# Eksperiment og simulering av ugelstadkuler i en magnetisk væske

Andreas Hellesøy

12.08.02

blabla

# FORORD

Denne hovedoppgaven er blitt til delvis da jeg holdt til på Institutt for Energiteknikk (IFE) ved Lillestrøm våren 2001, og delvis ved Fysisk Institutt ved Universitetet i Bergen høsten 2001 og våren og sommeren 2002. Et og et halvt år med hovedfagsstudium har gitt meg mye. Oppholdet på Østlandet gav meg utfordinger som starten på et halvannet år langt selvstendig studium. I Bergen fikk jeg lære at ikke alt er lært på et semester i Tigerstaden.

Mange gode mennesker har krysset min sti. Mine veiledere professor Geir Helgesen på IFE, og professor Jan Petter Hansen i Bergen, har ledet meg på den rette sti og hjulpet meg i oppoverbakkene. Med Jan Petters visjoner og Geirs erfaring med ugelstadkuler har de vært et radarpar.

Jeg vil gjerne takke alle på fysikk-avdelingen på IFE for kjekke kaffe-pauser, lunsjer og innblikk i forskjellige forskningsfelt som for eksempel hydrogenlagring. Spesielt vil jeg takke Kai deLange Kristiansen og Jørgen Akselvoll ved IFE som begge har vært svært hjelpsomme og hyggelige i både faglig og sosial sammenheng. Arne T. Skjeltorp må heller ikke glemmes. Han er på en måte dette hovedfagets bestefar (se referansene), og vet mye om det meste.

I Bergen har det heller ikke vært mangel på gode hjelpere og venner. Først og fremst vil jeg nevne Ingrid Sundvor som har vært med meg gjennom hele fysikk-studiet fra første kurs til og med hovedfag. Hun slo meg dog på målstreken, men vi har hatt mange bra og mange usaklige diskusjoner opp gjennom studiet. De andre på atomfysikk-gruppen i Bergen fortjener også en stor takk for sitt engasjement for alle mine små og store problemer.

Thierry Matthey ved Institutt for Informatikk i Bergen er (delvis) mannen bak simuleringsverktøyet jeg har brukt i denne oppgaven. Han har tilbrakt så mange timer for å løse mine programmeringsproblemer og ulykker at hvis jeg skulle betalt ham en krone i timen hadde jeg vært en fattig mann. Heldigvis har han stilt opp gratis og gladelig uansett vær og vind.

Min solstråle gjennom studietiden, Synne Selvik, har vært fantastisk tålmodig og tilgivende når kveldene på kontoret har vært lange og alt som ikke handlet om fysikk ble plassert lengst ned på prioriteringslisten.

Til slutt vil jeg takke familie, svigerfamilie, venner og svigervenner for engasjement, oppmuntring og støtte.

Bergen, august 2002

Andreas Hellesøy

# Innhold

# Innledning

Magnetiske dipoler finnes overalt i naturen og i alle størrelsesordener. De minste er ”elementær-dipolene” som skapes av elektronenes bevegelse rundt atomkjernene. Den største i våre daglige liv er jordens magnetfelt, som har vært brukt av seilere og utforskere i mange hundre år for å finne veien frem og tilbake. Grekerne kjente til at ”lodestone” hadde magiske tiltrekkende egenskaper allerede 800 år før Kristus. Først på 1800-tallet begynte en fysisk beskrivelse av magnetisme å ta form.

I 1983 introduserte Arne Skjeltorp på Institutt For Energiteknikk (IFE) magnetiske hull i et magnetisk medium [1]. Hullene oppfører seg som dipoler med dipolmoment motsatt rettet av det eksterne feltet. Effekten kan sees på som en magnetisk analog til Arkimedes lov eller elektronhull i en elektrisk leder.

Det magnetiske mediumet er et ferrofluid som består av små monodomaine magnetiske partikler løst i en væske. Hullene er små plastikkballer kalt ugelstadkuler. Disse lages med en teknikk oppfunnet av professor J. Ugelstad ved Norges Tekniske Høyskole (NTH) [2]. Teknikken er unik fordi den kan produsere kuler i mikrometerstørrelse med svært lite standardavvik i størrelsen. Både ugelstadkuler og ferrofluider har funnet mange anvendelsesområder [3]. Ugelstadkuler brukes blant annet innenfor medisin og elektronikk. Ferrofluider brukes blant annet som dynamisk forsegling og i støtdempere [4].

Etter at Skjeltorp introduserte magnetiske hull har det blitt utført mange forsøk og simuleringer på disse kulene. To doktorgradsavhandlinger (Geir Helgesen (1990)[5] og Sigmund Clausen (1997)[7]) og en rekke artikler er produsert i den forbindelse. Piotr Pieranski ved Institute of Molecular Physics, Polish Academy of Sciences, har også bidratt til mye av arbeidet. Ugelstadkuler i en magnetisk væske er et todimensjonalt system, da kulene er plassert mellom to glassplater. Systemet er blant annet brukt for å modellere krystallisering, faseoverganger [6] og sprekkdannelse.

Skjeltorp foreslo i 1993 å benytte flette-teori for å beskrive bevegelsene til et lite antall dipoler i et magnetisk roterende felt[8][9]. Clausens doktorgradsavhandling citeclausen beskriver flette-teori som mulig bevegelses-koding.

Kai deLange Kristiansen ble i 2000 ferdig med sin hovedoppgave [10] som videre rettferdiggjør bruken av fletteteori. Modellen gav ikke helt samsvarende resultater med eksperimenter. Noen faktorer som for eksempel grensebetingelsene mellom glass og ferrofluid var ikke med i modellen.

M. Warner og R.M. Hornreich beregnet i 1985 at grensebetingelsene for magnetiske hull som i eksperimentene ovenfor løses ved hjelp av en uendelig rekke imaginære speilbilder av

dipolene kulene skaper [11]. Renauld Toussaint, post.doc. ved Fysisk institutt, UiO, har også gjort beregninger av konsekvensene av speildipolene [12].

I denne oppgaven er disse speilbildene og en støyfaktor tatt med i modellen. Denne raffineringen av modellen fører til bedre samsvar mellom eksperiment og simulering. Modellen er benyttet på ProtoMol, et molekyldynamikk (MD) simuleringsprogram utviklet ved blant annet Institutt for informatikk ved UiB. Å implementere krefter og variabler i ProtoMol slik at det kunne brukes til å simulere ugelstadkuler i en magnetisk væske er oppgavens andre hovedbeskjeftigelse.

Kapittel 1 vil utlede modellen og utvidelsen med speildipoler og støy. Andre kapittel tar for seg det eksperimentelle oppsettet og hvordan et forsøk gjennomføres. I neste kapittel presenteres ProtoMol og hvilke hjelpeverktøy til simuleringene som finnes. Kapittel 4 vil presentere resultater fra eksperimenter og tidligere simuleringer og sammenligne disse med nye simuleringer gjort med ProtoMol. Til slutt i kapitlet presenteres også en tyngre simulering med flere hundre kuler. Etter konklusjonen i kapittel 5 følger tilleggene. De to første gir en matematisk utledning av henholdsvis det effektive dipolmomentet til en kule og stokesflyt omkring en kule. Tredje tillegg gir en innføring i knute- og fletteteori. Siste tillegg lister kilde-koden til de viktigste programmene eller programdelene som er brukt i forbindelse med oppgaven og gir en kort kommentar til hver av disse.

# Kapittel 1

# Matematisk modell

Dette kapitlet vil gi en presentasjon av den matematiske modellen som er brukt til å simulere ugelstadkuler i en magnetisk væske. Modellen uten såkalte speileffekter utledes først, før speileffektene innføres og forskjellene mellom de to modellene diskuteres. En støyfaktor introduseres også. Til slutt kommenteres noen faktorer som modellen ikke inneholder.

## 1.1 Magnetiske hull

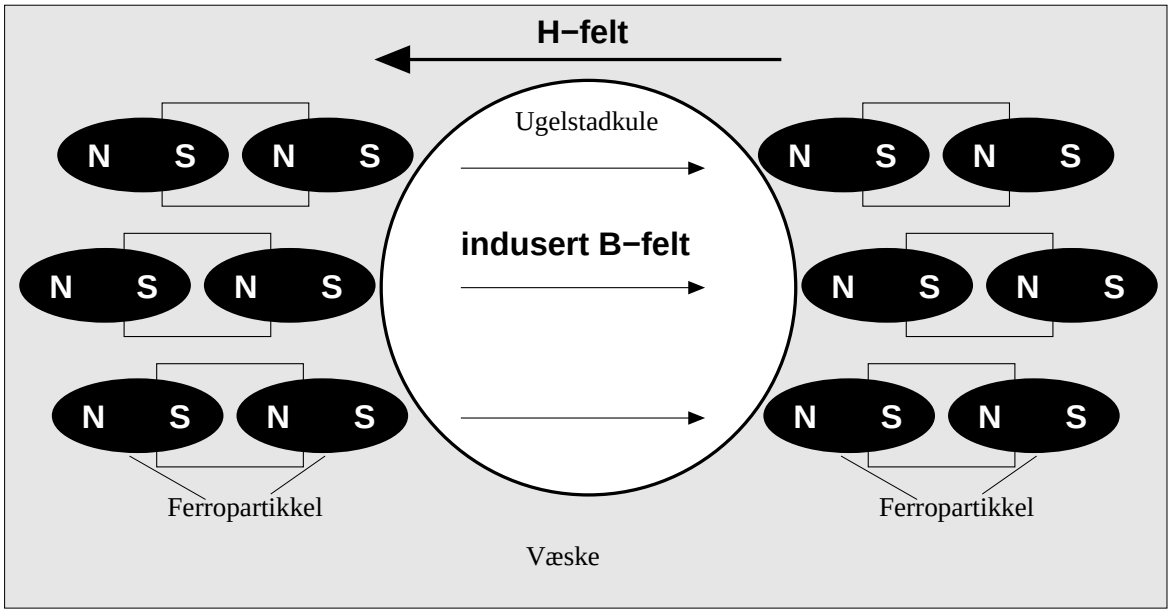
Magnetiske hull oppstår når et ellers homogent magnetisk medium blir fortrengt til fordel for et annet mindre magnetisk medium. Denne modellen skal beskrive små runde polystyrene-kuler, kalt ugelstadkuler, i en magnetisk væske, også kalt et ferrofluid. Kulene har ingen magnetisk susceptibilitet. Det magnetiske mediumet er et ferrofluid som består av små magnetiske partikler, ideelt sett uendelig små, løst i en væske. Ferrofluidet har en susciptebilitet $\chi_f$. I et ytre magnetfelt med intensitet $\vec{H}$ uten den magnetiske væsken og kulene, vil kulene virke som dipoler med dipolmoment $\vec{\sigma}$, motsatt rettet av det ytre magnetfeltet (se figur 1.1). Dette kan sees på som en analog til Archimedes lov om oppdrift, hvor et medium med lavere magnetisk intensitet fortrenger et med høyere intensitet. Uttrykket for $\vec{\sigma}$ er:

$$\vec{\sigma} = -V\chi_{eff}\vec{H}, \tag{1.1}$$

hvor V er volumet til kulen, $\chi_{eff}$ er den effektive susceptibiliteten mellom kulen og den magnetiske væsken, hvis presise form fremkommer når grensebetingelsene for magnetfeltet på overflaten til kulen tas i betraktning. Utledningen av $\chi_{eff}$ er gjort i tillegg A og resulterer i

$$\chi_{eff} = \frac{\chi_f}{1 + 2\chi_f/3}, \tag{1.2}$$

der $\chi_f$ er susceptibiliteten til den magnetiske væsken.




**Figur 1.1:** Illustrasjon av prinsippet om magnetiske hull. Polene til ferro-partiklene kanselerer hverandre i væsken, men på overflaten til ugelstadkulen vil det oppstå en polarisasjon.

## 1.2 Første tilnærming

Potensialet U til et antall, $n$, dipoler i et magnetisk felt er gitt ved:

$$U(\vec{r_1}, \vec{r_2}, ..., \vec{r_n}, t) = \sum_{i>j}^{n} \left\{ \frac{\sigma^2(t)}{r_{ij}^3} - \frac{3 \cdot [\vec{\sigma}(t) \cdot \vec{r}_{ij}]^2}{r_{ij}^5} \right\}, \tag{1.3}$$

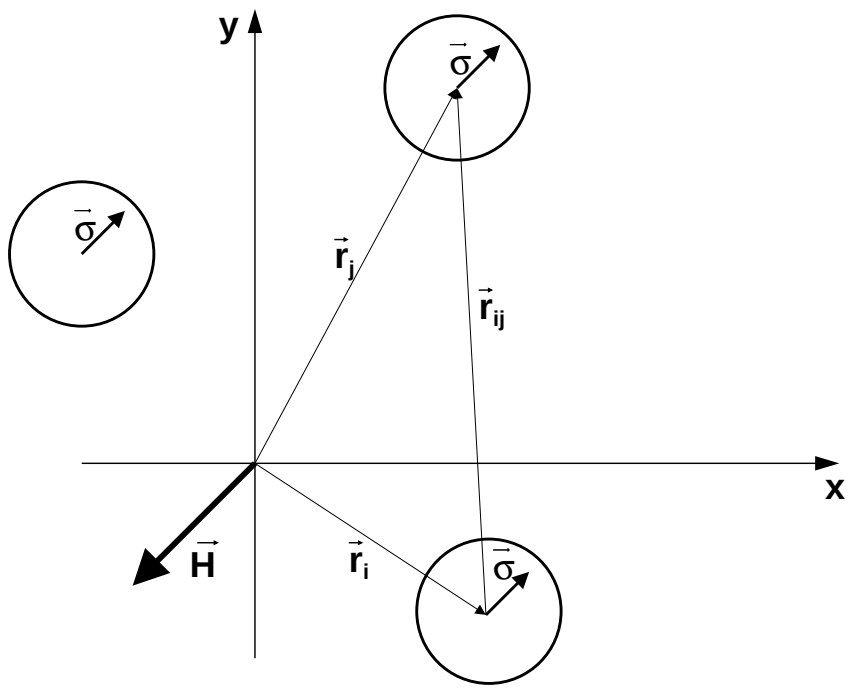
for $r_{ij} > d$, der $d$ er kulediameteren, og $\vec{r}_{ij}$ er avstandsvektoren mellom kule $i$ og $j$ (se figur 1.2). Summasjonstegnet skal forstås som summen over alle kuleparene $ij$. Kraften på kule $i$ fra de andre kulene blir dermed

$$F_i = \sum_{i \neq j} \left\{ -3\frac{\sigma^2(t) \cdot \vec{r}_{ij}}{r_{ij}^5} + 15\frac{[\vec{\sigma}(t) \cdot \vec{r}_{ij}]^2 \cdot \vec{r}_{ij}}{r_{ij}^7} - 6\frac{\vec{\sigma}(t) \cdot [\vec{\sigma}(t) \cdot \vec{r}_{ij}]}{r_{ij}^5} \right\} \tag{1.4}$$

Det siste leddet i ligning 1.4 gir et dreiemoment på kuleparet $ij$. Dreiemomentet er et resultat av en potensialdal mellom to kuler som går parallellt med feltretningen. Figur 1.3, hvor feltretningen er lagt langs x-aksen, viser potensialet til en kule med en annen kule plassert i origo. Potensialet er sylinder-symmetrisk om feltretningen.

Når kulene er i bevegelse vil det også virke viskøs motstandskraft på kulene. En enkel tilnærming til denne friksjonskraften er Stokesflyt på en kule gitt ved:

$$F_{stokes} = 3\pi\eta d\vec{v}, \tag{1.5}$$




**Figur 1.2:** Illustrasjon av magnetiske hull

der $\eta$ er viskositeten til væsken, $d$ diameteren til kulen og $\vec{v}$ er kulens relative hastighet til væsken. Utledningen av Stokesflyt på en kule er gitt i tillegg B og bygger på at væskebevegelsen foregår ved svært lave Reynolds-tall.

Reynoldstallet, $Re$, til en strømning er forholdet mellom de viskøse og de inertielle kreftene på væsken:

$$Re = av\rho/\mu, \tag{1.6}$$

der $a$ er den karakteristiske lengden (diameter på kulen), $\rho$ tettheten til væsken og $\mu$ er viskositeten til væsken. To kuler med diameter 50$\mu$m som følger et felt på 1Hz i en væske med tetthet på 1.02g/cm$^3$ og viskositet på 6mPa·s[1][2] gir $Re = 1.34$e-3 og resulterer i en laminær strøm [18].

Bevegelsesligningen for én kule er

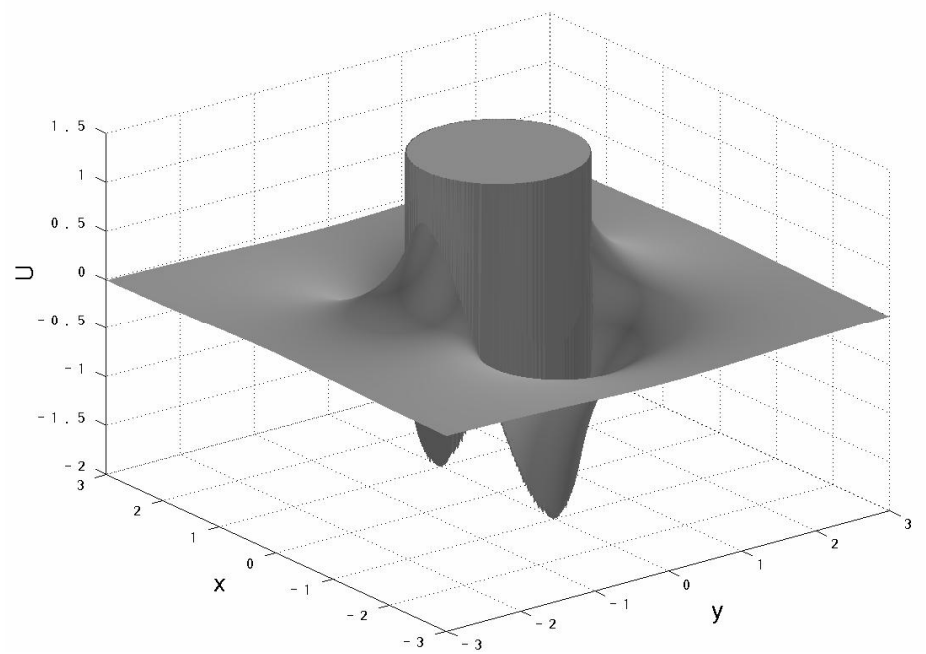
$$F_m + \xi\dot{x} = m\ddot{x}, \tag{1.7}$$

der $F_m$ er dipol-dipol-kreftene som virker på kulen, $\xi = 3\pi\eta d$ for stokes-flyt, $m$ er massen til kulen og $\dot{x}$ og $\ddot{x}$ representerer henholdsvis kulens hastighet og akselerasjon. Divisjon med $\eta$ gir

$$\dot{x} = (m/\xi)\ddot{x} - 1/\xi F_m. \tag{1.8}$$

[1]Datatabellen for EMG909 brukt i forsøkene angir en tetthet på 1.02g/cm$^3$ og en viskositet på 6mPa·s.

[2]Beregnet på reynoldskalkulatorene http://www.processassociates.com/process/dimen/dn_rey.htm og http://www.efunda.com/formulae/fluids/calc_reynolds.cfm.




**Figur 1.3:** Potensialet til en kule når en annen kule plassert i origo. Dipolmomentet i x-retning, $\vec{\sigma}$=[1,0,0]. Sirkelen rundt origo er forbudt"område hvor kulene ville overlappet.

Faktoren, $m/\eta$, i akselerasjonsleddet blir, for en kule med radius 50$\mu$m og en massetetthet på 1.05g/cm$^3$ i en væske som beskrevet ovenfor, 2.43e-5.

Den viskøse motstanden og dipol-dipol-vekselvirkningene er de to dominerende kreftene i systemet. En rimelig tilnærming til systemet er:

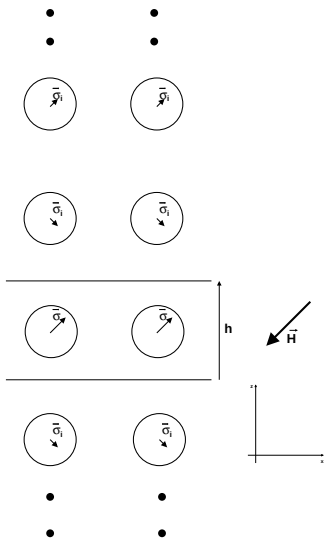
$$\vec{F}_m + \vec{F}_f = 0, \tag{1.9}$$

Denne tilnærmingen har vært brukt i en rekke arbeider (for eksempel [10] og [7]) og har i mange tilfeller gitt god overensstemmelse med eksperimenter.

## 1.3 Speil-effekter

Dipol-dipol-vekselvirkningen gitt i ligning 1.4 tar utgangspunkt i at dipolene ligger i et uendelig homogent medium. I forsøkene gjort i oppgaven kan dette ikke sies å være en god tilnærming da avstanden mellom dekkplatene på prøven ofte er i strørrelsesorden 2 kulediametre. Avstanden fra de magnetiske hullene og til grenseflaten mellom den magnetiske væsken og dekk-glassene på prøven er ikke relativt sett veldig stor. Grenseflatebetingelsene mellom glass og ferrofluid kan uttrykkes ved [21]

$$H_{||}^0 \quad = \quad H_{||} \text{ og} \tag{1.10}$$




**Figur 1.4:** Skjematisk tegning av speileffekt-prinsippet

$$B_{\perp}^{0} \;=\; B_{\perp}, \tag{1.11}$$

og må derfor tas med i modelleringen av systemet. En metode for å løse slike grense-problemer i elektrodynamikken er å innføre imaginære speilbilder. Disse speilbildene tilsvarer perturbasjonen grensebetingelsene påfører systemet (se f.eks. [17]).

Renaud Toussaint [12] har gjort omfattende beregninger på magnetiske hull med slike grensebetingelser og har vist at en uendelig serie av speildipoler oppfyller dem (figur 1.4). Med innføring av glassplatene er ikke lenger problemet sylindersymmetrisk. Koordinatsystemet legges slik at grenseflatene ligger i $xy$-planet og sentrum mellom glassplatene i $z = 0$. (Dette vil gjelde for resten av oppgaven hvis ikke annet er spesifisert.) Avstanden mellom glassplatene kalles $h$. Beregningene til Toussaint viser videre at det $i$'te speilbildet, $\sigma^i$, vil være lik den reelle kulen med en faktor $\kappa^{|i|}$, men $z$-komponenten, $\sigma_z^i$, vil veksle som

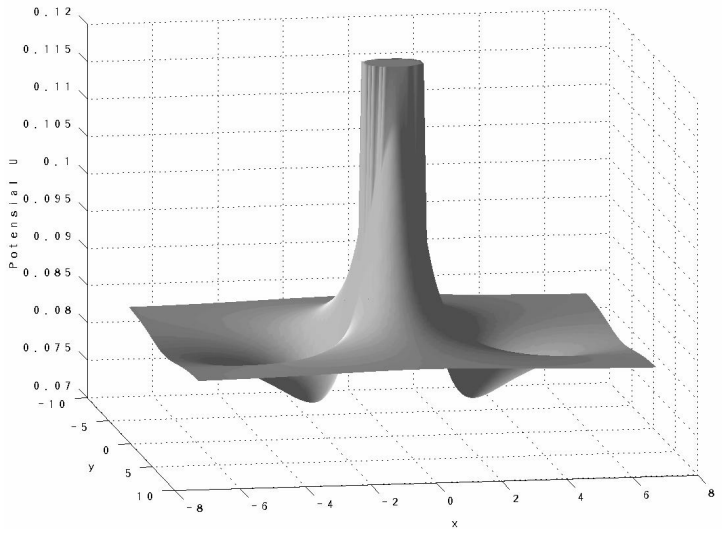
$$\sigma_z^i = (-1)^{|i|}\kappa\sigma_z, \tag{1.12}$$

der $\sigma_z$ er $z$-komponenten til det magnetiske hullets dipolmoment og $\kappa$ er gitt som:

$$\kappa = \frac{\chi_f}{\chi_f + 2}. \tag{1.13}$$

Her regnes glassplatene som perfekt plane og ikke-magnetiske.

Med en typisk $\chi_f \simeq 1$ og en $d \simeq 0,5h$, vil speil-effekten av to kuler som ligger inntil hverandre gi en korreksjon på ca. 10% (se figur 1.5 NB! størrelsen på potensialet er mye mindre enn på figur 1.3). Det er også interressant å legge merke til at potensialet ikke synker monotont mot sentrum, men har en bunn ved omkring $x$=2 i eksempelet på figur 1.5. Speileffekten resulterer også i en potensialdal med bunn i sentrum mellom de to glassplatene. Retningen til en



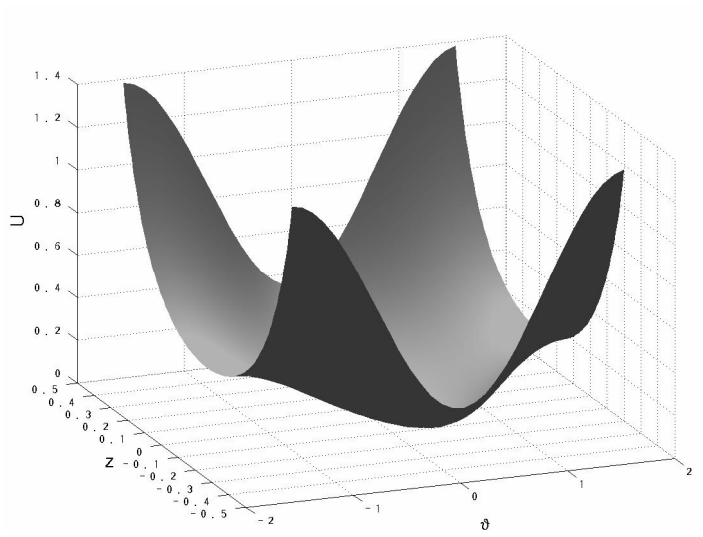

**Figur 1.5:** Potensialet omkring et magnetisk hull med $\vec{\sigma}$ =[1,0,0], $h = 2d$ og $\kappa = 1/3$ som følge av vekselvirkningen med sitt eget speilbilde og speilbildet til en kule som er sentrert i origo. Kun første speilbilde (dvs $i = \pm 1$) er tatt med i beregningen. Sirkelen i origo er området hvor kulene overlapper hverandre.

kule og dets speilbildes dipolmoment, henholdsvis $\vec{\sigma}$ og $\vec{\sigma^i}$ vil alltid ligge i samme retning i xy-planet. Derfor vil det selv for en enkelt isolert kule virke en kraft mot $z = 0$. Dette gjelder uansett om $\vec{\sigma}$ har komponent i $z$-retning eller ikke. På figur 1.6 vises potensialet til en isolert kule. $\vartheta$ angir vinkelen mellom dipolen og $xy$-planet. Potensialbunnen er for dipol parallell med $xy$-planet ($\vartheta = 0$) og i $z = 0$.

Tilpasningen til grensebetingelsene med glassplatene, og dermed innføringen av speilbilde-dipolene, fører til to viktige endringer. På grunn av symmetrien om sentrum mellom glassplatene vil det alltid virke en kraft på dipolene mot $z = 0$. Som vist på figur 1.5 har potensialet mellom en kule og en annens kule speilbilder en bunn før kulene berører hverandre. Utenfor denne avstanden fører speilbildene til en ekstra tiltrekningskraft, mens de ved kortere avstander senker tiltrekningskraften mellom to kuler.

## 1.4 Støy

Små ujevnheter i væsken og på glassplatene samt brownske bevegelser kan forekomme i forsøk. Dette introduseres med en tilfeldig kraft, $S(t)$, som virker individuelt på hver partikkel. $S(t)$ har en maksimumsverdi, $A$, og er såkalt hvit støy. Det er vanskelig å beregne hvor kraftig slik støy bør være uten å vite sikkert hva som er årsaken til støyen. Uregelmessigheter i bevegelsesmønsteret til kulene i forsøk gjør det naturlig å tro at et innslag av tilfeldighet eksisterer. Målet med støyen er å få simuleringene til å fremvise uregelmessigheter når dette




**Figur 1.6:** Potensialet til et isolert magnetisk hull med $\vec{\sigma}$ =[1,0,sin($\theta$)], h=2d og $\kappa = 1/3$ som følge av vekselvirkningen med sitt eget speilbilde. Kun første speilbilde (dvs i=$\pm$1) er tatt med i beregningen.

fremkommer i eksperimentene. Samtidig skal støyen ikke være så kraftig at det fører til uregelmessigheter der eksperimentene ikke viser uregelmessigheter.

