

Verfahren zur automatisierten Erkennung planarer Mechanismen in Videosequenzen

Kai Lawrence

30. August 2021

**Verfahren zur automatisierten Erkennung
planarer Mechanismen in
Videosequenzen**

**Bachelor-Thesis
am Fachbereich Maschinenbau
Studiengang Maschinenbau**

vorgelegt von: Kai Lawrence

Matrikelnummer: 7090084

Erstprüfer: Prof. Dr.-Ing. Stefan Gössner

Zweitprüfer: Prof. Dr. Flavius Guias

Abgabetermin: 06.09.2021

University of Applied Sciences And Arts, Fachhochschule Dortmund
30. August 2021

Kai Lawrence

Verfahren zur automatisierten Erkennung planarer Mechanismen in Videosequenzen

Fachbereich Maschinenbau, 30. August 2021

Prüfer: Prof. Dr.-Ing. Stefan Gössner und Prof. Dr. Flavius Guias

Betreuer: Prof. Dr. Stefan Gössner

University of Applied Sciences And Arts, Fachhochschule Dortmund

Fachbereich Maschinenbau

Sonnenstraße 96-100

44139 and Dortmund

Zusammenfassung

In dieser Arbeit wird untersucht, welche Informationen anhand der Unterschiede einzelner Bilder in einer Videosequenz gewonnen werden können, um einen sich darin befindlichen planaren Mechanismus zu rekonstruieren. Die dafür durchgeführten Versuche sind mit gängigen Webtechnologien implementiert. Hierfür werden die Pole der ebenen Bewegung für sich bewegende Glieder ermittelt. Außerdem werden Mittel gesucht mit denen die unterschiedlichen Glieder eines Mechanismus getrennt, so dass diese isoliert untersucht werden können. Schlussendlich werden die daraus ermittelten Pole der ebenen Bewegung genutzt, um ein Modell des Mechanismus zu erstellen.

Abstract

This work investigates what information can be obtained from the differences between individual images in a video sequence in order to reconstruct a planar mechanism within it. The experiments carried out for this purpose are implemented with common web technologies. For this purpose, the poles of planar motion are determined for moving links. In addition, means are sought with which the different links of a mechanism can be separated so that they can be examined in isolation. Finally, the resulting poles of plane motion are used to create a model of the mechanism.

Inhaltsverzeichnis

1 Einleitung	1
2 Erstellung der Versuchsumgebung	3
2.1 Die index.html	3
2.2 simulation	5
2.2.1 createElements	6
2.2.2 run	6
2.2.3 register	7
2.3 Aufbau der Versuchsseiten	8
2.4 Definition eines mec2 Modells	9
3 Informationsgewinnung aus Videosequenzen	11
3.1 compareImages	12
3.2 Bestimmung des kleinsten umfassenden Kreises	13
3.3 Schnittpunkte von Geraden	16
3.3.1 Der Umgang mit Ungenauigkeit	17
3.3.2 Nutzung des Schwerpunktes mehrerer Datenpunkte	20
3.4 Bestimmung der Regressionsgeraden	21
3.4.1 Orthogonale Regression	23
3.5 Vergleich des Verlaufes von Randpunkten	24
3.6 Auswertung der Beobachtungen	25
4 Ermittlung des optischen Flusses	27
4.1 Optischer Fluss nach Lucas-Kanade	28
4.1.1 Implementation des Lucas-Kanade Algorithmus	29
4.2 Andere Methoden	31
4.2.1 Optischer Fluss nach Farneback	31
4.2.2 Ermittlung von Bewegung durch maschinelles Lernen	32
4.3 Betrachtung der Ergebnisse	33
5 Ermittlung des Momentanpols	35
5.1 Implementation	36

5.2	Das drehende Rad	37
5.3	Gestellglieder	39
5.3.1	Drehendes Gestellglied	39
5.3.2	Translatives Gestellglied	40
5.4	Beobachtungen	40
6	Zuordnung von Datenpunkten zu Gliedern	43
6.1	Zuordnung von Geraden an Punkte	43
6.2	k-Means Algorithmus	45
6.3	Dijkstra Algorithmus	46
6.3.1	Nutzung des Korrelationskoeffizienten	49
6.4	Kombination der Ansätze	51
6.5	Rekonstruktion eines Mechanismus durch die Relativpole	52
6.5.1	Ermittlung der Anzahl der gesuchten Punkte	53
6.5.2	Betrachtung aller Relativpole	54
7	Zusammenfassung und Ausblick	57
Literatur		59
Erklärung über selbständige erbrachte Leistungen		69

Einleitung

In dieser Arbeit wird die automatisierte Erkennung von planaren Mechanismen durch wahrgenommene Bewegung in Bildsequenzen versucht.

Die Lösung des Problems ist interessant, da diese Aufgabe typischerweise von einem Menschen auch ohne Hintergrund in den Ingenieurwissenschaften problemslos gelöst werden kann. Selbst ohne Bewegung würde ein Mensch vermutlich mit geringen Abweichungen die beweglichen Gelenkpunkte und dessen vermeintliche Bewegung bestimmen können. Hierfür wird auf Erfahrung und Intuition zurückgegriffen, über die ein Programm typischerweise nicht verfügt. Neuronale Netzwerke könnten sicherlich trainiert werden, um die Gelenkpunkte eines Mechanismus anhand einer Momentaufnahme zu bestimmen. In dieser Arbeit soll jedoch untersucht werden, welche Informationen über planare Mechanismen innerhalb von Videosequenzen auf analytischem Wege gewonnen werden können.

Für die Bestimmung der Posen bekannter kinematischer Ketten existieren Lösungen, welche bereits in der Praxis angewandt werden. Diese Ansätze werden unter dem Begriff der Pose-Estimation zusammengefasst. Besonders prominent sind hierrunter Ansätze, welche die Pose eines auf Menschen definierten Modells ermitteln und visualisieren [Pap+18; @Goo21b; @Goo21a]. Im Unterschied zur Problemstellung dieser Arbeit, in der die kinematischen Ketten ermittelt werden sollen, gehen solche Methoden von bereits bekannten kinematischen Ketten aus.

Die Erkennung der Bewegung soll durch den Vergleich von Bildern in Videosequenzen erfolgen. Ziel ist das Untersuchen von Algorithmen, welche die Bewegung



Abb. 1.1: Versuch `bilder1_1.html`. Im linken Bild wird der Mechanismus mit darüberliegendem `mec2` Modell gezeigt. Das mittlere Bild zeigt den ursprünglichen Mechanismus. Im rechten Bild ist zur verbesserten Ansicht nur das `mec2` Modell zu sehen. Ein Mensch ist durch Erfahrung und Intuition in der Lage diesen Mechanismus direkt zu modellieren.

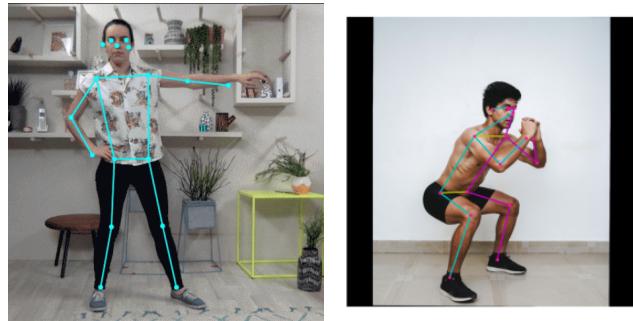


Abb. 1.2: Ermittlung der Pose einer kinematischen Kette auf Modellbasis von Menschen durch PersonLab [@Goo21b] (links) und MoveNet [@Goo21a] (rechts).

einzelner Glieder ermitteln. Die Ergebnisse sollen dann genutzt werden, um die Pole der ebenen Bewegung für die entsprechenden Glieder zu finden. Da ein Mechanismus aus mehreren Gliedern besteht, müssen Methoden entwickelt werden, um diese Glieder zu isolieren. Anhand der Gliedebenen und derer Momentanpole werden die übrigen Relativpole ermittelt. Unter den Relativpolen können dann die Gelenkpunkte gesucht werden.

Es werden folgende Annahmen getroffen, welche teilweise durch die Nutzung der Physik-Engine `mec2` vorgegeben sind. So sind die Glieder des Mechanismus inkompressibel und unverbieglich. Entsprechend kann auch Knickung nicht auftreten. Das Gestell ist unbeweglich und auch keines der Gelenke weist Spiel auf. Des Weiteren wird angenommen, dass alle translativen Elemente als Gestellglieder auftreten.

Die Mechanismen werden mit Hilfe von `mec2` simuliert, deren Animation daraufhin als Videosequenz untersucht werden. Die Verfahren sind alle in den Standardwebtechnologien HTML und JavaScript implementiert. Die entsprechenden Versuche können unter <https://aka.klawr.de/ba#1> gefunden werden. Alle Versuche wurden mit dem Webbrowser Mozilla Firefox 91.0.1 auf Windows 10 durchgeführt.

Erstellung der Versuchsumgebung

„ Wenn Sie der Meinung sind, dass gutes Design teuer ist, sollten Sie sich die Kosten für schlechtes Design ansehen.

— Ralf Speth
Ehem. Geschäftsführer bei Jaguar Land Rover

2.1 Die index.html

Die Versuche sind jeweils eigenständige HTML-Seiten, welche über ein HTML-Iframe Element in einer zentralen index.html untersucht werden können. Die Versuche sind wiederum in einzelne Sachverhalte gruppiert. Für diese Gruppierung wird in der index.html ein tests Objekt definiert.

```

1 const tests = {
2     pendel: {
3         title: "Drehpunkt bei gegebenem Glied.",
4         test: [
5             "kleinster umfassender Kreis": [
6                 "Erster Versuch",
7                 "Filtern bekannter Punkte",
8                 "Pfad des Mittelpunktes",
9                 "Drehpunkt via Orthogonale",
10                ],
11             "schnittpunkte": [
12                 "Weitest entfernte Punkte",
13                 "5 weiteste Punkte",
14                 // ...
15             ]
16         ]
17     }
18 }
```

Listing 2.1: Ausschnitt der Definition des tests Objekts in der index.html.

Über die Schlüssel des tests Objekts werden die Gruppen festgehalten. Die Werte von tests definieren die in dieser Gruppe gemachten Versuche. Es sind entsprechend Listen von Titeln für die Versuche darin enthalten.

Die automatische Einbindung der Versuchsdateien geschieht über die analoge Ordnerstruktur des Projektes. Jede Eigenschaft des `tests` Objekts hat im `src` Ordner relativ zum Projektverzeichnis einen genauso bezeichneten Ordner. Dieser enthält die einzelnen HTML-Versuchsseiten, welche ebenfalls nach einem festen Schema benannt sind. Die Versuchsgruppen sind `pendel`, `opticalflow`, `momentanpol`, `gruppen` und `bilder`. Der in Abbildung 2.1 gezeigte Test ist entsprechend unter dem Relativpfad `src/pendel/pendel1_4.html` zu finden. Anhand dieser Strukturierung werden alle Versuche durch den in Listing 2.2 beschriebenen Aufruf dynamisch in die `index.html` eingebunden.

```

1  Object.entries(tests).forEach((kv, i) => {
2      const title = create('h2', null, kv[1].title);
3      const details = create('details',.byId(' sidenav'));
4      const summary = create('summary', details, kv[1].title);
5
6      Object.entries(kv[1].test).forEach((e, j) => {
7          const innerDetails = create('details', details);
8          const innerSummary = create('summary', innerDetails);
9          innerDetails.classList.add('innerDetails');
10         innerSummary.innerHTML = e[0];
11
12         const ul = create('ul', innerDetails);
13
14         for (let m = 0; m < e[1].length; ++m) {
15             const name = `${kv[0]} + (j + 1)_${m + 1}.html`;
16             const li = create('li', ul, e[1][m] || name);
17             li.addEventListener('click', () => {
18                 byId('iframe').src = `src/${kv[0]}/${name}`;
19                 Array.from(document.getElementsByTagName('li'))
20                     .forEach(e => e.style.listStyle = 'disc');
21                 li.style.listStyle = 'inside';
22             });
23         }
24     });
25 });

```

Listing 2.2: Iteration über das `tests` Objekt zur Befüllung der Navigationsleiste.

Hier wird für jeden Eintrag im `tests` Objekt ein neues HTML-Details Element erstellt. Über die Listen der einzelnen Einträge wird iteriert, um die einzelnen Gruppen gemeinsam in ein weiteres HTML-Details Element zu platzieren¹. Diese Listen werden durch HTML-Listen Elemente gerendert. Für die Einträge der Listen werden entsprechende `EventListener`² registriert, welche die `src` Eigenschaft des HTML-Iframe

¹Hier als `innerDetails` bezeichnet.

²Für mehr Information über `EventListener` siehe <https://aka.klawr.de/ba#2>.

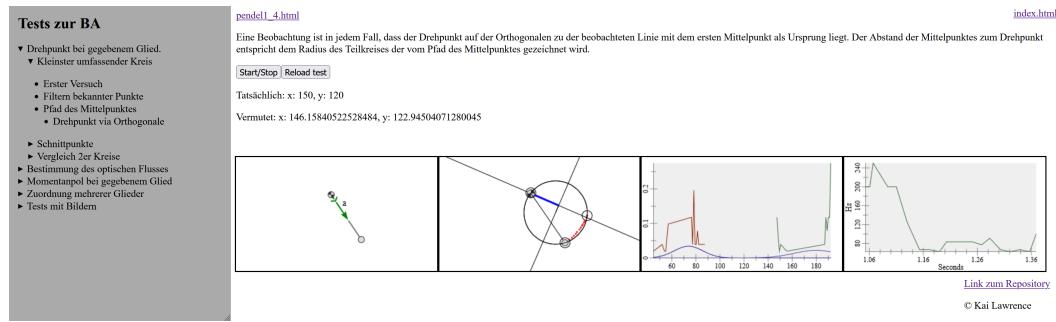


Abb. 2.1: Bild der `index.html`, wobei in der Navigationsleiste der Versuch zur Ermittlung des Drehpunktes ausgewählt ist. Der Hauptteil der Seite wird durch das HTML-Iframe Element gefüllt, welches den entsprechenden Versuch enthält.

Elements aktualisieren. Sollte der Pfad einer Datei nicht diesem Aufbau entsprechen, so wird ein Fehler geworfen.

Für die Formatierung der `index.html` wurden die entsprechenden Elemente mit Hilfe von CSS gestaltet, worauf hier jedoch nicht weiter eingegangen werden soll. Der Quelltext kann unter <https://aka.klawr.de/ba#3> eingesehen werden.

2.2 simulation

Die über viele Versuche wiederkehrenden Funktionalitäten sind in eine zentrale Bibliothek ausgelagert worden. Durch die zentrale Definition wird der Aufwand zur Wartung des Codes und zur Behebung von Fehlern stark reduziert. Dieses Prinzip ist unter dem Akronym **DRY**³ bekannt [And21].

Das `simulation` Objekt beinhaltet alle Referenzen zu den HTML-Elementen, auf welche im Zuge der Versuche zugegriffen werden kann sowie den Referenzen zu den einzelnen `CanvasRenderingContext2D` Objekten, welche von `g2` genutzt werden, um auf die entsprechenden HTML-Canvas Elemente zu zeichnen. Außerdem werden unter Anderem globale Konstanten wie die Höhe und Breite der HTML-Canvas Elemente zentral bereitgestellt.

Auf die in `simulation` definierten Funktionen soll im Folgenden näher eingegangen werden. Diese bestimmen die Schnittstelle, welche die Versuchsseiten implementieren müssen. Alle nachfolgenden Funktionen sind als Eigenschaften des `simulation` Objektes zu verstehen.

³don't repeat yourself, zu deutsch wiederhole dich nicht.

2.2.1 createElements

Die HTML-Elemente, welche innerhalb der Testseiten verwendet werden, sind grundsätzlich stets die Gleichen. Aus diesem Grund kann das Befüllen der Testseiten mit den entsprechenden HTML-Elementen zentral definiert werden. Hier werden alle Interaktionen, welche für die Versuche notwendig sind sowie vier HTML-Canvas Elemente zur einheitlichen Darstellung der Versuchsmetriken definiert. Auf `createElements` wird in Kapitel 2.3 genauer eingegangen.

2.2.2 run

Die `run` Funktion sorgt dafür, dass alle Versuche unter vergleichbaren Bedingungen ablaufen.

```
1  async run(step) {
2      this.model?.tick(1 / 60);
3      await this.g.exe(this.ctx1);
4      this.time_reset = performance.now();
5      step();
6      this.updateTimesChart().exe(this.ctx_times);
7
8      if (this.running) {
9          this.rafdId = requestAnimationFrame(() => {
10              this.run(step)
11          });
12      }
13  },
```

Listing 2.3: Definition der `simulation.run` Funktion.

Existiert ein `mec2` Modell, so wird dessen Simulation an dieser Stelle um einen Zeitschritt vorangetrieben und zur Animation verwendet. Die Definition des `mec2` Modells findet in den Versuchsdateien statt. Innerhalb der an `run` übergebenen `step` Funktion darf dieses Modell nicht modifiziert werden, da es als Quelle für die Bildsequenzen dient.

Anschließend wird der `CanvasRenderingContext2D` des ersten HTML-Canavs Elements `ctx1` gerendert⁴. Auf dieses HTML-Canvas Element sollte ausschließlich hier zugegriffen werden, damit der Versuch nicht versehentlich Einfluss auf die Zeichnung nimmt und sich somit selbst beeinflusst.

⁴Vor der `exe` Funktion steht hier ein `await`. Wenn Bilder geladen werden, muss die `exe` Funktion leicht modifiziert werden. Notwendig ist dies in der Versuchsgruppe `bilder2` (siehe <https://aka.klawr.de/ba#4>). Dementsprechend ist die `run` Funktion als `async` deklariert.

Nachdem Rendern der Videosequenz wird die `run` übergebene `step` Funktion ausgeführt. Diese Funktion implementiert den jeweiligen Versuchsalgorithmus und wird dementsprechend in den jeweiligen Kapiteln der Versuche beschrieben.

Abschließend wird hier noch die Dauer der `step` Funktion vermerkt. In `updateTimesChart` wird der aktuelle Zeitpunkt mittels `performance.now`⁵ ermittelt und einer Liste hinzugefügt, welche die Historie der Ausführungszeiten der einzelnen `step` Aufrufe festhält. Es wird außerdem gemessen, wie viel Zeit seit dem Beginn der Aufzeichnung vergangen ist, um eine sinnvolle Darstellung der Daten im Graphen zu ermöglichen. Dieser Graph wird anschließend auf einem HTML-Canvas Element durch den `exe(ctx_times)` Aufruf gezeichnet. `ctx_times` ist der Kontext des vierten der von `createElements` erstellten HTML-Canvas Elementen.

