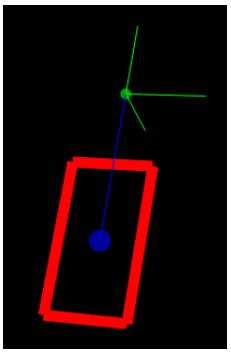

Figure 8.1.: Projection of the simulated hanger.

Fig. 8.2 shows the projected hanger at different stages of a sequence using integrated states.

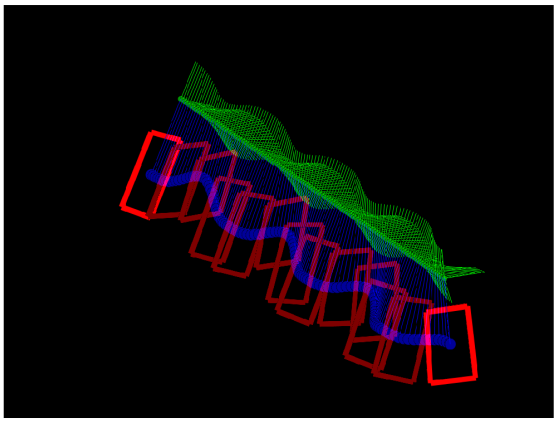

Figure 8.2.: The hanger sequence.

The deviation from the ground truth for each state variable was chosen as a test criteria and the filter was initialized close to the initial states used in the creation of the sequence. A run of 100 tests where performed to average the results and the effect of different camera placement was also investigated.

8.1.1. Single camera head on

Placing the camera directly ahead of the hanger was tried first, that is the camera was placed in the world origin with its z-axis aligned with the worlds y-axis to point at the hanger. The periodic oscillations of the hanger means that the pivot frame will align with the camera axes the most in this configuration.

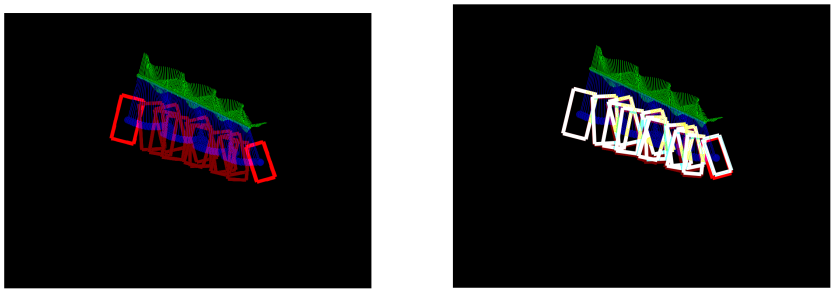

Figure 8.3.: The hanger seen from the front. Left shows the filter results overlayed in white.

As seen by both Fig. 8.3 and table 8.1 the results seem pretty accurate with the exception of the rotation around the z-axis, γ. This is to be expected as the current camera configuration is not information rich with respect to this kind of rotation.

x	y	z	$\dot{x}$	$\dot{y}$	$\dot{z}$
0.008	**0.051**	0.012	0.006	0.011	0.004

(a) Position [m] and linear velocities [m/s].

α	β	γ	$\dot{\alpha}$	$\dot{\beta}$	$\dot{\gamma}$
0.504	0.544	**5.184**	1.597	1.798	1.093

(b) Orientation [deg] and angular velocities [deg/s].

Table 8.1.: RMSE of the filter states to the ground truth using a single camera head on.

8.1.2. Single camera from an angle

In stead of pointing head on the camera is now moved in the world frame and pointed in the direction of the hanger again. This yields images of the hanger sequence from the side instead. The new camera configuration is shown in Fig. 8.4.

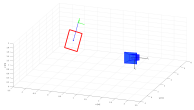

Figure 8.4.: Camera at an angle.

The results in 8.2 shows good improvement from the head on case with respect to the z-axis rotation and velocities. The slightly worse deviations of the other states is a witness that there is no perfect camera placement. Compared to the head on camera there is no doubt that having the camera viewing from the side is the favorable configuration as far as accuracy goes.

x	y	z	$\dot{x}$	$\dot{y}$	$\dot{z}$
0.031	**0.051**	0.022	0.010	0.012	0.005

(a) Position [m] and linear velocities [m/s].

α	β	γ	$\dot{\alpha}$	$\dot{\beta}$	$\dot{\gamma}$
0.386	0.601	**1.854**	1.189	2.114	0.639

(b) Orientation [deg] and angular velocities [deg/s].

Table 8.2.: RMSE of the filter states to the ground truth using a single camera from an angle.