## 1.5 Korreksjoner til modellen

I det følgende kommenteres faktorer som ikke er med i denne modellen av ugelstadkuler i et roterende magnetfelt, men som fortjener en kommentar.

- **Hydrodynamiske effekter** Lineariseringen av den viskøse motstandskraften med hensyn på hastigheten er nok den mest diskutable tilnærmingen gjort i simuleringen. Kathleen Mahoney [13] har gjort en grundigere undersøkelse av hydrodynamiske vekselvirkninger mellom ikke-magnetiske kuler i en magnetisk væske. Hovedsaklig dreier doktorgradsavhandlingen hennes om hydrodynamiske vekselvirkninger når frekvensen på det roterende magnetfeltet er relativt høy. Ferropartiklene i væsken roterer med det eksterne feltet og skaper små virvler og utgjør et komplekst hydrodynamisk system. Vekselvirkningen mellom to kuler bygger hun på Jeffrey og Onishis [14] artikkel om kuler ved lave Reynolds-tall.

  Kopling mellom partikler og væskedynamikk vil være svært komplisert [13] og har derfor ikke vært undersøkt i dette arbeidet.

  Strømningen til væsken vil også påvirkes av glassplatene som i mange forsøk ligger relativt nærmt kulene.

- **Den magnetiske væskens homogenitet** I simuleringen regnes væsken som et homogent magnetisk medium. En magnetisk væske består av små ”monodomaine” magnetiske partikler med utstrekning i et viskøst men umagnetisk medium. For væsken brukt i forsøkene i denne oppgaven, EMG 909, er de magnetiske partiklene i størrelsesorden 10nm, med en tetthet på 3,6%. Ugelstadkulene som er brukt er omtrent fire ordener større enn ferropartiklene. Utifra dette kan man anta at ugelstadkulene ser et homogent magnetisk felt.

- **Uniform kulestørrelse** Ugelstadkulene lages med en nøyaktighet i diameteren på rundt 3%. Selv om dipolmomentet øker med tredje potens av diameteren, minker den viskøse motstanden på kulen proporsjonalt med diameteren. Variasjoner på 3% kan tenkes å gi utslag i dynamikken til kulene.

- **Harde kuler** For å simulere kulenes utstrekning brukes et eksponentielt potensial. Dette potensialet skal ideelt sett gi kulene en diameters avstand når de er kommet til ro ved siden av hverandre med en gitt feltstyrke og susceptibilitet. Hvor skarp denne potensialtoppen skal være avhenger blant annet av hvor store tidssteg vi tillater i simuleringen. Det er opplagt at et slikt eksponentielt potensial ikke vil representere kulenes virkelige fysiske hardhet.

- **Speildipolene** Modellen med speildipoler forutsetter at kulene er i et homogent magnetfelt. Når to kuler er nærmt hverandre vil de påvirke hverandres speilbilder. Effekten av speilbildene er i utgangspunktet liten, og forstyrrelsen mellom kulene er neglisjert.

- **Aksellerasjonsledd** I bevegelsesligningen brukt i modellen, ligning 1.9, er aksellerasjonsleddet neglisjert da viskositeten til den magnetiske væsken er svært stor.

- **Støy** Støy-modellen beskrevet i kapittel 1.4 er relativ enkel. Mer sofistikerte modeller med variasjon i H-felt og brownsk Gauss-fordeling kan øke forståelsen for støy-fenomenene i eksperimenter.

# Kapittel 2

# Eksperiment

Forsøkene gjort i forbindelse med oppgaven ble utført på avdeling for fysikk ved Institutt For Energiteknikk (IFE) ved Kjeller utenfor Oslo. Dette kapitlet vil gi en oversikt over oppsettet til forsøk med ugelstadkuler i en magnetisk væske.

## 2.1 Forsøkets hoveddeler

De essensielle delene i et forsøk er små latex-kuler kalt ugelstadkuler[1] som ligger i en magnetisk væske. Et roterende magnetfelt produseres av to par spoler.

### 2.1.1 Ugelstadkuler

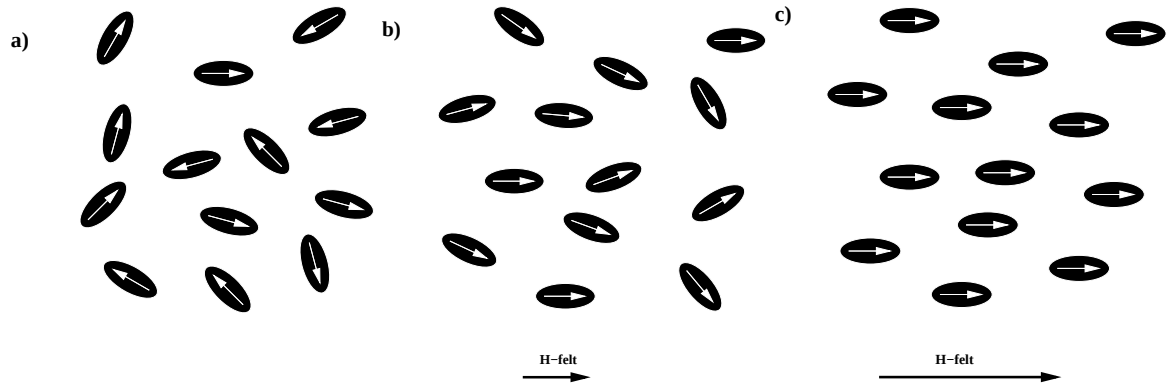
Prof. John Ugelstad oppdaget i 1980 en metode for å lage små polysteren-kuler med stor nøyaktighet. Kulene er svært sirkulære og lages nå med en usikkerhet i diameteren på$\pm 3\%$. Dynal Particles[2] leverer kuler med diameter fra $0.5\mu$m - 100$\mu$m. Kulene brukt til forsøk i forbindelse med denne oppgaven består av 98% polystyren og 2% divinylbenzen (DVB) og har en diameter på $50\mu$m hvis ikke annet er spesifisert.

Ugelstadkulene kan også lages av andre sammensetninger. De kan også gis et belegg med f.eks. gull eller et magnetisk materiale, og dermed bli elektisk ledende eller magnetisk. Elektrisk ledende ugelstadkuler kan brukes i mikro-elektronikk, og magnetiske kuler blir brukt i medisinsk behandling. Ugelstadkuler blir også brukt i LCD-skjermer for å lage et rom til krystallene under glassplaten som beskytter krystallene.

Ugelstadkulene i forsøkene knyttet til denne oppgaven brukes for å skape magnetiske hull i et magnetisk medium som beskrevet i kapittel 1.

[1] Ugelstadkuler skrives med liten forbokstav og uten bindestrek etter norsk språkråds mal. (http://www.sprakrad.no/sos)

[2] Dynal Particles, 2001 Lillestrøm, http://www.particles.no/




**Figur 2.1:** a) Ferropartikler uten et ytre magnetisk felt. b) Ferropartikler i et moderat magnetisk felt. c) Ferropartikler i et kraftig magnetisk felt.

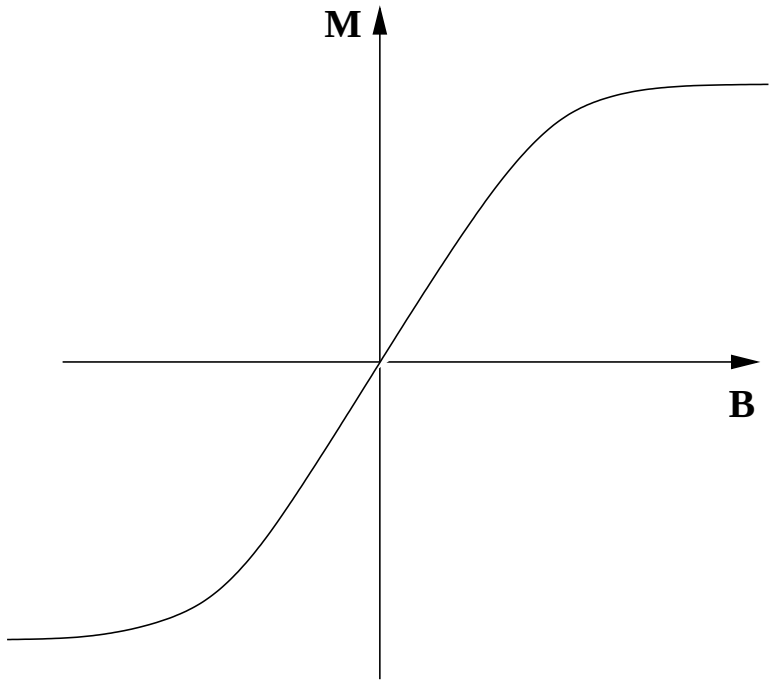
### 2.1.2 Magnetisk væske

Den andre hovedingrediensen i forsøkene er en magnetisk væske. Det finnes flere slike væsker på markedet, og i forbindelse med denne oppgaven er et ferrofluid kalt EMG 909[3] benyttet. EMG 909 består av magnetitt ($Fe_3O_4$) i en syntetisk isoparafinsk løsning. Størrelsen på magnetitt-partiklene er 10nm med en volumprosent på 3,6%. Uten et ytre magnetisk felt vil ikke ferro-partiklene ha noen netto magnetisering da termisk energi vil gi hver partikkel en tilfeldig retning (figur 2.1.a). Med et ytre magnetfelt vil dipolmomentet til partiklene ha en foretrukket retning. Hvis magnetfeltet ikke er for sterkt vil fremdeles partiklene være noe uordnet (figur 2.1.b), men ved økende magnetfelt vil partiklene bli mer og mer ordnet. Naturligvis vil det da være et metningspunkt hvor alle dipolmomentene er ordnet i samme retning som det ytre magnetfeltet (figur 2.1.c). Siden ferro-partiklene er så små vil de reagere umiddelbart på endringer i det ytre magnetfeltet, og brownske bevegelser vil raskt ødelegge partiklenes orden hvis feltet avtar. Hvis magnetfeltet øker er dreiemomentet stort i forhold til tregheten til partiklene. Relaksasjonstiden (responstiden) er typisk omkring $10^{-9}$s [3]. Dette gjør at ferrofluidet oppfører seg som et superparamagnetisk materiale uten hysteresis-sløyfe (figur 2.2).

### 2.1.3 Spolene

Spolene er laget ved avdeling for fysikk på IFE. De produserer et homogent magnetfelt med en usikkerhet på mindre enn 6% i sentrum av prøven. En mer nøyaktiv beskrivelse av spole-settet er gitt i Kristiansens hovedoppgave [10].

[3] EMG 909 lages av Ferro Tec, http://www.ferrotec-europe.de/




**Figur 2.2:** Skjematisk diagram over magnetisering vs magnetisk flux-tetthet.

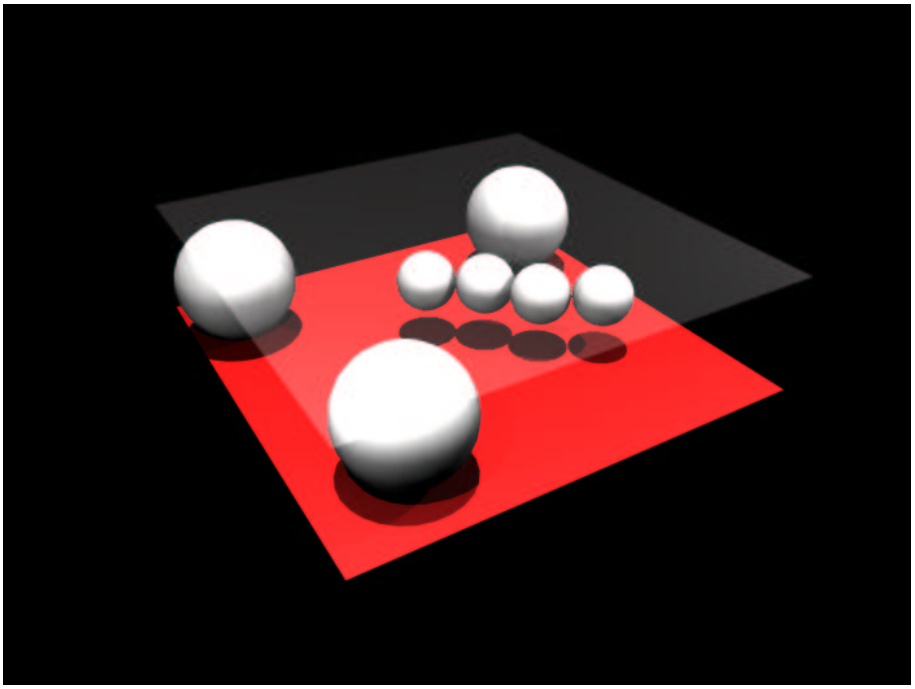
## 2.2 Klargjøring av prøver og oppstilling

Ugelstadkuler med en diameter fra $1\mu$m til $100\mu$m kan nødvendigvis ikke studeres i detalj med det blotte øye[4]. Et vanlig lysmikroskop er likevel godt nok for å gi bilder med god oppløsning til analyser. På laboratoriet på IFE står et mikroskop[5] til optisk forstørrelse av ugelstadkulene. Den magnetiske væsken som er brukt, EMG 909, er rød og svært lite gjennomsiktig. Derfor må ugelstadkulene legges i et tynt lag med ferrofluid for å være synlige. Noen større ugelstadkuler sørger for at mellomrommet mellom glassplatene er konstant; lite nok til at væsken er gjennomsiktig, men stor nok til at ugelstadkulene som skal studeres kan bevege seg fritt mellom glassplatene (se figur 2.3). De større ugelstadkulene må spres jevnt men likevel så sparsommelig at det er områder fri for disse hvor forsøket kan gjennomføres. I oppgaven har de større ugelstadkulene en diameter på $98\mu$m der ikke annet er spesifisert.

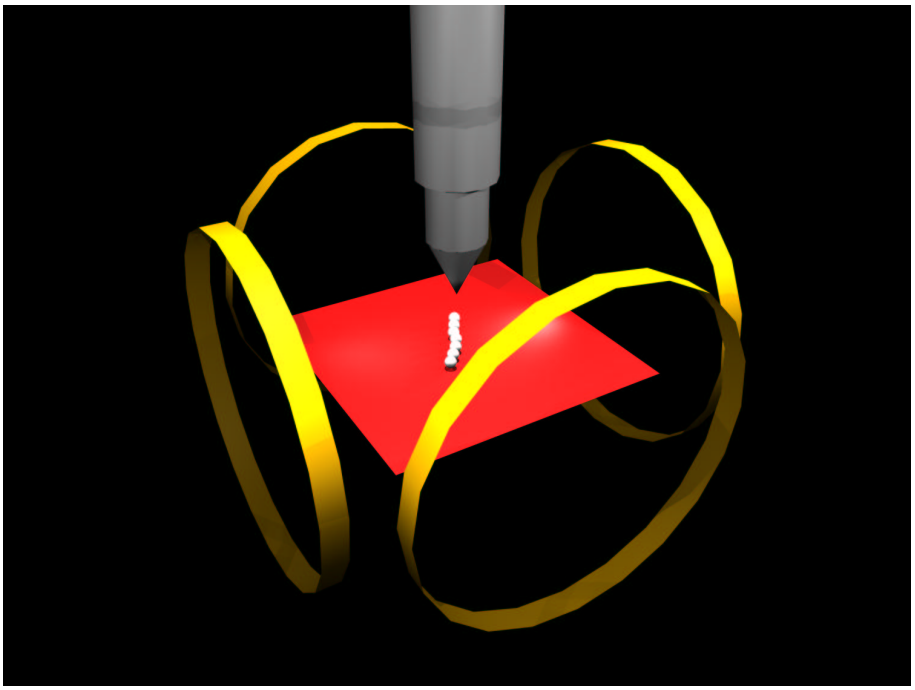
Et dekkglass plasseres over prøven og lim rundt kanten på dekkglasset forsegler prøven og sørger for at det hele holder seg på plass. Ugelstadkulene som skal studeres føres så til en egnet plass på prøven ved hjelp av en håndholdt magnet. Magnetfeltet til forsøket lages ved hjelp av to par spoler (se figur 2.4). Disse er koplet til en sinusgenerator og en variabel spenningsgenerator som gjør det mulig å produsere et sirkulært eller elliptisk roterende

[4]Selv om ugelstadkulenes bevegelser og flettemønster er umulig å følge uten bruk av mikroskop, kan de største (diameter $> 25\mu$m) sees som små prikker eller støvkorn hvis de legges på en glassplate som holdes opp mot lyset. Dette har gjort det mye lettere å isolere et bestemt antall kuler i forbindelse med et forsøk.

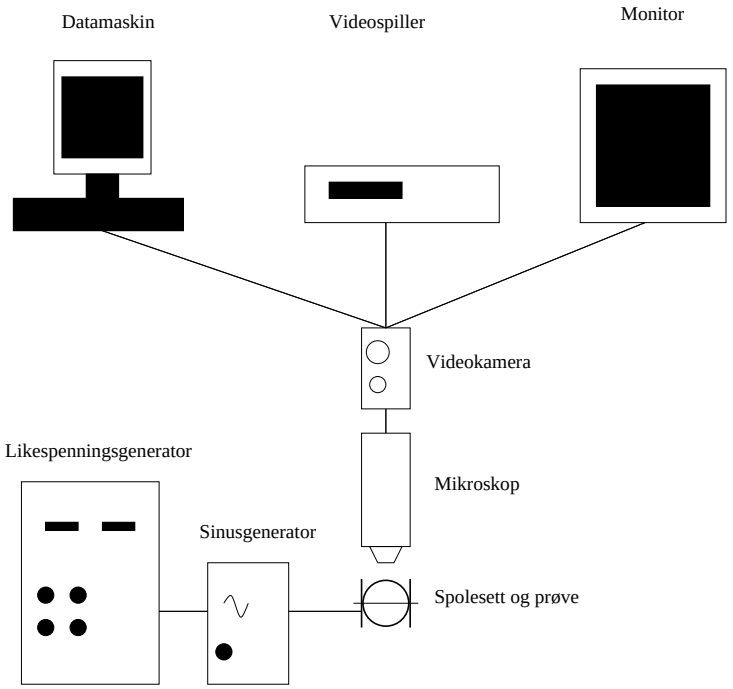
[5]Nikon Optiphot Biological mikroskop



**Figur 2.3:** Illustrasjon av større ugelstadkuler som sørger for at avstand mellom glassplatene er to ganger større enn diameteren til kulene som skal studeres.



**Figur 2.4:** Illustrasjon av oppstilling.




**Figur 2.5:** Skematisk diagram over eksperimentoppsett

magnetfelt. På mikroskopet er det montert et videokamera[6] som er koplet til en Silicon Graphics Indigo (SGI) datamaskin[7], en videospiller[8] og en monitor[9] for opptak og analyser (se figur 2.5).

Gjennom mikroskopet oppfattes ferrofluidet som ensfarget rødt og ugelstadkulene fremstår som gul-hvite kuler da de fortrenger den røde væsken. Figur 2.6 viser et videobilde fra et forsøk med 7 kuler. Bildene kan nå analyseres direkte på datamaskinen eller tas opp med videospiller for senere analyse.

## 2.3 Forsøk

Når en akseptabel prøve er blitt satt sammen og klargjort[10] for et eksperiment må spolene varmes opp til en stabil temperatur ved å la det gå strøm gjennom spolene. Prøven må også oppnå en stabil temperatur da viskositeten til ferrofluidet er temperaturavhengig. Sigmund

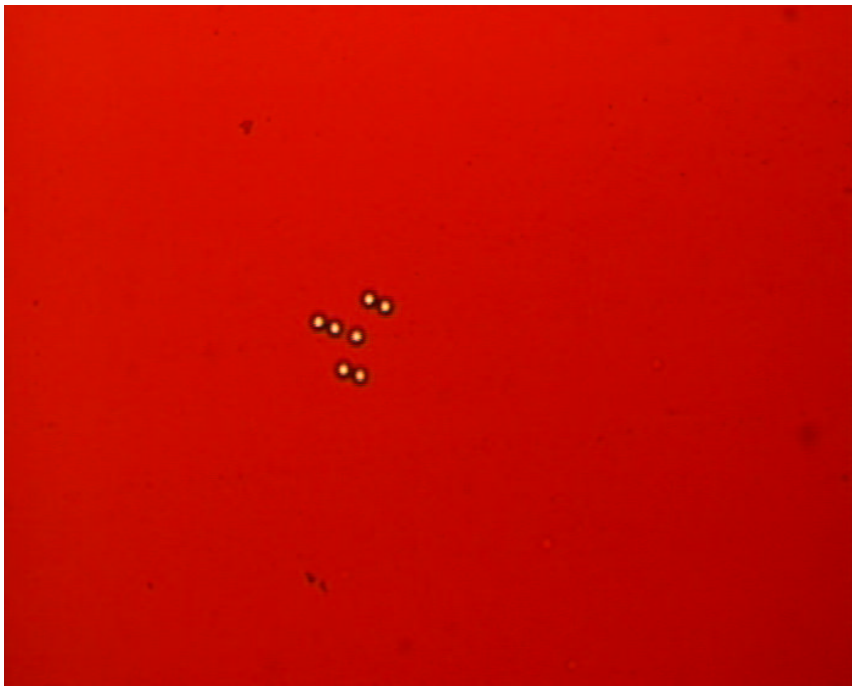
[6] Videokameraet er av typen JVC 3CCD KY-F55 BE. Kameraet har S-VHS-oppløsning (mer enn 500 linjer) og farge CCD-brikker.

[7] Silicon Graphics Indigo IMPACT arbeidsstasjon.

[8] JVC HR-DVS-2 videospiller med miniDV opptaksmulighet som gjør opptak av høy kvalitet til senere analyser mulig.

[9] Sony Triniton monitor

[10] Det er ikke lett å få de større ugelstadkulene til å ligge spredt nok og jevnt nok fordelt på prøven, samtidig som det ikke er for mange ugelstad-kuler av størrelsen som skal studeres.

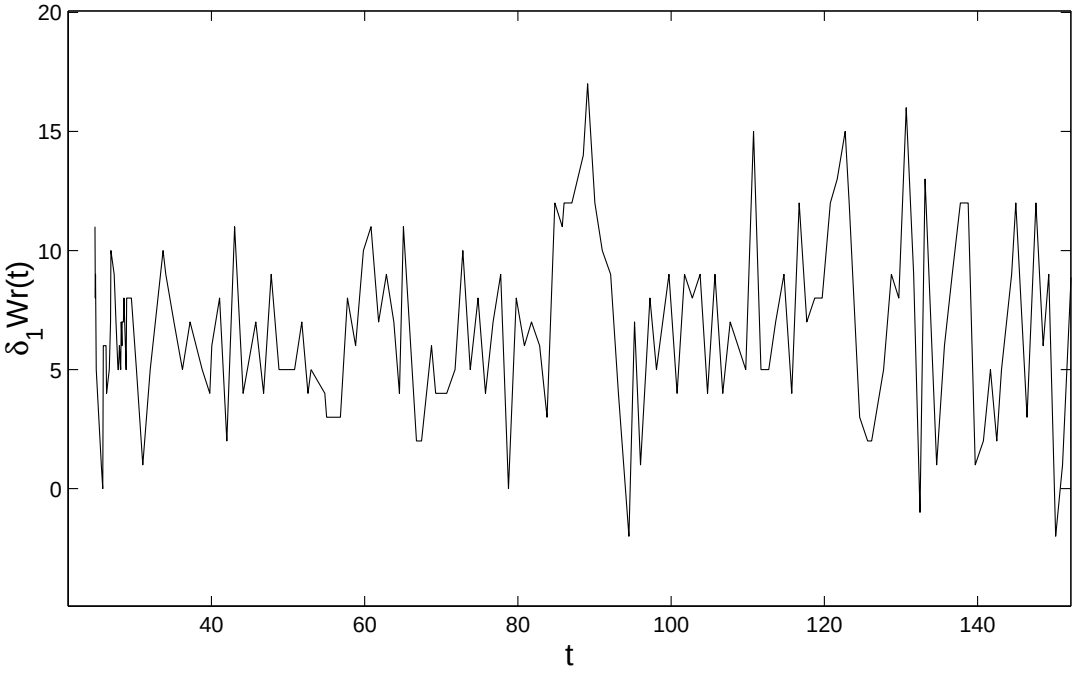


**Figur 2.6:** Eksempel på videobilde fra mikroskop.

Clausen og Geir Helgesen har skrevet et program for SGI-maskiner som henter videobilder og beregner koordinatene til partiklene på bildene, slik at analyse av eksperimentet kan gjøres i reell tid. Innhenting og konvertering av bildene er dog svært arbeidskrevende. Dette gjør at bildefrekvensen som kan analyseres i reell tid er lav, cirka. 0.5 Hz, og frekvensen på det drivende magnetfeltet må være tilsvarende lav.

Alternativt kan forsøket spilles inn på video og deretter analyseres. Fordelene er at man kan få en bildefrekvens på opptil 30Hz, og at opptakene med noe tilleggsutstyr kan analyseres på en annen datamaskin enn den som står på laboratoriet. En programpakke for å finne koordinatene til partiklene fra videobildene, find.C og trace.C, er listet i tillegg D. Figur 2.7 viser et plot fra et opptak av et forsøk med 7 kuler.

Et problem ved analyser av eksperiment er at frekvensen på det roterende magnetfeltet har en usikkerhet. Ved beregning av statistiske størrelser som for eksempel vridningshastigheten (se kapittel 4), er det viktig å kjenne rotasjonshastigheten til feltet nøyaktig.




**Figur 2.7:** Vridningshastigheten (se kapittel 4.4.3) fra et forsøk med 7 kuler i et roterende felt, tatt opp med videospiller og senere analysert.

# Kapittel 3

# Simulering

Simuleringsprogrammet brukt i denne oppgaven er ProtoMol[1] og er utviklet på Institutt for informatikk ved Universitetet i Bergen [25] i samarbeid med University of Notre Dame, USA. Dette kapitlet vil gå gjennom tilpassningen av ProtoMol til modellen beskrevet i kapittel 1 og diskutere problemer knyttet til simuleringen.

## 3.1 Generelt om ProtoMol

ProtoMol er et program utviklet for molekyldynamikk-simuleringer. Programmet kan kjøres parallellt på de største super-datamaskinene i Norge. Fleksibilitet er et nøkkelord, da integratorer og krefter lett kan endres og velges. Bruk av ProtoMol på andre systemer enn molekylsystemer (i dette tilfellet ugelstadkuler i en magnetisk væske) vil være med på å teste fleksibiliteten til programmet. Dette, sammen med at programmet utvikles delvis ved UiB, er bakgrunnen for valget av ProtoMol som simuleringsverktøy.

## 3.2 Tilpasningen av ProtoMol

Veien fra den atomære verden til ugelstadkulenes verden involverte nye utfordringer for ProtoMol. Ved å splitte de nye kreftene som skulle implementeres i ProtoMol opp i ulike kraftobjekter kan kreftene beregnes med allerede veloptimiserte algoritmer. Et kraftobjekt er en rutine (i C++) som forteller ProtoMol hva som skal beregnes og hvilken metoder som skal brukes for å beregne kraften. I samarbeid med Thierry Matthey ble ProtoMol tilpasset ugelstadkuler i en magnetisk væske med et ytre roterende magnetfelt. Resultatet av ProtoMols tilpassning ble tre kraftobjekter:

- MagneticDipole

[1] **PROTO**type **MOL**ecular dynamics

- MagneticDipoleMirror
- Friction.

Koden til kraftobjektene er listet i tillegg D.

I samsvar med kapittel 1.3 er bevegelsesplanet i simuleringene lagt i $xy$-planet med sentrum mellom glassplatene i $z = 0$.