Anschließend wird die `run` Funktion durch `requestAnimationFrame` erneut aufgerufen, sofern der Versuch nicht pausiert wird⁶.

2.2.3 register

`register` ist die Funktion, welche von den einzelnen Tests aufgerufen wird, um die HTML-Seite zu befüllen und die Versuchsalgorithmus zu injizieren.

In `register` wird zunächst der Titel festgelegt, welcher dem Namen der HTML-Datei entspricht. Dieser Titel wird in einem HTML-Anker Element platziert, damit diese in der `index.html` einfacher zugeordnet werden können. Es dient außerdem als Hyperlink, über welchen isoliert auf den Versuch zugegriffen werden kann, was sich während der Entwicklung als hilfreich herausgestellt hat.

An dieser Stelle wird außerdem eine Funktion definiert, welche beim Laden der Seite ausgeführt wird. Diese Funktion instanziert alle Variablen, welche für die Erstellung der Versuche notwendig sind.

Es wird zudem einmalig die `run` Funktion aufgerufen, um das erste Bild zu rendern⁷. Die an `register` übergebene `step` Funktion wird an `run` weitergereicht. Es wird außerdem noch der Knopf zum Starten und Pausieren der Tests definiert. Die Betätigung dieses Knopfes schaltet mittelbar die `simulation.running` Variable um.

Das `simulation` Objekt enthält noch weitere Variablen und Funktionen, welche Einfluss auf den Ablauf der Tests haben. Um die Beschreibung der `index.html` jedoch

⁵Für mehr Information zu `performance.now` siehe <https://aka.klawr.de/ba#5>.

⁶`running` kann beispielsweise durch einen Start/Stop Knopf umgeschaltet werden.

⁷Da `simulation.running` mit den Wert `false` initialisiert wird, wird innerhalb des `run` Aufrufs `requestAnimationFrame` nicht ausgeführt.

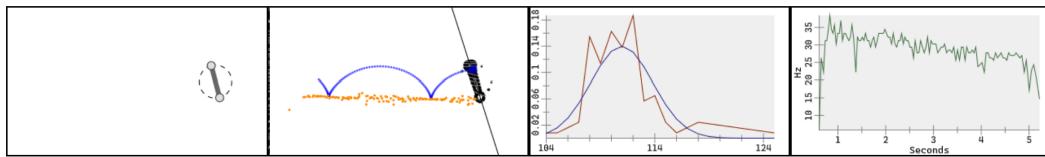


Abb. 2.2: Beispiel für die Canvasse. Hier wird der Versuch `momentanpol1_3` gezeigt. Von links nach rechts: Bild, welches den Eingang darstellt. Bild, welches visualisiert was in dem Test untersucht wird. Darstellung gesammelter Daten innerhalb des Tests. Bild zur Ermittlung der Performanz des Versuchs.

auf das Wesentliche zu reduzieren, sei auf den Quellcode hingewiesen, welcher unter <https://aka.klawr.de/ba#6> eingesehen werden kann.

2.3 Aufbau der Versuchsseiten

Die Versuche unterscheiden sich, abgesehen von der an die `register` Funktion übergebenen Funktion, vor allem durch die eingebundenen Dateien. Alle Versuchsdateien binden `g2.full.js`, `mec2.min.js` und `simulation.js` ein. Neben diesen existieren noch andere Hilfsbibliotheken, welche nicht von allen Versuchen genutzt werden. Jede Versuchsdatei beinhaltet zusätzlich eine kurze Beschreibung des Algorithmus.

Wie bereits angemerkt, dient die `createElement` Funktion der Abstraktion der Struktur aller Versuchsseiten. Diese Funktion wurde in Kapitel 2.2.1 eingeführt und soll nun konkret beschrieben werden.

Die vier erstellten HTML-Canvas Elemente sollen in den Versuchen jeweils vergleichbare Rollen übernehmen. Das Erste wird dazu verwendet den Versuchsalgorithmus mit Bildern zu versorgen. Im zweiten HTML-Canvas Element werden die Ergebnisse des Versuchsalgorithmus visualisiert. Die Visualierung der Daten hilft bei der Beurteilung des Versuchserfolgs und offenbart möglicherweise Erkenntnisse. Das dritte HTML-Canvas Element soll Statistiken über die ermittelten Daten zeigen. In den meisten Tests wird dieses Element genutzt, um die ermittelten Koordinaten in einem Graphen darzustellen. Das Rendering des zweiten und dritten HTML-Canvas Element wird durch die `step` Funktion übernommen. Das Befüllen des vierten HTML-Canvas Elementes wurde bereits in Kapitel 2.2.2 beschrieben.

2.4 Definition eines `mec2` Modells

In den meisten Versuchen wird ein `mec2` Modell verwendet, um eine kontrollierte Videosequenz eines Mechanismus zu erstellen. Hier wird die Beschreibung eines Pendels durch `mec2` gezeigt. Eingliedrige Modelle dieser Art werden in der ersten Versuchsreihe verwendet. Es wird durch den Drehpunkt, den Endpunkt und deren konstante Länge zueinander definiert. Bewegt wird das Modell, sofern es keinen Antrieb hat, lediglich durch das Gewicht des Endpunktes. Ein Beispiel für die Definition eines solchen Modells wird in Listing 2.4 und in Abbildung 2.1 gezeigt.

```
1 simulation.model = {  
2     gravity: true,  
3     nodes: [  
4         { id: 'A0', x: 150, y: 100, base: true },  
5         { id: 'A1', x: 230, y: 130 }  
6     ],  
7     constraints: [  
8         { id: 'a', p1: 'A0', p2: 'A1', len: { type: 'const' } }  
9     ]  
10};
```

Listing 2.4: Definition eines Pendels in `mec2` inklusive Animation.

Das `mec2` Modell wird in der entsprechenden Versuchsdatei definiert, damit dieses in den in Kapitel 2.2 beschriebenen Funktionen korrekt referenziert werden kann.

Dieses Modell besteht aus zwei `nodes`, welche die Endpunkte des Pendels darstellen. Einer dieser beiden Punkte wird als `base` deklariert, was ihn mit dem Gestell verbindet. Aus diesem regulären JavaScript Objekt wird ein `mec2` Modell, indem es durch `mec.model.extend` erweitert wird. Dieser Vorgang wurde bereits in Kapitel 2.2.3 beschrieben. Die Animation wird durch die `simulation.run` Funktion gesteuert, welche ebenfalls in Kapitel 2.2.2 beschrieben wurde.

Informationsgewinnung aus Videosequenzen

„ Was kann ich wissen?
Was soll ich tun?
Was darf ich hoffen?“

— Immanuel Kant
Kritik der reinen Vernunft

Zunächst soll untersucht werden, welche Informationen aus Videosequenzen gewonnen und die daraus abgeleiteten Erkenntnisse genutzt werden können. Es wird zunächst mit einem einzelnen Glied in Form eines einfachen Pendels gearbeitet. Dieses stellt im Grunde ein einzelnes Glied eines beliebigen Mechanismus dar. Die Methoden zur korrekten Erkennung eines Pendels sind also notwendige Werkzeuge für einen Ansatz, der die Glieder eines Mechanismus einzeln betrachtet.

Als zentrales Ziel dieser Methoden wird die Bestimmung des Drehpunkts eines Pendels gesetzt; es sei jedoch angemerkt, dass dieser als Absolutpol lediglich einen Sonderfall des Momentanpols darstellt, auf dessen Bestimmung in Kapitel 5 eingegangen wird. Die Erkennung des Drehpols wird als vereinfachtes Ziel genommen, um herauszufinden mit welchen Informationen überhaupt gearbeitet werden kann.

Um die Bewegung innerhalb einer Bildsequenz zu erkennen, werden hier stets zwei Bilder miteinander verglichen, welche zeitlich etwa 16,7 Millisekunden auseinanderliegen. Die Frequenz wird vorgegeben durch `requestAnimationFrame` und beträgt im Optimalfall 60 Hertz. Dieser Optimalfall sei dadurch definiert, dass die `step` Funktion aus `run` (s. Listing 2.3) weniger als 16,7 Millisekunden benötigt¹.

¹Neben der an `run` übergebenen `step` Funktion werden natürlich noch andere Funktionen wie `model.tick` und `updateTimesChart` ausgeführt, diese sollen hier jedoch nicht betrachtet werden. Sie werden daher auch für die Zeitmessung innerhalb von `run` ausgeschlossen.

3.1 compareImages

Da sich in der modellierten Bildsequenz nur das Pendel bewegt, reicht es aus die Bilder pixelweise zu vergleichen und die Koordinaten der ungleichen Pixel aufzuzeichnen. Hierfür wurde eine Funktion `stepCompareImages` definiert, welche durch die an die `register` Funktion übergebene Funktion aufgerufen wird. Diese Funktion speichert den aktuellen Inhalt des ersten HTML-Canvas Elementes durch `cnv1.getContext('2d').getImageData(0, 0, cnv1.width, cnv1.height).data`. Dieser wird dann mit dem in der letzten Iteration gespeichertem Bild verglichen. Da der erste Aufruf dieser Funktion keinen Vergleich mit einem vorangegangenen Bild zulässt, wird geprüft, ob ein solches existiert und es wird eine Iteration abgewartet bevor der erste Vergleich stattfindet.

Für solche Vergleiche wird eine Klasse `PointCloud` definiert. Diese Klasse beinhaltet alle Funktionen, welche auf und mit Punktfolgen benötigt werden. Unter anderem wird hier eine `fromImages` Funktion bereitgestellt, welche zwei Bilder auf Unterschiede untersucht und diese als Punktfolge zurückgibt.

```
1  fromImages(image1, image2, width, height) {
2      const difference = [];
3      for (let y = 0; y < height; ++y) {
4          for (let x = 0; x < width; ++x) {
5              const i = y * width + x;
6              if (image1[i * 4] !== image2[i * 4]) {
7                  difference.push({ x, y });
8              }
9          }
10     }
11
12     return new PointCloud(difference);
13 }
```

Listing 3.1: Definition der `fromImages` Funktion, welche eine statische Funktion der `PointCloud` Klasse darstellt.

Diese Funktion speichert eine `difference` Liste, welche jeweils Objekte mit den Koordinaten von gefundenen Unterschieden speichert. Dieser Ansatz zeichnet sich durch seine Einfachheit aus, birgt allerdings offensichtliche Nachteile. So würde die Bewegung von Dingen, welche für keinen Pixel eine Farbänderung verursachen, nicht erkannt werden. Es werden also nur dort Änderungen erkannt, wo sich die Farbe des entsprechenden Pixels ändert. Es ist außerdem zu beachten, dass durch die Betrachtung der reinen Änderung der Pixel sich bewegende Objekte stets zweimal abzeichnen. Dass heißt wenn die beiden Bilder verglichen werden, dass genau jene

Pixel unterschiedlich sind, an denen das Ding im ersten Bild und im zweiten nicht mehr ist und umgekehrt. Ohne weitere Analyse lässt sich jedoch nicht so einfach bestimmen in welche Richtung sich das Element bewegt hat. Außerdem ist mit dieser Methode nicht von einer exakten Abmessung auszugehen, sondern die ermittelten Koordinaten ergeben eher eine Art Punktfolge, bei der von einer gewissen Streuung auszugehen ist.

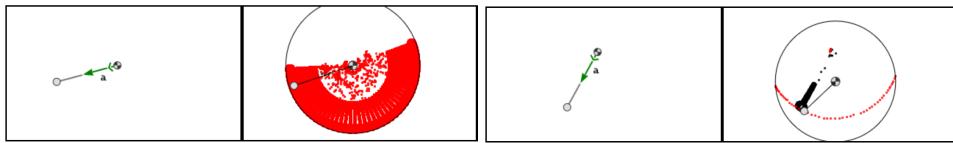
An die `stepCompareImages` Funktion kann wiederum eine Funktion übergeben werden, welche dann mit dem von `fromImages` zurückgegebenen `PointCloud` Objekt aufgerufen wird. Diese Sammlung an Koordinaten wird verwendet, um sie auf Regelmäßigkeiten zu untersuchen. Wie bereits angemerkt, wird als erstes versucht ein Pendel zu rekonstruieren. Im Nachfolgenden sollen für diese Aufgabe unterschiedliche Ansätze untersucht werden.

3.2 Bestimmung des kleinsten umfassenden Kreises

Die Koordinaten von Pixeln, für die in zwei Bildern ein Unterschied gemessen wird, werden in einem Objekt der dafür erstellten `PointCloud` Klasse festgehalten. Diese Koordinaten sollten bei der Drehung eines Gliedes um einen festen Punkt einen Bogen sichtbar machen, der Aufschluss über die Position dieses Drehpunktes gibt. Die genaue Bestimmung eines solchen Bogens durch den Mittelpunkt, den Radius und den eingeschlossenen Winkel anhand der Punktfolge zu ermitteln, erscheint zunächst schwierig. Stattdessen wird beim ersten untersuchten Ansatz der kleinste Kreis bestimmt, welcher alle bisher gesammelten Datenpunkte umfasst. Der kleinste umfassende Kreis kann in linearer Zeit² ermittelt werden. Der entsprechende Algorithmus wurde von Nimrod Megido [Meg83] entwickelt und die Implementation wird von <https://aka.klawr.de/ba#7> bezogen. Eine anschauliche Demo kann unter <https://aka.klawr.de/ba#8> [@Nay21] betrachtet werden.

Um einen solchen Kreis zu bestimmen, müssen zunächst die Koordinaten, für die eine Änderung erkannt wurde, festgehalten werden. Da zu Beginn der Aufnahme die Region, welche durch die Punkte gefüllt wird, sehr schnell an Fläche zunimmt, ändern sich die Koordinaten und der Radius des Mittelpunktes des kleinsten umfassenden Kreises schnell. Sobald sich das drehende Glied einer halben Umdrehung nähert, verändert sich die Definition des kleinsten umfassenden Kreises augenscheinlich gar nicht mehr und dessen Mittelpunkt scheint mit einer sehr kleinen Abweichung dem

²Das heißt die Dauer des Algorithmus steht in linearem Verhältnis zur Menge der Datenpunkte.



- (a) Versuch pendel1_1.html. Im ersten Can-
vas wird das Pendel gezeigt. Im zweiten
Bild werden die Punkte gezeigt, welche
gespeichert wurden um daraus den kleinsten
umfassenden Kreis dieser Punkte zu
ermitteln.
- (b) Versuch pendel1_2.html. Hier werden
die Punkte, welche jeweils nicht zum Kreis
beitragen gefiltert. Der eingeschlossene
Winkel ist jedoch kleiner, weshalb der Mit-
telpunkt des Kreises hier nicht dem Dreh-
punkt entspricht.

Abb. 3.1

tatsächlichen Drehpunkt zu entsprechen. Es wird bei diesem Versuch der zuletzt ermittelte Mittelpunkt als Drehpunkt der Gliedebene aufgefasst.

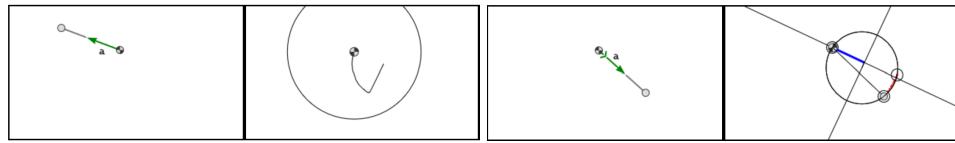
Zur Visualisierung des Modells wird das andere Ende des Pendels vorerst durch den Punkt definiert, der die größte Distanz zum prognostizierten Drehpunkt aufweist.

Im ersten Versuch zeigt sich, dass der Ansatz, die Menge aller erfassten Punkte zu speichern, sehr schnell ein Performanzproblem verursacht. Es ist glücklicherweise nicht notwendig alle Punkte zu behalten. Für jeden Kreis, welcher über den zeitlichen Verlauf gebildet wird, können alle Punkte verworfen werden, welche nicht zur Erstellung dieses Kreises verwendet wurden. Dies sind je nach Verteilung der Punktwolke entweder drei oder auch nur zwei Punkte, da ein Kreis genau über zwei oder drei Punkte definierbar ist.

Dafür wurde die Funktion des genutzten Algorithmus angepasst, so dass er nicht mehr nur den Kreis sondern auch die entsprechenden Punkte zurückgibt, welche für die Erstellung des entsprechenden Kreises genutzt wurden³. Außerdem wird nach der Ermittlung des Kreises zwischen den bereits bekannten und der aktuell ermittelten Punktwolke unterschieden, so dass die Liste der bekannten Punkte keine Duplikate enthält.

Diese Anpassung reduziert die Menge der Punkte die im ersten Test in einer Sekunde ermittelt werden von 19880 auf 294 Datenpunkte. Von diesen 294 Datenpunkten sind lediglich 69 den bekannten Datenpunkten zuzuordnen. Die anderen 225 Punkte lassen sich den gemessenen Veränderungen der letzten beiden verglichenen Bildern zuordnen. Es zeigt sich, dass in vielen Fällen die Menge der Datenpunkte reduziert werden kann ohne das Einbußen bei der Genauigkeit der Versuche in Kauf genommen werden müssen.

³Die entsprechende Änderung kann auf <https://aka.klawr.de/ba#9> nachvollzogen werden.



- (a) Versuch pendel1_3.html. Im dritten Versuch wird gezeigt, dass der Mittelpunkt des über die Zeit sich ändernden kleinsten umfassenden Kreises einen voraussichtlich interessanten Pfad bildet.
- (b) Versuch pendel1_4.html. Dieser Versuch zeigt, dass der durch den Mittelpunkt kreisierte Pfad genutzt werden kann, um bei Drehungen weit unter 90° den Drehpunkt zu bestimmen.

Abb. 3.2

Dieser Ansatz erkennt für ein Glied, das sich mindestens 180° um den definierten Punkt dreht, einen sehr nahe am tatsächlichen Wert gelegenen Drehpunkt. Der offensichtliche Nachteil besteht jedoch darin, dass alle Glieder, die sich um weit weniger als 180° drehen, mit Sicherheit kein gutes Ergebnis liefern, wie im zweiten Test nachvollzogen werden kann. Nähere Untersuchungen zum kleinsten umfassenden Kreis können jedoch weitere Erkenntnisse bringen. Zunächst ist der Pfad, welcher vom Mittelpunkt des über die Zeit wachsenden kleinsten umfassenden Kreis interessant. Für die ersten 90° , die sich das Glied dreht, bewegt sich der Mittelpunkt auf der Mittelsenkrechten der Geraden, welche durch die beiden Punkte des ersten Kreises gebildet wird. Der Abstand zum Drehpunkt ist ebenfalls bekannt, da bekannt ist, dass er exakt dem Radius dieses Kreises entspricht⁴. Die weiteren 90° werden durch einen Kreisbogen mit dem ersten Mittelpunkt des eben beschriebenen Pfades als Mittelpunkt beschrieben. Abschließend landet der Mittelpunkt des kleinsten umfassenden Kreises im Drehpunkt des betrachteten Gliedes.