8.1.3. Two cameras

For the last test a second camera was added in while keeping the configuration of the first. Both cameras are now watching the hanger from different point of views with the first camera (blue) having a steeper angle to the hanger as seen in Fig. 8.5.

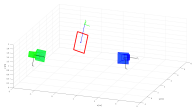

Figure 8.5.: Two cameras.

The most noteworthy result of adding the second camera is the depth improvement; a 51mm error is improved to 13mm at a 5m distance.

At first glance the single camera from an angle could almost rival the double camera

configuration but this imposes two problems; how can one ensure to always view the hanger from an angle, secondly this conclusion rests on the assumption that a single camera can deliver the same frame rate as two cameras.

x	y	z	$\dot{x}$	$\dot{y}$	$\dot{z}$
0.008	**0.013**	0.022	0.007	0.007	0.004

(a) Position [m] and linear velocities [m/s].

α	β	γ	$\dot{\alpha}$	$\dot{\beta}$	$\dot{\gamma}$
0.300	0.429	**1.964**	1.054	1.608	0.774

(b) Orientation [deg] and angular velocities [deg/s].

Table 8.3.: RMSE of the filter states to the ground truth using two cameras.

8.1.4. Effect of frame-rate

An industrial camera typically deliver ~ 30 fps. A new test with a single camera at an angle at this frame rate shows worse performance than the double camera setup as seen in 8.4

x	y	z	$\dot{x}$	$\dot{y}$	$\dot{z}$
0.123	**0.213**	0.047	0.018	0.028	0.009

(a) Position [m] and linear velocities [m/s].

α	β	γ	$\dot{\alpha}$	$\dot{\beta}$	$\dot{\gamma}$
0.749	0.745	**3.198**	2.322	2.550	0.693

(b) Orientation [deg] and angular velocities [deg/s].

Table 8.4.: RMSE of the filter states to the ground truth using a single camera at an angle at 30fps.

Note also that compensating with a smaller step size for the numerical integration in the filter had negligible effect on these results. It is concluded that viewing the hanger from more than one position is necessary to get better positional accuracy while keeping the frame rate high is equally important.

8.2. Filter convergence

The first thing to check in a real world application of the particle filter is naturally that the filter detects the object properly. To do so the hanger was placed stationary in the field of view of both cameras, the filter was then run until its output seemed to converge. Reprojecting the filter poses onto the images taken gave a good indication if the filter had converged on the right object or not.

Despite a close initialization it soon became apparent that adding the extra freedom of 3D movement let the filter find any high contrast area in stead of the actual object as can be seen in Fig. 8.6.

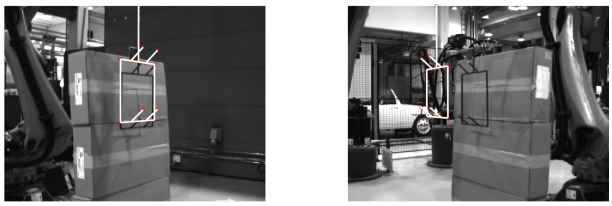

Figure 8.6.: The particle filter failed to detect the correct object due to background clutter.

In some cases the filter did not converge and the number of particles was pushed to 500 000 without luck. The observation model of the filter is clearly too simplistic for robust detection. In theory the filter can hypothesize the object to be anywhere it finds most contrast with the added degree of freedom, this puts extra burden on the uniqueness of the object to be picked up on. To help with the problem white paper was added to the hanger to enrich the contrast response, this proved quite effective as seen in Fig. 8.7

Figure 8.7.: The particle filter now detects the object correctly due to added contrast around the hanger.

This simple experiment highlighted the problem of increasing the state vector; the object

is no longer restricted to a certain position and it is therefore *much more* susceptible to clutter in the background. The only way that seem to get around the problem is highlighting the object in some way or do a rework of the observation model.

8.3. Linear tracking

As [28] already shows the effectiveness of tracking a swinging motion it was only natural to check the accuracy of the added states, namely the position in 3D space. To do so a hanger was suspended on a manipulator with a calibrated tool for the pivot point. The end-effector was then sent in linear paths along the work area of the filter while measurements where taken by both the filter and the manipulator. Since the particle filter was calibrated for the base of the manipulator in question the results are directly comparable. A velocity of about $0.06\frac{\mathrm{m}}{\mathrm{s}}$ was chosen to mimic an industrial pick and place operation.