### 3.2.1 Ytre varierende magnetfelt

I utgangspunktet var ikke ProtoMol tilpasset tidsavhengige krefter. Magnetfeltet roterer og det var nødvendig å innføre en synlig tids-variabel i kraftobjektet til dipol-dipol-vekselvirkningene. Dette var ikke gjort tidligere for andre krefter, men en peker til topologiobjektet i ProtoMol løste problemet.

### 3.2.2 Dreiemoment

Som beskrevet i kapittel 1 oppstår et dreiemoment på et kulepar som ikke ligger parallellt med det ytre magnetiske feltet. Kraftobjektet som behandler dipol-dipol-vekselvirkningen er radiell. Kraften mellom kulene returneres derfor ikke som en vektor da integratoren antar at kraften er radiell. Dette ble løst ved å endre avstandsvektoren mellom partiklene og på den måten ”lure” integratoren til å beregne en ikke-radiell kraft.

### 3.2.3 Friksjonskraft

Kraftobjektene i ProtoMol som behandler vekselvirkning mellom to partikler inneholder kun relative koordinater til partiklene, altså avstand mellom partiklene. Derfor måtte et annet type kraftobjekt behandle friksjonskraften til partiklene. Denne kraften virker på enkeltpartikler uavhengig av andre partikler; i alle fall i denne modellen av systemet. ProtoMol har kraftobjekt-typer som kan behandle en slik uavhengig partikkelmodell. Et nytt kraftobjekt som tar en konstant, $k$, som argument ble laget[2]. Kraftobjektet beregner på grunnlag av denne konstanten en kraft som er proporsjonal med hastigheten til partikkelen[3], $F = k\vec{v}$.

### 3.2.4 Speileffekter

Som beskrevet i kap 1.3 må imaginære speilbilder av kulene innføres i modellen for å tilfredsstille grensebetingelsene for systemet. Vekselvirkningen med speildipolene er egentlig bare en perturbasjon av dipol-dipol-vekselvirkningen og er således en topartikkel vekselvirkning, og må løses med samme type kraftobjekt som den ordinære dipol-dipol-vekselvirkningen. Problemet er at perturbasjonen er avhengig av avstanden til

[2] Konstanten $k$ tilsvarer $\xi$ brukt i ligning 1.7.

[3] Fordi energienheten i ProtoMol kcal/mol er hastighetene multiplisert med 48.88 når de hentes av dette kraftobjektet.

grensene, slik at det er nødvendig med partiklenes absolutte $z$-koordinater for å beregne perturbasjonen. Kraftobjektene for topartikkel-vekselvirkning inneholder ikke partiklenes absolutte koordinater. Heldigvis resulterer en partikkels vekselvirkning med sine egne speilbilder i et potensial med bunn i $z = 0$. Partiklene "presses" mot $z = 0$, og det er antatt i kraftobjektet for dipol-dipol-vekselvirkningen at kulene ligger i sentrum mellom de to glassplatene. Den sentrerende kraften på grunn av grensebetingelsene er i all hovedsak vekselvirkningen mellom en dipol og dets egne speilbilder[4] Et nytt kraftobjekt måtte lages for at dipol-selvspeilbilde-vekselvirkningen kunne beregnes. Kraftobjektet er av samme type som friksjonskraften og virker på enkeltpartikler. Denne kraften gjør at partiklene tvinges mot sentrum mellom glass-platene og tilnærmingen gjort i dipol-dipol-kraftobjektet, at dipolene ligger i $z = 0$, er rettferdiggjort. Simuleringer viser også at dipolene ligger svært nær sentrum hele tiden.

I alle simuleringene med speildipoler i oppgaven er det kun benyttet 1. tilærming. Det vil si at kun første speilbilde over og under grenseflatene er benyttet i beregningen.

### 3.2.5 Støy

Støyen beskrevet i kapittel 1.4 ble for enkelhets skyld lagt inn i kraftobjektet for friksjon. Kraften tar argumentet -rnd $n$ i konfigurasjonsfilen, der $n$ bestemmer amplituden til støyen. Kraften virker ved hvert tidssteg i simuleringen og har en komponent i hver romlige retning. En kraftkomponent beregnes ved

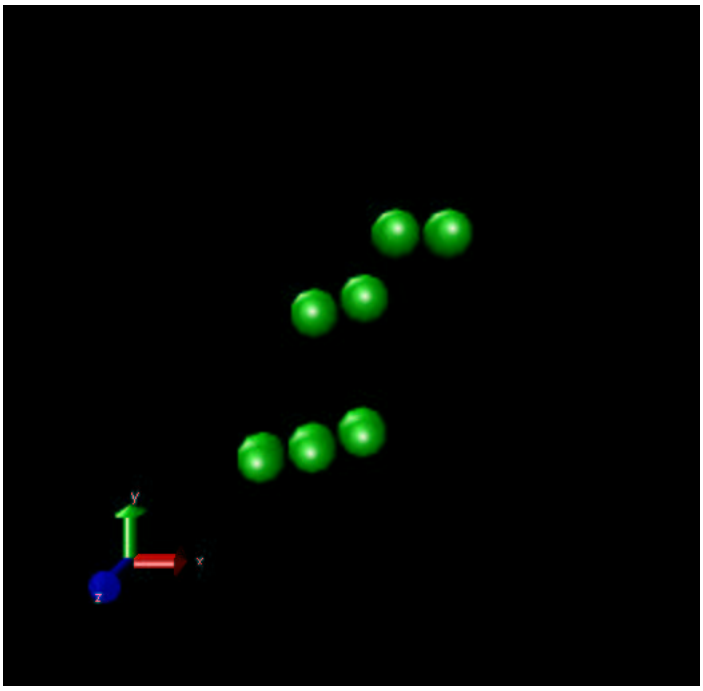
$$F^{\psi}_{støy} = \Phi_{rnd} n, \tag{3.1}$$

der indeksen til kraftkomponenten, $\psi$, indikerer en romlig retning og $\Phi_{rnd}$ er en funksjon som returnerer et tilfeldig tall mellom -0.5 og 0.5. Funksjonen som returnerer et tilfeldig tall er innebygget i ProtoMol.

## 3.3 Grafisk grensesnitt - VMD

Å få visualisert simuleringene er åpenbart en stor fordel. En visualisering er viktig for å sjekke at simuleringene går riktig for seg. Åpenbare feil oppdages lett ved å se resultatet av en simulering grafisk. Dessuten gir det en mulighet til intuitivt å sammenholde simulering og eksperiment. Det er ingen innebygget visualiserings-applikasjon i ProtoMol, men utskriftene er tilpasset blant annet et visualiseringsprogram kalt VMD[5] - Visual Molecular Dynamics. VMD har en glatt og naturlig fremstilling av kulene i forsøket. Figur 3.1 viser et øyeblikksbilde fra en simulering med 7 kuler. VMD har i tillegg et relativt lettfattelig brukergrensesnitt, men muligheter for mer avanserte funksjoner som å lage animasjoner med mer.

[4] Selv uten dipol-selvspeilbilde-vekselvirkningen ville kulene sentreres mot midten på grunn av vekselvirkningen med de andre dipolenes speilbilder. Men denne kraften er mye mindre da avstanden fra en dipol til sitt eget speilbilde er kortere enn til andres speilbilder, samt at vinkelen mellom en dipol og dets egne speilbilder gir maksimalt utbytte




**Figur 3.1:** Snapshot fra VMD som viser 7 kuler i et roterende magnetfelt

## 3.4 Raytracing

ProtoMol kan skrive ut BSDL-filer[6] som er lik spesifikasjonene til språket *RayShade*[26] som brukes av raytrace-programmet i rammeverket BOOGA[7]. Denne måten å visualisere simuleringene på er mer arbeidskrevende enn å se simuleringene direkte i VMD. Fordelen er at mye penere bilder og animasjoner til presentasjoner og liknende kan lages, og det er stort sett fantasien og arbeidsinnsatsen som setter grenser for hvordan simuleringene kan fremstå. Et eksempel på hva Raytrace er kapabel til er frank.sinatra.mpg som finnes på den medfølgende cd'en.

## 3.5 Datautskrift

Selv om visualisering av simuleringene er viktig, er de faktiske data som posisjoner og koordinater uunnværlige. Statistiske analyser og nøyaktige beregninger krever fakta på bordet. I tillegg til vanlige xyz-koordinater kan ProtoMol gi datautskrifter som ProteinDataBank-filer

---

i $z$-retning.

[5]VMD er utviklet ved University of Illinois, Beckman Institute for Advanced Science and Technology. Programmet er gratis og kan lastes ned fra http://www.ks.uiuc.edu/Research/vmd/

[6]**B**OOGA **S**cene **D**escription **L**anguage

[7]**B**erne's **O**bject–**O**riented **G**raphics **A**rchitecture er utviklet på Universitetet i Bern[23].

(PDB) som er mye brukt i molekylær simulering og Dynamical Coordinates Data (DCD), et binært format som brukes av VMD. Utskriftene kan inneholde hastighet, posisjon, kraft og energi. Programmene laget i forbindelse med denne oppgaven benytter xyz-formatet.

## 3.6 Kjøring

Protomol trenger minimum 4 filer for å kjøre. To av filene (En .par-fil og en .psf-fil) definerer molekylære strukturer og diverse parametre for molekylære krefter. Psf-filen inneholder også massen til partiklene. En .pdb- eller .xyz-fil inneholder de initielle posisjonene til partiklene. Det er også mulig å bestemme de initielle hastighetene i en hastighetsfil. Konfigurasjonsfilen angir navnet på alle disse filene samt hvilken krefter som skal benyttes og hvilken iterasjonsmetode som skal brukes. En typisk konfigurasjonsfil er listet på program 1. Kommentarene etter # gir en kort forklaring til hver linje.

## 3.7 Bruken av ProtoMol

ProtoMol har vist seg som et svært fleksibelt simuleringsverktøy. Bruk av ProtoMol har gitt forståelse for hvordan systemet kan generaliseres. Både større og mindre simuleringer har vært gjennomført på forskjellige maskiner, fra eldre linux-maskiner til den nye IBM regatta multiprossessormaskinen plassert i Høyteknologisenteret i Bergen. Andre simuleringer som er gjort med ProtoMol er ioner i coloumbkrystaller og vanndråper i gassfase[8].

[8] Video av vanndråpene på cd'en

```
temperature 100  #To create initial random velocities
firststep 0  #Defines the first step in case there is a restart etc.

numsteps 50000000  #Number of integration-steps

cellsize 20

outputfreq 200      #Frequence of output
restartfreq 100000  #How often to create restart-files
seed 1231 #random

posfile ../7s.xyz  #File containing the initial positions in xyz-format
psffile ../7s.psf  #File containing among other molecular parameteres the mass
parfile ../7s.par  #File containing (non)bond-parameteres used for molecules
usecharmm28parfile no  #Spesifies which type of charmm force-file is used

finxyzposfile 7s.out.pos.xyz  #Outputfile containing the final positions in xyz-format
finxyzvelfile 7s.out.vel.xyz  #Outputfile containing the final velosities in xyz-format
dcdfile 7s.out.dcd            #Outputfile containing the trajectories of the particles
                              #in dcd-format used by VMD
xyzposfile 7s.out.pos.xyz     #Outputfile containing the positions in xyz-format
restartfile 7s.out.restart    #Name of the restartfiles

boundaryConditions Normal  #Normal boundaries, i.e. no periodic boundaries, i.e. vacuum
cellManager Cubic  #Shape of the cell for the cell-algorithm


Integrator {
  level 0 Leapfrog {  #leapfrog-integrator
        timestep .1   #Length of one timestep

    force MagneticDipole  #The dipole-dipole and the dipole-mirror interactions force
        -algorithm NonbondedSimpleFull  #Calculates all pairs of dipoles
        -chi .61                  #Effective susceptibility of the spheres
        -r .5                     #Radius of the spheres
        -omega -0.0024898         #Angular frequency of the spheres
        -phi 0.0                  #Initial angel of the field
        -H 3.130916 3.130916 0.0  #Magnetic flux-density in x, y and z-direction
        -d 1.94                   #Separation between boundaries (glass-plates)
                                  #0.0 turns mirroreffects off.

    force Friction  #This force contains the friction and the noise
        -k -10.0    #Frictionforce = k*velocity*48.88. The velocities are multiplied
                    #with 48.88 due to the integrationmethod
        -rnd 0.01   #The amplitude of the noise.
                    #Noiseforce=noiseamplitude*random number
                    #between -1 and 1 in each direction.

    force MagneticDipoleMirror  #Calculates the centering force between a dipole
                                #and its own mirror-images.
        -chi .61                  #as for MagneticDipole
        -r .5                     #as for MagneticDipole
        -H 3.130916 3.130916 0.0  #as for MagneticDipole
        -d 1.94                   #as for MagneticDipole. Cannot be swiched off.
   }
}
```

**Program 1:** Eksempel på en konfigurasjonsfil for 7 kuler i et roterende magnetisk felt.

# Kapittel 4

# Resultater

Dette kapitlet presenterer resultater av simuleringer og eksperimenter gjort på ugelstadkuler i en magnetisk væske. Eksperimentene og simuleringene er sammenlignet for 2, 7 og mangepartikkelsystemer.

Simuleringene er gjort med ProtoMol og enhetene i programmet er tilpasset den molekylære verden[1]. Dette utgjør ikke noe stort problem da de viktigste størrelsene til systemet kan gjøres dimensjonsløse.

Systemet med to kuler er relativt sett enklere enn med flere kuler. Sammenligning mellom eksperimenter og simuleringer gir en indikasjon på om modellen er rett.

Sammenligningen med syv kuler er denne oppgavens hovedbeskjeftigelse. De bygger på Kai deLange Kristiansens hovedoppgave [10], men nye modeller er benyttet for simuleringene. I denne oppgaven er tre modeller benyttet og noen har vist seg å gi bedre samsvar med eksperimentene enn andre. I analysen av syv kuler er det brukt et spesielt matematisk verktøy kalt flette-teori. En kort presentasjon av flette-koding vil bli gitt før en oversikt over resultatene fra forsøkene med 7 kuler.

Til slutt presenteres en simulering av 500 kuler i et konstant magnetfelt. Kulene danner kjeder parallellt med magnetfeltet og gjennomsnittlig kjedelengde sammenlignes med eksperiment.

## 4.1 Dimensjonsløse størrelser

Ugelstadkuler i en magnetisk væske er et kraftig dempet system. Da aksellerasjonsleddet i bevegelsesligningen er neglisjerbart er det enkelt å utlede noen dimensjonsløse enheter som er praktisk å benytte ved simuleringer og eksperimenter. To forskjellige tidsenheter og en frekvensenhet vil bli utledet.

[1] ProtoMol bruker Ångstrøm som lengdeenhet, femtosekund som tidsenhet, amu som masseenhet og kcal/mol som energienhet.

### 4.1.1 Tiden ved konstant felt

Det er flere fornuftige måter å definere en tidsenhet ved et konstant felt. I denne oppgaven defineres en tidshenhet ved et konstant felt uten rotasjon som tiden det tar to kuler som ligger parallellt med feltet å gå fra en avstand 2 kulediametre til de treffer hverandre, $\tau_{ff}$. Uten speileffekter finner man fra den radielle komponenten i ligning 1.9 at

$$\tau_{ff} = \frac{4\sigma^2}{\xi d^3}(1 - 1/8). \tag{4.1}$$

slik at den dimensjonsløse tiden $\tau_{ff}$ blir $\tau_{ff} = t_{sim}/\tau$, der $t_{sim}$ er den simulerte tiden med samme parametre som $\tau$.

### 4.1.2 Tiden ved et roterende felt

Dipol-feltet kulene skaper er symmetrisk om magnetfeltets retning. Derfor gjentar systemet seg ved hver halve periode av det drivende magnetfeltet. For et system med et roterende magnetfelt er det derfor naturlig å definere en tidsenhet som

$$\tau_{rf} = \frac{t_{felt}}{2}, \tag{4.2}$$

der $\tau_{felt}$ er perioden til det roterende feltet.

### 4.1.3 Rotasjonshastigheten til feltet

Ved lave rotasjonshastigheter klarer to kuler (uten påvirkning fra andre) å følge feltet. Men ved en bestemt hastighet blir den viskøse motstanden større enn den tangentielle kraften på kulene, som er størst ved en faseforskjell, $\theta$, på $\pi/4$ mellom feltet og avstandsvektoren til kulene (se figur 4.1).

$\omega_c$ er den maksimale vinkelfrekvensen to kuler klarer å følge et sirkulært felt. Denne fås fra den tangentielle komponenten i ligning 1.9 og gir

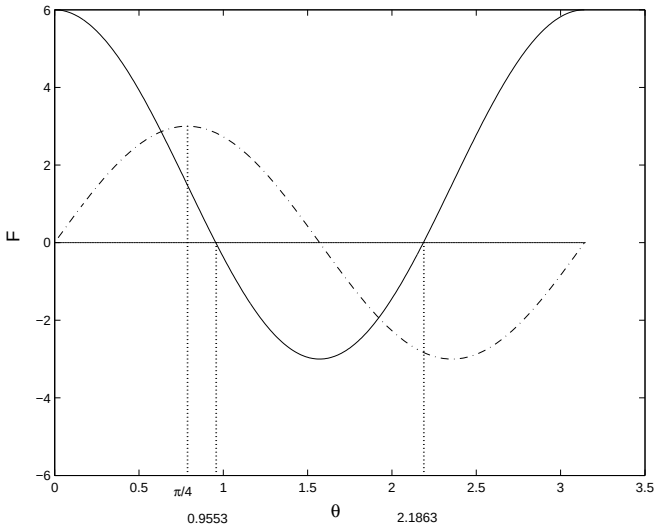
$$\omega_c = \frac{3\sigma^2}{d^5\xi} = \frac{H^2\chi^2\pi^2 d}{12\xi}. \tag{4.3}$$

Med speilbilder øker $\omega_c$ til

$$\omega_c^{speil} = \omega_c + \frac{3\kappa\sigma^2}{(d^2 + D^2)^2 d \cdot \xi} \tag{4.4}$$

med første tilnærming av speildipoler. Den dimensjonsløse vinkelfrekvensen, $\omega$, defineres som

$$\omega = \omega_{sim}/\omega_c, \tag{4.5}$$
$$\omega = \omega_{sim}/\omega_c^{speil}, \tag{4.6}$$
$$\omega = \omega_{lab}/\omega_c^{lab}, \tag{4.7}$$




**Figur 4.1:** Kraften, $F$, i tangentiell (stiplet) og radiell (heltrukket) retning for to kuler med dipolmoment = 1.0 og med en avstand = 1.0 uten speileffekter. Faseforskjellen, $\theta$, i radianer. Speildipoler vil ikke endre ekstremalverdier eller nullpunkt, men svekke den radielle kraften noe og styrke absoluttverdien til den tangentielle kraften.

der de to første ligningene gjelder for simuleringer henholdsvis uten og med bruk av speildipoler og siste ligning for eksperimenter. I eksperimentene må $\omega_c^{lab}$ finnes ved å prøve med forskjellige frekvenser[2]. (Videre i oppgaven vil det ikke spesifiseres hvilken $\omega_c$ som benyttes så lenge det ikke er av betydning eller tydelig fremkommer av sammenhengen.)

Hvis vinkelfrekvensen er høyere enn $\omega_c$ vil faseforskjellen mellom feltet og avstandsvektoren til kulene bli så stor at dipol-dipol-vekselvirkningen blir frastøtende. Når feltet drar fra kulene på denne måten kalles det faseslipp[3][15].

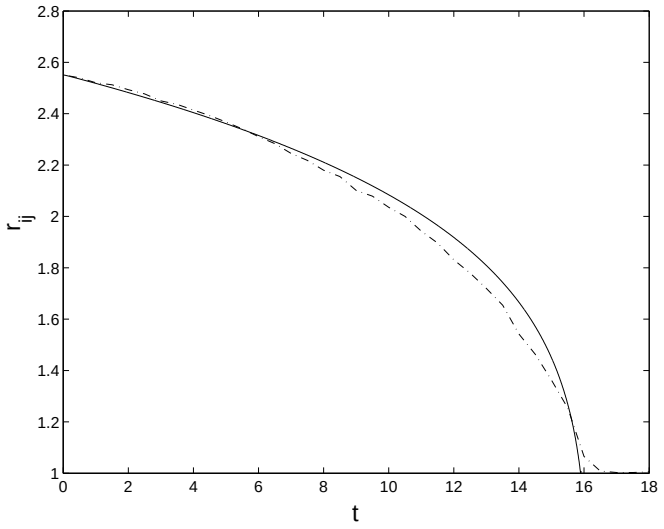
To systemer med samme antall kuler og $\varepsilon$ (og $\kappa$ og $D$ ved bruk av speildipoler) oppfører seg identisk hvis de har samme $\omega$.

## 4.2 2 kuler

Et system med to kuler er relativt enkelt. Med et roterende felt følger kulene feltet eller det oppstår faseslipp. Med et konstant felt i $xy$-planet (og uten felt i $z$-retning) vil kulene bevege seg langs hverandres feltlinjer.

[2]Dette gir en liten usikkerhet i $\omega_c^{lab}$, men den er relativt liten. Den største usikkerhetsfaktoren ved bestemmelse av $\omega_c^{lab}$ er om viskositeten er lik for to forskjellige prøver og om feltet er sirkulært.

[3]Ved svært høye frekvenser (i forsøk fra ca. 10Hz og oppover) vil kuleparet rotere mot feltets rotasjonsretning. I tillegg vil hver kule rotere rundt seg selv. Dette er grunnet vekselvirkninger mellom ferro-partiklene og ugelstadkulene[13].




**Figur 4.2:** To kuler i et konstant magnetisk felt. Tiden er i sekunder ved forsøket. Avstanden, $r_{ij}$, i kulediametre. Den punkterte linjen er fra et eksperiment utført av Geir Helgesen ved IFE, den heltrukne linjen er simulering med første tilnærming av speildipoler. Samme $\chi_{eff}$ og plateseparasjon i simulering som i eksperiment.

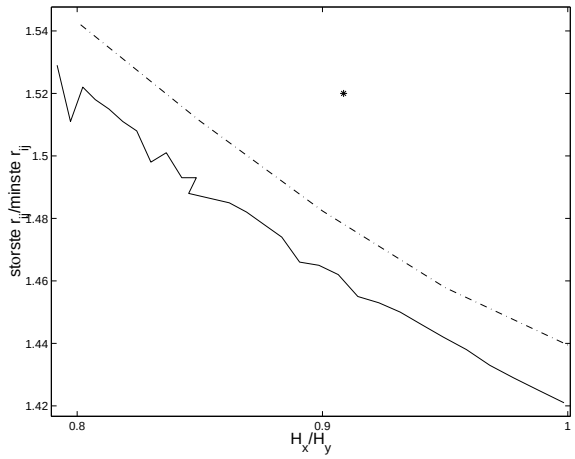
### 4.2.1 Konstant felt

To initielt separerte kuler i et konstant (ikke roterende) magnetfelt parallellt med avstandsvektoren mellom kulene tiltrekker hverandre. Ved å sammenligne separasjonen, $r_{ij}$, versus tiden, $t$, kan man se om den drivende mekanikken stemmer overens. Figur 4.2 viser avstanden mellom to kuler som funksjon av tiden under et forsøk[4] gjort av Geir Helgesen og en simulering med ProtoMol. Grafen viser god overensstemmelse mellom eksperiment og simulering.

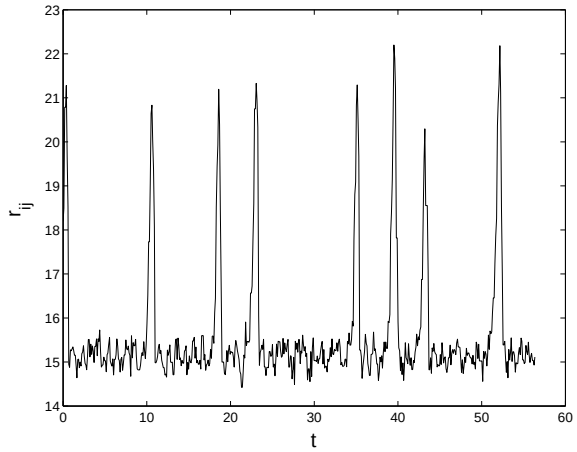
### 4.2.2 Faseslipp

Hvis den tangentielle hastigheten til kulene er for stor, slik at den viskøse motstanden blir større enn den maksimale tangentielle dipol-dipol-kraften, vil det ikke oppstå noen stabil faseforskjell mellom dipolmomentet og avstandsvektoren mellom kulene. Når $\theta$ så øker til 0.9553 ($cos^{-1}(1/\sqrt{3})$) vil den radielle kraften være frastøtende til faseforskjellen igjen blir større enn 2.1866 ($cos^{-1}(-1/\sqrt{3})$)(se figur 4.1). Maksimal separasjonsavstand ved faseslipp varierer en del med anisotropien til feltet (se figur 4.3). Ved å se på denne avstanden får man en annen indikasjon på hvor godt simuleringene samsvarer med eksperimentene. Med et anisotropt felt varierer separasjonsavstanden noe med hvilken vinkel feltet har i det faseslipp oppstår. Figur 4.4 viser et typisk plot av avstanden mellom to kuler i et forsø med relativt beskjedent anisotropt felt, $\varepsilon = 0.91$, ved en frekvens like over den kritiske.

[4]Video av forsøket finnes på cd'en: Twospheres.mov




**Figur 4.3:** Maksimal separasjon ved faseslipp som funksjon av anisotropien i feltet ($H_x/H_y$). Heltrukket linje representerer simuleringer uten bruk av speildipoler. Stiplet linje for simuleringer med speildipoler ($D = 1.94d$) sammenholdt med et eksperimentelt punkt.




**Figur 4.4:** Typisk avstand mellom to kuler i et forsø med anisotropt felt $\varepsilon = 0.91$ med vinkelfrekvens like over den kritiske. Avstanden er i bildepunkter og tiden i halve perioder av feltet.

På simuleringene og eksperimentet følger kulene med feltet i omtrent 5 til 6 perioder før faseslipp inntrer. Det er vanskelig å kontrollere at faseslipp skjer ved den vinkelen som gir størst separasjon. Derfor er det den største separasjonen av et antall faseslipp som er med på grafen på figuren. Simuleringene uten speileffekter har bare 5 til 10 faseslipp per simulering mens simuleringene med speileffekter har 15 til 20. Dette er grunnen til at kurven uten speileffekter ikke er så jevn som den med speileffekter. I tillegg ligger målepunktene tettere for simuleringene uten speileffekter.

Det er kun et eksperimentelt målepunkt. Dette fordi det var kun opptak med $\varepsilon = 0.91$[5] Usikkerheten i forsøket på grunn av oppløsning på opptaket er ca. 5%. Den menneskelige faktor kan spille en rolle når prøven skal plasseres i sentrum av spoleparene. Disse to usikkerhetsfaktorene kan være årsaken til at den eksperimentelle maksimalavstanden er noe større en den simulerte. Likevel viste 5 eksperiment med variende H-felt og frekvens samme maksimalavstand.

## 4.3 Knuter og fletter

Syv kuler, drevet av et magnetfelt, danser rundt hverandre, skifter partnere og mønster. Umiddelbart virker det ikke som om det finnes noen god måte å analysere kulenes bevegelse på, men en matematisk teori kalt flette-teori er kanskje et godt alternativ.

Dette kapitlet vil bare gi en rask oversikt over det aller mest nødvendige for å forstå metoden som er brukt for å analysere ugelstadkulenes bevegelser. En grundigere gjennomgang av knute og flette-teori er gitt i tillegg C. Dessuten finnes det mye litteratur om dette feltet (for eksempel [20]).

Knuteteorien oppstod på 1800-tallet og den første til å gi knute-teori en fysisk betydning var Lord Kelvin (William Thomson). Han la fram en hypotese om at atomene var knuter i eteren. Gauss var også tidlig ute med en elektromagnetisk teori som involverte flette-teori. I det 20. århundre ble det ikke gjort noen store fremskritt i knute-teori før V.F.R. Jones i 1986 lanserte en ny invariant[6], nå kalt Jones-polynomet. I dag benyttes knuteteori i flere interressante forskningsfelt som DNA-analyser [20] og i kjemi, statistisk fysikk og kvantefelt-teori [22].