Es kann außerdem beobachtet werden, dass der kleinste umfassende Kreis, solange er eine Gerade bildet, durch drei Punkte definiert wird⁵. Sobald das Glied jedoch eine Drehung von über 90° vollführt, wird der Kreis nur noch durch zwei Punkte beschrieben. Diese drei Punkte beinhalten die beiden Punkte des zuerst gebildeten Kreises⁶ und den letzten erfassten Endpunkt des Pendels. Sobald eine Drehung von über 90° vollführt wurde, ist der Kreis, der durch den ersten sowie den letzten Drehpunkt definiert wird, jedoch größer, weshalb der Drehpunkt nicht mehr zum Kreis beiträgt.

Aus diesen Erkenntnissen lässt sich ein Algorithmus bilden, welcher auch für Bewegungen weit unter 90° eine hinreichend gute Vorhersage über die Position des

⁴Das ist äquivalent zur Hälfte der Gliedlänge.

⁵Mit Ausnahme der ersten Iteration, wo er allein durch die Endpunkte des Gliedes beschrieben werden kann.

⁶Tatsächlich wird der Kreis in mehr als nur einer Iteration durch nur zwei Punkte definiert wird. Dies kann jedoch auf die Streuung der Punkte zurückgeführt werden.

Drehpunktes bringt. Solange die Anzahl der Punkte, die den letzten Kreis bilden, bei über zwei liegt, wird hierfür die Steigung vom ersten zum letzten Punkt im Pfad des Mittelpunktes des kleinsten umfassenden Kreises gemessen. So kann der Drehpunkt erheblich früher mit hinreichender Genauigkeit bestimmt werden. Auf die Berechnung wird hier nicht näher eingegangen, weil es sich gezeigt hat, dass obwohl die Erkenntnisse aus diesem Versuch sehr hilfreich sind, die Methoden zur Ermittlung des kleinsten umfassenden Kreises nicht weiter verwendet werden. Der hier genutzte Quellcode kann jedoch unter <https://aka.klawr.de/ba#10> eingesehen werden.

3.3 Schnittpunkte von Geraden

Als ein weiterer Ansatz soll nun untersucht werden, ob der Drehpunkt durch die Schnittpunkte von der durch die Punktwolke definierten Geraden ermittelt werden kann. Die entsprechende Gerade soll durch die Punktwolke ermittelt werden.

Zunächst soll die Gerade anhand der am weitesten voneinander entfernten Punkte definiert werden. Diese Gerade sollte in etwa denselben Winkel haben wie der Winkel des tatsächlichen Gelenkes. Hierbei wird implizit angenommen, dass die Glieder visuell länger als sie breit sind. Die Länge wird hierbei durch den Abstand der Gelenke und die Breite entsprechend durch den größten Abstand der Punkte definiert, welche orthogonal zu dieser Längenlinie stehen.

Die am weitesten voneinander entfernten Punkte einer `PointCloud` lassen sich durch die auf dieser Klasse definierten Funktion `getMaxDist` finden⁷. In dieser Funktion wird für jeden Punkt der Punkt gesucht, welcher den größten Abstand zu diesem hat. Es wird außerdem festgehalten welches Paar den größten Abstand hatte. Diese beiden Punkte können genutzt werden, um ein Objekt der Klasse `Line` zu erstellen. Diese Klasse stellt alle Funktionalitäten bereit, welche hier für den Umgang mit Geraden benötigt werden. Eine auf diese Weise definierte Gerade wird dann in einer Liste gespeichert und die nächste Iteration folgt. Bereits ab der zweiten Iteration können dann die Schnittpunkte dieser Geraden genutzt werden, um den Drehpunkt zu bestimmen.

Die Ermittlung des Schnittpunktes zweier Geraden ist durch die `intersection` Funktion der `Line` Klasse definiert. Der Schnittpunkt zweier Geraden berechnet sich durch

⁷Der entsprechende Quellcode kann unter <https://aka.klawr.de/ba#11> eingesehen werden.

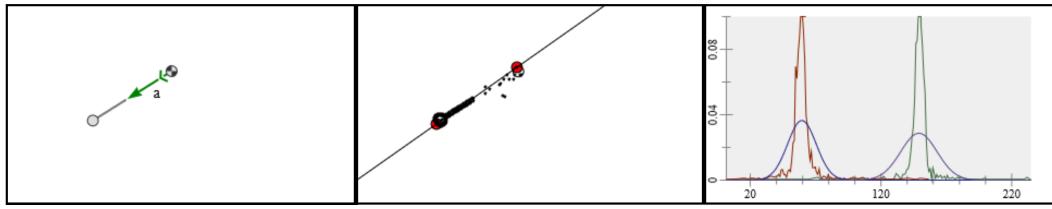


Abb. 3.3: Versuch pendel2_1.html. Links ist das Eingangsvideo. In der Mitte werden mit den roten Kreisen die aktuell am weitesten entfernten Punkt gezeigt. Der Erwartungswert aller Schnittpunkte ist als transparenter Kreis dargestellt. Im rechten Canvas kann gesehen werden, dass die Ergebnisse durchaus zuverlässig sind, jedoch einzelne Ausreißer die Gauß-Verteilung niedrig halten.

$$p_x = \frac{b_2 - b_1}{m_1 - m_2} \quad (3.1)$$

$$p_y = mp_x + b$$

Für die Berechnung von p_y können entweder die Parameter der ersten oder zweiten Gerade genommen werden.

```

1  intersection(otherLine) {
2      const x = (this.b - otherLine.b) / (otherLine.m - this.m);
3      const y = this.m * x + this.b;
4      return { x, y };
5  }

```

Listing 3.2: Definition der `intersection` Funktion, welche eine Funktion der `Line` Klasse darstellt.

3.3.1 Der Umgang mit Ungenauigkeit

Im Optimalfall würden alle gemessenen Daten exakt den gesuchten Punkten entsprechen. Allerdings ist bei den Messungen mit Streuung zu rechnen. Für die Verarbeitung der Daten wurden zwei weitere Klassen erstellt, welche die Daten verwalten, die bei der Bestimmung der gesuchten Punkte hilfreich sind. Die `Data` Klasse wird genutzt, um Daten zu verwalten. Hier werden Funktionen für die Berechnung von Erwartungswert, Varianz, Standardabweichung und die Gauß'sche Normalverteilung bereitgestellt. Außerdem bietet sie Funktionen für die Visualisierung der Daten im `g2.chart` Befehl.

Die zweite Klasse ist die `DataXY` Klasse. Sie bietet die Möglichkeit statt einzelner Werte Punkte zu verwalten. Punkte sind in diesem Fall definiert als Objekte mit

den Eigenschaften `x` und `y`. Ein `DataXY` Objekt hat die Eigenschaften `x` und `y`, welche wiederum als `Data` Objekt vorliegen. Dementsprechend bietet `DataXY` Funktionen an, um diese Daten gemeinsam auf einem `g2.chart` zeichnen zu können.

In der `Data` Klasse ist ein `data` Objekt definiert, welches die gesammelten Daten beinhaltet. Statt einer Liste wird eine Map verwendet, deren Schlüssel den Messwerten entsprechen und die dazugehörigen Werte die Anzahl darstellen, wie oft diese gemessen wurden⁸. Entsprechend wurde eine `add` Funktion für die `Data` Klasse definiert, welche wie folgt aussieht:

```

1  add(a) {
2      const r = Math.round(a);
3      if (Number.isSafeInteger(r)) {
4          this.data.set(r, this.data.get(r) + 1 || 1);
5      }
6  };

```

Listing 3.3: Definition der `add` Funktion, welche dazu genutzt wird der `Data` Klasse neue Werte hinzuzufügen.

Die Entscheidung eine Map statt einer Liste zu nutzen basiert auf den handlicheren Verarbeitungsmöglichkeiten der Daten. Es sind beispielsweise weniger Iterationen notwendig, wenn der Erwartungswert berechnet werden soll. Listen bieten jedoch die, in dieser Arbeit häufig verwendete, `reduce` Funktion. Diese wurde entsprechend für die `Data` Klasse implementiert.

```

1  reduce(callback, acc) {
2      const itr = this.data.entries();
3      acc = acc !== undefined ? acc : itr.next();
4      for (let cur = itr.next(); !cur.done; cur = itr.next()) {
5          acc = callback(acc, cur.value);
6      }
7      return acc;
8  }

```

Listing 3.4: Definition der `reduce` Funktion, um die Standardfunktion der Liste nachzubilden.

Die Vorhersage eines gesuchten Punktes wird hier stets durch den Erwartungswert aller vorangegangen Messungen bestimmt. Der Erwartungswert μ wird berechnet durch [Kla14, S. 143]

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad (3.2)$$

⁸Für mehr Informationen zu Maps siehe <https://aka.klawr.de/ba#12>.

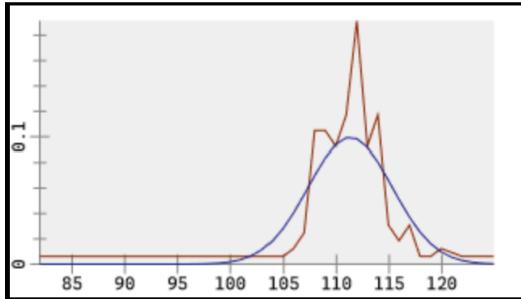


Abb. 3.4: Ein Beispiel zu einem durch g2.chart gezeichneten Graphen. Die X-Achse bezeichnet die Koordinate des gefundenen Punktes und die Y-Achse den Anteil der auf dieser Koordinate gefundenen Punkte. Hier wird Versuch momentanpol1_1.html gezeigt.

```

1  get mu() {
2      return this.reduce((pre, cur) => pre + cur[0] * cur[1], 0)
3      / this.length;
4 }
```

Listing 3.5: Definition der `mu` Funktion, welche den Erwartungswert der Daten in `Data` berechnet.

Um verschiedene Ansätze miteinander vergleichen zu können, soll außerdem noch die Standardabweichung berechnet werden. Hierfür berechnet sich zunächst die Varianz durch

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2 \quad (3.3)$$

```

1  get variance() {
2      const mu = this.mu;
3      return this.reduce((pre, cur) =>
4          pre + (((cur[0] - mu) ** 2) * cur[1]), 0)
5          / (this.length - 1);
6 }
```

Listing 3.6: Definition der `variance` Funktion, welche die Varianz der Daten in `Data` berechnet.

`mu` wird hier zwischengespeichert, damit dieser nicht für jede Iteration innerhalb des `reduce` Aufrufs erneut berechnet werden muss. Die Standardabweichung σ ist entsprechend die Quadratwurzel der Varianz.

Um die entsprechenden Ergebnisse mit der Gauß'schen Normalverteilung zu vergleichen wurde eine Funktion geschrieben, welche vom `g2.chart` genutzt werden

kann, um diese parallel zu den gesammelten Daten zu rendern. Die Funktion für die Gauß'sche Normalverteilung ist gegeben durch

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3.4)$$

```

1 gaussianDistribution(x) {
2     const mu = this.mu;
3     const variance = this.variance;
4     const nominator = Math.exp(-((x - mu) ** 2) / (2 * variance));
5     const denominator = Math.sqrt(2 * Math.PI * variance);
6     return nominator / denominator;
7 }
```

Listing 3.7: Definition der `gaussianDistribution` Funktion, welche die Normalverteilung für die gegebene Varianz und den Erwartungswert formt.

Diese Funktion kann dann vom `g2.chart` aufgerufen werden, um den entsprechenden Graphen mit Daten zu füllen. Die Implementation eines solchen Aufrufs wird in Listing 3.8 gezeigt.

```

1 getChart(limit) {
2     const sim = simulation;
3     const data = this.alignForChart(limit);
4     const fn = (i) => this.gaussianDistribution(i);
5
6     return g2().clr().view({ cartesian: true }).chart({
7         x: 20, y: 20, b: 280, h: 150,
8         funcs: [{ data }, { fn, dx: 1 },],
9         xaxis: {}, yaxis: {},
10    });
11}
```

Listing 3.8: Definition der `getChart` Funktion der `Data` Klasse.

Die `alignForChart` Funktion bereitet die `data` Map für die Darstellung durch `g2.chart` auf. Ein entsprechend gezeichneter Graph kann in Abbildung 3.4 gesehen werden.

3.3.2 Nutzung des Schwerpunktes mehrerer Datenpunkte

Idealerweise sollten alle Schnittpunkte der berechneten Geraden dieselben Koordinaten haben. Wäre dies der Fall, dann wäre die Varianz gleich null und bereits nach zwei Iterationen hätte man ein optimales Ergebnis. Da dies jedoch nicht der Fall ist, wird für jede neue Gerade jeder Schnittpunkt mit den vorangegangen Geraden

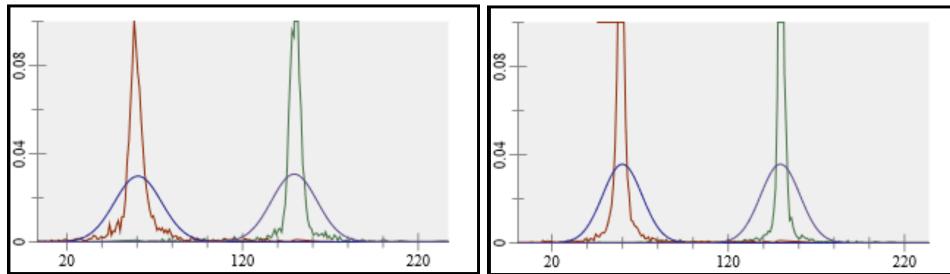


Abb. 3.5: Die Graphen zeigen jeweils die Verteilung der ermittelten Drehpunkte für die X- und Y-Achse. Orange ist jeweils der Y-Wert und Grün der X-Wert. Die darüber liegenden bläulichen Funktionen sind die entsprechenden Gauß-Verteilungen. Um die beiden Graphen besser vergleichbar zu machen wurde die Y-Achse auf den Wert 0.1 begrenzt. Tatsächlich scheint das Nutzen mehrere Punkte die Standardabweichung reduziert zu haben. Nach der Ermittlung von 100 Geraden konnte die Standardabweichung der Y-Achse von 13.4 auf 11.2 und die der X-Achse von 13.0 auf 11.2 reduziert werden.

dem `Data` Objekt übergeben. Diese Methode liefert schnell einen Erwartungswert, der nah an dem tatsächlichen Wert liegt. Dies gilt auch für vollzogene Winkel weit unter 180° .

Es zeigt sich, dass der Winkel der Geraden, die durch die am weitesten voneinander entfernten Punkte gebildet werden, eine sehr starke Streuung aufweist. Da im Abgleich zweier zeitlich aufeinander folgenden Bilder sowohl das Pendel des ersten als auch des zweiten Bildes enthalten ist. Um diesem Problem zu entgegenzuwirken sollen nun die Schwerpunkte mehrerer am weitesten auseinander liegenden Punkte ermittelt und die Prozedur mit diesen wiederholt werden⁹.

Die Standardabweichung für dieselbe Menge an Iterationen bei der Nutzung des Schwerpunktes der fünf am weitesten auseinanderliegenden Punkte reduziert sich tatsächlich leicht, wie in Abbildung 3.5 gesehen werden kann.

3.4 Bestimmung der Regressionsgeraden

Die Gerade, welche den geringsten Fehler gegenüber einer Menge an Punkten darstellt, wird als Regressionsgerade bezeichnet. Die Regressionsgerade nach der Gauß'schen Methode der kleinsten Quadrate wird durch [Pap16, S. 694]

⁹Für die Berechnung des Schwerpunktes wird davon ausgegangen, dass das Gewicht der Punkte gleich ist. Die Berechnung ist also äquivalent zu der des Erwartungswertes.

$$\begin{aligned}\bar{x} &= \sum_{i=1}^n x_i, \quad \bar{y} = \sum_{i=1}^n y_i \\ m &= \left(\sum_{i=1}^n (x_i y_i) - n \bar{x} \bar{y} \right) \left(\sum_{i=1}^n x_i^2 - n \bar{x}^2 \right)^{-1} \\ b &= \bar{y} - m \bar{x}\end{aligned}\tag{3.5}$$

berechnet. Diese Berechnung unterstellt jedoch einen vernachlässigbar kleinen Fehler auf der X-Achse. Dies resultiert hier in einer horizontalen Linie, wenn sie vertikal sein sollte. Die Berechnung durch die Methode der kleinsten Quadrate hat jedoch ein anderes Problem aufgezeigt. Wenn zwei Bilder die eine Drehung darstellen miteinander verglichen werden, indem jene Pixel behalten werden, für die eine Änderung zu erkennen ist, dann werden typischerweise mehr Änderungen erkannt, je weiter man sich radial von Drehpunkt entfernt. Diese Beobachtung ist nicht überraschend, denn die absolute Geschwindigkeit eines Punktes nimmt mit Abstand zum Drehpunkt zu. Diese Tatsache sorgt jedoch dafür, dass die Dichte der Punktfolke in einigen Regionen höher ist, was die Berechnung der Regressionsgerade beeinflusst. Um dieses Problem zu umgehen, soll die Dichte der Punktfolken limitiert werden. Diese Überlegung ist inspiriert durch die in der Bilderkennung verwendeten Methode der Non-Max Suppression [Gér19, S. 486]. Diese wird verwendet, um durch Bilderkennung klassifizierte Objekte zu filtern, die direkt nebeneinander liegen. Berechnet wird das Objekt dann unter Nutzung eines Konfidenzwertes. Da die Punktfolke jedoch keine Konfidenzwerte aufweist, kann dieser Ansatz stark vereinfacht werden. Hierfür wurde in der `PointCloud` Klasse die `removeOverlaps` Funktion definiert, welche für ein `PointCloud` Objekt ein neues gefiltertes Objekt zurückgibt.

```

1  removeOverlaps(dist = 5) {
2      let copy = [...this.points];
3      const survivor = [];
4      while (copy.length) {
5          const pt = copy.pop();
6          survivor.push(pt);
7          copy = copy.filter(
8              (rec) =>
9                  Math.abs(rec.x - pt.x) >= dist ||
10                 Math.abs(rec.y - pt.y) >= dist
11         );
12     }
13     return new PointCloud(survivor);
14 }
```

Listing 3.9: Definition der `removeOverlaps` Funktion der `PointCloud` Klasse.