To accompany the findings of section 8.2 the backside of the hanger was covered in white paper in hope that the detection would improve without affecting the dynamics much. The filter proved sensitive to initialization of the velocities and the best results where accomplished by letting the filter converge on the hanger while the pivot was still at rest in the x,y and z directions. Fig. 8.8 shows the filter estimates and the measured, ground truth, positions. It is seen that the filter tracks the linear movement with overall decent accuracy.

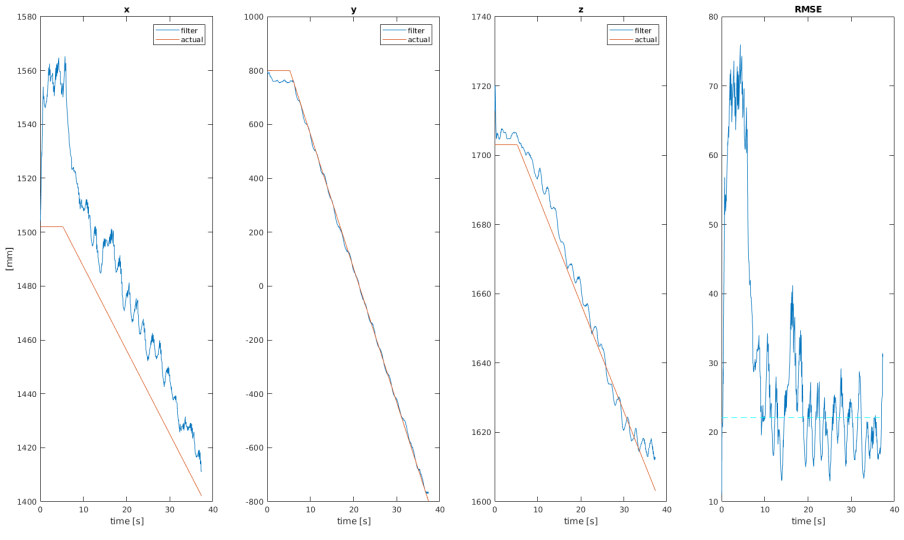

Figure 8.8.: The particle filters estimates superimposed on the measured position.

As expected the worst accuracy lies in the depth direction (here x). Eliminating the depth states, both coordinates and velocity, would about halve the error to below 1cm. This is possible in many industrial applications where the environment could be constructed to support the 2D case. Additionally constraining the states would most likely help with the convergence/detection problem of the filter discussed in section 8.2.

Table 8.5 shows the mean error of each dimension after the movement was started[1].

x	y	z
0.019	0.008	0.004

Table 8.5.: RMSE of the position estimates [m] using two cameras.

The depth error is, unsurprisingly, slightly larger than the simulated ones. Even though the object is closer in the real world experiment (about 1.5m) the frame rate was worse and therefore worse results may be expected. However; the vertical and horizontal accuracy are equal to or better than the simulation hinting at the frame rate having a smaller effect than first suspected, at least at close range.

The simulations where performed with testing the limits of the particle filter in mind and although this was not possible to recreate in the real world experiments a nice consistency was observed between the two tests.

Even though not measured directly it is still assumed that the orientation estimates where tracked well based on the smooth sinusoidal looking output of the filter shown in Fig. 8.9.

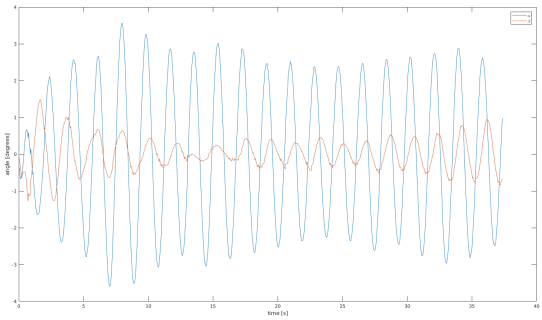

Figure 8.9.: The particle filters angular estimates of α and β.

One of the most interesting results to take away from 8.8 is the larger deviation when the

[1]To be comparable to the simulated results

parameters are static for the first few seconds; the figure clearly shows that the accuracy increases when the pivot is moving.

The test where also performed with an increase from 100k to 500k particles with no positive change in accuracy. A few tests at 50% increased speed was also conducted with little noticeable change in accuracy, higher speeds could not be tested due to limitations of the test setup.

Finally it is noted that the deviations seen are worst case in that the calibration of the pivot point where tricky. Some of the seemingly constant offsets in the depth direction especially could very well be due to poor calibration or slightly wrong measurements of the hanger.

8.4. Motion compensation

The final experiment consisted of simply testing the system as a whole. The objective of this thesis has been to accurately estimate the pose (and velocities) of a swinging object and handle it as if it was at rest.