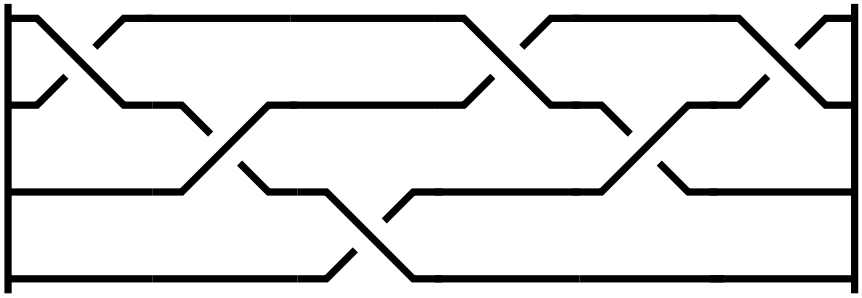
Flette-teori er en undergren til knute-teorien. Peter Guthrie Tait (1831 - 1901) var den første som laget en matematisk teori om fletter. En matematisk flette består av to eller flere tråder som er festet til et plan i den ene enden. De er flettet og spunnet rundt hverandre[7]. De andre endene er så festet til et annet plan. Et eksempel er vist på figur 4.5.

Fletten kan beskrives ved hjelp av såkalte flette-operatorer. En fletteoperator tilsvarer at to tråder i fletten krysser hverandre, og beskriver hvilke tråder som krysser og om den øverste av de to trådene krysser over eller under den andre. Nummeret på den øverste tråden telt ovenfra i fletten angis i nedre indeks på operatoren, mens en positiv øvre indeks tilsvarer en overkrysning og en negativ øvre indeks en underkrysning (se figur C.7). Øvre indeks tar kun verdiene 1 eller

[5] Med lik strøm gjennom begge spoleparene som produserer magnetfeltet (se figur 2.4) får feltet en anisotropi lik 0.91.

[6] Se tillegg C.

[7] To tråder kan naturligvis ikke flettes, bare spinnes rundt hverandre.



**Figur 4.5:** Eksempel på en skematisk tegning av en matematisk flette.

-1. Fletten på figur 4.5 kan således beskrives ved operator-rekken, kalt flette-ordet, $\sigma_1^1\sigma_2^{-1}\sigma_3^1\sigma_1^1\sigma_2^{-1}\sigma_1^1$. (Operatorene $\sigma_i^j$ må ikke forveksles med dipolmomentet $\vec{\sigma}$ fra kapittel 1.)

## 4.4 Resultater med 7 kuler

Med litt kunnskap om flette-teori er det nå mulig å diskutere hvordan bevegelsesmønsteret til kulene skal analyseres.

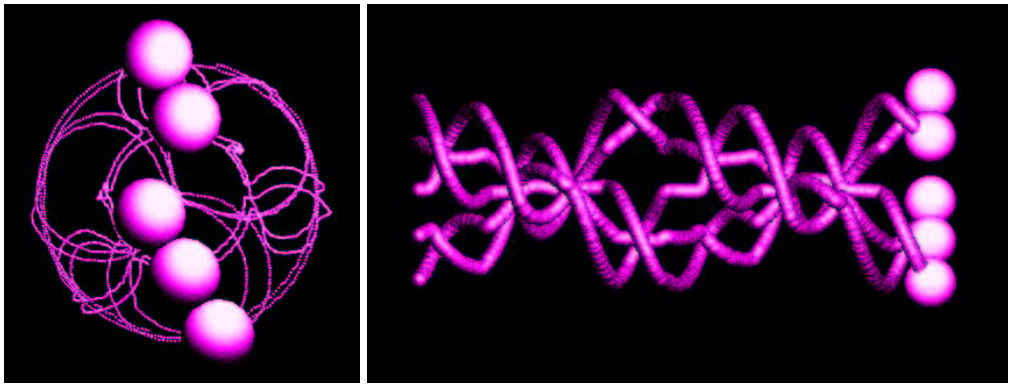
### 4.4.1 Fra kuler til fletter

Sett ovenifra, det vil si gjennom mikroskopet eller vinkelrett på bevegelsesplanet, ser bevegelses-trajektorene til ugelstadkulene i det roterende magnetfeltet ut som på figur 4.6.a. Hvis det hele sees fra siden (det vil si i bevegelsesplanet), og bevegelsen strekkes ut i tid ser bevegelsene ut til å danne en flette som på se figur 4.6.b. Dette er utgangspunktet for å bruke flette-teori i analysen av kulenes bevegelse. Flette-kodingen av bevegelsene gir ikke like mye informasjon om kulenes bevegelse som de fullstendige tid-rom-koordinatene, men forhåpentligvis nok til å vise essensen i bevegelsesmønsteret.

### 4.4.2 Vridningen, Wr(t)

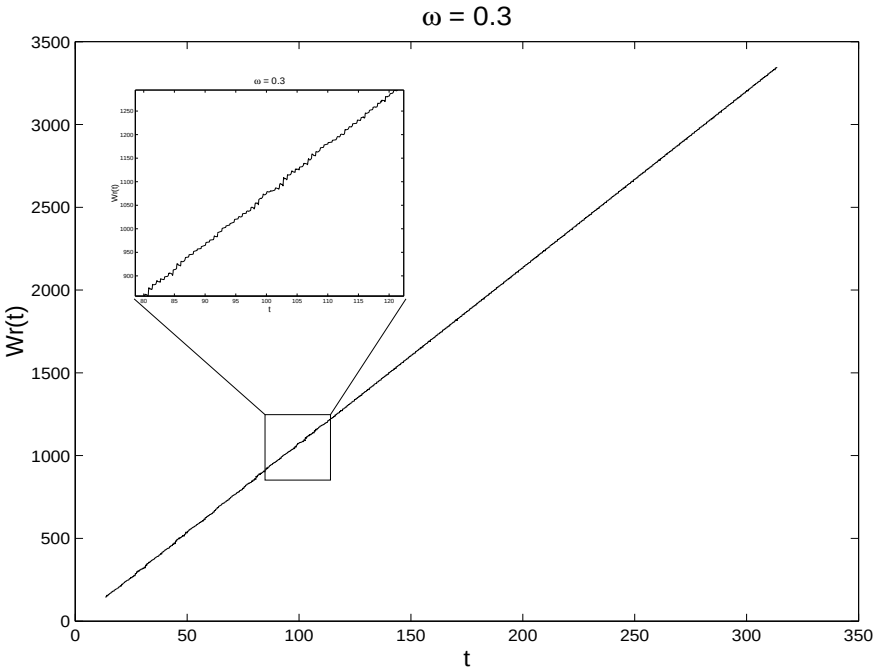
En topologisk invariant[8] for fletter er vridningen til fletten. Denne beregnes ved å summere de øvre indeksene i operatorene for fletten slik at fletten på figur 4.5 blir 1-1+1+1-1+1=2. Vridningen til en flette er et mål på hvor mye fletten er vridd eller viklet. En overkrysning (en økning i vridningen) er definert som en krysning i samme retning som magnetfeltet. Derfor vil vridningen stort sett øke med tiden. Underkrysninger skjer ved faseslipp og når to grupper av kuler krysser hverandre. Figur 4.7 viser et typisk plot av vridningen til et system som fuksjon av tiden. Som man ser på figuren er vridningen i det store og det hele lineær som funksjon av tiden, men på en mindre skala er det variasjoner i stigningen.

[8] Se kapittel C

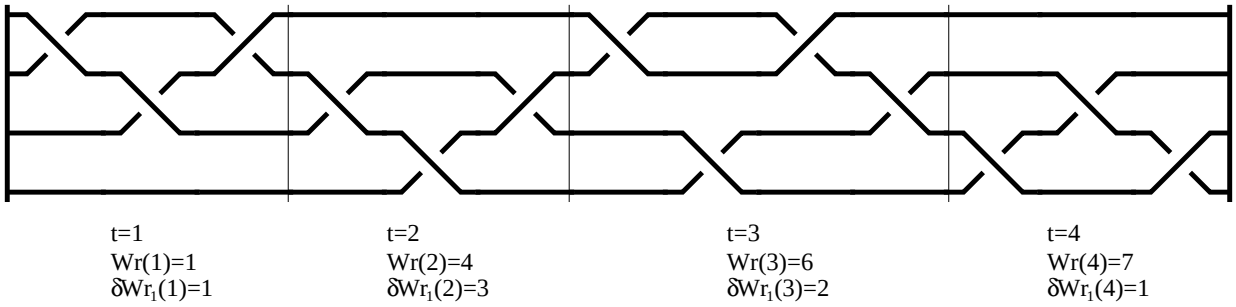


(a) (b)

**Figur 4.6:** Bevegelsestrajektorene til fem kuler. (a) Sett vinkelrett på bevegelsesplanet. (b) Sett i bevegelsesplanet med tiden langs horisontal-aksen.




**Figur 4.7:** Et typisk plot av vridningen til et system med syv kuler. Tiden t er i halve perioder av det roterende magnetfeltet. Sirkulært felt med relativ frekvens 0.3.




**Figur 4.8:** Tidsinndeling av en flette med vridningen, $Wr(t)$ og vridningshastigheten, $\delta Wr_1(t)$.

### 4.4.3 Vridningshastigheten, $\delta$Wr$_i$(t)

Vridningshastigheten er et mål på hvor raskt vridningen øker. Den deriverte til vridningen ville likevel ikke vært et riktig mål på hastigheten. Vridningen endres når to eller flere kuler endrer innbyrdes posisjon på $y$-aksen. Derfor er endringen i vridningen diskret og vridningshastigheten må beregnes utifra et visst antall hele tidssteg. Antallet tidssteg, $i$, det er midlet over vises i indeksen til vridningshastigheten, $\delta$Wr$_i$(t).

$$\delta Wr_i(t) = Wr(t) - Wr(t-i). \tag{4.8}$$

### 4.4.4 Tre modeller

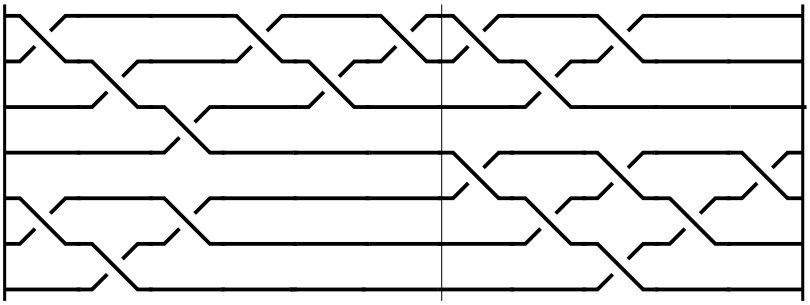
I de følgende simuleringene er tre ulike modeller benyttet:

- **Uten speildipoler** - Simuleringer hvor grensebetingelsene som fører til speilbilder av dipolene ikke er tatt med i beregningen.
- **Med speildipoler** - Simuleringer med speildipoler i første tilnærming, det vil si for $i = \pm 1$. $\kappa$ og plateseparasjon, $D$, er de samme som i forsøkene.
- **Med speildipoler og støy** - Som simuleringer med speildipoler, men i tillegg er det ved hver iterasjon lagt til en kraft på hver partikkel i en tilfeldig retning (hvit støy) med en amplitude på 0.1% av den radielle kraften mellom to kuler som ligger parallellt med feltretningen ved feltets maksimale verdi.

### 4.4.5 Regulære bevegelsesmønster

Ved lave nok rotasjonshastigheter bryter ikke kjeden av kuler opp. Mønsteret blir helt regulært og med syv kuler blir[9] $\delta Wr(t) = 21$. For noe høyere hastigheter vil det dannes kjeder på 3 og 4

[9]Det er ingen indeks på denne vridningshastigheten da den er helt regulær så det vil ikke være noen forskjell på hastigheten midlet over flere perioder. Generelt er vridningshastigheten for $n$ kuler som ikke bryter opp $(\delta Wr(t))_n = (n-1) + (n-2) + \cdots + 1 = n(n-1)/2$.



**Figur 4.9:** Fletten til en gruppe på 4 fire kuler som gir en kule til en gruppe på tre.

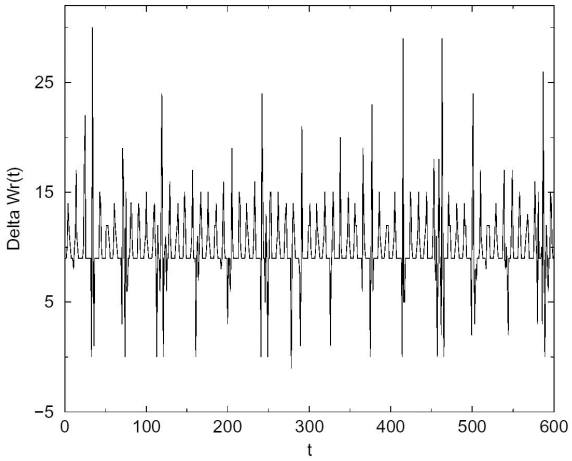
kuler, der kjeden med 4 kuler ofte vil gi en kule til kjeden med 3 kuler. Figur 4.9 viser flettemønsteret for overgangen, og figur 4.10 viser $\delta Wr_1(t)$ for simuleringer og eksperiment med $\omega = 0.20$ og $\varepsilon = 1$.

De største toppene og bunnene i eksperimentell vridningshastighet kan skyldes problemet med nøyaktig å bestemme eksperimentell frekvens. Vridningen er som beskrevet i kapittel 4.4.3 diskret. Hvis perioden for analysene avviker noe fra den eksperimentelle perioden kan et tidsvindu mellom to perioder enten dekke over to krysningspunkt (for stort tidsvindu i analysen) eller havne midt mellom to krysninger (for lite tidsvindu i analysen). Hvis man ser bortifra disse ekstreme verdiene viser den eksperimentelle grafen et relativt regulært mønster.

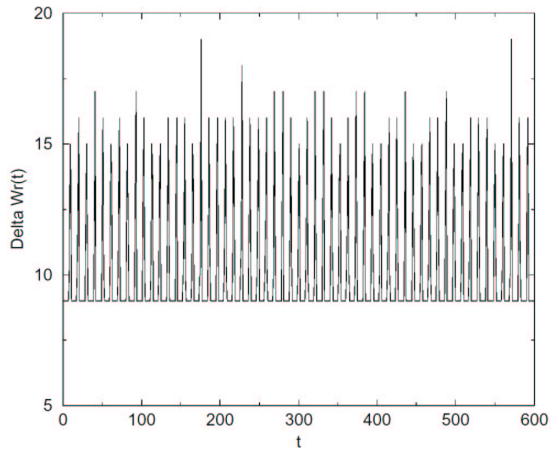
Det er tydelig at speildipolene (figur (d) og (e)) påvirker mønsteret til kulene, selv om simuleringer med disse også gir et regulært mønster. Eksperimentet på figur 4.10 er hentet fra Kai deLange Kristiansens hovedoppgave [10]. Med grupper på 3 og 4 kuler gir en halv omdreining en økning i vridningen på 9, som også er typetallet for alle grafene på figur 4.10. Høyere vridningshastigheter oppstår når to grupper krysser hverandre. Sammenligner man (b) og (c) er disse omtrent like regulære. Toppene i vridningshastigheten er noe høyere i (c), men det blir kompensert for med en lavere vridningshastighet i neste tidssteg. Støy (e) har ingen synlig virkning på vridningshastigheten.

### 4.4.6 Komplekse mønster

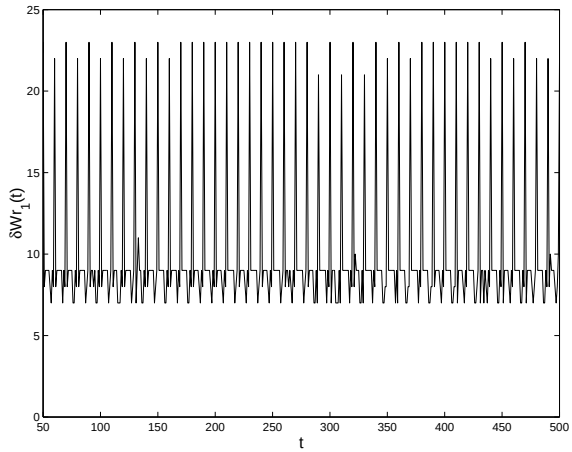
Ved å innføre et anisotropt felt med en faktor $\varepsilon = 0.8$ og øke vinkelhastigheten på feltet til $\omega = 0.4$ oppstår mer ustabile grupper av kuler. Figur 4.11 viser at innføring av speilbilder av dipolene (b og c) er det lille nødvendigefor å skape kompleksitet i bevegelsen til kulene. Uten speildipoler (a) faller kulenes bevegelse inn i et regulært mønster, noe som ikke ser ut til å skje med ved bruk av speildipoler. Simuleringer med opp til $4000\tau_{rf}$ viser fortsatt et uregelmessig mønster. For slike felt blir effekten av speildipoler mye mer tydelig enn ved lavere hastigheter (figur 4.10). Figur 4.13 viser at det er stor forskjell i fordelingen av $\delta Wr_1(t)$ med og uten speileffekter. Med anisotropi i feltet har kjedene lett for å ryke når de står parallellt med feltets svakeste retning. Overraskende nok har simuleringene uten speildipoler vridning med høyere gjennomsnittshastighet enn simuleringene med speildipoler (se figur 4.12), selv om den i startfasen begynner med en noe lavere verdi. Grunnen til den noe høyere vridningshastigheten



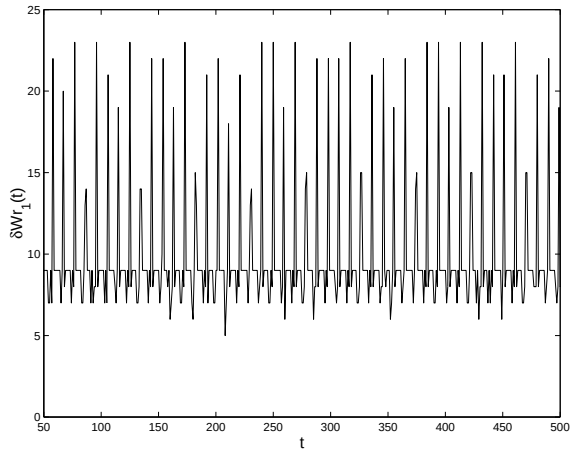

(a)



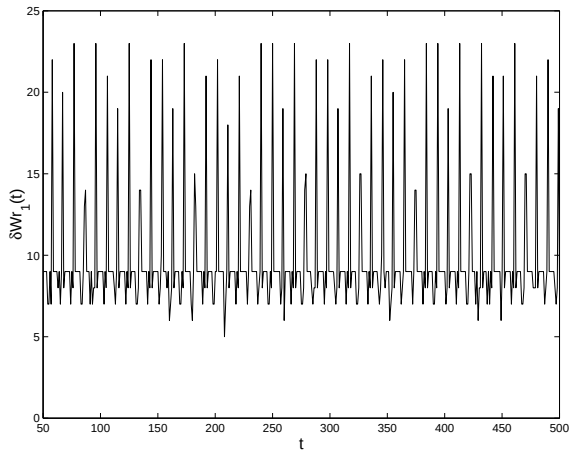

(b)




(c)



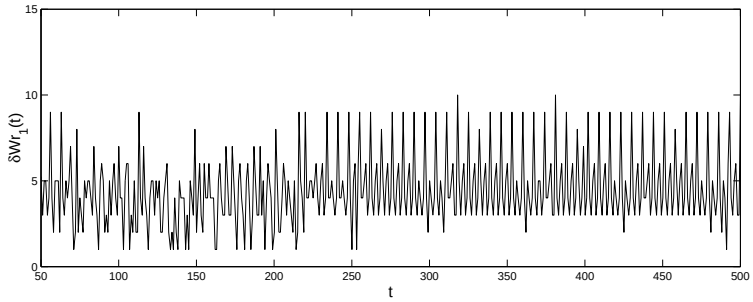

(d)



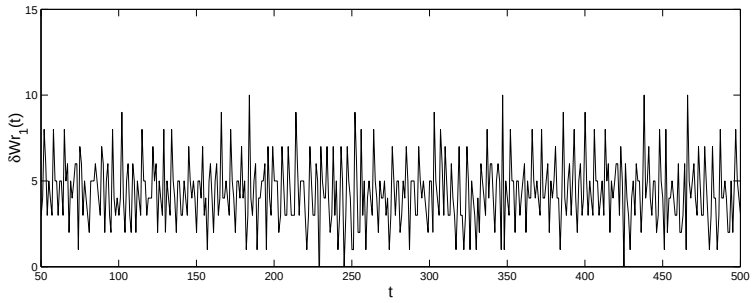

(e)



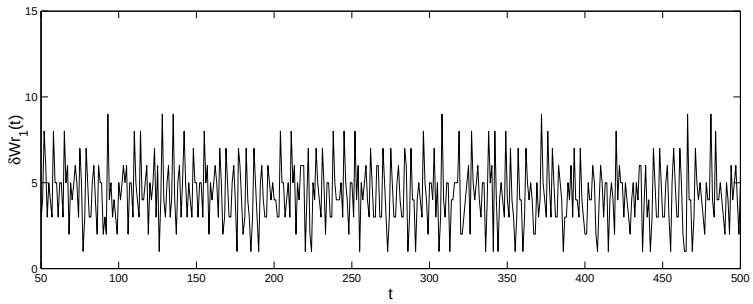
**Figur 4.10:** Vridningshastigheten til syv kuler med $\omega = 0.2$ og $\varepsilon = 1$. (a) Eksperiment, (b) Simulering gjort av Kristiansen[10] (c) Simulering uten speileffekter eller støy, (d) Simulering med speileffekter. (e) Simulering med speileffekter og støy.




(a)

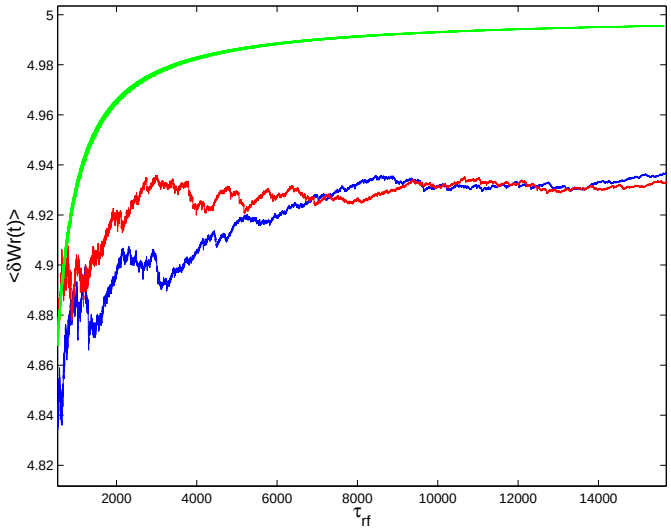



(b)




(c)

**Figur 4.11:** Vridningshastigheten til syv kuler med $\omega = 0.4$ og $\varepsilon = 0.8$. (a) Simulering uten speileffekter eller støy, (b) Simulering med speileffekter. (c) Simulering med speileffekter og støy.



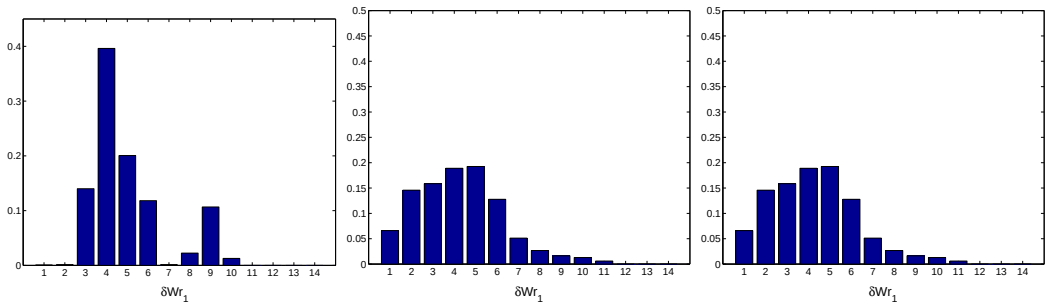

**Figur 4.12:** $< \delta Wr(t) >$ for 7 kuler i et felt med $\omega = 0.4$ og $\varepsilon = 0.8$. Grønn kurve: Uten speildipoler eller støy. Rød kurve: Med speildipoler, uten støy. Blå kurve: Med speildipoler og støy.

ved simuleringer uten bruk av speildipoler kan være at kulene får en større avstand under faseslipp ved bruk av speildipoler, og dermed minker sjansen for å bli med en lengre kjede på en rundtur. Simuleringen med støy (figur 4.11.c) ser ikke ut til å skille seg vesentlig fra den tilsvarende simuleringen uten støy (figur 4.11.b).

### 4.4.7 Kaos

Kuler i et felt med kraftig anisotropi og høy vinkelhastighet utgjør et svært ustabilt system. Tilfeldigheter avgjør bevegelsesmønsteret til kulene. Figur 4.14 viser $\delta Wr_1(t)$ til 7 kuler i et felt med $\omega = 0.401$ og $\varepsilon = 0.681$. Eksperimentet (a) viser store uregelmessigheter. Simuleringen uten speildipoler (b) viser seg å kollapse etter cirka 250 $\tau_{rf}$. Kulene samler seg i en klump som ligger og vugger i takt med feltet mens et kulepar spinner på hjørnet av konfigurasjoen[10]. Kulene kan samle seg på forskjellige måter ved en kollaps, men felles for alle er at de har havnet i en stabil konfigurasjon uten særlig bevegelse. Med speildipoler (c) viser simuleringene et mye mer regulært mønster enn eksperimentene viser. Nærmest eksperimentene ligger simuleringen med speildipoler og støy (d). Støyen er ikke større enn i simuleringen med regulære mønster (figur 4.10). Det er likevel nok til å hindre systemet i å komme inn i periodiske bevegelser.

[10]En video som illustrerer en kollaps ligger på den medfølgende cd'en.




(a) (b) (c)

**Figur 4.13:** Sluttfordelingen av $\delta Wr_1$ for syv kuler med $\omega = 0.4$ og $\varepsilon = 0.8$ etter cirka 15000 perioder. (a) simulering uten speileffekter eller støy, (b) simulering med speileffekter, uten støy, (c) simulering med speileffekter og støy.