Um die Berechnung der Regressionsgerade für diesen Anwendungsfall zu korrigieren, wird auf eine andere Methode zur Berechnung dieser zurückgegriffen.

3.4.1 Orthogonale Regression

Während die vorher angesprochene Regressionsgerade den Abstand der Punkte in Y-Richtung zu minimieren versucht, ist der nachfolgende Ansatz dazu gedacht den Fehler der Punkte orthogonal zur gesuchten Geraden zu minimieren [Jür20, S. 140].

$$m = \frac{\sigma_y^2 - \sigma_x^2 + \sqrt{(\sigma_y^2 - \sigma_x^2)^2 + 4\sigma_{xy}^2}}{2\sigma_{xy}} \quad (3.6)$$

$$b = \bar{y} - m\bar{x}$$

Diese Gleichung nutzt die Varianz der einzelnen Datensammlungen, um damit die entsprechende Steigung der Geraden zu ermitteln. Die Berechnung dieser Parameter kann aus der `Data` Klasse wiederverwendet werden, was in Listing 3.6 gezeigt wird. Hierfür muss also nur noch die Ermittlung der Kovarianz σ_{xy}^2 implementiert werden¹⁰. Diese kann passend der `DataXY` Klasse hinzugefügt werden, welche für die Tests die entsprechenden Punkte bereithält.

```

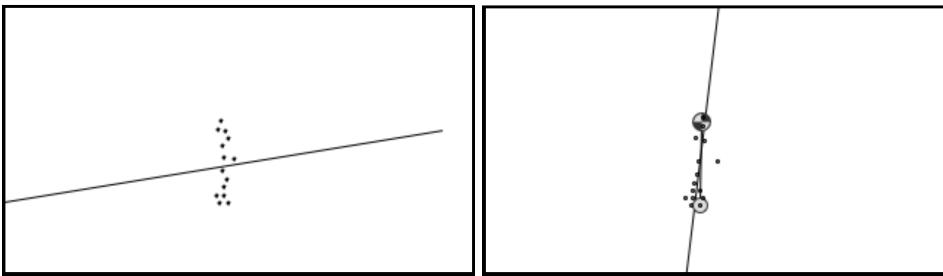
1  get() covariance() {
2      const xmu = this.x.mu;
3      const ymu = this.y.mu;
4
5      return this.pts.reduce((pre, cur) =>
6          pre + (cur.x - xmu)(cur.y - ymu), 0)
7          / (this.pts.length - 1);
8 }
```

Listing 3.10: Implementation der `covariance` Funktion in der `DataXY` Klasse.

Hier werden die entsprechenden Erwartungswerte analog zu Listing 3.6 festgehalten, damit diese nicht öfters ermittelt werden müssen.

Die Erzeugung der Regressionsgerade wird auf der `Line` Klasse definiert werden. Die Implementation kann in Listing 3.11 nachvollzogen werden.

¹⁰Die Varianz nutzt in ihrer Berechnung jeweils die Anzahl der Datenpunkte. Diese könnte aus der Gleichung gekürzt werden, was in einer abschließenden Implementation berücksichtigt werden sollte, um ein paar Rechenschritte zu sparen.



(a) Versuch pendel2_4.html

(b) Versuch pendel2_5.html

Abb. 3.6: Links wird die durch die Methode der kleinsten Quadrate ermittelte Regressionsgerade gezeigt. Diese geht von einem vernachlässigbar kleinem Fehler auf der X-Achse aus, weshalb die Gerade horizontal gezeichnet wird. Rechts wird die für den Anwendungsfall passendere orthogonale Regression genutzt. Im rechten Bild wird zudem wieder der Drehpunkt und ein vermuteter Endpunkt des Pendels gezeichnet, was im linken Bild keinen Sinn hat.

```

1 static fromRegressionLine(pts, g) {
2     const data = new DataXY(pts);
3
4     const sy = data.y.variance;
5     const sx = data.x.variance;
6     const sxy = data.covariance;
7
8     const m = (sy - sx + Math.sqrt((sy - sx) ** 2 + 4 * sxy**2))
9         / (2 * sxy);
10    const b = data.y.mu - m * data.x.mu;
11
12    return new Line({m, b});
13 }
```

Listing 3.11: Erstellung von Line Objekten durch die orthogonale Regressionsgerade für Datenpunkte

Die so ermittelte Regressionsgerade ist nun für jede Steigung der Punktwolke definiert, wie in Abbildung 3.6b gesehen werden kann.

3.5 Vergleich des Verlaufes von Randpunkten

Die letzte Versuchsreihe, die gemacht werden soll, ist der Abgleich zweier durch die Punktwolke entstandener Pfade von Punkten. Die ausgewählten Punkte bestehen jeweils aus dem Punktpaar mit größter Distanz pro Abgleich. Hierfür wird zur Ermittlung zunächst dieselbe Methodik verwendet wie in Kapitel 3.3.

Allerdings muss hier sichergestellt werden, dass jeweils der korrekte Nachfolger für einen Pfad der dazugehörigen Liste hinzugefügt wird. Der bisherige Ansatz beinhaltet keine Information über die Zugehörigkeit von einzelnen Punkten in aufeinanderfolgenden Bildabgleichen. Um den Verlauf solcher Messdaten über Zeit nutzbar zu machen, wurde die `Groups` Klasse erstellt. Entsprechend wird bei jeder Iteration der Abstand des Punktpaares mit dem letzten Eintrag der entsprechenden Listen verglichen und so gruppiert. Zuständig für diesen Abgleich ist die `addPoints` Funktion.

Um mit dieser Information den Drehpunkt des Pendels zu ermitteln, reicht es dann aus die Länge des gesamten Pfades zu messen, da der Gesamtweg des Mittelpunktes erheblich kürzer sein sollte als der des Endpunktes. Diese Überlegung entspricht nicht der Beobachtung. Das liegt vor allem daran, dass die in Kapitel 3.4 gemachte Beobachtung, dass je weiter der Abgleich zweier Bilder vom tatsächlichen Drehpunkt entfernt ist die absolute Geschwindigkeit höher und die Dichte der Punktewolke entsprechend in solchen Regionen höher ist. Das hat zur Folge, dass die Streuung der über die Iteration unterschiedlichen Punktewolken erheblich zur Länge des Pfades der Bewegung des eigentlichen Drehpunktes beiträgt. Die Nutzung mehrerer maximal distanzierter Punkte, wie es in Kapitel 3.3.2 gemacht wurde, scheint diesen Fehler zu verstärken. Minimieren lässt sich dieser, indem der kleinste umfassende Kreis um die beiden gesammelten Gruppen gebildet wird. Derjenige mit dem kleineren Radius stellt dann den Mittelpunkt dar.

3.6 Auswertung der Beobachtungen

Die hier besprochenen Ansätze haben viele Erkenntnisse gebracht, welche Informationen aus Videosequenzen gewonnen werden können. Es wurden für die Struktur der Versuche vier zentrale Klassen implementiert, welche im Verlauf dieser Arbeit eine zentrale Rolle spielen werden¹¹.

Die `PointCloud` Klasse ordnet den Umgang mit Punktewolken. Bisher wird diese Klasse vor allem genutzt, um die Regressionsgerade zu bilden. Es werden in den nachfolgenden Kapiteln weitere Funktionen dieser Klasse genutzt, um beispielsweise die Punkte der Punktewolken einzelnen Gliedern eines Mechanismus zuzuordnen.

Die `Line` Klasse beinhaltet alle Methoden, welche auf der Basis von Geraden gemacht werden können. Hier werden Funktionen für die Berechnung der Schnittpunkte

¹¹Es existieren vereinzelt Klassen, welche in Relation zu diesen vier Klassen stehen, wie zum Beispiel `DataXY` zu `Data`.

definiert sowie die Berechnung von Winkelhalbierenden zweier Geraden. Auch die Ermittlung der Orthogonalen einer Geraden und andere Hilfsfunktionen werden hier definiert.

Die `Group` Klasse wurde in Kapitel 3.5 kurz angesprochen, findet allerdings vor allem in Kapitel 6 Anwendung. Die Zuordnung von Punkten im Verlauf über mehrere Iterationen hat sich als ein notwendiges Werkzeug für die Berechnung von Relativpolen herausgestellt.

Als besonders wichtig hat sich die statistische Analyse der Messungen gezeigt. Die `Data` Klasse beinhaltet alle Funktionen, welche zur Berechnung dieser notwendig ist. Unter anderem werden deren Methoden auch genutzt, wenn sie für die Berechnung anderer Funktionen hilfreich sind, wie das in Listing 3.11 der Fall ist. Der Erwartungswert ist der intuitive Ansatz für die Ermittlung der Koordinaten gesuchter Punkte, wenn diese eine Streuung aufweisen. Die Standardabweichung ermöglicht es die unterschiedlichen Ansätze qualitativ zu vergleichen.

Jede der hier genannten Klassen hat zudem entsprechende `draw` Funktionen, welche es ermöglichen die durch diese Klassen verwalteten Objekte visuell darzustellen, um so die Ergebnisse zumeist im zweiten Canvas der Versuchsreihen zu rendern.

Im Folgenden soll noch eine weitere Möglichkeit untersucht werden, um nutzbare Informationen aus Veränderungen in Bildern zu ermitteln nämlich über die Berechnung des optischen Flusses.

Ermittlung des optischen Flusses

„Ich glaube, es ist verlockend, wenn das einzige Werkzeug, das man hat, ein Hammer ist, alles zu behandeln, als ob es ein Nagel wäre.“

— Abraham Maslow
US-amerikanischer Psychologe

Die wahrgenommene Bewegung von Objekten innerhalb zweier Bilder wird als optischer Fluss (zu engl. Optical Flow) bezeichnet [Rus10, S. 939]. Der optische Fluss kann durch ein Vektorfeld interpretiert werden, wobei die Vektoren jeweils die relative Bewegung des betrachteten Punktes darstellen. Die Bewegung eines Pixels innerhalb zweier Bilder lässt sich durch

$$I_n(x, y) = I_{n+1}(x + \Delta x, y + \Delta y) \quad (4.1)$$

darstellen. Hier stellt n die Iteration, x und y jeweils die Koordinaten und t die Helligkeit des Objektes dar. Durch die Berechnung der Bewegung aller Pixel lässt sich so ein Vektorfeld erstellen.



Abb. 4.1: Hier werden zwei Bilder einer Bildsequenz gezeigt. Das dritte Bild zeigt das Vektorfeld an, welches die Bewegung der einzelnen Pixel im nächsten Bild vermutet [Rus10, S. 941; BBM09].

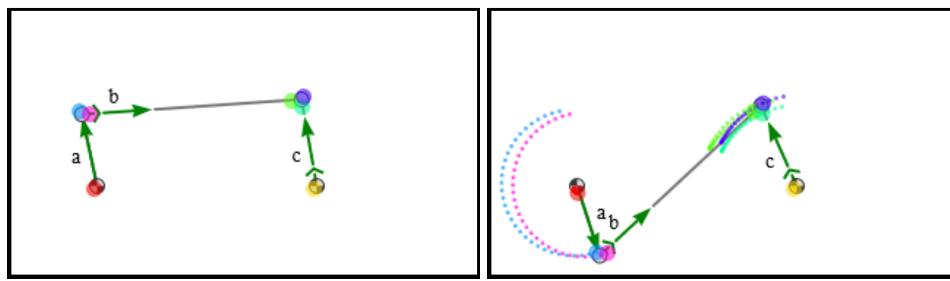
4.1 Optischer Fluss nach Lucas-Kanade

Eine nutzbare Implementation des optischen Flusses wurde von Bruce Lucas und Takeo Kanade bereits 1981 vorgeschlagen [LK81]. Der vorgeschlagene Ansatz trifft jedoch mehrere Annahmen. Die erste Annahme ist, dass die Bewegung nicht sehr groß ist. Zu schnelle Bewegungen können durch diesen Ansatz wahrscheinlich nicht oder nur schlecht erkannt werden. Eine weitere Annahme ist, dass die zu untersuchende Bildsequenz in Graustufen immernoch nutzbar ist. Es werden in dieser Methode die Helligkeitswerte der Pixel untersucht. Eine Bewegung welche keine Änderung der Helligkeitswerte verursacht, kann also offensichtlich nicht erkannt werden.

Die Bewegung sollte durch einen Abgleich mit benachbarten Pixeln gefunden werden, indem der entsprechende durch eine Bewegung erzeugte Fehler minimiert wird. Definiert man das erste Bild für die entsprechenden Pixelwerte als eine Funktion $F(x)$ und die Pixelwerte des zweiten Bildes entsprechend als $G(x)$, so wird also jener Vektor gesucht, welcher den Fehler zwischen diesen beiden Funktionen minimiert. Beschreiben diese Funktionen stattdessen Regionen auf entsprechenden Bildern, so können die Vektoren gesucht werden, welche für die Bewegung dieser Regionen innerhalb der Bilder die geringsten Fehler aufweisen. Auf diese Weise kann versucht werden die Bewegung einzelner Objekte zu verfolgen.

Zunächst ist die Größe der Regionen unbekannt, deshalb wird ein pyramidischer Ansatz genutzt. Pyramidisch bedeutet, dass um Regionen von Interesse Quadrate unterschiedlicher Größe definiert werden, für welche der Fehler entsprechend berechnet wird. Diese Fehler können dann verglichen werden und das passendste Quadrat wird genutzt, um die Verschiebung der Region innerhalb des Bildes zu beschreiben. Anhand dieses Ansatzes kann die ungefähre Größe des sich bewegenden Objektes untersucht werden. Eine Implementation dieses Ansatzes wird von Jean-Yves Bouget beschrieben [Bou00].

Was für eine Nutzung nun noch fehlt, ist die Festlegung der entsprechenden Regionen von Interesse. Hierfür wird ein Algorithmus genutzt, welcher in einem Bild interessante Punkte anhand von Kantenerkennung identifiziert. Entwickelt wurde dieser von J. Shi und C. Tomasi [ST94] und wird im Nachfolgenden genutzt, um die Menge der zu verfolgenden Punkte zu bestimmen.



- (a) Versuch `opticalflow2_1.html`. Hier werden die Punkte gezeigt, welche durch die `cv.goodFeaturesToTrack` Funktion bei einem `mec2` Modell ermittelt werden.
- (b) Versuch `opticalflow2_2.html`. Durch die `cv.calcOpticalFlowPyrLK` Funktion werden die ermittelten Punkte über die Animation hinweg verfolgt.

Abb. 4.2

4.1.1 Implementation des Lucas-Kanade Algorithmus

Um diesen Algorithmus in die nun bestehende Struktur zu implementieren, wird für die `Group` Klasse eine weitere Eigenschaft definiert. Diese Eigenschaft verweist auf ein Objekt der `LucasKanade` Klasse. Objekte der `LucasKanade` Klasse enthalten wiederum alle Parameter, welche für die Nutzung der Algorithmen notwendig sind.

Die Funktionen zur Bestimmung interessanter Punkte und deren Verfolgung wird von `opencv.js` bezogen [@Ope21]. `opencv`¹ ist eine Sammlung von Funktionen im Bereich der Bildverarbeitung und des maschinellen Sehens. Implementiert ist `opencv` in C++. Für eine Vielzahl von anderen Programmiersprachen wie zum Beispiel Python, Java und JavaScript gibt es jedoch entsprechende Schnittstellen.

Für JavaScript wird das globale `cv` Objekt genutzt, um auf die Funktionen zuzugreifen. Für die Bestimmung von vermeintlich sinnvollen Regionen zur Verfolgung wird die `cv.goodFeaturesToTrack` Funktion verwendet, welche den Shi-Tomasi Algorithmus implementiert. Die hier übergebenen Parameter sind ein in Graustufen konvertiertes Bild, eine `cv.Mat` Objekt, in welchem die Ergebnisse gespeichert werden, die maximale Anzahl der Punkte, welche zur Verfolgung gesucht werden, die minimale Qualität einer Kante² und der minimale euklidische Abstand der Punkte zueinander. Des Weiteren können hier spezielle Regionen von Interesse übergeben werden und die durchschnittliche Größe der Blöcke, welche bei der Berechnung genutzt werden.

Nach der Bestimmung der zu verfolgenden Punkte wird versucht diese durch `cv.calcOpticalFlowPyrLK` zu verfolgen. Die `calcOpticalFlowPyrLK` Funktion benötigt als

¹ `opencv` ist eine Abkürzung für Open Computer Vision.

² Die Qualität einer Kante wird hier ins Verhältnis zur Kante mit bester Qualität gestellt. Die Qualität einer hier genutzten Kante berechnet sich durch `cv.cornerMinEigenVal`, worauf hier jedoch nicht weiter eingegangen wird. Mehr Information kann unter <https://aka.klawr.de/ba#13> gefunden werden.

Parameter zunächst die beiden Bilder in zeitlicher Abfolge. Der dritte Parameter beinhaltet die Punkte, welche verfolgt werden sollen und der vierte Parameter ist das `cv.Mat` Objekt, welcher die berechneten Punkte enthalten wird. Alles Weitere sind Parameter zum Einstellen des Algorithmus wie etwa der Status, einem Objekt zur Fehlerbehandlung, der Größe der Fenster für den pyramidischen Ablauf und die Anzahl der Iterationen von den im Verlauf der Berechnung größer werdenden Regionen. Abschließend wird dem Algorithmus ein `criteria` Parameter übergeben, welcher eine Abbruchbedingung für die iterative Suche darstellt. Ein Abbruch findet dann statt, wenn eine maximale Anzahl von Iterationen erreicht ist oder die Bewegung unterhalb eines Schwellenwertes vermutet wird.

Die in der `LucasKanade` Klasse definierte `step` Funktion ist in Listing 4.1 abgebildet.

```

1  step(frame) {
2      if (this.very_first) {
3          this.very_first = false;
4          return;
5      }
6      if (!this.first_indicator) {
7          this.goodFeaturesToTrack(frame);
8          this.first_indicator = true;
9      }
10
11     cv.cvtColor(frame, this.frameGray, cv.COLOR_RGBA2GRAY);
12     this.calcOpticalFlowPyrLK();
13     const points = this.getPoints();
14
15     this.frameGray.copyTo(this.oldGray);
16     this.p0 = new cv.Mat(points.length, 1, cv.CV_32FC2);
17     for (let i = 0; i < points.length; i++) {
18         this.p0.data32F[i * 2] = points[i].x;
19         this.p0.data32F[i * 2 + 1] = points[i].y;
20     }
21
22     return points;
23 }
```

Listing 4.1: Implementation der `step` Funktion der `LucasKanade` Klasse.