Due to time restrictions it was not accomplished to test the particle filter output directly on the robotic manipulator safely. Any safety, excluding motor/joint limitations, had to be implemented server side, for such a large and powerful manipulator this was not something to take lightly.

Even though the joint movements are regulated through code they could still be very fast and violent given the wrong input. This was a limitation of the implementation between filter and manipulator rather than the concept in it self.

Nevertheless it was demonstrated how the KUKA manipulator could be programmed on-line to run smooth paths with typical movements of the spherical pendulum with a moving pivot. The only difference to the complete system is that the poses given to the controller where generated by the dynamic equations presented in 3.1 rather than the estimates of the particle filter.

The implemented robot controller showed great response and had no problems with accelerating up to and keeping track with given poses[2].

[2] A small video-snippet is given among the attached files.

9. Concluding remarks

In this thesis a way of implementing and using a particle filter to track an object suspended in a single pivot was given. Further a small study on camera calibration together with a new optimization was presented. Lastly the particle filters linear tracking abilities where quantitatively tested and a way of controlling KUKA manipulators for the desired system was demonstrated.

In section 8.3 the particle filter showed acceptable accuracy in tracking and estimating the Cartesian position of the pivot from 1.5-2m away with a mean deviation of about 20mm. None of the tests indicated that the filter was near its maximum capabilities with respect to both speed and distance.

However it is important to note that the most inaccurate dimension was depth and it was also found that static parameters resulted in worse estimates. In section 8.2 there where also discovered problems with the uniqueness of the object given the simplicity of the observation model to the particle filter; a correct convergence was hard to achieve when the object was allowed to move in 3D space.

The takeaway from this is that, if possible, the movements of the tracked object should be restricted to avoid parameters of little motion to better the convergence as well as the robustness of the particle filter.

The proposed optimization of the camera calibration was shown in 7.2 to have poorer accuracy than its traditional counterpart despite promising simulations in 7.1. Both methods did however give very decent results when paired with an automatic calibration routine utilizing most of the cameras field of view.

A case study of moving a robotic manipulator using the estimates of the particle filter was planned but never fully reached due to being restricted on time and therefor the inability to perform safe tests. Section 8.4 did however demonstrated how the controller of 6.3.2 smoothly controls a KUKA manipulator on-line suitable for the desired purpose.

It is the authors strong belief that with some further work the proposed system should be able to perform pick-and-place operations on a swinging conveyor with a moving pivot. The preliminary results are promising enough to warrant further investigation.

10. Future work

As this thesis simply outlines a particular and limited way of achieving the desired goals there are many ways to proceed. Some issues not addressed here include initialization of the particle filter, live parameter estimation and occlusions of the object.

Both initialization and parameter estimation may very well have to be handled automatically in an industrial application and are natural progressions from the current system. Parameter estimation may or may not be necessary dependent on the variation of parts and if loading/unloading (which changes the center of mass significantly) takes place.

In addition it was seen how more degrees of freedom made convergence more difficult, an issue not seen in [20]. Some more research must be spent on mitigating the convergence problems: either by improving the observation model of the filter, reducing the dimensions the object can travel in, or by some other means. Incidentally this also ties in with the initialization problem and could be regarded as a logical place to pick up first.

Appendices

A. Source code

As this thesis had a significant amount of programming in it, and to further future research, the entirety of the source code[1] is attached and a simple usage documentation is given here.

The main file in the base directory, although mostly untested, gives the construction and usage of the complete system. Almost everything needs to be compiled with the NVCC compiler of CUDA, local Makefiles shows the flags and dependencies used.

Calibration

Contains the automatic calibration routine written in MATLAB which store the end-effector poses in a text file. The main C++ program then moves the manipulator via KUKAVARPROXY while capturing images at each halt. The second MATLAB script can be used to estimate the camera matrices in the manipulators base frame, the results are stored in a formatted text file marked by the cameras serial numbers.

Camera

Contains C++ camera convenience classes for the Vimba API to use either as single frame capture or continuous, asynchronous captures.

Data

Mostly contain automatically generated text files to store camera matrices and test results. One important exception is the config file which contain the 3D points of line segments given in the pivot frame. Points farther down the file are weighted more.

Filter

Contains the CUDA class to run the models of the particle filter. The configuration such as number of particles, noise levels and center of mass placement is done through

[1] Complete source can also be found at `https://github.com/HakonHystad/motion_compensation`

a separate header file.

Robot

The robot class is a simple C++ header wrapper over an existing KUKAVARPROXY implementation to move the manipulator from A to B giving the pose in either meter and XYZ-euler angles in radians or millimeters and ZYX-euler angles in degrees.