## 4.5 Mangepartikkel-system

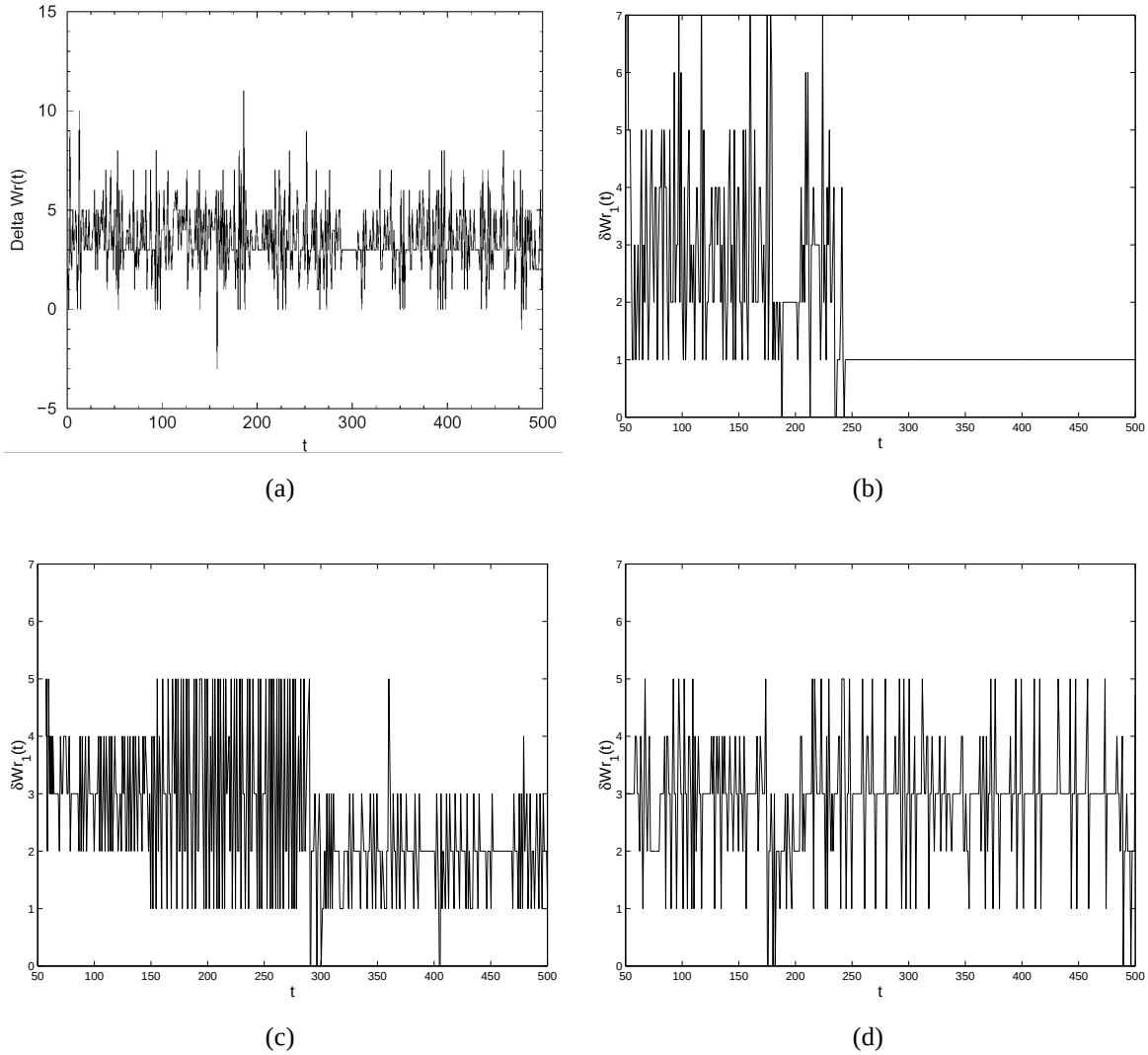
Forsøk gjort på IFE av flere hundre kuler i et konstant, ikke-roterende felt er også simulert. Geir Helgesen og Josef Cernak har arbeidet med dette systemet på lyslaboratoriet på IFE [29]. Kulene danner kjeder som igjen hekter seg sammen med andre kjeder og lager lengre kjeder[11]. Figur 4.16 viser systemet ved 4 forskjelige tider i en simulering. I forsøket er kulene bare en 1/10 av plateseparasjonen og speildipolene har minimal effekt. Figur 4.15 viser gjennomsnittslengden av kjedene $< L(t) >$, gitt som:

$$< L(t) >= N_p/N_k(t), \tag{4.9}$$

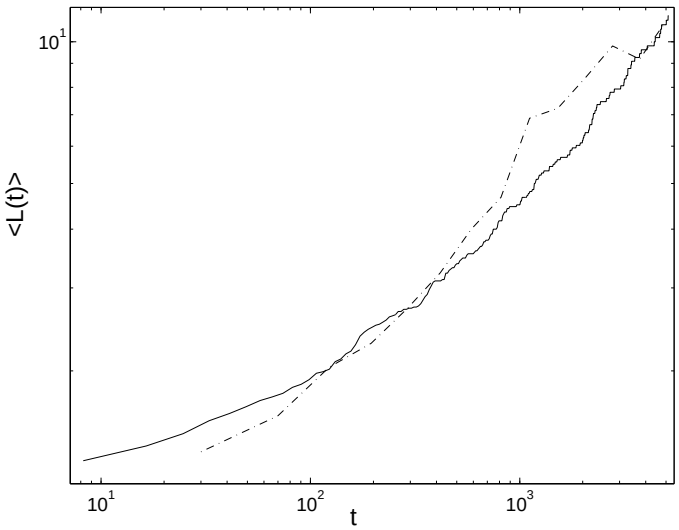
der $N_p$ er antall partikler og $N_k(t)$ er antall kjeder. En tidsenhet i simuleringen er definert som for fastfelt i kapittel 4.1.1.

ProtoMol gjennomførte simuleringen på et knapt døgn med 16 prossessorer på Gridur som står i Trondheim på NTNU og verensstemmelsen med eksperiment er bra. Videre simuleringer og forsøk vil vise bedre om ProtoMol er riktig arbeidsverktøy for denne oppgaven.

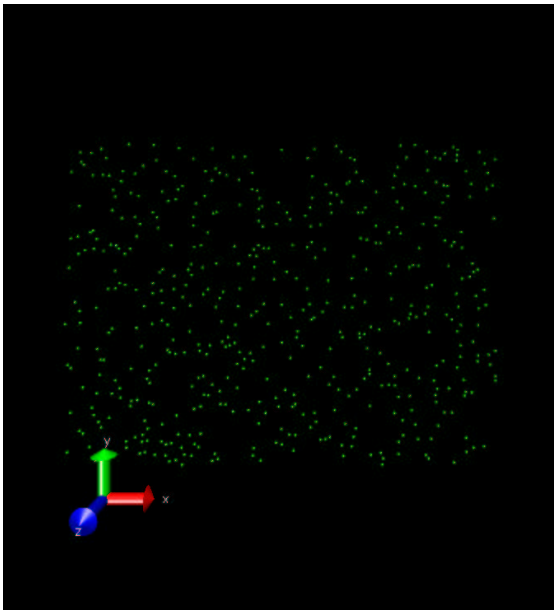
[11] En video av en simulering finnes på cd'en.



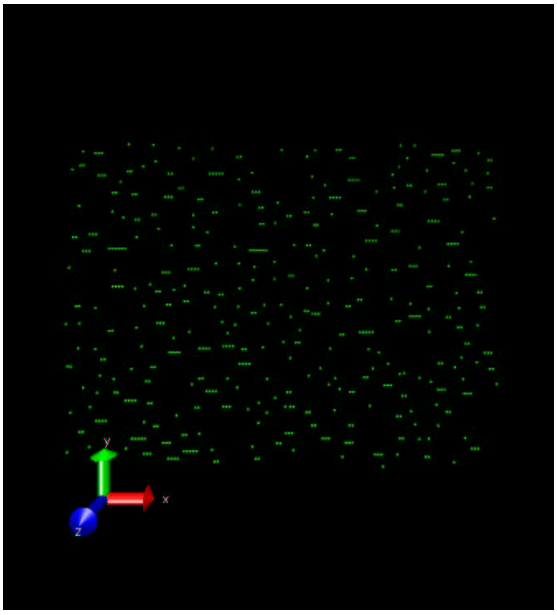

**Figur 4.14:** Vridningshastigheten til syv kuler med $\omega = 0.401$ og $\varepsilon = 0.681$. (a) Eksperiment, (b) Simulering uten speileffekter eller støy, (c) Simulering med speileffekter. (d) Simulering med speileffekter og støy.



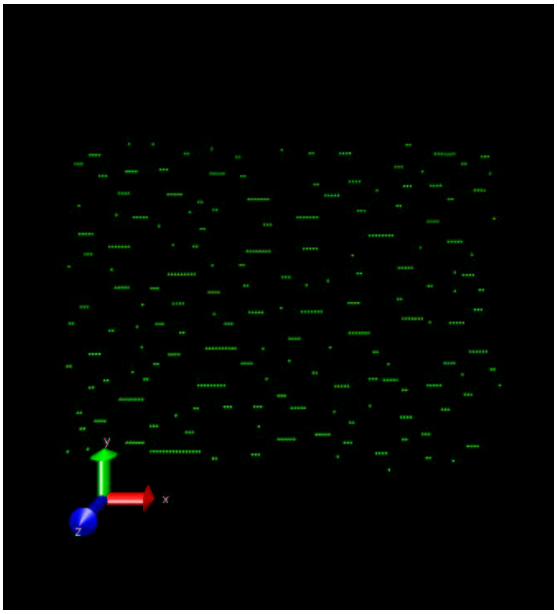

**Figur 4.15:** $< \ L(t) \ >$ for 500 kuler initielt tilfeldig fordelt på et område 135x100 kule-diametre. Stiplet linje: eksperimentelle data. Heltrukket linje: simulering uten bruk av speildipoler. Eksperimentelle data fra et forsøk utført av Geir Helgesen på IFE.



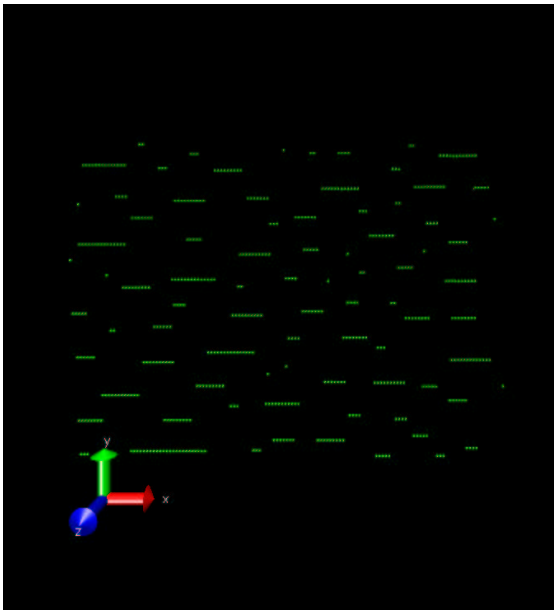

(a)




(b)




(c)




(d)

**Figur 4.16:** Øyeblikksbilder fra simuleringer med 500 kuler i et ikke-roterende magnetisk felt ved tiden: (a) $0\tau_{ff}$ (b) $100\tau_{ff}$ (c) $1000\tau_{ff}$ (d) $5000\tau_{ff}$.

# Kapittel 5

# Konklusjon

Oppgaven har presentert den matematiske modellen brukt i simuleringene og hvordan eksperimentene er utført. En gjennomgang av hvordan ProtoMol er tilpasset modellen er gitt. Videre er resultater med forskjellige modeller og systemer vist og diskutert. Arbeidet med å raffinere modellen for ugelstadkuler har vært vellykket og gitt noen interessante resultater.

For det første har det vist seg at grensebetingelsene med glassplatene som løses ved hjelp av speilbilder av dipolene kan ha vesentlig innvirkning på dynamikken i systemet. Dette gjør seg gjeldende ved større vinkelhastigheter eller med anisotropi i feltet hvor systemet er mer ustabilt. Selv om det er mulig å se forskjeller ved lave vinkelhastigheter også, er forskjellene mellom simuleringer med og uten bruk av speildipoler ikke så dominerende under slike forhold.

For det andre har litt støy gitt et mer komplekst bevegelsesmønster for svært ustabile systemer. Støyen som har vært brukt har likevel vært så svak at den ikke har hatt innvirkning på mer stabile systemer. Støymodellen brukt i denne oppgaven er svært enkel og gir ikke større forståelse for hvor eventuell støy i eksperimentene oppstår, men viser likevel at en slik støy ikke er usannsynlig.

Selv om overensstemmelsen mellom eksperiment og simulering fremdeles ikke er fullstendig er det tydelig at de to faktorene nevnt ovenfor spiller en vesentlig rolle i modellen. Årsaken til avvikene kan være den enkle hydrodynamiske tilnærmingen.

Simuleringene med to kuler under faseslipp stemte heller ikke 100%, men det er gjort for få forsøk å sammenligne med til å trekke noen endelig konklusjon i så hendseende.

Prosjektet med å implementere krefter og dynamiske variable i ProtoMol tilpasset forsøkene med ugelstadkuler i en magnetisk væske var vellykket. ProtoMol har vist seg å være et fleksibelt og pålitelig simuleringsprogram i forbindelse med oppgaven.

Når det gjelder simuleringer av større systemer med flere hundre kuler har disse vært utført i et beskjedent omfang. I skrivende stund er en ny hovedfagsstudent ved UiB, Fadia Jaber, i gang med å se nærmere på slike store systemer. Forhåpentligvis kan ProtoMol også der vise seg å være et godt verktøy for å løse slike oppgaver.

# Tillegg A

# Effektivt dipolmoment

En kule med permeabilitet $\mu_k$ og radius R plasseres i et ferrofluid med permeabilitet $\mu_f$. Et ytre homogent magnetfelt, $\vec{H}_0$, påtrykkes systemet (se figur A.1). Det må nå defineres et potensial for innsiden av kulen og ett for utsiden. I tillegg må grensebetingelsene for magnetfelt være oppfylt:

$$H_{ft} = H_{kt} \quad \text{og} \quad \mu_f H_{fr} = \mu_k H_{kr}, \tag{A.1}$$

der $t$ og $r$ refererer henholdsvis til den tangentielle og den radielle komponenten av H, og $f$ og $k$ refererer henholdsvis til fluid og kule. Potensialet i ferrofluidet hvis kulen ikke hadde vært tilstede er

$$\phi_0 = -\vec{H}_0 \vec{r}. \tag{A.2}$$

Vi antar at potensialene kan skrives på formen:

$$\begin{aligned} \phi_{r>R} &= A_1 r \cos\theta + A_2 r^{-2} \cos\theta \\ \phi_{r<R} &= B_1 r \cos\theta + B_2 r^{-2} \cos\theta, \end{aligned} \tag{A.3}$$

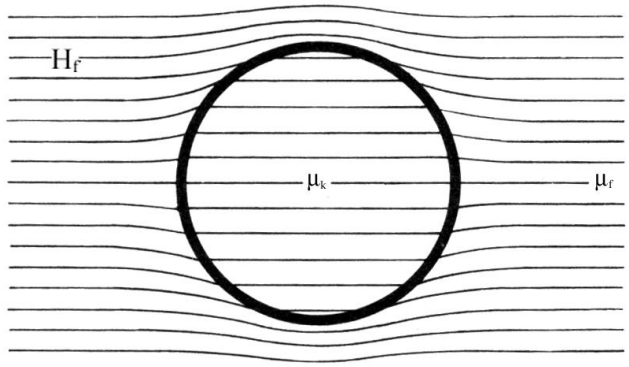
der $A_{1,2}$ og $B_{1,2}$ skal bestemmes. $\phi_{r>R}$ er potensialet utenfor kulen og $\phi_{r<R}$ er potensialet på innsiden av kulen.

Når r$\rightarrow \infty$ må $\phi_{r>R}$ gi samme potensial som om kulen ikke var tilstede, slik at $A_1 = -H_0$. Videre kan vi argumentere for at $B_1 = 0$, da potensialet ikke kan gå mot uendelig når $r \rightarrow 0$. Ligningene A.1 er dermed redusert til:

$$\begin{aligned} \phi_{r>R} &= -E_0 r \cos\theta + A_2 r^{-2} \cos\theta \text{ og} \\ \phi_{r<R} &= B_1 r \cos\theta. \end{aligned} \tag{A.4}$$

Den første grensebetingelsen, at den tangentielle delen av $\vec{H}$ må være kontinuerlig, er ekvivalent med at $\phi_{r>R} = \phi_{r<R}$. Dette gir:

$$B_1 = AR^{-3} - H_0. \tag{A.5}$$




**Figur A.1:** Ikke-magnetisk kule i et ferrofluid med ytre magnetisk felt. (Tegning tatt fra Bleaney & Bleaney (1976) og manipulert)

Normalkomponentene til H i grenseflaten er:

$$H_{kr} = -\left(\frac{\partial \phi_{r<R}}{\partial r}\right)_{r=R} = B_1 \cos\theta \tag{A.6}$$
$$H_{fr} = -\left(\frac{\partial \phi_{r>R}}{\partial r}\right)_{r=R} = H_0 \cos\theta + 2A_2 R^{-3} \cos\theta$$

A.6 innsatt i den andre grensebetingelsen i A.1 gir:

$$-B_1 = (\mu_f/\mu_k)(H_0 + 2A_2 R^{-3}). \tag{A.7}$$

Løsningen av ligning A.5 og A.7 er

$$B_1 = -\frac{3\mu_f}{\mu_k + 2\mu_f} H_0 \qquad og \quad A_2 = \frac{\mu_k - \mu_f}{\mu_k + 2\mu f} R^3 H_0, \tag{A.8}$$

slik at potensialet innenfor og utenfor kulen er gitt ved henholdsvis

$$\phi_k = -\frac{3\mu_f}{mu_k + 2\mu_f} H_0 r \cos\theta \text{ og} \tag{A.9}$$
$$\phi_f = -(1 - \frac{R^3}{r^3}\frac{mu_k - muf}{mu_k + 2\mu_f}) H_0 r \cos\theta. \tag{A.10}$$

Endringen i feltet i ferrofluidet etter at kulen ble plassert der er

$$\Delta\phi_f = \frac{\mu_k - \mu_f}{\mu_k + 2\mu_f}\frac{R^3}{r^3} H_0 r \cos\theta \equiv \frac{\vec{\sigma}\cdot\vec{r}}{4\pi r^3}, \tag{A.11}$$

der

$$\vec{\sigma} = 4\pi \frac{\mu_k - \mu_f}{\mu_k + 2\mu_f} R^3 \vec{H}_0 \tag{A.12}$$

er det effektive dipolmomentet til kulen. Hvis $\mu_k = \mu_0$ og $\mu_f = \mu_0(1 + \chi_f)$, der $\mu_0$ er permeabiliteten i vakuum, blir det effektive dipolmomentet

$$\vec{\sigma} = -\frac{4\pi R^3}{3}\frac{\chi_f}{1 + \frac{2}{3}\chi_f}\vec{H}_0 = -V\chi_{eff}\vec{H}_0, \tag{A.13}$$

der V er volumet til kulen, og $\chi_{eff}$ den effektive susceptibiliteten.

Det må legges til at resultatet av den effektive susceptibiliteten er kritisk avhengig av at dipollegemet er en kule. Dipollegemer med andre enkle geometriske former kan beregnes analytisk, men en numerisk tilnærmelse er i de fleste realistiske sammenhenger nødvendig.

# Tillegg B

# Stokesflyt på en kule

Denne utledningen av den viskøse motstanden på en kule forutsetter en ikke-komprimerbar væskeflyt med svært lavt Reynolds-tall, $Re \ll 1$. Løsningen ble første gang presentert av Stokes i 1851. Utgangspunktet er ligningen for jevn flyt rundt en kule med radius R, plassert i en uniform strøm U (figur B.1),

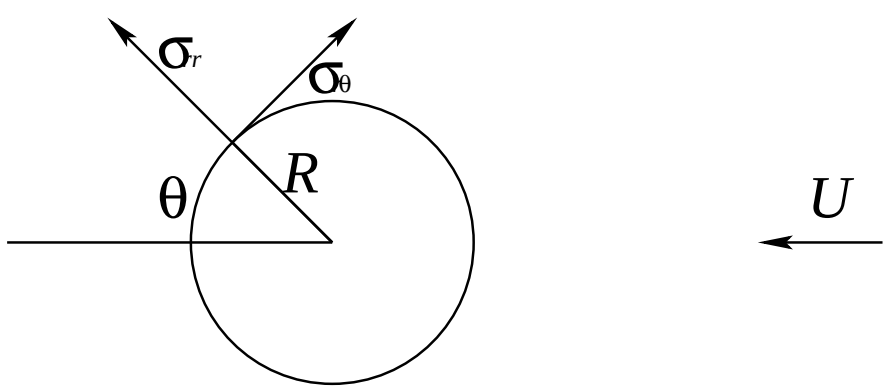
$$\rho \vec{u} \cdot \vec{\nabla} \vec{u} = -\vec{\nabla} p + \mu \nabla^2 \vec{u}, \tag{B.1}$$

der $\rho$ er tettheten til væsken, $\vec{u}$ er den lokale hastigheten til væsken relativt til kulesentrum, $p$ er det lokale trykket og $\mu$ er viskositeten til væsken [16]. Når de viskøse leddene dominerer, det vil si ved lave Reynoldstall, reduseres ligning B.1 til

$$\vec{\nabla} p = \mu \nabla^2 \vec{u}. \tag{B.2}$$

I kulekoordinater gir det

$$\frac{1}{\mu}\frac{\partial p}{\partial r} = \frac{\partial}{\partial r}\left(\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 u_r\right)\right) + \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial u_r}{\partial \theta}\right) - \frac{2}{r^2 \sin\theta}\frac{\partial}{\partial \theta}(u_\theta \sin\theta) \tag{B.3}$$




**Figur B.1:** De viskøse stresskomponentene på kule i en væske med strøm U.

$$\frac{1}{\mu r}\frac{\partial p}{\partial \theta} = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial u_\theta}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \theta}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial \theta}(u_\theta \sin\theta)\right) + \frac{2}{r^2}\frac{\partial u_r}{\partial \theta}. \tag{B.4}$$

Ved å ta virvlingen på ligning B.2 fås

$$0 = \nabla^2 \vec{\nabla} \times \vec{u} = \nabla^2 \vec{\omega}, \tag{B.5}$$

siden virvlingen til en gradient er null, og $\nabla^2$ kommuterer med $\vec{\nabla}\times$.

Systemet er aksesymmetrisk, slik at den eneste komponenten av $\vec{\omega}$ er $\omega_\phi$ som er vinkelrett på papir-planet i figur B.1. $\omega_\phi$ er gitt som

$$\omega_\phi = \frac{1}{r}\left[\frac{\partial r u_\theta}{\partial r} - \frac{\partial u_r}{\partial \theta}\right] \tag{B.6}$$

For en aksesymmetrisk flyt kan det defineres en strømfunksjon $\psi$ som i kulekoordinater er gitt ved [16]:

$$u_r \equiv \frac{1}{r^2 \sin\theta}\frac{\partial \psi}{\partial \theta} \quad \text{og} \quad u_\theta \equiv -\frac{1}{r\sin\theta}\frac{\partial \psi}{\partial r} \tag{B.7}$$

B.7 innsatt i B.6 og til slutt løst med hensyn på ligning B.5 gir

$$\left[\frac{\partial^2}{\partial r^2} + \frac{\sin\theta}{r^2}\frac{\partial}{\partial \theta}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial \theta}\right)\right]^2 \psi = 0. \tag{B.8}$$

Randbetingelsene for ligningen er

$$\psi(R,\theta) = 0 \quad (u_r = 0 \text{ på overflaten}) \tag{B.9}$$

$$\partial\psi/\partial r(R,\theta) = 0 \quad (u_\theta = 0 \text{ påoverflaten}) \tag{B.10}$$

$$\lim_{r\to\infty} \psi(r,\theta) = U r^2 \sin^2\theta \quad (\text{uniform flyt når } r \to \infty) \tag{B.11}$$

Strømningsfunksjonen for en uniform flyt er gitt som høyre side av B.11 [16]. Det er nå fornuftig å anta en separabel bølgefunksjon på formen

$$\psi = f(r)\sin^2\theta, \tag{B.12}$$

som innsatt i B.8 gir

$$f^{iv} - \frac{4f''}{r^2} + \frac{8f'}{r^3} - \frac{8f}{r^4} = 0. \tag{B.13}$$

Den generelle løsningen av B.13 er

$$f = Ar^4 + Br^2 + Cr + D/r. \tag{B.14}$$

B.11 krever at $A = 0$ og at $B = 1/2U$, mens grensebetingelsene ved kuleoverflaten gir $C = -3UR/4$ og $D = UR^3/4$. Løsningen av $\psi$ med randbetingelsene er altså

$$\psi = Ur^2 \sin 2\theta \left[\frac{1}{2} - \frac{3R}{4r} + \frac{R^3}{4r^3}\right]. \tag{B.15}$$

Hastighetskomponentene blir

$$u_r = U\cos\theta\left(1 - \frac{3R}{2r} + \frac{R^3}{2r^3}\right) \tag{B.16}$$

$$u_\theta = -U\sin\theta\left(1 - \frac{3R}{4r} - \frac{R^3}{4r^3}\right). \tag{B.17}$$

Ved å sette inn ligning B.3 og B.4 i ligning B.2 og benytte uttrykkene for $u_r$ og $u_\theta$ (B.7) fås

$$\nabla p = \frac{\partial p}{\partial r}dr + \frac{\partial p}{\partial \theta}d\theta = -\mu\frac{3}{2}UR\left(\frac{2\cos\theta dr}{r^3} + \frac{\sin\theta d\theta}{r^2}\right) \tag{B.18}$$

Integrasjon av B.18 gir

$$p(r) = -\frac{3R\mu U\cos\theta}{2r^2}, \tag{B.19}$$

der $\lim_{r\to\infty} p(r) \equiv 0$.

Kraft per enhetsareal normalt på kuleoverflaten er

$$F_i = \tau_{ij}n_j = (-p\delta_{ij} + \sigma_{ij})n_j = -pn_i + \sigma_{ij}n_j, \tag{B.20}$$

der $\tau_{ij}$ er den totale stress-tensoren og $\sigma_{ij}$ er den viskøse stresstensoren. Motstandskraften i strømningsretningen, $U$, er derfor

$$(-p\cos\theta + \sigma_{rr}\cos\theta - \sigma_{r\theta}\sin\theta)_{r=R} = \frac{3\mu U}{2R}, \tag{B.21}$$

der de viskøse stress komponentene er gitt ved

$$\sigma_{rr} = 2\mu\frac{\partial u_r}{\partial r} = 2\mu U\cos\theta\left[\frac{3R}{2r^2} - \frac{3R^3}{2r^4}\right] \tag{B.22}$$

$$\sigma_{r\theta} = \mu\left[r\frac{\partial}{\partial r}\left(\frac{u_\theta}{r}\right) + \frac{1}{r}\frac{\partial u_r}{\partial \theta}\right] = -\frac{3\mu UR^3}{2r^4}\sin\theta. \tag{B.23}$$

Ligning B.21 er altså vinkeluavhengig og det gjenstår bare å multiplisere med overflatearealet til kulen, $4\pi R^2$, for å finne den totale viskøse motstandskraften

$$F_{viskøs} = 6\pi\mu RU, \tag{B.24}$$

som altså har samme retning som strømmen $U$. Ligning B.24 kalles også Stokes motstandslov.

# Tillegg C

# Knute- og flette-teori

Knuter og fletter har vært brukt av mennesket til alle tider. De har vært viktig i konstruksjonen av redskaper av forskjellige slag. Incaindianerne brukte til og med knuter som bokstaver [20]. Til tross for knutenes lange historie er de matematiske knutene relativt unge.

## C.1 Historie

Knuteteori er en del av den matematiske grenen topologi og oppstod i det 19. århundre. Det første forsøket på å benytte knute-teori til å forklare den fysiske verden ble fremført av Lord Kelvin (William Thompson, 1824-1907) på midten av 1800-tallet som foreslo at atomer egentlig var knuter i eteren. Peter Guthrie Tait (1831-1901) fulgte opp ideen og laget den første systematiske katalogiseringen av knuter. Teorien døde hen, men kanskje var de bare 150 år før sin tid? J.C. Maxwell (1831-1879) undersøkte også knuter, men for ham representerte knutene lukkede strømsløyfer som genererte magnetfelt.

C. F. Gauss var den første fysikeren som introduserte flette-teori til en fysisk problem. I *Zur Electrodynamik* fra 1877 skriver han[1]:

> Fra *Geometria Situs*[2], som Leibnitz ante og som kun et par geometrikere (Euler og Vandermonde) fikk muligheten til å få et glimt av, er vi klar over at vi etter hundre og femti år ikke har oppnådd særlig mer enn ingenting.
>
> En hovedoppgave i *grenseland* mellom *Geometria Situs* og *Geometria Magnitudinis* blir å telle antallet omslyngninger mellom to lukkede eller uendelige linjer.

Etter dette gikk knuteteorien inn i en dvaletilstand som skulle vare helt til 1985, da V. F. R. Jones (1952- ) introduserte en ny polynomisk invariant[3] til knuter, i dag kjent som Jones-polynomet [30]. I dag anvendes knuteteori i blant annet biologi for å forstå

[1] Oversettelsen er gjort etter beste evne fra et sitat i I.M. James' 'History of Topology' [19].

[2] Ordet topologi dukker ikke opp i litteratur før 1920. Istedenfor er ordene *Geometria Situs* (situasjons-geometri) og *analysis situs* (situasjons-vitenskap) brukt

[3] Hvis to knuter er like har de samme invariant. Relasjonen holder generelt ikke andre veien, så en invariant kan ikke brukes til åbestemme om to knuter er like, men kan avgjøre om to knuter er ulike.

**Figur C.1:** 0-knuten.

DNA-molekylets funksjon som ikke er uavhengig av hvordan det er krøllet sammen. Knuteteori har også funnet anvendelse i kjemi og statistisk fysikk, og det er verdt å nevne at Jones' invariant hadde sitt utspring i statistisk fysikk. I denne oppgaven er fletteteori brukt for å analysere dynamikken til ugelstadkuler i en magnetisk væske.