Es wird mittels der `first_indicator` Eigenschaft geprüft, ob es sich um den ersten Aufruf mit einem Bild handelt. Entsprechend muss noch die `very_first` Variable genutzt werden, um den ersten Aufruf zu filtern. `goodFeaturesToTrack` wird nur einmal am Anfang ausgeführt, damit die Reihenfolge der gefundenen Punkte konsistent bleibt. Außerdem ist nicht sichergestellt, ob die vorgeschlagenen Regionen bei Veränderungen im Bild dieselben bleiben. Die gefundenen Punkte werden im `LucasKanade`

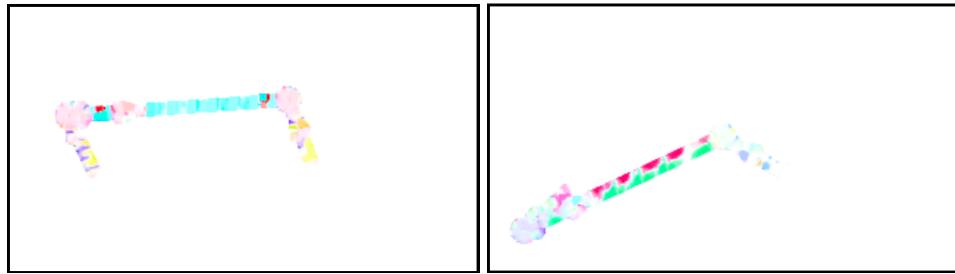


Abb. 4.3: Versuch `opticalflow1_1.html`. Hier wird der Versuch mit dem optischen Fluss nach Farneback versucht. Die Farben stellen die vorausgesagte Richtung der entsprechenden Pixel dar. Auf dem ersten Bild scheinen diese noch relativ eindeutig zu sein. Es zeigt sich jedoch, dass vor allem für etwas schnellere Bewegungen diese Berechnungen sehr ungenau werden und starke Streuungen zeigen. Eine sinnvolle Nutzung bedarf entsprechend tieferer Analyse.

Objekt entsprechend gespeichert. Nachdem das Bild in Graustufen übersetzt wurde, wird es in der `calcOpticalFlowPyrLK` Funktion genutzt, um die entsprechend neuen Koordinaten der Punkte zu ermitteln. Diese Punkte werden dann in einer `points` Liste gespeichert, welche die zuletzt ermittelten Koordinaten enthält. Daraufhin wird das `oldGray` Bild überschrieben, damit es bei der nächsten Iteration entsprechend als erstes Bild dienen kann. Ebenso wird `p0` für die nächste Iteration auf die zuletzt ermittelten Koordinaten gesetzt.

Der Rückgabewert dieser Funktion ist die `points` Liste. Aufgerufen wird diese Funktion durch die `lucasKanade` Funktion des `group` Objektes. Dieses ermittelt die Bildinformation von `cnv1` durch `cv.imread`, welches ein `cv.Mat` Objekt zurückgibt. Daraufhin wird die in Kapitel 3.5 bereits angedeutete Funktion `addPoints` genutzt, um die nun für die nächste Iteration ermittelten Koordinaten der Punkte der Liste an Punkten hinzuzufügen. Diese Punkte haben augenscheinlich eine sehr viel geringere Varianz als die Ansätze aus Kapitel 3.

4.2 Andere Methoden

4.2.1 Optischer Fluss nach Farneback

Bei der Ermittlung des optischen Flusses wird zwischen dichtem und spärlichem optischen Fluss unterschieden. Während der spärliche optische Fluss einzelne Regionen von Interesse im Bild verfolgt und entsprechend dessen Bewegung nachvollzieht, ist der Anspruch von Algorithmen zur Berechnung vom dichten optischen Fluss



Abb. 4.4: Die ersten beiden Bilder sind ein Auszug aus dem Flying Chair Dataset. Das zweite Bild versetzt die Stühle ein wenig, wobei die Bewegung, welche in der Kontrolle des Programmierers liegt entsprechend als Wert für das Training festgehalten wird. Im dritten Bild sieht man entsprechend die Vektoren, welche als wahre Werte gehalten werden. Die Farben zeigen hierbei die Richtung des Vektors und die Sättigung der Farbe gibt den Betrag an [Fis+15].

die Berechnung der Bewegung jedes einzelnen Pixels. Während ich auf die Implementierung nicht weiter eingehen möchte, da dieser Ansatz wegen augenscheinlich zu großen Varianzen nicht weiter verfolgt wird, verdient dieser dennoch eine Erwähnung. Die Implementation wurde nicht sonderlich in den Text integriert, der entsprechende Test kann jedoch unter `src/opticalflow/opticalflow1_1.html` eingesehen werden.

Für ein sich um einen festen Punkt drehendes Glied wäre zu erwarten, dass alle Vektoren zur Bestimmung der Bewegung denselben Einheitsvektor haben. Der entsprechende Betrag des Vektors sollte linear mit Abstand zum Drehpunkt zunehmen. Diese Information sollte dann genutzt werden können, um die Pole der ebenen Bewegung zu ermitteln.

Des Weiteren sollten die Glieder durch den Verlauf der gemessenen Geschwindigkeiten leicht unterscheidbar sein. Da jedoch die Ungenauigkeiten hier zu groß zu sein scheinen, wäre die Implementierung entsprechend komplex und wird hier nicht weiter Gegenstand sein.

4.2.2 Ermittlung von Bewegung durch maschinelles Lernen

Eine weitere Erwähnung verdienen Ansätze zur Ermittlung des optischen Flusses durch maschinellen Lernens. Sie versprechen eine höhere Genauigkeit bei vergleichbarer Performanz. Hierzu gehören Ansätze zur Segmentierung einzelner Objekte durch die Verfolgung der Punkttrajektorien [OMB14; KAB15]. Diese Segmentierungen könnten dann im Einzelnen analysiert werden, was die Problemstellung von Kapitel 6 darstellen wird.



Abb. 4.5: Vergleich der Genauigkeit von FlowNet gegenüber FlowNet 2.0. Die hier sichtbar verbesserte Genauigkeit ist laut den Autoren vier mal höher als beim Vorgänger [Ilg+16].

Auch für die Ermittlung des Vektorfeldes werden Neurale Netze entwickelt, welche eine höhere Genauigkeit und weniger Streuung der Ergebnisse versprechen. FlowNet [Fis+15] ist ein *Convolutional Neural Network*³, welches durch automatisiert erstellte Daten trainiert wurde. Es wurden hierfür beliebige Hintergründe verwendet, auf denen dann Bilder von Stühlen platziert wurden. Diese Stühle wurden dann für das Training entsprechend ein wenig versetzt, damit das neurale Netz mit diesen vorgegebenen *Ground Truth* Werten trainiert werden konnte. Es muss hierbei berücksichtigt werden, dass das neurale Netz nicht durch die Objekte, sondern dessen Bewegung trainiert wird.

Als eine Entwicklung von FlowNet wurde FlowNet 2.0 [Ilg+16] entworfen. FlowNet 2.0 hat einige Verbesserungen im Umgang mit den Trainingsdaten eingeführt. Diese Entwicklungen basieren auf den gefundenen Mängeln von FlowNet, welcher auf Daten der echten Welt keine Verbesserung zur bis dahin genutzten Methoden aufweist⁴. Zum einen wird hier ein größerer Fokus auf kleinere Änderungen in der Bewegung gelegt, was zu einer höheren Genauigkeit bei echten Bildern führt. Es zeigte sich außerdem, dass der Zeitplan, mit welchem die Trainingsdaten an das Netzwerk geliefert werden, Einfluss auf dessen Generalisierbarkeit hat.

4.3 Betrachtung der Ergebnisse

Die Bestimmung des optischen Flusses war zunächst die zugrundeliegende Überlegung zur Lösung dieser Arbeit. Tatsächlich kam die ursprüngliche Inspiration zu dieser Arbeit durch Abbildung 4.1. Es hat sich jedoch schnell gezeigt, dass es sehr schwierig sein könnte die Ungenauigkeiten der Vektorfelder unter Kontrolle zu bringen. Ansätze des maschinellen Lernens sind vielversprechend. Die Entwicklung dieser zeigt, dass die in dieser Arbeit untersuchten Algorithmen durch diese

³Zu deutsch in etwa *Faltendes neurales Netzwerk*.

⁴FlowNet hat allerdings gezeigt, dass maschinelles Lernen dieses Problem bearbeiten kann.

verbessert und schlussendlich die Genauigkeit erhöht werden kann. Zur Zeit der Bearbeitung ist jedoch der Mehraufwand einer Implementierung solcher Ansätze zu hoch. Um den Rahmen dieser Arbeit nicht zu übersteigen, finden diese hier daher keine Anwendung.

Die Implementation des Algorithmus nach Lucas-Kanade scheint jedoch eine stabile Methode zu sein, die Bewegung einzelner Punkte zu verfolgen. Dieser findet vor allem im nachfolgenden Kapitel Anwendung.

Ermittlung des Momentanpols

„ Die allgemeine ebene Starrkörperbewegung kann augenblicklich als reine Drehung um einen ausgezeichneten Punkt – den Momentanpol oder Geschwindigkeitspol – aufgefasst werden

— Stefan Gössner
Mechanismentechnik

In Kapitel 3 wurde die Ermittlung eines Drehpunktes von einem rotierendem Glied beschrieben. Diese Bewegung stellt einen häufigen Fall in Mechanismen dar, jedoch ist der feste Drehpunkt ein Spezialfall für die Bestimmung des Momentanpols einer Gliedebene. Der Momentanpol ist definiert als jener Punkt, welcher sich momentan bei einer Bewegung einer Gliedebene nicht bewegt. Das kann auch interpretiert werden, als der Punkt um den sich ein Element in dieser Momentbetrachtung dreht.

Es ist zu beachten, dass der Momentanpol ein virtueller Punkt und damit nicht an den untersuchten Körper gebunden ist. Tatsächlich kann entsprechend die Bewegung einer starren Gliedebene als ein Vektorfeld extrapoliert werden. Der Momentanpol wird in diesem dann als jener Vektor mit einem Betrag von 0 identifiziert. Wären die Methoden der Berechnung des optischen Flusses also mit nur sehr kleinem Fehler behaftet, würde sich so möglicherweise auch direkt der Momentanpol ermitteln lassen.

Die Position des Momentanpols kann konstant sein, wodurch dann der Drehpunkt definiert wird, welcher dann auch als Absolutpol bezeichnet wird. Ein translatives Element, dessen Bewegung durch die Abwesenheit einer Rotation definiert ist

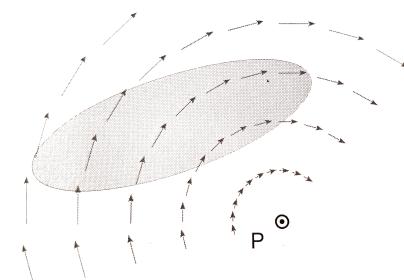


Abb. 5.1: Diese Betrachtung eines starren Körpers in der Ebene soll nahelegen, dass sich die Bewegung auch als Vektorfeld interpretieren lässt [Gös16].

dreht sich in dem Sinne um einen Momentanpol, welcher sich orthogonal zur Bewegungsrichtung in unendlicher Distanz befindet. Bei einer Parallelkurbel, bei welcher die Koppel keine Drehung aufweist, würde der Momentanpol ebenfalls stets im Unendlichen liegen.

Eine Möglichkeit zur Ermittlung des Momentanpols ist die Ermittlung der Bewegung durch den ersten Satz von Euler[Gös16]. Die Bewegung eines Körpers berechnet sich nach diesem durch

$$\vec{v}_A = \vec{v}_P + \omega \tilde{\vec{r}}_{PA} \quad (5.1)$$

Hier ist \vec{v}_A die momentane Geschwindigkeit eines Körperpunktes A . \vec{v}_P ist die Geschwindigkeit des Körperpunktes P . $\omega \tilde{\vec{r}}_{PA}$ entspricht der Geschwindigkeit des Punktes P nach A durch

$$\vec{v}_{AB} = \dot{\vec{r}}_{AB} = \omega \tilde{\vec{r}}_{AB}. \quad (5.2)$$

Wird für Gleichung 5.1 der Punkt P nun als der Momentanpol festgelegt, kann die Geschwindigkeit $\vec{P} = 0$ gesetzt werden, so dass sich Gleichung 5.1 zu

$$\vec{v}_A = \omega \tilde{\vec{r}}_{PA} \quad (5.3)$$

vereinfacht. Diese kann dann nach r_{AP} umgestellt werden,

$$\vec{r}_{AP} = \frac{\tilde{\vec{v}}_A}{\omega} \quad (5.4)$$

so dass der Momentanpol P über die Beziehung $\vec{r}_P = \vec{r}_A + \vec{r}_{AP}$ ermittelt werden kann.

Diese Formel soll nun angewendet werden, um den Momentanpol von Elementen zu bestimmen deren Bewegung in Videosequenzen nachvollzogen wird.

5.1 Implementation

Die `group` Klasse hat eine entsprechende `momentanpol` Funktion erhalten. Diese implementiert hier zunächst eine Berechnung des Erwartungswertes für einen Punkt.

```

1  momentanpol(threshold = 1) {
2      /**
3      * Ermittlung von p1, p2, m1, m2
4      */
5
6      const v = { x: p1.x - p2.x, y: p1.y - p2.y };
7      let dw = ((w1 - w2) + Math.PI) % Math.PI;
8      dw = dw > Math.PI / 2 ? dw - Math.PI : dw;
9
10     return {
11         x: (p1.x + p2.x) / 2 - v.y / dw,
12         y: (p1.y + p2.y) / 2 + v.x / dw,
13     }
14 }

```

Listing 5.1: Definition der `momentanpol` Funktion, welche in der `group` Klasse definiert ist
Die Ermittlung der Punkte `p1` und `p2` sowie der Winkel `w1` und `w2` wurde hier aus Platzgründen weggelassen.

`p1` und `p2` sind jeweils der letzte Punkt beziehungsweise jener der vor `threshold` Iterationen hinzugefügt wurde. Analog dazu sind `w1` und `w2` die Winkel der Geraden, welche als letztes und vor `threshold` Iterationen hinzugefügt wurden. Hierfür hat die `Group` Klasse die Eigenschaft `lines` bekommen, welche in etwa dieselbe Aufgabe wie die `pts` Eigenschaft erfüllen soll. Diese Liste wird entsprechend von den Tests analog zu `pts` befüllt. Es muss hier noch garantiert werden, dass in jedem Fall ein Winkel kleiner π eingesetzt wird und anschließend wird dieser noch auf einen Wert gesetzt dessen Betrag in jedem Fall kleiner $\pi/2$ ist. Damit wird sichergestellt, dass wenn durch den eingeschlossenen Winkel geteilt wird in jedem Fall der kleinere Betrag genommen wird. Ein ansonsten daraus resultierender Fehler würde sich bei der Berechnung des sich translativ bewegenden Gliedes zeigen.

5.2 Das drehende Rad

Als erster Test soll der Momentanpol eines sich drehenden Rades bestimmt werden. Von diesem Rad soll eine Speiche sichtbar sein, welche es erlaubt die Drehung zu erkennen. Interessant ist die Bestimmung des Momentanpols eines Rades, weil dieses als erster der bisher gemachten Tests keinen Absolutpol darstellt. Bekannt ist, dass die Drehung eines Rades im Wälzpunkt liegt. Ansonsten würde dieses nicht schlupffrei rollen¹. Es wird also erwartet, dass die über die Zeit ermittelten

¹Bei einem rutschendem Rad würde sich der Momentanpol entsprechend im Mittelpunkt des Rades befinden.

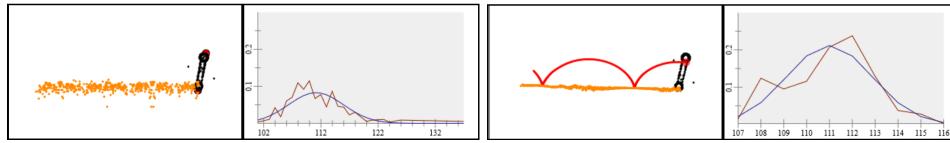
Momentanpole eine Linie bilden, welche sich genau auf der Ebene befindet, auf der das Rad abrollt.

Für die Animation des Rades wurde eine g2 Animation erstellt, da sich dieses Modell so leichter darstellen lässt als mit einem `mec2` Modell. Das Rad wird durch eine Linie und einen Kreis definiert. Das Rollen wird durch den `g2.use` Befehl simuliert, indem die Rotation durch eine Laufvariable bestimmt wird und die Position in X-Richtung durch $x = x_0 + r * i * \pi$, wobei x_0 entsprechend der Startwert ist, r der Radius des Kreises und i die Laufvariable, welche hier den Radianen bestimmt. Die Animation wird dann an `simulation.g` übergeben. Da kein `mec2` Modell definiert ist, werden die entsprechenden Befehle ignoriert (wie in Kapitel 2.2 beschrieben).

Zur Ermittlung des Momentanpols ist es nun notwendig die absolute Geschwindigkeit eines auf dem Glied befindlichen Punktes sowie die Rotationsgeschwindigkeit der Gliedebene zu ermitteln. Die absolute Geschwindigkeit wird zunächst ermittelt, indem wie in Kapitel 3.5 beschrieben die Bewegung der äußersten Randpunkte gemessen wird. Unter der Voraussetzung, dass diese in etwa denselben Punkt auf dem Glied bezeichnen, sollten diese eine ausreichende Approximation darstellen. Die `Group` Klasse speichert hierbei den Pfad der gemessenen Punkte und ordnet diese entsprechend zu. Durch den Gradienten zweier Punkte in dieser Liste wird so eine Geschwindigkeit berechnet. Die Rotationsgeschwindigkeit wird ähnlich gemessen, wie in Listing 5.1 zu sehen ist.

Wird der Gradient über dieselbe Anzahl an Iterationen für die Winkelgeschwindigkeit wie für die absolute Geschwindigkeit berechnet, so kann Gleichung 5.4 genutzt werden, um die Position des Momentanpols zu ermitteln. Dieser stellt hier wieder eine Schätzung dar, da die Eingangsparameter mit einer Ungenauigkeit behaftet sind. Des Weiteren ist anzumerken, dass die Anzahl der Iterationen zwischen den beiden zu vergleichenden Bildern im vorhinein festgelegt werden muss. Ist diese Zahl zu klein oder zu groß, so folgt daraus eine größere Standardabweichung. Diese optimale Anzahl ist jedoch je nach Geschwindigkeit des sich bewegenden Gliedes unterschiedlich, so dass zur automatisierten Ermittlung weitere Untersuchung notwendig sind. Für den Versuch des Rades wird hier lediglich die Y-Koordinate betrachtet. Tatsächlich befindet sich der Erwartungswert des Momentanpols unter Berücksichtigung der Standardabweichung in etwa im Wälzpunkt des Rades.