The RSI class is meant as a C++ base (header) class for handling the multithreaded RSI communication, inheritance can then be used to override the update function so that the prediction between given poses/states can be implemented after ones desire. This is exemplified through the integration of the dynamic equations of a spherical pendulum.

Static test

A single CUDA main file to either test the convergence of the filter on a static object or logging the filter output together with measurements of the manipulator to test tracking capabilities. A little quirk: To get correct file placement the executable needs to run from the base directory of the project.

References

[1] Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996.

[2] Ahmad Ali, Abdul Jalil, Jianwei Niu, Xiaoke Zhao, Saima Rathore, Javed Ahmed, and Muhammad Aksam Iftikhar. Visual object tracking—classical and contemporary approaches. *Frontiers of Computer Science*, 10(1):167–188, Feb 2016.

[3] Peter I. Corke. *Robotics, vision and control : fundamental algorithms in MATLAB*, volume 73 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, 2011.

[4] Yi Cui, Fuqiang Zhou, Yexin Wang, Liu Liu, and He Gao. Precise calibration of binocular vision system used for vision measurement. *Optics express*, 22(8):9134, 2014.

[5] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

[6] Shaoyan Gai, Feipeng Da, and Xianqiang Dai. A novel dual-camera calibration method for 3d optical measurement. *Optics and Lasers in Engineering*, 104:126–134, 2018.

[7] Gene H Golub. Matrix computations, 1996.

[8] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F - Radar and Signal Processing*, 140(2):107–113, 1993.

[9] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International Journal of Computer Vision*, 103(3):267–305, 2013.

[10] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, 2nd ed. edition, 2003.

[11] Richard I. Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997.

[12] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. pages 1106–1112.

[13] T. Honda, S. Takahashi, H. Takauji, and S. Kaneko. Vision-based tracking with extended kalman filter. *IFAC Proceedings Volumes*, 44(1):9644–9649, January 2011.

[14] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[15] N. Kantas, A. Doucet, S.S. Singh, and J.M. Maciejowski. An overview of sequential monte carlo methods for parameter estimation in general state-space models. *IFAC Proceedings Volumes*, 42(10):774 – 785, 2009. 15th IFAC Symposium on System Identification.

[16] KUKA Roboter GmbH. *KUKA.RobotSensorInterface 2.3*, version kst rsi 2.3 v1 en edition, May 2009.

[17] Lawrence M. Murray, Anthony Lee, and Pierre E. Jacob. Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805, 2016.

[18] T. A. Myhre and O. Egeland. Parameter estimation for visual tracking of a spherical pendulum with particle filter. In *2015 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 116–121, Sept 2015.

[19] T. A. Myhre and O. Egeland. Estimation of crane load parameters during tracking using expectation-maximization. In *2017 American Control Conference (ACC)*, pages 4556–4562, May 2017.

[20] Torstein Anderssen Myhre. *Vision-based control of a robot interacting with moving and flexible objects*. PhD thesis, Trondheim, 2016.

[21] Hui Pan, Na Li Wang, and Yin Shi Qin. A closed-form solution to eye-to-hand calibration towards visual grasping. *Industrial Robot: An International Journal*, 41(6):567–574, 2014.

[22] F. C. Park and B. J. Martin. Robot sensor calibration: solving ax=xb on the euclidean group. *IEEE Transactions on Robotics and Automation*, 10(5):717–721, 1994.

[23] George Poyiadjis, Arnaud Doucet, and Sumeetpal S. Singh. Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98(1):65–80, March 2011.

[24] A.S. Sabbi and M. Huber. Particle filter based object tracking in a stereo vision system. volume 2006, pages 2409–2415. IEEE, 2006.

[25] Fred Sanfilippo, Lars I. Hatledal, H. Zhang, Matthew Fago, and K. Y. Pettersen. Jopenshowvar: An open-source cross-platform communication interface to kuka robots. *2014 IEEE International Conference on Information and Automation (ICIA)*, pages 1154–1159, 2014.

[26] Thomas B. Schön, Adrian Wills, and Brett Ninness. System identification of nonlinear state-space models. *Automatica*, 47(1):39 – 49, 2011.

[27] R. Smits. KDL: Kinematics and Dynamics Library. `http://www.orocos.org/kdl`.

[28] A. Myhre Torstein and Egeland Olav. Tracking a swinging target with a robot manipulator using visual sensing. *Modeling*, 37(1):53–62, 2016.

[29] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, 1991.

[30] Greg Welch and Gary Bishop. An introduction to the kalman filter. 8, 01 2006.

[31] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.