## C.2 Topologi

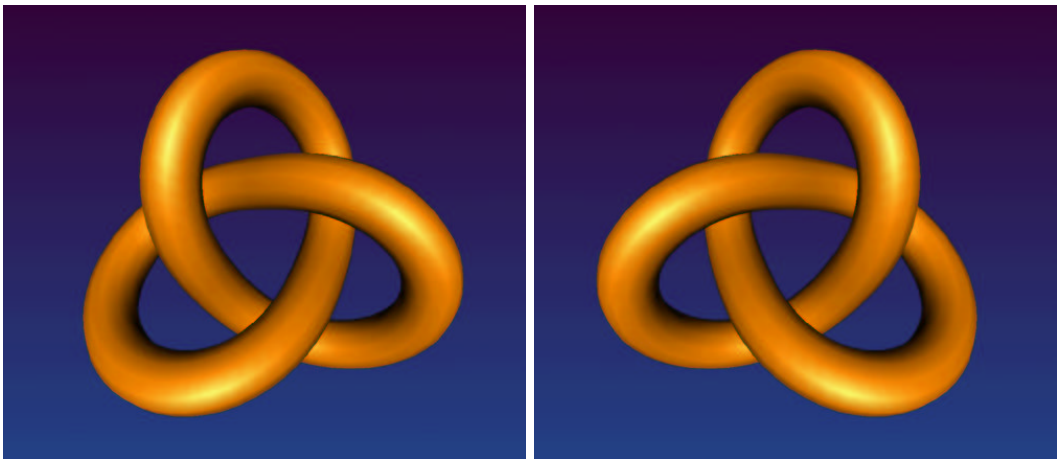
Knute- og flette-teori hører inn under feltet topologi. Selv om topologiens opprinnelse kan spores tilbake flere hundre år, var det Poincaré, rundt forrige århundreskifte, som fikk satt topologi på matematikernes dagsorden. I dag dekker topologi et viktig felt innen matematikk. I.M. James har redigert *History of Topology* [19] som gir en grundig innføring i topologiens historie med bidrag fra mange anerkjente matematikere.

## C.3 Knuter

Den materielle analogien til en matematisk knute er en tråd, viklet og knyttet sammen, med endene på tråden limt sammen. Den enkleste knuten, 0-knuten, er en sirkel (figur C.1), mens den enkleste ikke-trivielle knuten kalles en tresløyfeknute[4]. Tresløyfeknuten er en vanlig kjerringknute med endene limt sammen og kommer i en høyrehendt og en venstrehendt versjon (figur C.2) [5].

[4]Tresløyfeknute er fritt oversatt fra det engelske ordet trefoil-knot.

[5]En høyrehendt person som knyter en tresløyfeknute vil får knuten til høyre på figur C.2 som resultat, mens en venstrehendt vil ende opp med knuten til venstre.



**Figur C.2:** De to tresløyfeknutene. Den venstrehendte til venstre og den høyrehendte til høyre.

Den venstrehendte og den høyrehendte tresløyfeknuten er ikke ekvivalente. For at to knuter skal være topologisk ekvivalente må det være mulig å transformere den ene knuten til den andre (og vice versa) ved hjelp av de fire såkalte reidermeister-grepene[6] (figur C.3). Hvis knuten hadde vært fysisk ville reidermeister-grepene bety at knuten kan strekkes og dras i så mye en ønsker, men den kan ikke kuttes. Dessuten må den ikke roteres i andre retninger enn i planet den betraktes.

Det er ikke mulig med disse grepene å komme fra den vestrehendte til den høyrehendte tresløyfeknuten og derfor er de heller ikke ekvivalente. Det er heller ikke nødvendigvis slik at en tilsynelatende komplisert knute har en annen topologisk verdi enn en enklere variant. Knuten på figur C.4 er i virkeligheten en 0-knute[7].

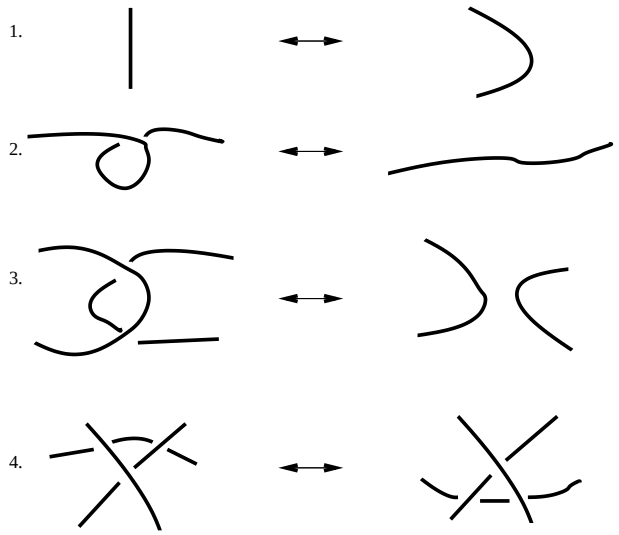
Problemet er at det er et uendelig antall måter å kombinere reidermeister-grepene på. Så generelt sett er det ikke mulig å være sikker på at to knuter ikke er ekvivalente. Dette er hovedproblemet i knute-teorien.

Knuter katalogiseres etter hvor mange krysninger av tråden de enklest mulige versjonene av knutene har. Tresløyfeknuten har 3 krysninger. Det er bare tresløyfeknuten som har tre krysninger. Likeledes kun én knute med fire krysninger. Men så stiger antallet raskt. To forskjellige knuter har fem krysninger, tre med seks, syv med syv, 21 med åtte, og med 13 krysninger finnes det 9988 forskjellige. Det finnes opplagt et ubegrenset antall forskjellige knuter, og i dag er det beregnet 1.701.935 forskjellige knuter med 16 eller færre krysninger.

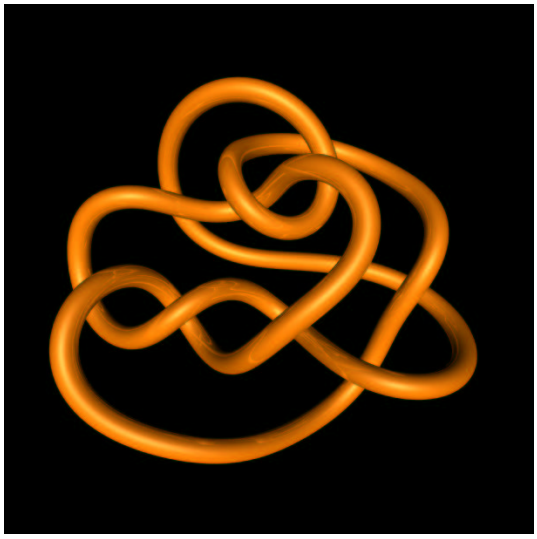
To eller flere knuter som er vevd inn i hverandre kalles en lenke. På denne måten kan knute-kartet utvides og på figur C.5 er en oversikt over noen av de enkleste knutene og lenkene.

[6] Kurt Werner Friedrick Reidermeister (1893-1971) beviste i 1920-årene at to ekvivalente knuter kunne transformeres til hverandre ved hjelp av disse grepene.

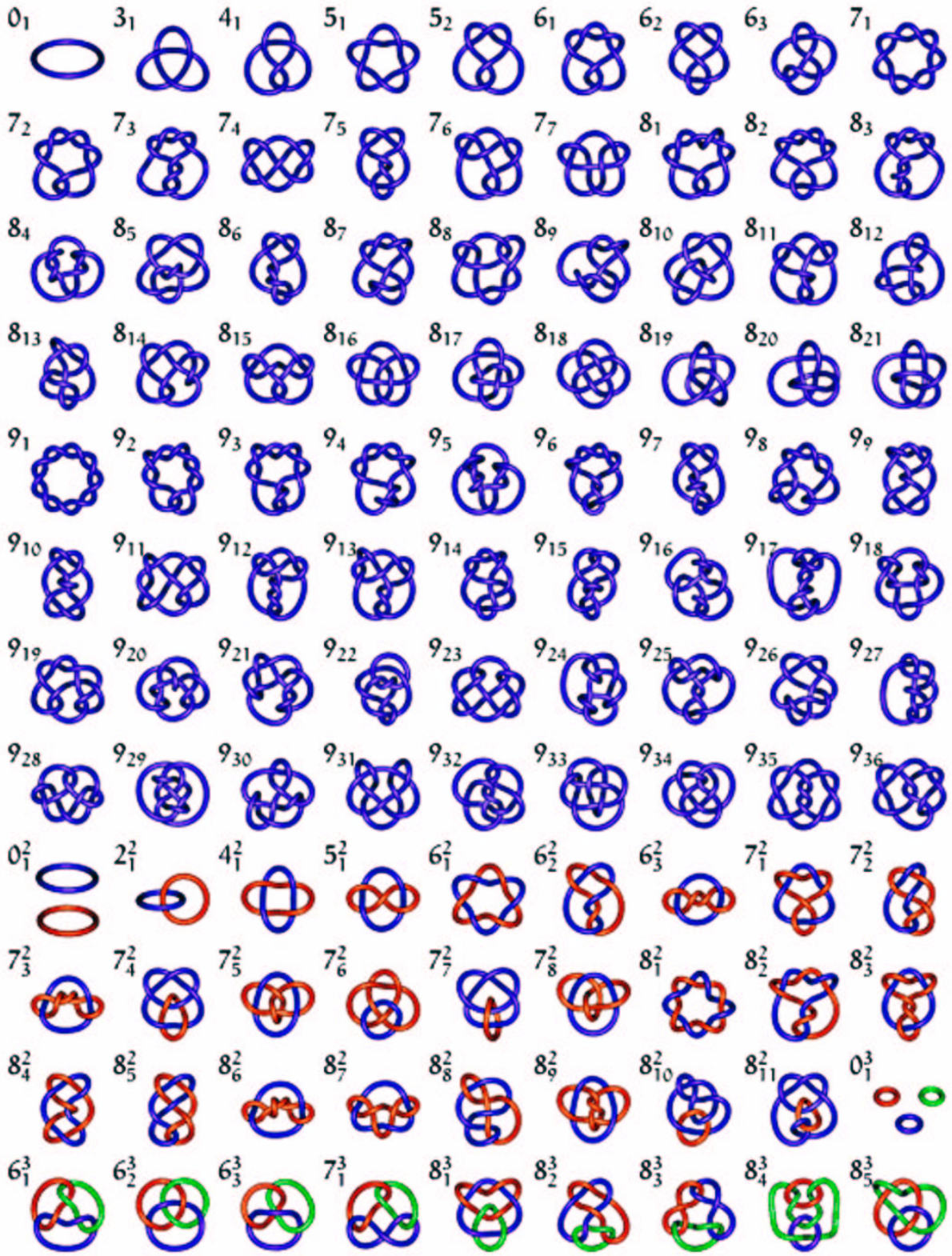
[7] På den medfølgende cd'en finnes en video som viser at knuten faktisk er en 0-knute.



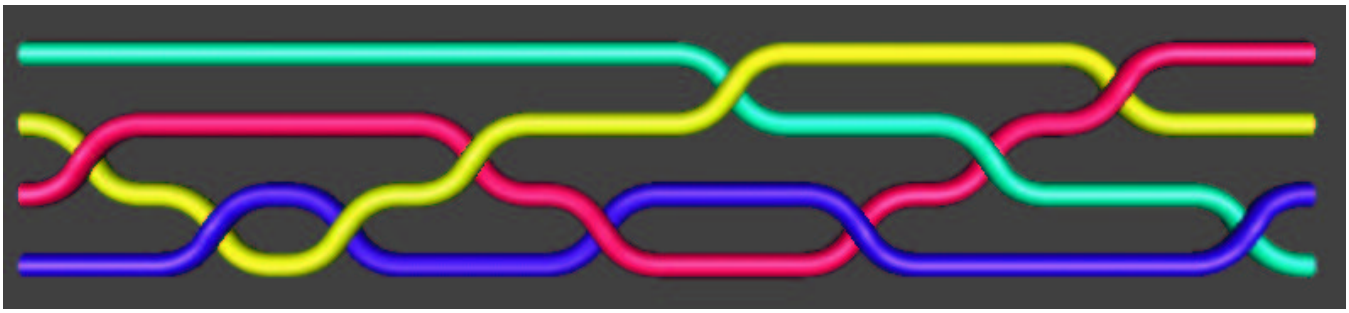

**Figur C.3:** De fire reidermeister-grepene. Nullte reidermeister-grep forteller at tråden ikke er stiv, men kan bøyes og strekkes. Første grep gjør det mulig å fjerne en krøll på tråden. Grep nummer to gir anledning til å separere to tråder som overlapper hverandre, mens tredje grep forteller hvordan det er mulig å flytte en tråd fra ene siden av et krysningspunkt til den andre.



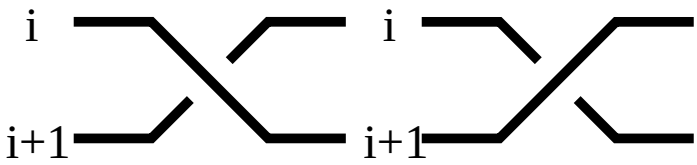
**Figur C.4:** Denne knuten kan ved hjelp av de fire reidermeister-grepene gjøres om til en 0-knute.




**Figur C.5:** En tabell over noen knuter og lenker. Hovedtallet tilsvarer antall krysninger i knuten eller lenken. Øvre indeks forteller hvor mange tråder linkene har, og det siste tallet gir hver unike knute eller lenke et unikt tall. Denne tabellen skiller ikke mellom høyre- og venstre-hendte knuter.



**Figur C.6:** Eksempel på en flette med 4 tråder.




**Figur C.7:** Til venstre en $\sigma_i$: Tråd i krysser over tråd i+1. Til høyre en $\sigma_i^{-1}$: Tråd i krysser under tråd i+1.

## C.4 Fletter

En matematisk flette har mange likhetstrekk med de matematiske knutene, men i motsetning til knutene er trådene i de matematiske flettene ikke limt sammen, men festet til hvert sitt plan i hver sin ende. Figur C.6 viser et eksempel på en flette med fire tråder.

Flette-teorien definerer en operator som tilsvarer at to tråder krysser hverandre. Hvis den i'te tråden krysser over den (i+1)'te tråden betegnes dette med $\sigma_i$. Hvis den krysser under er betegnelsen $\sigma_i^{-1}$ (se figur C.7). Øvre indeks til en fletteoperator betegner hvilken type krysning det er, og nedre indeks hvilken tråder som krysser hverandre.

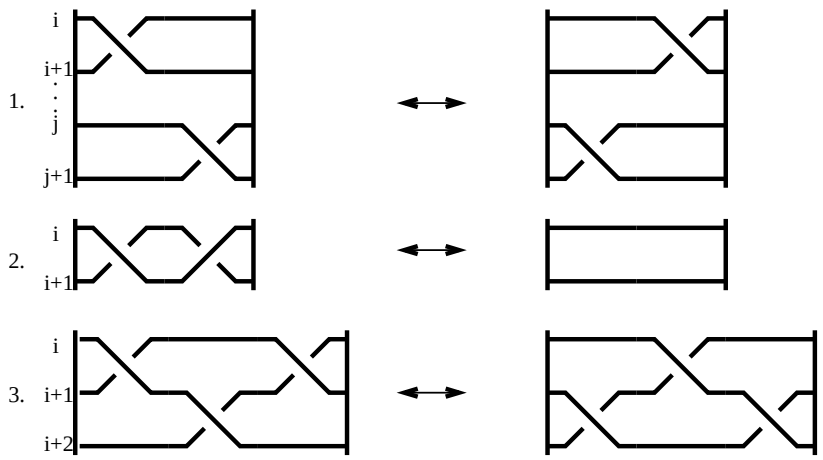
Ved å gjøre dette ved hver krysning kan fletten skrives som en sekvens med operatorer. Fletten på figur C.6 har flettekode $\sigma_2^{-1}\sigma_3^{-1}\sigma_3^{-1}\sigma_2^{-1}\sigma_3^{1}\sigma_1^{-1}\sigma_3^{1}\sigma_2^{1}\sigma_1^{-1}\sigma_3^{-1}$.

Reidermeister-grepene har sine analoger i flette-teori og her gir de også i praksis bare begrensninger på kutting av trådene (figur C.8). Trådene kan eller ikke løses fra veggen de er festet i.

En topologisk invariant er vridningen[8] til en flette. Vridningen er summen av de øvre indeksene til flette-operatorene. En flette, $f$, beskrevet med operatorene $\sigma_{i_1}^{a_1}\ \sigma_{i_2}^{a_2}\ \sigma_{i_3}^{a_3}\cdots\sigma_{i_k}^{a_k}$, har en vridning lik

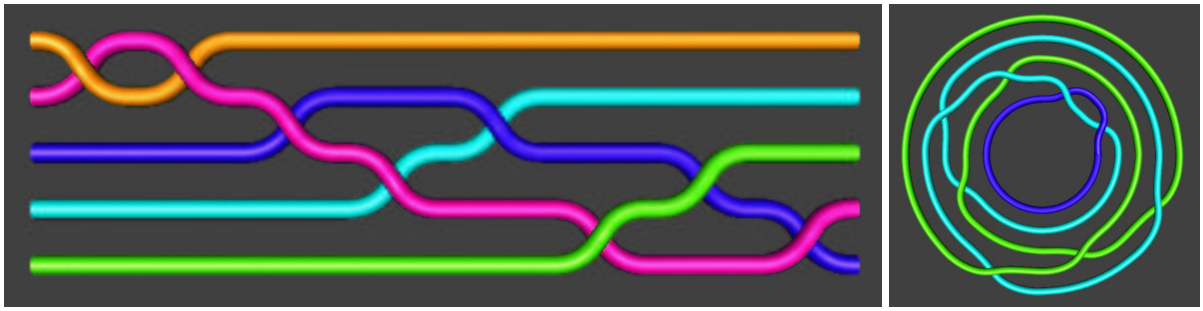
$$Wr(f) = \sum_{j=0}^{k} a_j. \tag{C.1}$$

[8]Det er lite norsk litteratur om knute-teori, og ordet vridning er min oversettelse av writhe som i 'The Oxford Pocket Dictionary' [28] beskrives slik: **writhe** - *twist or roll oneself about (as) in acute pain;*, altså vri seg i smerte.




**Figur C.8:** De tre reidermeister-grepene for fletter.

Ved å lime endene på fletten sammen, den øverste på høyresiden med den øverste på venstresiden og så videre nedover, får man en knute eller en lenke. Figur C.9 viser en flette og dens respektive lenke[9]



**Figur C.9:** Fletten til venstre lukkes og resultatet er lenken til høyre.

Flere andre egenskaper ved fletter og knuter kan utledes, men berøres ikke i denne oppgaven. Kunio Murasugis bok *Knot Theory & It's Applications* [20] og J.S. Birmans *Braids, Links and Mapping Class Group*[27] gir en god innføring i knute og flette-teori.

[9] .

# Tillegg D

# Programmer

I dette kapitlet listes noen av programmene eller deler av dem som er brukt i forbindelse med denne oppgaven. Alle programmene som er listet i dette kapitlet er laget i løpet av hovedfagsarbeidet og i tilknytning til det. En kort forklaring til programmene vil bli gitt. Programmene listes som de er og følgelig vil ofte kommentarer i programmene være på engelsk.

## D.1 Krefter i ProtoMol

I ProtoMol defineres kreftene i .C-filer (c++ kilde-filer), men siden ProtoMol baseres mye på templates er det nødvendig (og mer effektivt) å definere noen av de ikke-statiske operasjonene i .h-filer (c++ header-filer). Det varierer med de forskjellige kreftene om det er .C-filen eller .h-filen som inneholder selve beregningen av kraften. Oppbygningen av ProtoMol er komplisert, men mye er forklart i Thierry Mattheys doktorgradsavhandling [24].

### D.1.1 MagneticDipole

Den første kraften som ble implementert i ProtoMol var MagneticDipole. .C-filen definerer navnet på kraften, men .h-filen inneholder beregningene. Kraften ble først implementert uten speileffekter som ble lagt til senere.

**MagneticDipoleForce.h**

34-38: Definerer størrelser som ikke behøver å beregnes for hver gang kraften kalles.
54-55 : Henter tid og invers avstand mellom kuleparet som beregnes.
61 : H-felt beregnes.
66-68 : Kraft og energi mellom paret beregnes uten speileffekter.
71-105 : Kraft og energi pga speileffekter til første tilnærming legges til hvis ikke speileffekter er slått av.
107-108: Kraft for å hindre kulene i å overlappe legges til.
110 : Størrelsen på kraften beregnes.
111 : Avstandsvektoren normeres og legges i samme retning som kraften. Dette gjøres fordi kraften returneres til integratoren som en størrelse, mens avstanden returneres som en vektor. Integratoren antar at kraften er radiell.

```
1   /* -*- c++ -*- */
    #ifndef MAGNETICDIPOLEFORCE_H
    #define MAGNETICDIPOLEFORCE_H

5   #include "GenericTopology.h"
    #include "EnergyStructure.h"
    #include "ExclusionTable.h"
    #include "mathutilities.h"
    //#include "Report.h"
10  #include "ParameterType.h"
    #include<string>
    using std::string;
    //using namespace Report;

15  //_____________________________________________________________ MagneticDipoleForce
    // This force calculates the force between two dipoles created by a magnetic
    // field. It returns the absolute value of the force and changes the
    // distance-vector (diff) since the force is usually not radial. It includes the
    // mirror effect (1. aprox, i.e. only one mirror-image on each side of the
20  // boundary. It also assumes that the dipoles lie in the middle between the two
    // boundaries.
    // http://www.ife.no/media/504_Thesis.pdf
    // http://www.ife.no/tillegg/index.jsp?projectId=3047&tilleggId=1035
    //
25
    class MagneticDipoleForce {
      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      // Constructors, destructors, assignment
      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
30  public:
      MagneticDipoleForce():myChi(0.0),myR(0.0),myOmega(0.0),myPhi(0.0),myHx(0.0),myHy(0.0),myHz(0.0),myD(0.0){}
      MagneticDipoleForce(Real chi, Real radius, Real omega, Real phi, Real Hx,Real Hy, Real Hz, Real D):
        myChi(chi),myR(radius),myOmega(omega),myPhi(phi),myHx(Hx),myHy(Hy),myHz(Hz),myD(D){
        // Shortcuts
35      volum     = 4.0/3.0*M_PI*power<3>(myR);
        expfactor = -1000.0*myChi*myChi*myR*myHx*myHx;
        realChi   = myChi/(1-2.0/3.0*myChi);
        kappa     = realChi/(realChi+2.0);
      }
40
      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      // New methods of class MagneticDipoleForce
      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    public:
```

```
void operator()(Real &energy,
                Real &force,
                Real rDistSquared,
                Coordinates &diff,
                const GenericTopology* topo,
                int atom1, int atom2,
                ExclusionClass excl) const {

  //defining variables used to calculate the force
  Real t = topo->time;
  Real inv_r = sqrt(rDistSquared);
  Real r = 1.0/inv_r;
  Real inv_r2 = rDistSquared;
  Real inv_r3 = inv_r2*inv_r;
  Real inv_r5 = inv_r3*inv_r2;
  Real inv_r7 = inv_r5*inv_r2;
  Coordinates H(myHx*cos(t*myOmega+myPhi),myHy*sin(t*myOmega+myPhi),myHz/(1.0+realChi));
  Coordinates sigma(-H*volum*myChi);
  Real sigmaDotSigma = sigma.dot(sigma);
  Real sigmaDotDiff = sigma.dot(diff);
  //Dipole-dipole effects:
  Coordinates F(diff*sigmaDotSigma*3.0*inv_r5
                - diff*power<2>(sigmaDotDiff)*15.0*inv_r7
                + sigma*6.0*sigmaDotDiff*inv_r5);
  energy = sigmaDotSigma*inv_r3-3*sigmaDotDiff*sigmaDotDiff*inv_r5;

  //Mirror-effect 1st approx (myD=0 turns mirroreffect off)
  if (myD != 0.0){
    // creating mirror-dipoles
    Coordinates sigma_m(sigma.x,sigma.y,-sigma.z);
    sigma_m *= kappa;
    Coordinates diff_m_lower(diff.x,diff.y, myD);
    Coordinates diff_m_upper(diff.x,diff.y,-myD);

    // mirror-self-coupling
    // Not possible without z-coordinates
    // Done in MagneticDipoleMirrorSystemForce*.[Ch]

    // mirror-dipole-coupling (assumes that the dipoles are all sentered at z=0)
    Real inv_r_m2 = 1.0/diff_m_upper.normSquared();
    Real inv_r_m3 = inv_r_m2*sqrt(inv_r_m2);
    Real inv_r_m5 = inv_r_m2*inv_r_m3;
    Real inv_r_m7 = inv_r_m5*inv_r_m2;

    // now the dipole - mirror-force is added to F
    Real sigma_mDotsigma = sigma_m.dot(sigma);
    Real sigma_mDotdiff_m = sigma_m.dot(diff_m_upper);
    Real sigmaDotdiff_m = sigma.dot(diff_m_upper);
    F += diff_m_upper*(sigma_mDotsigma*3.0*inv_r_m5
                       - sigma_mDotdiff_m*sigmaDotdiff_m*15.0*inv_r_m7)
      + sigma_m*sigmaDotdiff_m*6.0*inv_r_m5;
    energy += sigma_mDotsigma*inv_r_m3 - 3.0*sigma_mDotdiff_m*sigmaDotdiff_m*inv_r_m5;

    sigma_mDotdiff_m = sigma_m.dot(diff_m_lower);
    sigmaDotdiff_m = sigma.dot(diff_m_lower);
    F += diff_m_lower*(sigma_mDotsigma*3.0*inv_r_m5
                       - sigma_mDotdiff_m*sigmaDotdiff_m*15.0*inv_r_m7)
      + sigma_m*sigmaDotdiff_m*6.0*inv_r_m5;
    energy += sigma_mDotsigma*inv_r_m3 - 3.0*sigma_mDotdiff_m*sigmaDotdiff_m*inv_r_m5;

  }
  //Peak-potetial at the edge of the spheres:
  Real f = exp(expfactor*(r-2*myR))*inv_r;
  F += diff*f;
```

```
    force = F.norm();
    diff = F/force;
  }

  static void accumulateEnergy(EnergyStructure* energies, Real energy) {
    energies->otherEnergy += energy;
  }
  static Real getEnergy(const EnergyStructure* energies) {return energies->otherEnergy;}

  // Parsing
  static string getKeyword() {return keyword;}
  static string getIdString() {return keyword;}
  void getParameters(vector<ParameterType>& parameters) const{
    parameters.push_back(ParameterType("-chi",VarValType::REAL,VarVal(myChi)));
    parameters.push_back(ParameterType("-r",VarValType::REAL,VarVal(myR)));
    parameters.push_back(ParameterType("-omega",VarValType::REAL,VarVal(myOmega)));
    parameters.push_back(ParameterType("-phi",VarValType::REAL,VarVal(myPhi)));
    parameters.push_back(ParameterType("-H",VarValType::REAL,VarVal(myHx)));
    parameters.push_back(ParameterType("",VarValType::REAL,VarVal(myHy)));
    parameters.push_back(ParameterType("",VarValType::REAL,VarVal(myHz)));
    parameters.push_back(ParameterType("-d",VarValType::REAL,VarVal(myD)));

  }
  static unsigned int getParametersCount() {return 8;}
  static MagneticDipoleForce doMake(string& errMsg, const vector<VarVal>& values, int n) {
    Real chi   = values[n  ].getReal();
    Real radius= values[n+1].getReal();
    Real omega = values[n+2].getReal();
    Real phi   = values[n+3].getReal();
    Real Hx    = values[n+4].getReal();
    Real Hy    = values[n+5].getReal();
    Real Hz    = values[n+6].getReal();
    Real D     = values[n+7].getReal();
    return MagneticDipoleForce(chi,radius,omega,phi,Hx,Hy,Hz,D);
  }

  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  // My data members
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
public:
  static const string keyword;
private:
  Real myChi, myR, myOmega, myPhi, myHx, myHy, myHz, myD;
private:
  //Short cuts
  Real volum, expfactor, realChi, kappa;
};

//_____________________________________________________________________ INLINES

#endif /* MAGNETICDIPOLEFORCE_H */
```

## MagneticDipoleForce.C

5: Definerer navnet på kraften.

```
#include "MagneticDipoleForce.h"

//_____________________________________________________________ MagneticDipoleForce

const string MagneticDipoleForce::keyword("MagneticDipole");
```

### D.1.2 Friction

Friction er et annet type kraftobjekt enn MagneticDipole. Den virker på enkeltpartikler uten vekselvirkning med andre partikler. Beregningen av kraften er gjort i .C-filen.