Weitere Versuche wurden unternommen, indem nicht mehr die am weitesten von einander entfernten Punkte sondern die in Kapitel 4.1 verwendete Methode genutzt wird. Hierfür wird zunächst ein Punkt über den Shi-Tomasi Algorithmus gewählt. Dieser nutzt die am besten sichtbare Kante, welche in diesem Fall durch den von g2 definierten `nod` verursacht wird. Der Verlauf der nun verfolgten Punkte verspricht



(a) Versuch momentanpol1_1.html.

(b) Versuch momentanpol1_4.html.

Abb. 5.2: Die orangen Punkte sind hier die berechneten Momentanpole der entsprechenden Iteration. Der Verlauf ist von links nach rechts, entsprechend der Bewegung des Rades. Die Graphen zeigen jeweils die Verteilung der Momentanpole gemäß ihrer Y-Koordinate.

weniger Streuung der Koordinaten und entsprechend eine höhere Genauigkeit bei der Berechnung der Geschwindigkeit.

Des Weiteren soll nicht mehr die Gerade der am weitesten auseinander liegenden Punkte genutzt werden sondern die orthogonale Regressionsgerade, welche in Kapitel 3.4.1 beschrieben wurde. Diese verspricht ebenfalls weniger Streuung bei der Ermittlung des vom Glied eingelegten Winkels.

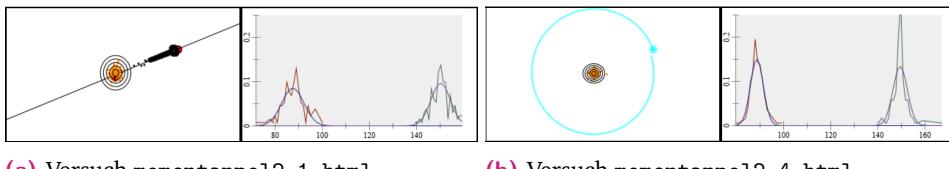
Die beschriebene Implementation erfordert hier jedoch nur eine andersartige Befüllung der `pts` und `lines` Eigenschaften des entsprechenden `group` Objektes. Die Berechnung des Momentanpols und die genutzten Funktionen sind ansonsten gleich. In Abbildung 5.2 ist zu sehen, dass die Berechnung in der Tat eine erheblich geringere Streuung aufweisen. Die gezeigten Ansätze überspringen jeweils 20 Iterationen. Dieser Wert ist experimentell ermittelt und sollte der Geschwindigkeit der Drehung angepasst werden. Die Standardabweichung hat sich durch den neuen Ansatz von 4.8 auf 1.9 reduziert.

5.3 Gestellglieder

5.3.1 Drehendes Gestellglied

Diese Herangehensweise soll nun für die Ermittlung des Drehpunktes eines Pendels, wie es in Kapitel 3 betrachtet wird, verwendet werden. Der Drehpol eines Gliedes, welches mit dem Gestell verbunden ist, sollte sich im Momentanpol befinden. Für diesen Versuch wurde entsprechend das Pendel aus Kapitel 3 verwendet.

Anhand von Abbildung 5.2 ist bereits abzusehen, dass die Standardabweichung erheblich geringer ist. Die Ellipsen um die Erwartungswerte stellen die Vertrauensbereiche dar. Jede Ellipse stellt hierbei ein Vielfaches der Standardabweichung dar.



(a) Versuch `momentanpol2_1.html`.

(b) Versuch `momentanpol2_4.html`.

Abb. 5.3: Hier stellen die orangenen Punkte die pro Iteration berechneten Momentanpole dar. Der grüne Graph steht jeweils für die Anzahl der X-Koordinaten der berechneten Momentanpole und der orangene Graph für die Y-Koordinaten. Die bläulichen Funktionen stellen jeweils die Gauß-Verteilungen dar.

Die Standardabweichung reduziert sich für die X- und Y-Koordinate jeweils von 4.2 und 4.7 auf 3.0 und 2.6 nach einer kompletten Umdrehung des Gliedes.

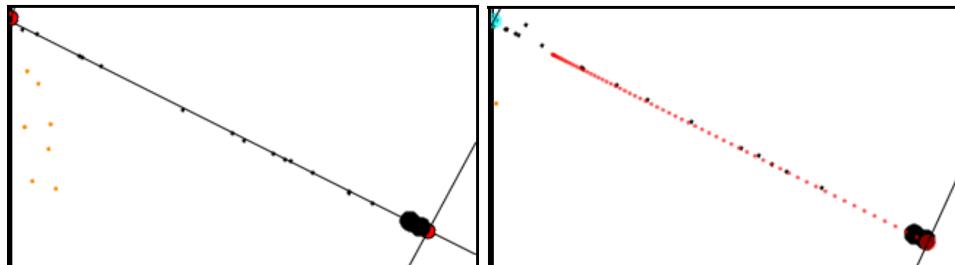
5.3.2 Translatives Gestellglied

Für ein sich translativ bewegendes Glied ist entsprechend zu erwarten, dass der Momentanpol sich im Unendlichen befindet. Die numerische Herangehensweise lässt hierbei also vermuten, dass die Varianz der berechneten Punkte sehr hoch ist. Ein eindeutiges Ergebnis zu bekommen, ist numerisch unwahrscheinlich. Im Gegenteil ließe hier ein besonders ungenaues Ergebnis den Schluss zu, dass es sich um ein Schubgelenk handelt. Um zu verhindern, dass sich der Momentanpol fälschlicherweise auf der Gliedebene des sich bewegten Elementes befindet, wird der eingeschlossene Winkel durch $dw = dw > \text{Math.PI} / 2 ? dw - \text{Math.PI} : dw$; korrigiert, wie in Listing 5.1 gesehen werden kann, damit jeweils der kleinere der beiden eingeschlossenen Winkel bevorzugt wird. Der rote Punkt in Abbildung 5.4b stellt hier den verfolgten Punkt dar und nicht den Momentanpol. Die Standardabweichung zu vergleichen scheint an dieser Stelle weniger sinnvoll. Sie ist in beiden Fällen sehr hoch und wird nur durch die Tatsache niedrig gehalten, dass `Data` die Punkte außerhalb des Canvas ignoriert, um die Gauß-Verteilung der gemessenen Werte nutzbar zu halten².

5.4 Beobachtungen

Es hat sich gezeigt, dass die naive Herangehensweise von den am weitesten entfernten Punkten in Verbindung mit der Regressionsgerade durchaus Erwartungswerte

²Teilweise auftretende extreme Ausschläge haben einen großen Einfluss auf die Varianz.



(a) Versuch `momentanpol3_1.html`.

(b) Versuch `momentanpol3_3.html`.

Abb. 5.4: Hier sind die Bewegungen eines sich translativ bewegenden Gelenkes nachvollzogen. Im zweiten Bild wird zudem die Gerade zwischen dem aktuellen Punkt und dem Momentanpol gezeichnet, welche hier in der Tat orthogonal zur Bewegungsrichtung steht. Auf dem Bild selber sollten keine Momentanpole zu finden sein.

liefert, welche dem tatsächlichen Momentanpol entsprechen. Werden jedoch lediglich Momentaufnahmen betrachtet, so sind diese wahrscheinlich nicht als gute Bezugspunkte für weitere Prognosen zur Rekonstruktion eines Mechanismus geeignet. Die Kombination aus der Ermittlung der Regressionsgeraden und der Verfolgung von Punkten durch den Lucas-Kanade Algorithmus weist jedoch eine geringere Streuung auf. Es kann also davon ausgegangen werden, dass auf diese Weise ermittelte Erwartungswerte nach nur wenigen Iterationen nutzbare Pole darstellen.

Bisher wurde die Ermittlung des Momentanpols eines isolierten Gliedes betrachtet. Da ein Mechanismus zwangsläufig aus mehreren Gliedern besteht soll nun betrachtet werden, wie ein mehrgliedriges Modell in seine Komponenten geteilt werden kann.

Zuordnung von Datenpunkten zu Gliedern

„Eine Kinematische Kette ist die Aneinanderreihung wenigstens dreier durch Elementenpaare beweglich miteinander verbundener Glieder“

— VDI2127
Getriebetechnische Grundlagen

Im vorangegangen Abschnitt wurde die Erkennung der Pole ebener Gliedebebenen durch ihre Bewegung betrachtet. Alle Methoden gingen jedoch davon aus, dass sich alle Datenpunkte einem Glied zuordnen lassen. Es ist entsprechend notwendig, bevor man die bereits besprochenen Methoden anwenden kann, eine Zuordnung der Punkte in entsprechende Gruppen zu unternehmen, welche dann an die bekannten Methoden weiter gegeben werden können. Die hier untersuchten Mechanismen folgen der Getriebedefinition aus [Gro14, G167] mit genau einem angetriebenem Glied. Für eine solche Zuordnung von Gliedern werden verschiedene Ansätze untersucht.

6.1 Zuordnung von Geraden an Punkte

Der zuerst untersuchte Ansatz geht von der Annahme aus, dass die hier untersuchten Mechanismen hinreichend durch Geraden beschrieben werden können. Die Punkte der Punktwolken lassen sich entsprechend einzelnen Geraden zuordnen. Diese Annahme umfasst ähnlich wie bei den vorangegangen Methoden, dass die Glieder immer länger sind als sie breit sind (s. Kapitel 3.3).

Hierfür wird ein Punkt aus den Datenpunkten genommen und dann eine Gerade definiert, welche am ehesten dem Glied entspricht, welche diesem Punkt zugeordnet werden kann. Dann werden alle Punkte genommen deren Abstand zu dieser Gerade kleiner ist als ein Schwellenwert, welcher vorher definiert werden muss.

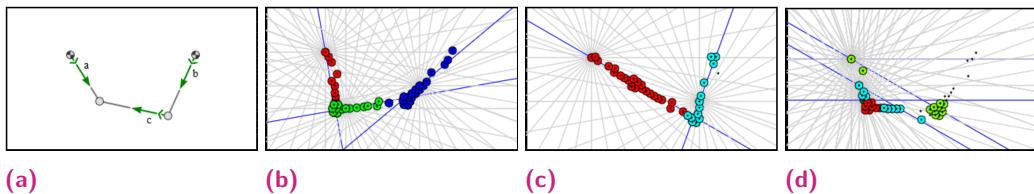


Abb. 6.1: Versuch gruppe1_1.html. In Bild 6.1a wird der Mechanismus gezeigt, dessen Glieder nun zugeordnet werden sollen. Bild 6.1b zeigt die Vorhersage zur Zuordnung dieser Punkte. Bild 6.1c zeigt die Zuordnung der Punkte in den Totlagen des Mechanismus. Hier wird das linke Glied und die Koppel einer einzelnen Gruppe zugeordnet. Bild 6.1c zeigt, dass der genutzte Algorithmus manchmal auch falsche vorhersagen trifft. Das liegt daran, dass zuerst die grüne Gruppe ermittelt wurde, bei welcher die Dichte der weit entfernten Punkte des Gelenkes des rechten Gliedes zu einer falschen Geraden geführt hat. Danach hat sich der Fehler in den übrigen Gruppen entsprechend fortgepflanzt.

Alle Punkte deren Abstand unter diesem Schwellenwert sind werden dann der ersten Gruppen zugeordnet und der Prozess wird mit allen ungruppierten Punkten wiederholt. Hierfür soll die Gerade ähnlich zur Regressionsgerade bestimmt werden. Als problematisch stellt sich jedoch heraus, dass die Methode zur Berechnung dieser Geraden nicht zwischen Punkten unterscheidet, welche zu einem Glied des gewählten Punktes gehören. Daher kann die Methode zur Berechnung der Regressionsgeraden in ihrer bisherigen Form nicht genutzt werden. Ein naiver Ansatz an dieser Stelle wäre die Berechnung der Fehler von willkürlich in die Ebene gelegten Geraden. Der Fehler berechnet sich dann durch die Summe der orthogonalen Abstände aller Punkte zu dieser Geraden. Damit Punkte, welche weit von der Geraden entfernt sind, einen geringeren Einfluss auf den Fehler haben, kann stattdessen eine Wurzel der Distanz verwendet werden. Wenn man den Fehler für genug Geraden im Winkel äquidistant zueinander auf diese Weise misst, kann die am besten passende Gerade gewählt werden und alle Punkte innerhalb des vorher angesprochenen Schwellenwertes dieser als Gruppe zugeordnet werden.

Eine entsprechende Funktion wurde auf der `PointCloud` Klasse definiert. Diese hält die Punkte vor, welche gruppiert werden sollen. Ziel der Funktion ist es, zunächst die Linien zu finden, welche den Mechanismus am ehesten repräsentieren. Zunächst wird eine Kopie der Punkteliste erstellt. Dann wird, solange es noch mehr als 10% der initial ungruppierten Punkte gibt, eine solche Linie ermittelt. Es wird zunächst der Punkt gewählt, welcher den geringsten Wert für die X-Koordinate aufweist. Dann wird für eine festgelegte Anzahl an Geraden, welche jeweils um den gleichen Winkel versetzt sind und einen gemeinsamen Schnittpunkt in dem zuvor gewählten Punkt haben, die orthogonale Distanz aller anderen Punkte gemessen. In Abbildung 6.1 ist die Anzahl dieser Punkte 36, welche durch die grauen Linien gezeigt werden. Von

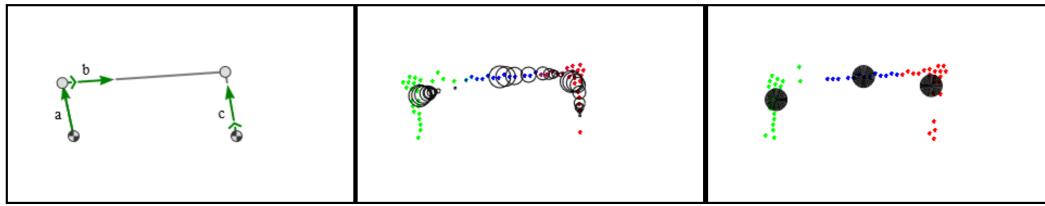


Abb. 6.2: Versuch gruppe2_1.html. Das linke Bild zeigt den Mechanismus. Das mittlere Bild zeigt die Gruppen in jeweils unterschiedlichen Farben an. Die Centroids werden bei jeder Iteration größer, um die Annäherung zu visualisieren. Es ist sichtbar, dass sie gegen einen Punkt konvertieren. Das rechte Bild zeigt einen darauffolgenden Aufruf des Algorithmus. Die Centroids des letzten Aufrufs werden hierbei als Ausgangspunkte gewählt. So bleiben die Gruppen über den Verlauf der Bildsequenz hinweg konstant. Entsprechend wandern die Centroids kaum über die internen Iterationen.

dieser orthogonalen Distanz wird jeweils noch die dritte Wurzel genommen, damit Punkte, die weiter weg von der Geraden sind, weniger Einfluss auf diese Summe haben. Ziel ist es, die Gerade zu finden, welche hierbei die geringste, als `score` bezeichnete, Summe aufweist. Ohne einen weiteren Ausgleichswert würden hier entsprechend jene Linien, welche wenig Punkte treffen bevorzugt werden. Daher wird der `score` noch durch die Anzahl der Punkte geteilt, deren Distanz zur Geraden unterhalb des Schwellenwertes liegt.

Der Schwellenwert sowie die Skalierung der Ausgleichswerte wurden experimentell bestimmt. Es ist davon auszugehen, dass Mechanismen, welche eine andere Form aufweisen als von `mec2` gezeichnete Modelle, andere Schwellenwerte zur korrekten Berechnung benötigen.

Nachdem die entsprechenden Linien bestimmt wurden, wird die auf der `Line` Klasse definierte statische Funktion `realignGroups` aufgerufen. Diese bestimmt für jede der Gruppen die entsprechende Regressionsgerade. Dann wird jeder Punkt der Geraden zugeordnet, welche die geringste orthogonale Distanz aufweist. Auf diese Weise werden die Punkte neu gruppiert.

6.2 k-Means Algorithmus

Eine andere Möglichkeit die Datenpunkte ihren Gruppen zuzuordnen ist der k-Means Algorithmus [Gér19, S. 241]. Dieser wird in der Statistik genutzt, um Datenpunkte in Gruppen einzuteilen. Ein offensichtlicher Nachteil dieses Ansatzes besteht darin, dass

die Anzahl der Gruppen von vorneherein bekannt sein muss¹. Eine Untersuchung, ob er für den vorliegenden Anwendungsfall nutzbar ist, scheint dennoch angebracht. Eine entsprechende Funktion ist in der `PointCloud` Klasse definiert. Hier werden zunächst zufällig k Punkte auf dem Bild ausgewählt. Diese Punkte werden als `Centroids` bezeichnet. Jeder Punkt der Punktwolke ordnet sich dann demjenigen `Centroid` zu, welcher die geringste euklidische Distanz aufweist. Dies sollte drei Gruppen ergeben, welche die entsprechenden Punkte enthalten. Daraufhin werden neue `Centroids` durch die Erwartungswerte dieser Gruppen definiert. Dieser Vorgang soll dann mehrfach (hier zehn mal) wiederholt werden.

Als eine Erweiterung des k -Means Algorithmus könnte noch der EM-Algorithmus untersucht werden, welcher weniger dazu tendiert Gruppen gleicher Größe zu erstellen [Gér19, S. 262]. Dies wird jedoch an dieser Stelle nicht mehr getan.

6.3 Dijkstra Algorithmus

Die letzte Variante die hier untersucht werden soll, um die Glieder des Mechanismus in entsprechende Gruppen zu zerteilen, ist die Zuordnung aller Datenpunkte zueinander. Hierfür soll das `PointCloud` Objekt in einen vollständigen Graphen überführt werden, auf welchem dann über eine Implementation des Dijkstra-Algorithmus die kürzesten Pfade ermittelt werden. Anschließend werden diese Pfade genutzt, um Geraden zu definieren, welche dann Rückschluss auf die zugrundeliegenden Gruppen geben können.

Zur Durchführung des Dijkstra Algorithmus wurde die `Dijkstra` Klasse erstellt. Der Konstruktor dieser Klasse nimmt hierfür ein `PointCloud` Objekt und optional einen Punkt² entgegen, welcher hier als `anchor` bezeichnet werden soll. Ziel ist es den kürzesten Pfad zu bestimmen, um vom `anchor` zu jedem Punkt des übergebenen `PointCloud` Objektes zu gelangen. Der offensichtlich kürzeste Weg ist laut der Dreiecksungleichung natürlich immer der direkte Weg vom `anchor` zum entsprechenden Punkt. Deshalb sollen die Distanzen zwischen den Werten jeweils potenziert werden. An den Konstruktor kann dafür der Parameter `warp` übergeben werden, welcher standardmäßig den Wert zwei einnimmt. Eine hohe Potenzierung begünstigt entsprechend Pfade über mehr Knoten des Graphen. Hierfür wird der Graph gebildet,

¹Es gibt jedoch Methoden, um diese Anzahl zu berechnen, zum Beispiel durch den Silhouettenkoeffizienten [Gér19, S. 247].

²Ein Punkt sei definiert als Objekt mit den Eigenschaften `x` und `y`. Wird kein Punkt übergeben wird als `anchor` ein Punkt der `PointCloud` genommen.

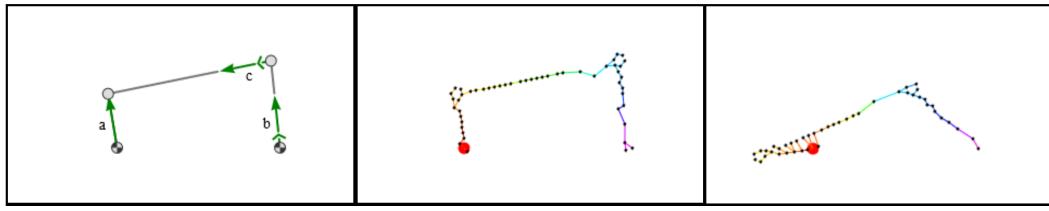


Abb. 6.3: Versuch gruppe3_2.html. Das linke Bild zeigt den Mechanismus. Das mittlere Bild zeigt die Verbindung des anchor, welcher im linken Gestellpunkt ist. Die Farbe der Verbindung zwischen den Punkten weist auf den dist Wert hin. Wie im rechten Bild zu sehen ist, werden die Verbindungen bei sehr kleinen eingeschlossenen Winkeln fehlerbehaftet.

indem alle Objekte des übergebenen `PointCloud` Objektes in ein Objekt überführt werden.

```

1 const compEdges = (p) => points
2   .map((sp, sidx) => ({
3     target: sidx,
4     weight: Dijkstra.euclDistance(p, sp) ** warp,
5   }))
6   .sort((a, b) => a.weight - b.weight)
7   .slice(1, numEdges + 1);
8
9 const graph = points.map((p, id) => ({
10   id,
11   dist: Infinity,
12   known: false,
13   edges: compEdges(p),
14   pred: -1,
15 }));

```

Listing 6.1: Im Dijkstra Konstruktor werden die Punkte des übergebenen `PointCloud` Objektes in einen Graphen überführt.

Die `id` entspricht dem Index des Punktes innerhalb der `points` Liste. `dist` beschreibt die Distanz, welche zum entsprechenden `anchor` ermittelt wurde. Hierbei ist zu beachten, dass es sich nicht um die euklidische Distanz, sondern um die Summe der durch `warp` beeinflussten Einzelpfade handelt. Dieser Wert ist also eher als Wert zum Vergleichen mit anderen Punkten zu verstehen. Er ist initial `Infinity`, damit er später bis zum kleinsten gefundenen Pfad reduziert werden kann. `known` wird genutzt, um bei der Berechnung jene Punkte überspringen zu können deren `dist` Wert bereits ermittelt wurde. Die `edges` Eigenschaft hält von jedem Punkt jeden anderen Punkt mittels des entsprechenden Index und der Distanz vor. Diese Distanz wird hier als `weight` bezeichnet, um zu unterstreichen, dass es nicht die euklidische Distanz wiederspiegelt, sondern diese mit dem `warp` Parameter potenziert wird. Zuletzt wird

`pred` genutzt, um jeweils den Vorgänger jedes Knotens auf dem Graphen ermitteln zu können.

Daraufhin wird dieser Graph als `unvisited` kopiert, um alle unbesuchten Punkte festzuhalten zu können. Dieser kopierten Liste wird der `anchor` an den Start angefügt³.

```
1  while (unvisited.length) {
2      unvisited.sort((a, b) => a.dist - b.dist);
3      const u = unvisited.shift();
4      u.known = true;
5
6      for (const { ldist, o } of u.edges
7          .map(e => ({ ldist: e.weight, o: graph[e.target] }))
8          .filter(({ o }) => !o.known)) {
9          const cdist = u.dist + ldist;
10         if (cdist < o.dist) {
11             o.dist = cdist;
12             o.pred = u;
13         }
14     }
15 }
```

Listing 6.2: Berechnung der `dist` Eigenschaft der einzelnen Objekte aus dem Dijkstra-Graphen.

Hier wird ausgenutzt, dass obwohl die Liste `unvisited` nur eine Kopie von `graph` ist, die Referenzen der einzelnen darin enthaltenen Objekte jedoch jeweils dieselben sind. So können die Elemente in `unvisited` dort jeweils der `dist` nach aufsteigend sortiert und verschoben werden und die Objekte in `graph` werden hierbei mit den Berechnungen aktualisiert. Das erste Objekt wird entsprechend aus der Liste entfernt und die `known` Eigenschaft wird auf `true` gesetzt. Bei der ersten Iteration ist dies in jedem Fall `anchor`, da `dist` offensichtlich 0 ist. Daraufhin wird für jedes Element in der `edges` Liste des entsprechenden Punktes die Distanz mit seiner eigenen addiert. Nach der ersten Iteration ist also davon auszugehen, dass jeder Punkt direkt mit dem `anchor` verbunden ist. Bei der zweiten Iteration werden alle Punkte aus `unvisited` wieder sortiert und jeder Punkt der nun zusammen mit der Distanz dieses Punktes eine kürzere Distanz aufweist wird seinen `dist` und entsprechend den `pred` Wert aktualisieren. Dadurch, dass die `dist` jeweils potenziert werden, werden hier für die Aufgabe nutzbare Pfade ermittelt.

³Mit der `id - 1` und einer `dist` von 0.

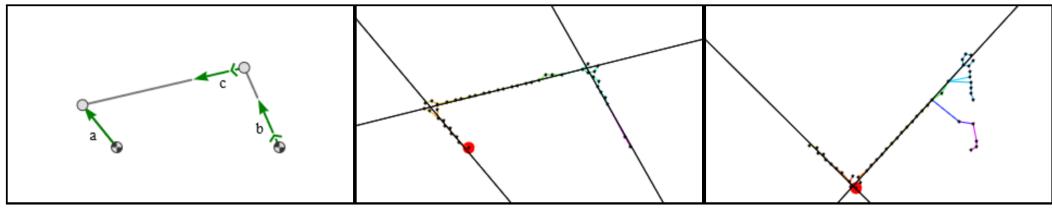


Abb. 6.4: Versuch gruppe3_3.html. Das linke Bild zeigt den Mechanismus. Das mittlere Bild zeigt die Regressionsgeraden, welche durch die vorgeschlagene Methode meistens ermittelt werden. Das rechte Bild zeigt einen entsprechenden Fehler, welcher der falschen Zuordnung der Pfade im Graphen zugrunde liegt.

6.3.1 Nutzung des Korrelationskoeffizienten

Der Dijkstra Algorithmus erstellt für die Punktwolke Pfade für jeden Punkt zu dem anchor. Diese Pfade soll jetzt dazu genutzt werden, um eine Linie zu bilden. Die Idee ist, dass wenn man die äußersten Punkte des Graphen nimmt, um von diesen zum anchor wandert, dann werden hierdurch Pfade gebildet, welche keine Ausläufer haben.

```

1 let ends = this.graph
2   .filter((n) => !this.graph.some(g => g.pred.id === n.id))
3   .sort((a, b) => b.dist - a.dist);
4
5 const winner = [];
6
7 while (ends.length) {
8   const occupied = (function f(o, arr) {
9     if (o.pred) arr = f(o.pred, arr);
10    return [...arr, o];
11  })(ends[0], []);
12  winner.push(ends.shift());
13
14  ends = ends.filter(u => {
15    for (let p = u.pred; p.pred; p = p.pred)
16      if (occupied.includes(p)) return false;
17    return true;
18  });
19}

```

Listing 6.3: Bestimmung der äußersten Knoten zur Bestimmung von Geraden durch den Korrelationskoeffizienten in der `groupsByCorrelation` Funktion innerhalb der Dijkstra Klasse.

Hierfür werden zunächst die Knoten im Graphen gesucht, welche keinen Vorgänger haben. Dann wird eine Liste dieser, welche als `ends` bezeichnet wird, der `dist` nach absteigend sortiert. Diese Liste wird dann gefiltert, indem der Punkt der jeweils die

größte Distanz hat andere Punkte filtert, nach dem Prädikat ob diese oder deren verkettete `pred` Eigenschaften gemeinsame Vorgänger haben.

Dieser Vorgang wird solange wiederholt, bis die `ends` Liste leer ist, so dass nur noch äußere Knoten vorgehalten werden, welche keine gemeinsamen Vorgänger zum `anchor` haben. Diese als `winner` bezeichneten Knoten werden dann über ihre Vorgänger iterieren, um zu prüfen ob sich mit diesen eine Gerade bilden lässt.

```

1 let lines = [];
2 for (const win of winner) {
3     let group = [];
4     for (let u = win; u.pred; u = u.pred) {
5         const correlation = PointCloud.correlation(
6             [...group, this.points[u.id]]);
7         if (Math.abs(correlation) ** group.length < minCorr) {
8             if (group.length > 2 &&
9                 group.length > this.points.length * minPercent) {
10                 lines.push(Line.fromRegressionLine(group, g));
11             }
12             group = [];
13         }
14         group.push(this.points[u.id]);
15     }
16     lines.push(Line.fromRegressionLine(group, g));
17 }
```

Listing 6.4: Bestimmung der äußersten Knoten zur Bestimmung von Geraden durch den Korrelationskoeffizienten in der `groupsByCorrelation` Funktion innerhalb der `Dijkstra` Klasse.

Die Punkte werden auf ihre Nähe zu einer Regressionsgeraden über den Korrelationskoeffizienten geprüft. Der Korrelationskoeffizient berechnet sich durch [Pap14]

$$r = \left(\left(\sum_{i=1}^n x_i y_i \right) - n \bar{x} \bar{y} \right) \div \sqrt{\left(\sum_{i=1}^n x_i^2 - n \bar{x}^2 \right) \left(\sum_{i=1}^n y_i^2 - n \bar{y}^2 \right)} \quad (6.1)$$

und dessen Berechnung ist als statische Funktion auf der `PointCloud` definiert.

In Listing 6.4 wird für jeden Eintrag aus `winner` der Vorgänger gewählt und diese werden einer Liste `group` hinzugefügt. Für diese Gruppe an Punkten wird dann der Korrelationskoeffizient berechnet. Ist dieser niedriger als ein Schwellenwert, wird noch geprüft ob eine Mindestanzahl an Punkten erreicht ist, damit aus diesen

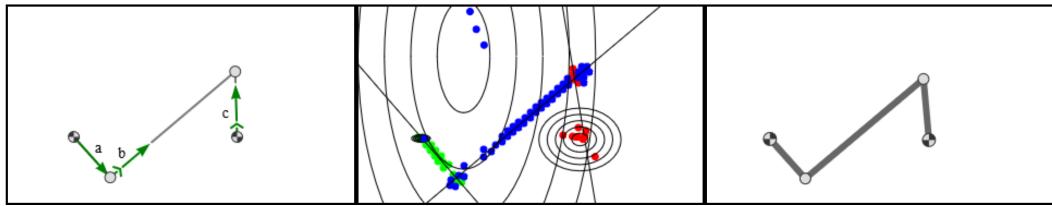


Abb. 6.5: Versuch gruppe4_1.html. Das linke Bild zeigt den Mechanismus. Das mittlere Bild zeigt die Regressionsgeraden, die Gruppen, sowie die Momentanpole, welche durch die Gruppen ermittelt werden. Die Kreise zeigen jeweils die Standardabweichung der Verteilungen als Ellipsen in ihren Vielfachen dar. Im rechten Bild ist das rekonstruierte Viergelenk zu sehen.

dann die Regressionsgerade gebildet werden kann⁴. Diese Geraden werden dann vorgehalten und könnten für weitere Analysen genutzt werden.

6.4 Kombination der Ansätze

Die Ermittlung der Pfade durch den Dijkstra Algorithmus führte zu der Idee, die dadurch ermittelten Distanzen zu nutzen, um die Centroids des k-Means Algorithmus zu verwenden. Dies führte zur Erstellung der `kMeansDijkstra` Funktion der `PointCloud`. Diese nutzt größtenteils dieselbe Implementation wie die `kMeansClustering` Funktion aus Kapitel 6.2, jedoch wird hier eine andere Methode zur Ermittlung des nächsten Centroids der Punkte genutzt.

```

1 function findNearestCentroid(pointIdx, dijkstras) {
2     return dijkstras.indexOf(dijkstras.reduce((pre, cur) =>
3         pre.graph[pointIdx].dist > cur.graph[pointIdx].dist ?
4             cur : pre));
5 }
```

Listing 6.5: Bestimmung des nächsten Centroids in der `kMeansDijkstra` Funktion.

Die `findNearestCentroid` Funktion von `kMeansClustering` nutzt für die Ermittlung des nächsten Centroids jeweils die euklidische Distanz. Stattdessen wird hier für jeden Centroid ein Graph erstellt, welcher dann die Distanzen zu den Punkten mittels der Dijkstra Implementation ermittelt. Auf diese Weise wird verhindert, dass Punkte in der Nähe von Totlagen des Mechanismus falsch zugewiesen werden. Diese Implementation zeigt weniger Fluktuation als die anderen Ansätze. Werden diese Gruppen dann noch durch die in Kapitel 6.1 erwähnte `realignGroups` Funktion den

⁴Der Standardwert für den Schwellenwert der Regressionsgeraden liegt bei 0,6 und die Mindestanzahl an Punkten liegt bei 10% der Gesamtzahl der Punkte des Graphen.

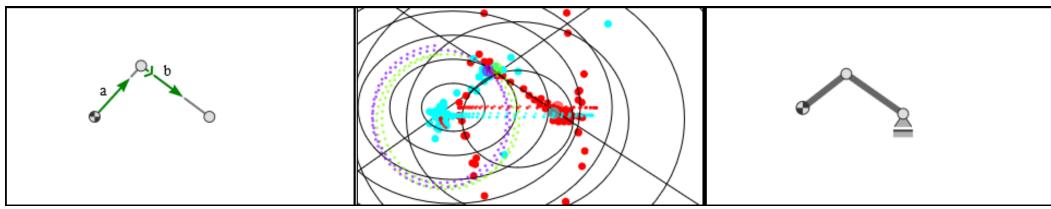


Abb. 6.6: Versuch gruppe4_2.html. Das linke Bild zeigt den Mechanismus. Das mittlere Bild zeigt die Regressionsgeraden, die Gruppen und die Momentanpole, welche durch die Gruppen ermittelt werden, sowie den Pfad der durch den Lucas-Kanade Algorithmus verfolgten Punkte. Die Kreise zeigen jeweils die Standardabweichung der Verteilungen als Ellipsen in ihren Vielfachen dar. Das Schubgelenk wird durch den verfolgten Punkt gebildet, dessen Pfad den höchsten Korrelationskoeffizienten als Betrag aufweist. Im rechten Bild ist das rekonstruierte Schubgelenk zu sehen.

Regressionsgeraden entsprechend angeglichen, so werden die Gruppierungen für einen durch `mec2` animierten Mechanismus nahezu stabil.

In Versuch `gruppe4_1.html` wird ein Viergelenk rekonstruiert. Die Absolutpole werden als jene definiert, welche eine geringe Standardabweichung aufweisen. Es ist davon auszugehen, dass die anderen Pole durch ihre Polbahn naturgemäß eine höhere Standardabweichung haben. Daraufhin werden die Relativpole als die Schnittpunkte der durch die Gruppen gezogenen Regressionsgeraden definiert. Der entsprechend rekonstruierte Mechanismus wird in Abbildung 6.5 gezeigt.

Ebenso wird ein Schubgelenk in Versuch `gruppe4_2.html` rekonstruiert. Hierfür wurde der Absolutpol wie in Versuch `gruppe4_1.html` nachgebildet. Das Schubgelenk wird durch den Korrelationskoeffizienten der durch den Lucas-Kanade Algorithmus ermittelten Pfade bestimmt. Der letzte Punkt des Pfades mit dem höchsten Betrag des Korrelationskoeffizienten ist entsprechend das Schubgelenk. Für diesen Versuch wird die von `mec2` definierte `base` und das zugehörige Glied, welche zur Definition des Schubgelenkes dort notwendig sind, daran gehindert gezeichnet zu werden. Das ist notwendig, da obwohl dort keine Bewegung vorhanden ist, dennoch kleine Unterschiede beim Rendern erkannt werden, was die Gruppierung fehlerhaft macht.

6.5 Rekonstruktion eines Mechanismus durch die Relativpole

Die vorangegangen Methoden geben scheinbar akzeptable Ergebnisse für Viergelenke. Sie scheitern jedoch, sobald die Anzahl der Glieder erhöht wird, wie in

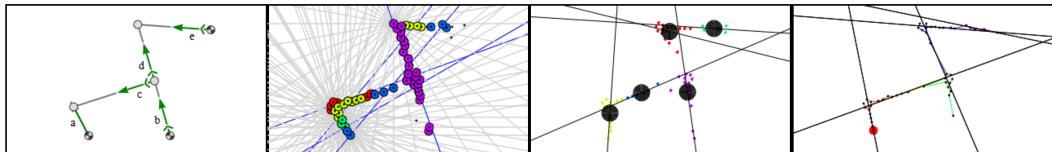


Abb. 6.7: Links ist der Ausgangsmechanismus, der durch die wattsche Kette gebildet wird. Von links nach rechts sind jeweils der Versuch gruppe1_3.html, welcher so offensichtlich keine nutzbaren Ergebnisse hervorbringt. gruppe2_6.html zeigt etwas bessere und der Versuch gruppe3_4.html produziert hier scheinbar die stabilsten Ergebnisse, jedoch sind auch diese für weitere Analysen kaum brauchbar.

Abbildung 6.7 gesehen werden kann. Aus diesem Grund soll in diesem Kapitel davon ausgegangen werden, dass sowohl die Ermittlung der Gliedebenen als auch die dazugehörigen Momentanpole ideal sind. Es wird untersucht, inwiefern die Menge der Relativpole nach den gesuchten Gelenken gefiltert werden können.