**FrictionExtendedForce.C**
85-90: Løkke som går gjennom alle partiklene
86: Beregner friksjonskraften. OBS! (*velocities)[i] er 48.88 ganger større enn hastigheten til partikkel i.
87-89: Legger til støy i alle romlige retninger.

```
1  /*
    * GNU compiler has a pretty bad handling of templates...
    */
   #if defined( TEMPLATE_IS_INCLUDE_FROM_H) || !defined(TEMPLATE_IN_HEADER)
5  // Only compile this file if either not GNU compiler running,
   // or, in case of GNU compiler, files is included from header.


   #ifdef TEMPLATE_IN_HEADER
10 # define INLINE inline
   #else
   # include "FrictionExtendedForce.h"
   # define INLINE
   #endif  // TEMPLATE_IN_HEADER
15
   #include "Topology.h"
   //_____________________________________________________________ FrictionExtendedForce


20 template<class TBoundaryConditions>
   FrictionExtendedForce<TBoundaryConditions>::FrictionExtendedForce():myF(0.0), myRnd(0.0){}

   template<class TBoundaryConditions>
   FrictionExtendedForce<TBoundaryConditions>::FrictionExtendedForce(Real f, Real random):myF(f), myRnd(random){}
25
   template<class TBoundaryConditions>
   INLINE void FrictionExtendedForce<TBoundaryConditions>::getParameters(vector<ParameterType>& parameters) const
     parameters.push_back(ParameterType("-k",VarValType::REAL,VarVal(myF)));
     parameters.push_back(ParameterType("-rnd",VarValType::REAL,VarVal(myRnd)));
30 }

   template<class TBoundaryConditions>
   INLINE Force* FrictionExtendedForce<TBoundaryConditions>::doMake(string&, const vector<VarVal>& values) const {
     Real f = values[0].getReal();
35   Real random = values[1].getReal();
     return new FrictionExtendedForce(f,random);
   }
   template<class TBoundaryConditions>
   INLINE void FrictionExtendedForce<TBoundaryConditions>::evaluate(const GenericTopology* topo,
40                                                                  const CoordinateBlock* positions,
                                                                    const CoordinateBlock* velocities,
                                                                    CoordinateBlock* forces,
                                                                    EnergyStructure* energies){
     doEvaluate(topo,positions,velocities,forces,energies,0,positions->size());
45 }

   template<class TBoundaryConditions>
   INLINE void FrictionExtendedForce<TBoundaryConditions>::parallelEvaluate(const GenericTopology* topo,
```

```
                                                                    const CoordinateBlock* positions,
                                                                    const CoordinateBlock* velocities,
                                                                    CoordinateBlock* forces,
                                                                    EnergyStructure* energies){
  unsigned int n = positions->size();
  unsigned int num = (unsigned int)topo->parallel->getAvailableNum();
  unsigned int count = 0;
  if(num >= n)
    count = n;
  else if(num*2 <= n)
    count = 2*num;
  else
    count = num;

  for(unsigned int i = 0;i<count;i++){
    if(topo->parallel->next()){
      int to = (n*(i+1))/count;
      if(to > (int)n)
        to = n;
      int from = (n*i)/count;
      doEvaluate(topo,positions,velocities,forces,energies,from,to);
    }
  }
}

template<class TBoundaryConditions>
INLINE void FrictionExtendedForce<TBoundaryConditions>::doEvaluate(const GenericTopology* topo,
                                                                   const CoordinateBlock* positions,
                                                                   const CoordinateBlock* velocities,
                                                                   CoordinateBlock* forces,
                                                                   EnergyStructure* energies,
                                                                   int from, int to){
  //const TBoundaryConditions &boundary =
  //  (dynamic_cast<const SemiGenericTopology<TBoundaryConditions>& >(*topo)).boundaryConditions;
  //Real t = topo->time;
  //report << debug << "Time ="<<t <<"[fs]"<< endr;
  for(int i=from;i<to;i++){
    (*forces)[i] += (*velocities)[i]*myF;
    (*forces)[i].x += (rnd_generator()-.5)*myRnd;
    (*forces)[i].y += (rnd_generator()-.5)*myRnd;
    (*forces)[i].z += (rnd_generator()-.5)*myRnd;
  }
}

template<class TBoundaryConditions>
INLINE void FrictionExtendedForce<TBoundaryConditions>::numberOfBlocks(const GenericTopology* topo,
                                                                       const CoordinateBlock* pos,
                                                                       unsigned int& n,unsigned  int& step){
  unsigned int count = pos->size();
  unsigned int num = (unsigned  int)topo->parallel->getAvailableNum();
  if(num >= count)
    n = count;
  else if(num*2 <= count)
    n = 2*num;
  else
    n = num;
  step = 1;
}

#undef INLINE
#endif // defined( TEMPLATE_IS_INCLUDE_FROM_H) || !defined(TEMPLATE_IN_HEADER)
```

### D.1.3 MagneticDipoleMirror

MagneticDipoleMirror virker som Friction kun på enkeltpartikler Denne krafttypen inneholder koordinater til filene og det er mulig å beregne kraften mellom en dipol og dets egne speilbilder. Denne kraften ble lagt til da det ikke var mulig å få et sentrerende potensial i MagneticDipole.

**MagneticDipoleMirrorSystemForce.C**

27-33: Beregner noen nødvendige størrelser fra input-parametrene.
64: e er energien ved dipol-selv-speil-vekselvirkning.
65-71: Løkke som går gjennom alle partiklene.
66-67: Avstand til første speilbilde henholdsvis oppe og nede.
68: Kraften på kulen som følge speilbildene.
69: Energien fra speil-selvspeil-vekselvirkningen økes.
72: Energien legges til den globale energien.

```
1  /*
    * GNU compiler has a pretty bad handling of templates...
    */
   #if defined( TEMPLATE_IS_INCLUDE_FROM_H) || !defined(TEMPLATE_IN_HEADER)
5  // Only compile this file if either not GNU compiler running,
   // or, in case of GNU compiler, files are included from header.


   #ifdef TEMPLATE_IN_HEADER
10 # define INLINE inline
   #else
   # include "MagneticDipoleMirrorSystemForce.h"
   # define INLINE
   #endif  // TEMPLATE_IN_HEADER
15
   //_____________________________________________________________ MagneticDipoleMirrorSystemForce
   // This force calculates the mirror - source effect, i.e. the main-effect that positions the spheres
   // at z = 0

20
   template<class TBoundaryConditions>
   MagneticDipoleMirrorSystemForce<TBoundaryConditions>::MagneticDipoleMirrorSystemForce():myChi(0.0), myR(0.0), m

   template<class TBoundaryConditions>
25 MagneticDipoleMirrorSystemForce<TBoundaryConditions>::MagneticDipoleMirrorSystemForce(Real chi, Real r, Real H>
     //Short cuts
     Real realChi = myChi/(1.0-2.0/3.0*myChi);
     Real volume = 4.0/3.0*M_PI*power<3>(myR);
     Real kappa = realChi/(realChi+2.0);
30   Coordinates H(myHx, myHy, myHz/(1+realChi));
     Coordinates sigma = -H*volume*myChi;
     Coordinates sigma_m = sigma*kappa; sigma_m.z = -sigma_m.z;
     myF = sigma.dot(sigma_m)*(3.0);
   }
35
   template<class TBoundaryConditions>
   INLINE void MagneticDipoleMirrorSystemForce<TBoundaryConditions>::getParameters(vector<ParameterType>& paramete
     parameters.push_back(ParameterType("-chi",VarValType::REAL,VarVal(myChi)));
     parameters.push_back(ParameterType("-r",VarValType::REAL,VarVal(myR)));
40   parameters.push_back(ParameterType("-H",VarValType::REAL,VarVal(myHx)));
     parameters.push_back(ParameterType("",VarValType::REAL,VarVal(myHy)));
     parameters.push_back(ParameterType("",VarValType::REAL,VarVal(myHz)));
     parameters.push_back(ParameterType("-d",VarValType::REAL,VarVal(myD)));
```

```
}

template<class TBoundaryConditions>
INLINE Force* MagneticDipoleMirrorSystemForce<TBoundaryConditions>::doMake(string&, const vector<VarVal>& value
  Real chi = values[0].getReal();
  Real r = values[1].getReal();
  Real Hx = values[2].getReal();
  Real Hy = values[3].getReal();
  Real Hz = values[4].getReal();
  Real d = values[5].getReal();

  return new MagneticDipoleMirrorSystemForce(chi,r,Hx,Hy,Hz,d);
}

template<class TBoundaryConditions>
INLINE void MagneticDipoleMirrorSystemForce<TBoundaryConditions>::evaluate(const GenericTopology* topo,
                                      const CoordinateBlock* positions,
                                      CoordinateBlock* forces,
                                      EnergyStructure* energies){
  Real e = 0.0;
  for(unsigned int i=0;i<topo->atoms.size();i++){
    Real z_upper = myD-(*positions)[i].z*2;
    Real z_lower = myD+(*positions)[i].z*2;
    (*forces)[i].z += myF*(1.0/power<4>(z_lower)-1.0/power<4>(z_upper));
    e += myF*(1.0/power<3>(z_lower) - 1.0/power<3>(z_upper));
  // Denne energien fr du pushe tilbake ett eller annet sted hvis den skal vre med  bestemme totalenergien.
  }
  energies->otherEnergy += e;
}
#undef INLINE
#endif // defined( TEMPLATE_IS_INCLUDE_FROM_H) || !defined(TEMPLATE_IN_HEADER)
```

## D.2 Flette

Flette er et program skrevet i C++ som finner flette-koden til partikler. Programmet består av

program.C: inneholder hovedrutinen og deklarasjoner

fil_rutiner.C: leser koordinater og konfigurasjon og skriver filer

fletting.C: beregner fletten til partiklene. Flette-rutinen bygger på yo.c i Kristiansens hovedoppgave [10]

statistikk.C: beregner statistiske størrelser av fletten

og noen haeder-filer

Konfigurasjon.h: Deklarasjon av konfigurasjonen

Partikkel.h: Deklarasjon av en partikkel

Partikkelblokk.h: Deklarasjon av vektor som inneholder alle partiklene i et tidssteg

Krysning.h: Deklarasjon av en krysning som inneholder bl.a. operator og tidspunkt

Flette.h: Deklarasjon av flette-vektoren, en vektor av krysninger

Statistikk.h: Deklarasjon av flere statistiske vektorer

rutiner.h: Deklarasjon av alle rutiner utenfor program.C

Flette tar et argument som er navnet på konfigurasjonsfilen.

Typisk konfigurasjon

```
antall partikler = 7
koordinatfil = 7s.out.pos.xyz
datafiltype = xyz
statistikkfil = stat.dat
heltidsfil = helstat.dat
flettefil = flette.dat
sannsynlighetsfil = p.dat
3dsannsynlighetsfil = p3.dat
innledning = 10
frekvens = 0.0079577
delta_t = 10
```

Kommentarene i filene forklarer det meste.

## program.C

```
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "Partikkel.h"
#include "Partikkelblokk.h"
#include "Krysning.h"
#include "Flette.h"
#include "Konfigurasjon.h"
#include "Statistikk.h"
#include "rutiner.h"

void main(int argc, char **argv)
  {
    Flette flette;
    Konfigurasjon konfig;
    Partikkelblokk pblokk_for, pblokk_etter; //partikler dette og forrige tidssteg.
    Krysning kryss;
    int i=0, steg = 0;

    // Statistiske funksjoner:
    Statistikk stat;

                // Leser inn konfigurasjon
    les_konfig(argv[1], konfig);

    // pner inn og utfil
    ifstream innfil(konfig.innfil);
    if(!innfil) {
      cout << "Kan ikke pne " << konfig.innfil << " for input\n";
      exit(-1); }
    else
      cout << "pner '" << konfig.innfil << "' som inputfil\n";

    pblokk_for.part = les_koord(innfil, konfig);  //Frste gang m to tidssteg
    pblokk_for.part = les_koord(innfil, konfig);  //leses inn
    pblokk_for.perm = finn_permutasjon(pblokk_for);  //permutasjonen til frste tidssteg


    ///////////////
    // HOVEDLOOP //
    ///////////////
    while (innfil)
      {
        steg ++;
        pblokk_etter.part = les_koord(innfil, konfig);  //leser inn nye partikler
        pblokk_etter.perm = finn_permutasjon(pblokk_etter);  //finner permutasjonen

                                // Bryter ut av lkken hvis det er slutt p filen:
        if (!innfil) { cout << "FRDI!! ( lese...)\n"; break;}

                                // Hvis det er endringer i innbyrdes y-posisjon mellom partiklene
                                // beregnes flette operatoren(e) som tilsvarer endringen.
        if (endring(pblokk_for, pblokk_etter, steg))
          {
            kryss.Operator = finn_operator(pblokk_for, pblokk_etter, steg);
            kryss.tid = steg*konfig.frekvens;  // tiden for krysningen beregnes
            flette.krysning.push_back(kryss);  // ny krysning legges til flette-vektoren
          }
        pblokk_for = pblokk_etter;  // ny blir gammel
      }
```

```
    // Utfrer de statistiske beregningene nr koordinatene er ferdig lest og fletten
                // ferdig laget
    stat.writhe = finn_writhe(flette);
    stat.writhe_hastighet = finn_writhe_hastighet(stat.writhe, flette, konfig.delta_t);
    stat.gj_writhe_hastighet = finn_gjennomsnitts_writhe_hastighet(stat.writhe, flette);
    stat.writhe_fluktuasjon = finn_writhe_fluktuasjon(stat.writhe_hastighet, stat.gj_writhe_hastighet);
    stat.p = finn_sannsynlighetsfordeling(flette, stat.writhe, konfig.antpart);
    stat.treD_p = treD_sannsynlighet(flette, stat.writhe, konfig.antpart);

    // Til slutt skrives de statistiske data til diverse filer
    skriv_fil(konfig, flette, stat, konfig.innledning);
    innfil.close;
  }
```

## fil_rutiner.C

```
#include <stdlib.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "Konfigurasjon.h"
#include "Partikkel.h"
#include "Partikkelblokk.h"
#include "Krysning.h"
#include "Flette.h"
#include "Statistikk.h"

//////////////////////////////////
// Leser inn konfigurasjonsfilen//
//////////////////////////////////
void les_konfig(char *filnavn, Konfigurasjon &konfig)
  {
    ifstream innfil(filnavn);
    if(!innfil)
      {
        cout << "Kan ikke pne " << filnavn << ".\n";
        exit(-1);
      }
    else
      cout << "pner '" << filnavn << "' som konfigurasjonsfil.\n";
    string filtypetekst;
    konfig.filtype = 0;
    innfil.ignore(30, '='); innfil >> konfig.antpart;
    innfil.ignore(30, '='); innfil >> konfig.innfil;
    innfil.ignore(30, '='); innfil >> filtypetekst;
    innfil.ignore(30, '='); innfil >> konfig.statfil;
    innfil.ignore(30, '='); innfil >> konfig.helstatfil;
    innfil.ignore(30, '='); innfil >> konfig.flettefil;
    innfil.ignore(30, '='); innfil >> konfig.pfil;
    innfil.ignore(30, '='); innfil >> konfig.p3fil;
    innfil.ignore(30, '='); innfil >> konfig.innledning;
    innfil.ignore(30, '='); innfil >> konfig.frekvens;
    innfil.ignore(30, '='); innfil >> konfig.delta_t;

    if (filtypetekst == "kai") konfig.filtype = 1;
    if (filtypetekst == "xyz") konfig.filtype = 2;
    if (filtypetekst == "xy")  konfig.filtype = 3;
    if (konfig.filtype == 0) { cout << "Ugyldig datafil!\n"; exit(-2); }
    innfil.close;
  }


//////////////////////////////////////////////
// Leser koordinater i et tidssteg fra       //
// koordinatfilen inn i en partikkel-vektor  //
//////////////////////////////////////////////
vector <Partikkel> les_koord(ifstream &innfil, Konfigurasjon &konfig)
  {
    int i, teller, antsteg, antpart, steg;
    float tid;
    Partikkel p;
    vector <Partikkel> pvec;
    string atomtype(10, ' ');  // Behves bare for  hoppe over atom-navnet

    if (konfig.filtype == 1)    // leser kai-fil
      for (i = 0; i < konfig.antpart; i++)
        {
          innfil >> p.x >> p.y;
```

```
            innfil >> steg;
            p.z = 0;
            pvec.push_back(p);
          }

      if (konfig.filtype == 2)    // leser xyz-fil
        {
          if (innfil.tellg() == 0)
            innfil >> antsteg;
          innfil >> antpart;
          for (i = 0; i < antpart; i++)
            {
              innfil >> atomtype >> p.x >> p.y >> p.z;
              pvec.push_back(p);
            }
        }

      if (konfig.filtype == 3)    // leser xy-fil
        {
          if (innfil.tellg() == 0)
            innfil >> antsteg;
          innfil >> antpart >> tid;
          for (i = 0; i < antpart; i++)
            {
              innfil >> atomtype >> p.x >> p.y;
              p.z = 0.0;
              pvec.push_back(p);
            }
        }

      return(pvec);
    }


  ////////////////////////////////////
  // Skriver ut statistiske filer. //
  ////////////////////////////////////

  void skriv_fil(Konfigurasjon &konfig, Flette &flette, Statistikk &stat, int innledning)
    {
      int i, j, steg;
      float tid;
      string sjekk(70,' ');

      // Statistiske data
      sjekk = konfig.statfil;
      if (sjekk != "ingen") {
      ofstream statfil(konfig.statfil);
      if(!statfil) {
        cout << "Kan ikke pne " << konfig.statfil << " for skriving\n";
        exit(-1); }
      cout << "Skriver " << konfig.statfil << endl;

      // I stat-filen skrives de fleste statistiske strrelsene i hver sin kolonne
      for (i = innledning; i < flette.krysning.size(); i++)
        statfil << flette.krysning[i].tid
            << "    "
            << stat.writhe[i]
            << "    "
            << stat.writhe_hastighet[i]
            << "    "
            << stat.gj_writhe_hastighet[i]
            << "    "
            << stat.writhe_fluktuasjon[i]
            << "    "
```

```
        << endl;
statfil.close; }

// Sannsynlighetsfordeling ved slutt
sjekk = konfig.pfil;
if (sjekk != "ingen") {
cout << "Skriver " << konfig.pfil << endl;
ofstream pfil(konfig.pfil);
if(!pfil) {
  cout << "Kan ikke pne " << konfig.pfil << " for skriving\n";
  exit(-1); }
for (i = 0; i < stat.p.size(); i++)
  pfil << stat.p[i] << endl;
pfil.close; }

// Sannsynlighetsfordeling over tid
sjekk = konfig.p3fil;
if (sjekk != "ingen") {
cout << "Skriver " << konfig.p3fil << endl;
ofstream p3fil(konfig.p3fil);
if(!p3fil) {
  cout << "Kan ikke pne " << konfig.p3fil << " for skriving\n";
  exit(-1); }
cout << "Skriver " << konfig.p3fil << endl;

// Ved plot i 3D i matlab er det greit  begrense antall steg.
// I hvertfall med en hjemmesnekret graf-funksjon
if (stat.treD_p.size() > 100) steg = (int) stat.treD_p.size()/100;
  else steg = 1;
for (i = innledning; i < stat.treD_p.size(); i += steg)
  {
    p3fil << i << "  ";
    for (j = 0; j < stat.treD_p[i].size(); j ++)
      p3fil << stat.treD_p[i][j] << "  ";
    p3fil << endl;
  }
p3fil.close; }

// Statistiske data med hele tidssteg
sjekk = konfig.helstatfil;
if (sjekk != "ingen") {
ofstream helstatfil(konfig.helstatfil);
if(!helstatfil) {
  cout << "Kan ikke pne " << konfig.helstatfil << " for skriving\n";
  exit(-1); }
cout << "Skriver " << konfig.helstatfil << endl;
tid = 0.0;
for (i = innledning; i < flette.krysning.size(); i++)
  if (flette.krysning[i].tid > tid)
    {
      helstatfil << flette.krysning[i].tid
          << "    "
          << stat.writhe[i]
          << "    "
          << stat.writhe_hastighet[i]
          << "    "
          << stat.gj_writhe_hastighet[i]
          << "    "
          << stat.writhe_fluktuasjon[i]
          << "    "
          << stat.signert_krysningstall[i]
          << endl;
      tid += 1.0;
    }
helstatfil.close; }
```

```
    // Fletteoperatorfil
    sjekk = konfig.flettefil;
    if (sjekk != "ingen") {
    ofstream flettefil(konfig.flettefil);
    if(!flettefil) {
      cout << "Kan ikke pne " << konfig.flettefil << " for skriving\n";
      exit(-1); }
    cout << "Skriver " << konfig.flettefil << endl;
    for (i = innledning; i < flette.krysning.size(); i++)
      for (j = 0; j < flette.krysning[i].Operator.size(); j++)
        flettefil << flette.krysning[i].tid
                << "  "
                 << flette.krysning[i].Operator[j]
                << endl;
    flettefil.close; }
  }
```

## fletting.C

```
#include <vector>
#include "Partikkel.h"
#include "Partikkelblokk.h"

////////////////////////////////////////////////////////
// Finner permutasjonen til partiklene i partikkelblokken //
////////////////////////////////////////////////////////
vector <int> finn_permutasjon(Partikkelblokk &pblokk)
        {
                vector <int> perm(pblokk.part.size());
                int i, j, teller;

                for (i = 0; i < pblokk.part.size(); i++)
                        {
                                teller = 0;
                                // sjekker hvor mange partikler som har hyere y-verdi enn partikkel i
                                for (j = 0; j < pblokk.part.size(); j++)
                                        if (pblokk.part[i].y > pblokk.part[j].y && i != j)
                                                teller += 1;
        // perm[0] er nummeret p verste partikkel osv.
                                perm[teller] = i;
                        }
                return(perm);
        }

////////////////////////////////////////////
// Sjekker om det er noen endring i inbyrdes //
// plassering i y-retning blant partiklene   //
////////////////////////////////////////////
bool endring(Partikkelblokk pf, Partikkelblokk pe, int steg)
        {
                int i;
                bool endring = 0;
                if (pf.part.size() != pe.part.size())
                        {
                                cout << "Endring i partikkel antall. Pingler ut! steg = " << steg
                                <<endl;
                                exit(-2);
                        }
                for (i = 0; i < pf.part.size(); i++)
                        if (pf.perm[i] != pe.perm[i]) endring = 1;
                return(endring);
        }

//////////////////////////////////////////////
// Finner operatorene som definerer endringene //
// mellom to tidssteg.                         //
//////////////////////////////////////////////
vector <int> finn_operator(Partikkelblokk pf, Partikkelblokk pe, int steg)
        {
                vector <int> perm, neste_perm(pe.part.size());
                int i, j, tmp;
                bool endring = 1;

                for (j = 0; j < pe.part.size(); j++)
                        for (i = 0; i < pe.part.size(); i++)
                                if (pf.perm[j] == pe.perm[i])
                                        {
                                                neste_perm[j] = i;
                                                if (i!=j) endring = 1;
                                        }
```

```
        while (endring)
                {
                        endring = 0;
                        for (i = 0; i < pf.part.size() - 1; i++)
                                if (neste_perm[i+1] < neste_perm[i])
                                        {
                                                if (pe.part[pf.perm[i]].x < pe.part[pf.perm[i+1]].x)
                                                        perm.push_back(i + 1);
                                                else
                                                        perm.push_back(-i - 1);
                                                tmp = neste_perm[i];
                                                neste_perm[i] = neste_perm[i+1];
                                                neste_perm[i+1] = tmp;
                                                tmp = pf.perm[i];
                                                pf.perm[i] = pf.perm[i+1];
                                                pf.perm[i+1] = tmp;
                                                endring = 1;
                                        }
                }
        return(perm);
}
```

## statistikk.C

```
#include <vector>
#include "Konfigurasjon.h"
#include "Krysning.h"
#include "Flette.h"

//////////////////////////////////
// Finner vridningen til fletten //
//////////////////////////////////
vector <int> finn_writhe(Flette &f)
  {
    int i, j, writhe_teller = 0;
    vector <int> w;
    for (i = 0; i < f.krysning.size(); i++)
      {
        for (j = 0; j < f.krysning[i].Operator.size(); j++)
          if (f.krysning[i].Operator[j] > 0)
            writhe_teller += 1;
          else
            writhe_teller -= 1;
        w.push_back(writhe_teller);
      }
    return(w);
  }


/////////////////////////////////////////////////////
// Finner vridningshastigheten til vridningsvektoren //
/////////////////////////////////////////////////////
vector <float> finn_writhe_hastighet(vector <int> &writhe, Flette &flette,
int delta_t)
  {
    int i, j, forrige_writhe = 0;
    vector <float> writhe_hastighet(flette.krysning.size(), 0.0);

    // Lper gjennom alle krysningene
    for (i = 0; i < flette.krysning.size(); i++)
      {
        j = i;
        // Teller bakover til en krysning ved tiden t-delta_t
        while (flette.krysning[i].tid
            < (flette.krysning[j].tid + (float) delta_t) && j != 0)
          j--;
        // Sjekker om forrige krysning l nrmere t-delta_t
        if (flette.krysning[i].tid - delta_t - flette.krysning[j].tid
          >flette.krysning[j+1].tid - flette.krysning[i].tid + delta_t)
            j++;
        // vridningsforskjellen / delta_t
        writhe_hastighet[i] = (float) (writhe[i] - writhe[j])/delta_t;
        if (j == 0) writhe_hastighet[i] = 0;
      }
    return(writhe_hastighet);
  }

//////////////////////////////////////////////////////////////////
// Finner gjennomsnittlig vridningshastighet som funksjon av tiden //
//////////////////////////////////////////////////////////////////
vector <float> finn_gjennomsnitts_writhe_hastighet(vector <int> &writhe, Flette
&flette)
  {
    int i;
    vector <float>gj_hastighet;
    gj_hastighet.push_back(0.0);
    for (i = 1; i < flette.krysning.size(); i++)
```

```
      gj_hastighet.push_back(writhe[i]/flette.krysning[i].tid);
    return(gj_hastighet);
  }

////////////////////////////////////////////////////
// Variasjonene rundt det endelige gjennomsnittet //
////////////////////////////////////////////////////
vector <float> finn_writhe_fluktuasjon(vector <float> &writhe_hastighet, vector <float> &gj_writhe_hastighet)
  {
    int i;
    vector <float> writhe_fluktuasjon;
    for (i = 0; i < writhe_hastighet.size(); i++)
      if (writhe_hastighet[i] > 0)
        writhe_fluktuasjon.push_back(writhe_hastighet[i]-gj_writhe_hastighet[writhe_hastighet.size()-1]);
      else
        writhe_fluktuasjon.push_back(0);
    return(writhe_fluktuasjon);
  }

//////////////////////////////////////////////
// Sannsynlighetsfordelingen av hastighetene //
//////////////////////////////////////////////
vector <int> finn_sannsynlighetsfordeling(Flette &flette, vector <int> writhe, int antpart)
  {
    float teller = 1.0;
    int i, forrige = 0;
    int maks;
    maks = (antpart*(antpart - 1)+(antpart-1)*(antpart-2))/2;
    vector <int> p(maks, 0);

    // For  finne fordelingen kan ikke hastigheten vre "kontinuerlig"
    // Hastighetsfordelingen er ogs kun for hastigheter med delta_t=1
    for (i = 0; i < writhe.size(); i++)
      if (flette.krysning[i].tid > teller)
        {

          teller += 1.0;
          p[writhe[i]-writhe[forrige]] += 1;
          forrige = i;
        }
    return(p);
  }

////////////////////////////////////////////////////////////////////
// Lager en vektor med vektorer av sannsynlighetsfordeling over tid //
////////////////////////////////////////////////////////////////////
vector <vector <float> > treD_sannsynlighet(Flette &flette, vector <int> writhe, int antpart)
  {
    vector <vector <float> > treD_sannsynlighet;
    float teller = 1.0;
    int i, j, forrige = 0;
    int maks;
    maks = (antpart*(antpart - 1)+(antpart-1)*(antpart-2))/2;
    vector <int> p(maks, 0);
    vector <float> p_abs(maks, 0);
    for (i = 0; i < writhe.size(); i++)
      if (flette.krysning[i].tid > teller)
        {
          teller += 1.0;
          p[writhe[i]-writhe[forrige]] += 1;
          forrige = i;
          for (j = 0; j < maks; j++)
            p_abs[j] = p[j]/flette.krysning[i].tid;
          treD_sannsynlighet.push_back(p_abs);
        }
```

```
        return(treD_sannsynlighet);
    }
```

## Konfigurasjon.h

```
class Configuration
        {
                public:
                char picfilename[70];
                char outfilename[70];
                int threshold;
                int numofpics;
                int minsize, maxsize;
        };
```

## Partikkel.h

```
class Particle
        {
                public:
                float x,y,z;
                vector <Pixel> pix;
        };
```

## Partikkelblokk.h

```
class Partikkelblokk
        {
                public:
                vector <Partikkel> part;
                vector <int> perm;
        };
```

## Krysning.h

```
class Krysning
        {
                public:
                vector <int> Operator;
                float tid;
        };
```

## Flette.h

```
class Flette
        {
                public:
                vector <Krysning> krysning;
        };
```

## Statistikk.h

```
class Statistikk
        {
                public:
                vector <int> writhe;
                vector <float> writhe_hastighet;
                vector <float> gj_writhe_hastighet;
                vector <float> writhe_fluktuasjon;
                vector <float> signert_krysningstall;
                vector <float> sum_signert_krysning;
                vector <int> p;
                vector <vector <float> > treD_p;
        };
```

## rutiner.h

```
// leserutiner
void les_konfig(char *filnavn, Konfigurasjon &konfig);
vector <Partikkel> les_koord(ifstream &innfil, Konfigurasjon &konfig);

// fletterutiner
vector <int> finn_permutasjon(Partikkelblokk &pblokk);
bool endring(Partikkelblokk pf, Partikkelblokk pe, int steg);
vector <int> finn_operator(Partikkelblokk pf, Partikkelblokk pe, int steg);

// statistiske rutiner
vector <int> finn_writhe(Flette &f);
vector <float> finn_writhe_hastighet(vector <int> &writhe, Flette &flette, int delta_t);
vector <float> finn_gjennomsnitts_writhe_hastighet(vector <int> &writhe, Flette &flette);
vector <float> finn_writhe_fluktuasjon(vector <float> &writhe_hastighet, vector <float> &gj_writhe_hastighet);
vector <int> finn_sannsynlighetsfordeling(Flette &flette, vector <int> writhe, int antpart);
vector <vector <float> > treD_sannsynlighet(Flette &flette, vector <int> writhe, int antpart);

// skriverutiner
void skriv_fil(Konfigurasjon &konfig, Flette &flette, Statistikk &stat, int innledning);
```

## D.3 Setupmaker

Program i C som setter opp den tilfeldig fordeling av et gitt antall partikler innenfor et gitt område og med en gitt minimumsavstand mellom partiklene. Tar 'setup.conf' som konfigurasjonsfil.

**setup.conf**

1-3: Definerer boksen kulene befinner seg i.
4: Minimum avstand mellom partiklene.
5: Antall partikler.

```
delta x = 18
delta y = 0.01
delta z = 0.01
min delta r = 1.0
ant part = 15
```

**setupmaker.c**

```
/*********************************************/
/* Genererer en psf og xyz-fil med valgfritt */
/* antall partikler over et valgfritt volum. */
/*********************************************/

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

int main(int argc, char **argv)
  {
    FILE *outf, *inf, *outf2;
    char *infile, *outfile, _progName;
    char boss;
    int i, j, antpart, ok;
    float coord[3][1000], dx, dy, dz, r, test;
    char *filename = "";
    srand48( (unsigned int)time( NULL ) ); //seeder srand vha tiden

    // Opening infile and outfile
    if ((inf = fopen ("setup.conf", "r")) == NULL)
      {
        printf("This infile does not compute!\n");
        exit (-1);
      }
    // Reading config-file
    for (i = 0; i < 9; i++) fscanf (inf, "%c", &boss);
    fscanf (inf, "%f\n", &dx);
    for (i = 0; i < 9; i++) fscanf (inf, "%c", &boss);
    fscanf (inf, "%f\n", &dy);
    for (i = 0; i < 9; i++) fscanf (inf, "%c", &boss);
    fscanf (inf, "%f\n", &dz);
    for (i = 0; i < 13; i++) fscanf (inf, "%c", &boss);
    fscanf (inf, "%f\n", &r);
    for (i = 0; i < 10; i++) fscanf (inf, "%c", &boss);
    fscanf (inf, "%d\n", &antpart);
    close (inf);
```

```
  if ((outf = fopen ("out.xyz", "w")) == NULL)
    {
      printf("Cannot creat outfile!\n");
      exit (-1);
    }
  if ((outf2 = fopen ("out.psf", "w")) == NULL)
    {
      printf("Cannot creat outfile!\n");
      exit (-1);
    }

  for (i = 0; i < antpart ; i++)
    {
      do
        {
          coord[0][i] = drand48()*dx;
          coord[1][i] = drand48()*dy;
          coord[2][i] = drand48()*dz;
          ok = 1;
          for (j = 0; j < i; j++)
            if ((coord[0][i]-coord[0][j])*(coord[0][i]-coord[0][j])
               +(coord[1][i]-coord[1][j])*(coord[1][i]-coord[1][j])
               +(coord[2][i]-coord[2][j])*(coord[2][i]-coord[2][j])
               < r*r) ok = 0;
        } while (ok == 0);
    }
  // Skriver xyz-fil
  fprintf(outf, "%d\n", antpart);
  fprintf(outf, "psf og xyz generator (c) 2002 mr. Hellesoy... dx = %f, dy = %f, antpart = %d\n", dx, dy, ant
  for (i = 0; i < antpart ; i ++)
    {
      fprintf(outf, "CAL  %f  %f  %f\n", coord[0][i], coord[1][i], coord[2][i]);
    }
  close (outf);
  fprintf(outf2, "PSF\n\n");
  fprintf(outf2, "  1 !NTITLE\n");
  fprintf(outf2, " REMARKS Generated by mr Hellesoy's xyz and psf maker.\n\n");
  fprintf(outf2, "     %d !NATOM\n", antpart);
  for (i = 0; i < antpart ; i++)
    {
      fprintf(outf2, "     %d XXXX   %d IONE X   CAL   0.4   1   0\n", i, i);
    }
  close (outf2);
}
```

## D.4 Length

Et program i C som beregner lengden av kjeder med kuler. Lengden beregnes ved en rekursiv prosedyre som starter i en partikkel, merker den, ser etter naboer, går til en nabo, merker den osv. osv. Når det ikke er flere kuler igjen i kjeden starter programmet på nytt med en tilfeldig kule. Egentlig teller ikke programmet lengden på kjeder, men antallet kuler som henger sammen. Konfigurasjonsfilen 'conf' behøves for å starte programmet. Programmet gir to utskrifts-filer: En med midlere kjedelengde og tid og en med fordeling av lengdene over tid. I tillegg kan lengdefordelingen vises grafisk under analysen.

**conf**

1: Utskrifts-frekvens.
2: Xyz-formatet inneholder ikke tid, og delta t definerer tiden mellom hvert steg.
3: Maksimum avstand mellom to partikler som berører hverandre.
4: Navn på xyz-filen.
5: graf=1 viser en fordeling av kjedelengdene under analysen.

```
delta n = 1
delta t = 1.0
r = 1.05
filename = 500s.cutoff.xyz
graf = 0
```

**length.c**

```
/*****************************************/
/*  Beregner midlere lengde av kjeder over */
/*  tid fra .xyz-filer fra protomol.       */
/*  Avstand som regnes som berrende       */
/*  defineres i konfigurasjonsfilen.       */
/*****************************************/

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <fstream.h>
#include <iostream.h>

int TRUE = 1, FALSE = 0;
int dn, lengde[50], l, teller, antkjeder;
float float1, float2;
FILE *outf, *inf, *inf2;
char *infile, *outfile, _progName, filename[20], streng[255];
char boss;
float dt;
bool graf;

    int n, m, i, j, antpart, antsteg;
    int maks = 0, hoyest = 0;
    float t, r;
    class partikkel
      {
```

```
        public:
        bool analysert;
        float x, y, z;
        int antnaboer;
        int nummer;
        partikkel *nabo[10]; //Det kan ikke vre mer enn ti nabokuler.
      };
    partikkel part[2000];  //Begrensning p 2000 partikler.
    int kjede[100];        //Begrensning p kjedelengde ved 100.


//Rutine som undersker om to partikler berrer hverandre
bool beroring(partikkel *p1, partikkel *p2, float avst)
  {
    if ((p1->x - p2->x)*(p1->x - p2->x)
       +(p1->y - p2->y)*(p1->y - p2->y)
      +(p1->z - p2->z)*(p1->z - p2->z) < avst)
      return (TRUE);
    else
      return (FALSE);
  }


// Finner hvilke partikler som grenser til partikkel *p som ikke allerede er underskt.
void sjekkpartikkel(partikkel *p)
  {
    int a;
    p->analysert = TRUE;
    for (a = 0; a < antpart; a++)
      {
        if (part[a].analysert == FALSE)
          if (beroring(p, &part[a], r) == TRUE)
            {
              p->nabo[p->antnaboer] = &part[a];
              p->antnaboer += 1;
            }
      }
    l += p->antnaboer;
    for (a = 0; a < p->antnaboer; a++)
      if (p->nabo[a]->analysert == FALSE) sjekkpartikkel(p->nabo[a]);
  }

// Leser konfigurasjonsfil
void hentkonfigurasjon()
  {
    ifstream inf("conf");  //pner konfigurasjonsfilen "conf"
    if (!inf) cout << "Kan ikke pne inputfil.\n";

    // Reading configfile:

    inf.ignore(20, '='); inf >> dn;
    inf.ignore(20, '='); inf >> dt;
    inf.ignore(20, '='); inf >> r;
    inf.ignore(20, '='); inf >> filename;
    inf.ignore(20, '='); inf >> graf;
    cout << "pner '" << filename << "' for input." << endl;
    inf.close();
  }


int main()
  {
    hentkonfigurasjon();
    ofstream outf("fordeling.dat");
    if (!outf) cout << "Kan ikke pne outputfil.\n";
```

```
ofstream outf2("gjlengde.dat");
if (!outf2) cout << "Kan ikke pne outputfil.\n";
ifstream inf(filename);
if (!inf) cout << "Kan ikke pne datafil.\n";
inf >> antsteg;

// Hovedrutine:
for (n = 0; n < antsteg; n = n + dn)
  {
    maks = 0;
    inf >> antpart;
    // Henter inn partikkel-koordinater
    for (m = 0; m < antpart; m++)
      {
        for (i = 0; i < 4; i++)
          inf.get(boss);  //Ikke all informasjonen i xyz-filen er interressant.
        inf >> part[m].x >> part[m].y >> part[m].z;
        part[m].nummer = m;
        part[m].analysert = FALSE;
        part[m].antnaboer = 0;
      }

    //Begynner med partikkel 1 og analyserer videre
    //Partiklene som er analysert blir merket, og telles ikke flere ganger.
    for (i = 0; i < antpart; i++)
      if (part[i].analysert == FALSE)
        {
          l = 0;
          sjekkpartikkel(&part[i]);
          lengde[l+1] += 1;
          if (l+1 > maks) maks = l+1;
          if (lengde[l+1] > hoyest) hoyest = lengde[l+1];
        };

    outf << "Tid: " << dt*n << endl;
    outf << "Maks: " << maks << endl;
    for (i = 1; i < maks + 1; i++)
      {
        outf << "L" << i << " = " << lengde[i] << endl;
        antkjeder += lengde[i];
      }
    outf << endl;

    // Skjermprint:
    cout << "Steg " << n;
    float1 = antpart; float2 = antkjeder;
    if (graf) cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
    cout << "  Gj.sn. kjedelengde: " << float1/float2 << endl;
    outf2 << n << "  " << float1/float2 << endl;
    if (graf)
      {
        cout << endl;
        for (i = 1; i < 21; i++)
          {
            if (i < 10) cout << " ";
            cout << i << " ";
            for (j = 0; j < lengde[i] && j < 41; j++)
              cout << "*";
            if (lengde[i] > 40) cout << "->";
            cout << endl;
          }
      }
    for (i = 1; i < maks + 1; i++)
      lengde[i] = 0;
```

```
        antkjeder = 0; hoyest = 0;

        // Hopper dn tidssteg
        for (i = 1; i < dn; i++)
          {
            //cout << "tidssteg " << n << ". dn = " << dn << " i = " << i << " ";
            inf >> antpart;
            //cout << "antpart = " << antpart << endl;
            inf.getline(streng, 255);
            for (j = 0; j < antpart; j++)
            {
              inf.getline(streng, 255);
              //cout << streng << endl;
            }
          }
      }
    outf.close;
    inf.close;
  }
```

## D.5 Find

Et program i C++ som finner posisjon til kuler fra en serie bilder fra et eksperiment. Programmet tar en konfigurasjonsfil som argument. I konfigurasjonsfilen spesifiseres blant annet hvor lyst og stort et område skal være som regnes som en kule. Gjennomsnittsposisjonen til bildepunktene til en kule regnes da som kulens sentrum. Find begynner programmet 'convert' som må konvertere bilder til Portable PixMap (ppm) format type P3 (ascii).

**En konfigurasjonsfil**

1: Første del av bildenavnene
2: Innfil
3: Lyshet på kule
4: Antal bilder
5: Minimum antall bildepunkter i en kule
6: Maksimum antall bildepunkter i en kule

```
picfilename = f30.e1.mpg
outfilename = utfil
threshold = 19000
numofpics = 10
minsize = 5
maxsize = 20
```

**find.C**

```
////////////////////////////////////////////////////////
// Program to find particles from pictures. Right     //
// now it's only using jpg-pics made from quick-time, //
// but it's easy to add other type of pictures too.   //
////////////////////////////////////////////////////////

#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "Pixel.h"
#include "Partikkel.h"
#include "Konfigurasjon.h"

void read_config(char *filename, Configuration &config)
  {
    // Opening configuration file
    ifstream infile(filename);
    if(!infile)
      {
        cout << "Cannot open " << filename << ".\n";
        exit(-1);
      }
    else
      cout << "Opening '" << filename << "' as configuration-file.\n";
    infile.ignore(30, '='); infile >> config.picfilename;
    infile.ignore(30, '='); infile >> config.outfilename;
    infile.ignore(30, '='); infile >> config.threshold;
```

```
    infile.ignore(30, '='); infile >> config.numofpics;
    infile.ignore(30, '='); infile >> config.minsize;
    infile.ignore(30, '='); infile >> config.maxsize;
    infile.close;
  }

void read_pic(vector <vector <int> > &pic, int &xsize, int &ysize)
  {
    // Reading ppm-ascii-picturefile
    int garb, i, j, number;
    ifstream picfile("tmp.ppm");
    picfile.ignore(30, '#');picfile.ignore(30, '#');
    picfile >> xsize;
    picfile >> ysize;
    picfile >> garb;
    for (i=0; i < ysize; i++)
      for (j = 0; j < xsize; j++)
        {
          picfile >> number >> number >> number;
          pic[j][i] = number;
        }
    picfile.close;
  }

// Making the picture black and white using the threshold
void make_bw(vector <vector <int> > &pic, int &xsize, int &ysize, int &threshold, vector <vector <int> > &bwpic
  {
    int i,j;
    for (i = 0; i < xsize; i++)
      for (j = 0; j < ysize; j++)
        if (pic[i][j] > threshold && i > 20 && i < xsize-20 && j > 20 && i < ysize - 20)
          bwpic[i][j] = 10;
        else
          bwpic[i][j] = 0;
  }

\\Writing the black and white picture for later viewing
void write_pic(vector <vector <int> > &bwpic, int &xsize, int &ysize)
  {
    int i,j;
    ofstream picfile("tmp2.ppm");
    picfile << "P2\n";
    picfile << xsize << " " << ysize << endl;
    picfile << "10\n";
    for (i = 0; i < ysize; i++)
      for (j = 0; j < xsize; j++)
        picfile << bwpic[j][i] << " ";
    picfile.close;
  }

void find_pix(int x, int y, vector <vector <int> > &bwpic, vector <Pixel> &p)
  {
    // This routine works mainly as distance.
    int a;
    Pixel pix;
    pix.x = x; pix.y = y; p.push_back(pix);
    int neighboors[4][2];
    int numofneighboors = 0;
    bwpic[x][y] =0;

    // Checking if there are any neighboors around.
    if (bwpic[x-1][y] == 10) {neighboors[numofneighboors][0]=x-1;neighboors[numofneighboors][1]=y; numofneighbo
    if (bwpic[x+1][y] == 10) {neighboors[numofneighboors][0]=x+1;neighboors[numofneighboors][1]=y; numofneighbo
    if (bwpic[x][y-1] == 10) {neighboors[numofneighboors][0]=x;neighboors[numofneighboors][1]=y-1; numofneighbo
    if (bwpic[x][y+1] == 10) {neighboors[numofneighboors][0]=x;neighboors[numofneighboors][1]=y+1; numofneighbo
```

```
    for (a=0;a<numofneighboors;a++)

    // If there are neighboors, check all of them the same way.
      if (bwpic[neighboors[a][0]][neighboors[a][1]] == 10) find_pix(neighboors[a][0], neighboors[a][1], bwpic,
  }

// Finding particles based on the bw-picture.
vector <Particle> find_particles(vector <vector <int> > &bwpic, int &xsize, int &ysize, int minsize, int maxsiz
  {
    vector <Particle> pblock;
    int i, j, k, l;
    for (i=0; i<xsize; i++)
      for (j=0; j<ysize;j++)
        if (bwpic[i][j] == 10)
          {
            Particle particle;
            vector <Pixel> p;
            find_pix(i,j,bwpic,p);
            int sumx=0, sumy=0;
            for (k=0;k < p.size(); k++)
              { sumx+=p[k].x; sumy += p[k].y; }
            particle.x = sumx/( (float) p.size() ); particle.y = sumy/( (float) p.size() );
            if (p.size() > minsize && p.size() < maxsize) pblock.push_back(particle);
          }
    return(pblock);
  }

void main(int argc, char **argv)
  {
    int i, j, frame, ysize, xsize, threshold, lastnumberofparticles;
    char picname[70], execline[150];
    vector <vector <int> > pic(2000, vector <int> (2000,0)); // double vector for the colourpicture.
    vector <vector <int> > bwpic(2000, vector <int> (2000,0)); // double vector for the black and white picture
    vector <Particle> pblock; //block of all the particles found
    Configuration config;

    read_config(argv[1], config);

    ofstream outfile(config.outfilename);
    outfile << config.numofpics << endl;

    for (frame = 1; frame < config.numofpics+1; frame++)
      {
        // When using quicktime to extract pictures
        // the filenames are given as "filename 01.jpg" "filename 02.jpg" etc.
        // ... "filename 123.jpg", so the first ten filenames start with a zero.
        if (frame > 9)
          sprintf(picname,"%s\\ %i.jpg",config.picfilename,frame);
        else
          sprintf(picname,"%s\\ 0%i.jpg",config.picfilename,frame);
        cout << picname << endl;

        // This uses convert which has to convert a picture to ppm type P3 (ascii)
        sprintf(execline,"convert %s tmp.ppm;", picname);
        system(execline);
        read_pic(pic, xsize, ysize);
        make_bw(pic, xsize, ysize, config.threshold, bwpic);
        write_pic(bwpic, xsize, ysize);

        // Showing the first picture for checking
        if (frame == 1) {sprintf(execline,"xv tmp2.ppm"); system(execline);}
        pblock = find_particles(bwpic, xsize, ysize, config.minsize, config.maxsize);

        // Checks if there’s a change in the number of particles.
        if (lastnumberofparticles != pblock.size() && frame > 1)
```

```
            {cout << "Change in number of particles from " << lastnumberofparticles << " to " << pblock.size() <<
          else
            lastnumberofparticles = pblock.size();
          outfile << pblock.size() << endl;
          for (i=0; i<pblock.size(); i++)
            outfile << pblock[i].x << " " << pblock[i].y << endl;
        }
      outfile.close;
    }
```

## Pixel.h

```
class Pixel
        {
                public:
                int x,y;
                vector <Pixel> neighbors;
                int neighborhood;
        };
```

## Partikkel.h

```
class Particle
        {
                public:
                float x,y,z;
                vector <Pixel> pix;
        };
```

## Konfigurasjon.h

```
class Configuration
        {
                public:
                char picfilename[70];
                char outfilename[70];
                int threshold;
                int numofpics;
                int minsize, maxsize;
        };
```

## D.6 Trace

Find ordner koordinatene slik at kulene med lavest x-verdi kommer først i xy-filen. Trace finner hvilke kuler som tilhører hvilke koordinater ved å la hver kule følge de koordinatene som er nærmest mellom to etterfølgende bilder. Hvis to kuler prøver å følge samme kule stopper programmet. Trace tar to argumenter, innfil og utfil.

**trace.C**

```
////////////////////////////////////////////////////
// Program to make the particles follow the right //
// coordinates made by find.C                     //
// It's quite straight foreward.                  //
////////////////////////////////////////////////////

#include <stdlib.h>
#include <math.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

void main(int argc, char **argv)
  {
    int numofsteps, numofparticles, step, i, j;
    if (argc < 3) { cout << "Usage: distance inputfile outputfile\n"; exit(-2);}
    ifstream infile (argv[1]);
    if(!infile) { cout << "Cannot open " << argv[1] << " as infile.\n";}
    ofstream outfile (argv[2]);
    infile >> numofsteps;
    outfile << numofsteps << endl;
    class Particle
      {
        public:
        float x,y;
        int taken;
      };

    // Assigning numbers to particles:
    infile >> numofparticles;
    outfile << numofparticles << endl;
    Particle particle[numofparticles];
    for (i = 0; i < numofparticles; i++)
      {
        infile >> particle[i].x >> particle[i].y;
        outfile << particle[i].x << "  " << particle[i].y << endl;;
      }

    // Main loop
    Particle newcoords[numofparticles];
    int follow;
    float dsquared;
    for (step = 1; step < numofsteps; step++)
      {
        infile >> numofparticles;
        for (i = 0; i < numofparticles; i++)
          {
            infile >> newcoords[i].x >> newcoords[i].y;
            newcoords[i].taken = 0;
          }
        for (i = 0; i < numofparticles; i++)
```

```
            {
              float mindist=1000.0;
              for (j = 0; j < numofparticles; j++)
                {
                  dsquared = (particle[i].x-newcoords[j].x)
                     *(particle[i].x-newcoords[j].x)
                     +(particle[i].y-newcoords[j].y)
                     *(particle[i].y-newcoords[j].y);
                  if (dsquared < mindist)
                    {
                      mindist = dsquared;
                      follow = j;
                    }
                }
              if (newcoords[follow].taken == 0)
                {
                  particle[i] = newcoords[follow];
                  newcoords[follow].taken = 1;
                }
              else
                {
                  cout << "Particle " << i << " tried to follow particle "
                       << follow << " which was already taken!\n";
                  exit (-2);
                }
            }

        // Writing new coords;
        outfile << numofparticles << endl;
        for (i = 0; i < numofparticles; i++)
          outfile << particle[i].x << "  " << particle[i].y << endl;
      }
  }
```

# Bibliografi


[1] A.T. Skjeltorp - *One- and Two-Dimensional Crystallization of Magnetic Holes* Phys. Rev. Lett. **51**, 2306 (1983).

[2] J. Ugelstad, P.C. Mørk, K.H. Kaggerud, T. Ellingsen, A. Berge - *Swelling of oligomer-polymer particles. New methods of preparation* Adv. Colloid Int. Sci. **13**, 101 (1980).

[3] R.E. Rosensweig - *An introduction to Ferrohydrodynamics* Chem. Eng. Comm. **67**, 1 (1988).

[4] K. Raj og R. Moskowitz - *Commercial Applications of Ferrofluids* Journal of Magnetism and Magnetic Materials **85**, 233-245 (1990).

[5] G. Helgesen - *Cooperative Processes for Magnetic Holes and Microspheres* doktorgradsavhandling ved UiO (1990)

[6] A.T. Skjeltorp, G. Helgesen - *Qualitative aspects of condensation ordering and aggregation* i *Phase trasitions in soft condensed matter* red. T. Riste og D. Sherrington, Plenum Press (1989)

[7] S. Clausen - doktorgradsavhandling (1997)

[8] A.T. Skjeltorp - *Condensation and ordering of colloidal spheres dispersed in a ferrofluid* Physica A, **213** 30 (1995)

[9] P. Pieranski, S. Clausen, G. Helgesen, A.T. Skjeltorp - *Braids plaited by magnetic holes* Phys. Rev. Lett. **77** 1620 (1996)

[10] K.dL. Kristiansen - *Some applications of knot and braid theory in experimental physics* hovedfagsoppgave ved UiO (2000)

[11] M. Warner, R.M. Hornreich - *The stability of quasi 2D lattices of magnetic holes* J. Phys. A, **18** 2325 (1985)

[12] R. Toussaint - *Interaction model for microspheres in a ferrofluid* preprint (2001)

[13] K. Mahoney - *Magnetization, multiparticle interactions, and chain dynamics in magnetic fluids under rapidly rotating external fields* doktorgradsavhandling ved M.I.T. (1999)

[14] D. J. Jeffrey og Y. Onishi - *Calculation of the resistance and mobility functions for two unequal rigid spheres in low Reynolds number flow* Journal of Fluid Mechanics **139**, 261 (1984)

[15] G. Helgesen, P. Pieranski, A.T. Skjeltorp - *Dynamic behavior of simple magnetic hole systems* Phys. Rev. A **42**, 7271 (1990)

[16] P.K. Kundu, I.M. Cohen - *Fluid mechanics* Academic Press (1990), ISBN 0-12-428770-0

[17] J.R. Reitz, F.J. Milford, R.W. Christy - *Foundations of electromagnetic theory* Addison-Wesley (1993), ISBN 0-201-52624-7

[18] E.M. Purcell - *Life at low Reynolds number* Am. J. Phys. **45**,3 (1977).

[19] I.M. James - *History of Topology* Elsevier (1999), ISBN 0-444-82375-1

[20] K. Murasugi - *Knot Theory & Its Applications*, Birkhäuser (1996), ISBN 0-8176-3817-2

[21] J.D. Jackson - *Classical Electrodynamics* John Wiley & Sons P.K. (1975), ISBN 0-471-43132-X

[22] Thomas Schücker - *Knots in chemistry and physics* New Journal of chemistry **17**, 655 (1993)

[23] S. Amann, C. Streit, and H. Bieri. *BOOGA – A component-oriented framework for computer grahics* GraphiCon, 193, Moscow, (1997).

[24] T. Matthey *Framework Design, Parallelization and Force Computation in Molecular Dynamics*. Doktorgradsavhandling ved Institutt for informatikk, UiB (2002).

[25] J. A. Izaguirre, J. Willcock, T. Matthey, Q. Ma, T. B. Slabach, T. Steinbach, S. Stender, G. F. V., and J. Mohnke. PROTOMOL*: An object oriented framework for molecular dynamics*. Tilgjengelig på nett via `http://www.cse.nd.edu/~lcls/Protomol.html` (2000–2002)

[26] `http://www-graphics.stanford.edu/~cek/rayshade/rayshade.html`, *RayShade* Web-side.

[27] J.S. Birman - *Braids, Links and Mapping Class Group* Annals of Math. Study, Princeton Univ. Press (1974)

[28] R.E. Allen - *The Pocket Oxford Dictionary* Clarendon Press (1991), ISBN 0-19-861133-1

[29] G. Helgesen, J. Cernak (upublisert).

[30] V.F.R. Jones, Bulletin of the American Mathematical Society **1** 12 (1985).