6.5.1 Ermittlung der Anzahl der gesuchten Punkte

Der Freiheitsgrad für einen planaren Mechanismus berechnet sich nach Grübler durch [Gös16]

$$F = 3n - 3 - b_1 - 2b_2 \quad (6.2)$$

wobei F den Freiheitsgrad, n die Anzahl der Glieder und der b -Index jeweils die Wertigkeit dieses Gelenkes darstellt. Hierbei sei anzumerken, dass ein b_1 jederzeit durch zwei b_2 ersetzt werden kann, wenn man entsprechend auch die Gliederzahl n um eins erhöht, was durch

$$\begin{aligned} F &= 3((n + c) - 1) - (b_1 - c) - 2(b_2 + 2c) \\ F &= 3n + 3c - 3 - b_1 + c - 2b_2 - 4c \\ F &= 3n - 3 - b_1 - 2b_2 \end{aligned} \quad (6.3)$$

gezeigt wird. c kann hierbei beliebig also auch als $c = b_1$ gewählt werden. Entsprechend darf in den folgenden Überlegungen von lediglich zweiwertigen Gelenken ausgegangen werden. Aus der Annahme heraus, dass die hier betrachteten Mechanismen jeweils zwangsläufige Mechanismen mit genau einem angetriebenen Glied sind lässt sich diese Gleichung durch

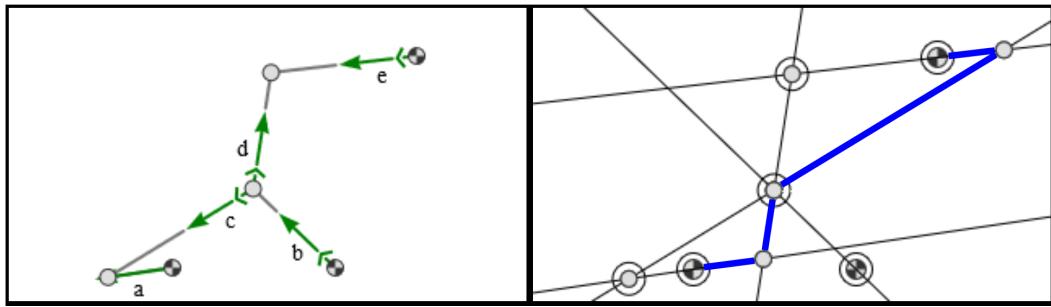


Abb. 6.8: Versuch gruppe4_3.html. Links ist der Ausgangsmechanismus. Fast alle Relativpole können durch die genannten Methoden als Gelenke ausgeschlossen werden. Jedoch bilden die nicht benachbarten Gliedebenen hier zwei weitere Kandidaten, welche ohne weitere Analyse nicht ausgeschlossen werden können.

$$1 = 3n - 3 - 2b_2 \quad (6.4)$$

$$b_2 = \frac{3n}{2} - 2$$

vereinfachen. So kann neben der Anzahl der gesuchten Gelenke des Weiteren festgestellt werden, dass die Gliederzahl der hier für die Berechnung genutzten kinematischen Ketten in jedem Fall einer geraden Zahl entspricht. Wird also während den Ermittlungen ein Schubgelenk vermutet, so zählt dieses entsprechend doppelt.

6.5.2 Betrachtung aller Relativpole

Die gemeinsame Drehung zweier Gliedebenen geschieht jeweils um einen Relativpol. Entsprechend sind alle gesuchten Gelenke auch Relativpole. Die Anzahl der Relativpole eines Mechanismus berechnet sich durch [Gös16]

$$k = \frac{n}{2}(n - 1) \quad (6.5)$$

Wobei n hier wieder die Anzahl der Glieder ist. Hier lassen sich in jedem Fall die durch die in den angeführten Methoden bestimmten Momentanpole der bewegten Gliedebenen ausschließen. Die Absolutpole können direkt als gefundene Gelenke identifiziert werden. Allerdings ist auch zu beachten, dass Relativpole im Schnittpunkt zweier Gliedebenen, welche gemeinsam mehr als einen Absolutpol aufweisen keine Gelenke darstellen können, da diese sonst starr wären. Die Relativpole, welche

durch zwei nicht benachbarte Gliedebenen gebildet werden, lassen sich dennoch nicht einfach ausschließen, so dass hier weitere Analysen notwendig sind.

Nach dem Satz von Aronhold-Kennedy [Gös16; KCH16], welcher auch als 3-Polsatz bezeichnet wird, befinden sich für drei bewegte Gliedebenen jeweils drei Relativpole auf einer Geraden. Die Punkte, welche nicht ausgeschlossen werden können, bewegen sich entsprechend auf einer solchen Geraden. Der Abstand dieser Gelenke müsste konstant zu mindestens einem benachbarten Gelenk sein, da die Annahme besteht, dass Schubgelenke stets mit dem Gestell verbunden sind. Hierüber können weitere Relativpole als Gelenke ausgeschlossen werden. Schubgelenke werden als Absolutpole im Vorfeld erkannt. Für diese können ansonsten die gleichen Methoden genutzt werden, was in den Versuchen `gruppe5_4.html` und `gruppe5_5.html` gezeigt wird.

Zusammenfassung und Ausblick

„ Wenn man die Daten lange genug foltert, werden sie alles gestehen.

— Ronald Harry Coase
Britischer Ökonom

In dieser Arbeit wurde untersucht, welche Informationen einer Bildsequenz von einem sich bewegendem planaren Mechanismus gewonnen werden kann. Es hat sich gezeigt, dass allein durch die Punktwolke der sich verändernden Pixel einiges ermittelt werden kann. Nachdem für die Versuche über den kleinsten umfassenden Kreis zwar viele nützliche Methoden entwickelt wurden, so hat dieser in einer abschließenden Lösung keinen Platz gefunden. Die Ermittlung der orthogonalen Regressionsgeraden hingegen hat sich hier als ein notwendiges Hilfsmittel herausgestellt, welches die Gliedebenen selber definiert und für die Ermittlung der Winkelgeschwindigkeiten genutzt wird. Außerdem wurden Methoden entwickelt, um die zunächst unzusammenhängenden Punkte über die zeitlichen Iterationen miteinander zu verbinden, um so auch darüber Informationen gewinnen zu können.

Die Nutzung der äußersten Punkte der Punktwolke wurde in späteren Versuchen durch den Shi-Tomasi Algorithmus ersetzt. Diese Punkte werden dann durch eine Implementation des Lucas-Kanade Algorithmus verfolgt, so dass aus diesen die Absolutgeschwindigkeit von auf den Gliedern ermittelten Punkten gemessen wird. Die Anwendung dieser Algorithmen hat hier den Fehler bereits maßgeblich reduziert, so dass bei einzelnen Gliedern die Momentanpole mit einer hinreichenden Genauigkeit ermittelt werden konnten.

Die Absolutgeschwindigkeit und die Winkelgeschwindigkeit werden genutzt, um den Momentanpol eines sich in der Ebene bewegenden Gliedes zu ermitteln. Nach der Ermittlung aller Momentanpole lassen sich dann daraus jene filtern, welche als Absolutpole interpretiert werden können. Daraufhin werden kinematische Gesetzmäßigkeiten angewandt, um nicht als Gelenk nutzbare Relativpole zu filtern, um den zugrundeliegenden Mechanismus zu rekonstruieren.

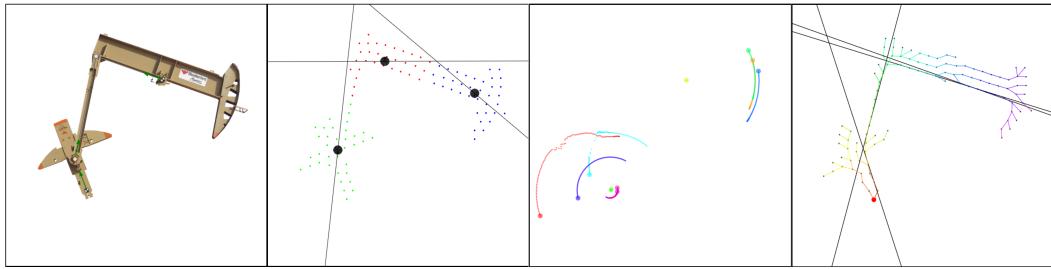


Abb. 7.1: Versuche `bilder2_3.html`, `bilder2_4.html` und `bilder2_6.html`. Im ersten Canvas wird der Pumpjack gezeigt [@Uhl21]. Im zweiten Bild wird der k-Means Algorithmus aus Kapitel 6.2 angewandt. Im dritten Bild wird der Lucas-Kanade Algorithmus aus Kapitel 4.1 genutzt. Im letzten Bild wird der Dijkstra Algorithmus aus Kapitel 6.3 gezeigt.

Die vorgeschlagenen Methoden beinhalten alle Messungenauigkeiten, welche es zunächst nicht zulassen kompliziertere Mechanismen zu rekonstruieren. Allerdings wurden Methoden aus dem Themengebiet der künstlichen Intelligenz vorgestellt, welche diese Fehler durch eine bessere Segmentierung der Glieder reduzieren können. Diese wurden jedoch auch zugunsten des Rahmens dieser Arbeit nicht weiter untersucht.

Für Versuche mit reellen Mechanismen wurden ebenfalls Versuche durchgeführt. Mit der Nutzung von einer Kamera mit Stativ unter Tageslicht waren die Störungen jedoch zu groß, als dass die `compareImages` Funktion sinnvolle Information ermitteln konnte¹. Hier ist es denkbar die Methoden zur Ermittlung von Änderung weniger sensibel zu machen um so die gesuchten Änderungen hervorzuheben.

Es wurden außerdem Versuche mit dem in `mecEdit` [Uhl19] enthaltenen Pumpjack durchgeführt, welche in der Versuchsgruppe `bilder` enthalten sind. Die Ergebnisse sind in Abbildung 7.1 zu sehen.

Schlussendlich ist zu sagen, dass eine komplett Rekonstruktion von Mechanismen durchaus denkbar ist. Die dafür notwendigen Technologien existieren, es bedarf an dieser Stelle nur einer sehr viel feinere Einstellung der Parameter oder entsprechend komplexeren Verfahren.

¹Der entsprechende Versuch ist unter <https://aka.klawr.de/ba#14> zu finden.

Literatur

- [And21] David Thomas Andrew Hunt. *Der pragmatische Programmierer*. Hanser Fachbuchverlag, 9. Apr. 2021 (zitiert auf Seite 5).
- [Bou00] Jean-yves Bouguet. “Pyramidal implementation of the Lucas Kanade feature tracker”. In: *Intel Corporation, Microprocessor Research Labs* (2000) (zitiert auf Seite 28).
- [BBM09] Thomas Brox, Christoph Bregler und Jitendra Malik. “Large displacement optical flow”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Juni 2009 (zitiert auf Seite 27).
- [Fis+15] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg et al. “FlowNet: Learning Optical Flow with Convolutional Networks”. In: (Apr. 2015). arXiv: 1504.06852 [cs.CV] (zitiert auf den Seiten 32, 33).
- [Gér19] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly UK Ltd., Okt. 2019. 819 S. (zitiert auf den Seiten 22, 45, 46).
- [Gös16] Stefan Gössner. *Mechanismentechnik vektorielle Analyse ebener Mechanismen*. Berlin: Logos Verlag Berlin GmbH, 2016 (zitiert auf den Seiten 35, 36, 53–55).
- [Gro14] Grote. *Dubbel Taschenbuch für den Maschinenbau*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014 (zitiert auf Seite 43).
- [Ilg+16] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia et al. “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks”. In: (Dez. 2016). arXiv: 1612.01925 [cs.CV] (zitiert auf Seite 33).
- [Jür20] Lothar Sachs Jürgen Hedderich. *Angewandte Statistik*. Springer-Verlag GmbH, Okt. 2020. 1054 S. (zitiert auf Seite 23).
- [KCH16] Hanfried Kerle, Burkhard J. Corves und Mathias Hüsing. *Getriebetechnik*. Vieweg+Teubner Verlag, Jan. 2016 (zitiert auf Seite 55).
- [KAB15] Margret Keuper, Bjoern Andres und Thomas Brox. “Motion Trajectory Segmentation via Minimum Cost Multicuts”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Dez. 2015 (zitiert auf Seite 32).
- [Kla14] Hermann Gebhard Klaus Eden. *Dokumentation in der Mess- und Prüftechnik*. Springer Fachmedien Wiesbaden, 6. Aug. 2014. 212 S. (zitiert auf Seite 18).
- [LK81] Bruce Lucas und Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI)”. In: Bd. 81. Apr. 1981 (zitiert auf Seite 28).

- [Meg83] Nimrod Megiddo. “Linear-Time Algorithms for Linear Programming in R^3 and Related Problems”. In: *SIAM Journal on Computing* 12.4 (Nov. 1983), S. 759–776 (zitiert auf Seite 13).
- [OMB14] Peter Ochs, Jitendra Malik und Thomas Brox. “Segmentation of Moving Objects by Long Term Video Analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.6 (Juni 2014), S. 1187–1200 (zitiert auf Seite 32).
- [Pap+18] George Papandreou, Tyler Zhu, Liang-Chieh Chen et al. “PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model”. In: (März 2018). arXiv: 1803.08225 [cs.CV] (zitiert auf Seite 1).
- [Pap16] Lothar Papula. *Mathematik für Ingenieure und Naturwissenschaftler: Band 03*. Vieweg + Teubner Verlag, 4. Apr. 2016 (zitiert auf Seite 21).
- [Pap14] Lothar Papula. *Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler ; mit zahlreichen Rechenbeispielen und einer ausführlichen Integraltafel*. Wiesbaden: Springer Vieweg, 2014 (zitiert auf Seite 50).
- [Rus10] Stuart Russell. *Artificial intelligence : a modern approach*. Upper Saddle River, New Jersey: Prentice Hall, 2010 (zitiert auf Seite 27).
- [ST94] Jianbo Shi und Tomasi. “Good features to track”. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*. IEEE Comput. Soc. Press, 1994 (zitiert auf Seite 28).
- [Uhl19] Jan Uhlig. “Entwicklung einer modularen Web-App zur interaktiven Modellierung und impulsbasierten Analyse beliebiger planarer Koppelmechanismen”. Magisterarb. Fachhochschule Dortmund, 2019 (zitiert auf Seite 58).

Webseiten

- [@Goo21a] Google. *MoveNet*. 2021. URL: <https://www.tensorflow.org/hub/tutorials/movenet> (besucht am 19. Aug. 2021) (zitiert auf den Seiten 1, 2).
- [@Goo21b] Google. *PoseNet*. 2021. URL: <https://www.tensorflow.org/hub/tutorials/movenet> (besucht am 19. Aug. 2021) (zitiert auf den Seiten 1, 2).
- [@Nay21] Nayuki. *Smallest enclosing circle*. 20. Aug. 2021. URL: <https://www.nayuki.io/page/smallest-enclosing-circle> (zitiert auf Seite 13).
- [@Ope21] OpenCV. *OpenCV*. 2021. URL: <https://www.tensorflow.org/hub/tutorials/movenet> (besucht am 16. Aug. 2021) (zitiert auf Seite 29).
- [@Uhl21] Jan Uhlig. *mecEdit*. 21. Aug. 2021. URL: <https://mecedit.com/> (zitiert auf Seite 58).

Abbildungsverzeichnis

1.1	Versuch <code>bilder1_1.html</code>	1
1.2	Posenbestimmung durch maschinelles Lernen	2
2.1	Bild der <code>index.html</code>	5
2.2	Bild der Canvasse	8
3.1	Versuche <code>pendel1_1.html</code> und <code>pendel1_2.html</code>	14
3.2	Versuche <code>pendel1_3.html</code> und <code>pendel1_4.html</code>	15
3.3	Versuch <code>pendel2_1.html</code>	17
3.4	Beispiel für in etwa normal verteilte Daten.	19
3.5	Daten der Versuche <code>pendel2_1.html</code> und <code>pendel2_2.html</code>	21
3.6	Versuche <code>pendel2_4.html</code> und <code>pendel2_5.html</code>	24
4.1	Vektorfelder durch optischen Fluss.	27
4.2	Versuche <code>opticalflow2_1.html</code> und <code>opticalflow2_2.html</code>	29
4.3	Versuch <code>opticalflow1_1.html</code>	31
4.4	Beispiele aus dem Flying Chair Dataset	32
4.5	Vergleich FlowNet zu FlowNet 2.0	33
5.1	Vektorfeld der ebenen Bewegung	35
5.2	Versuche <code>momentanpol1_1.html</code> und <code>momentanpol1_4.html</code>	39
5.3	Versuche <code>momentanpol2_1.html</code> und <code>momentanpol2_4.html</code>	40
5.4	Versuche <code>momentanpol3_1.html</code> und <code>momentanpol3_3.html</code>	41
6.1	Versuch <code>gruppe1_1.html</code>	44
6.2	Versuch <code>gruppe2_1.html</code>	45
6.3	Versuch <code>gruppe3_2.html</code>	47
6.4	Versuch <code>gruppe3_3.html</code>	49
6.5	Versuch <code>gruppe4_1.html</code>	51
6.6	Versuch <code>gruppe4_2.html</code>	52
6.7	Versuche <code>gruppe1_3.html</code> , <code>gruppe2_6.html</code> und <code>gruppe3_4.html</code> . . .	53
6.8	Versuch <code>gruppe4_3.html</code>	54
7.1	Versuche <code>bilder2_3.html</code> , <code>bilder2_4.html</code> und <code>bilder2_6.html</code> . .	58

Listingverzeichnis

2.1	Ausschnitt der Definition des <code>tests</code> Objekts in der <code>index.html</code>	3
2.2	Iteration über das <code>tests</code> Objekt zur Befüllung der Navigationsleiste. . .	4
2.3	Definition der <code>simulation.run</code> Funktion.	6
2.4	Definition eines Pendels in <code>mec2</code> inklusive Animation.	9
3.1	Definition der <code>fromImages</code> Funktion, welche eine statische Funktion der <code>PointCloud</code> Klasse darstellt.	12
3.2	Definition der <code>intersection</code> Funktion, welche eine Funktion der <code>Line</code> Klasse darstellt.	17
3.3	Definition der <code>add</code> Funktion, welche dazu genutzt wird der <code>Data</code> Klasse neue Werte hinzuzufügen.	18
3.4	Definition der <code>reduce</code> Funktion, um die Standardfunktion der Liste nachzubilden.	18
3.5	Definition der <code>mu</code> Funktion, welche den Erwartungswert der Daten in <code>Data</code> berechnet.	19
3.6	Definition der <code>variance</code> Funktion, welche die Varianz der Daten in <code>Data</code> berechnet.	19
3.7	Definition der <code>gaussianDistribution</code> Funktion, welche die Normalverteilung für die gegebene Varianz und den Erwartungswert formt.	20
3.8	Definition der <code>getChart</code> Funktion der <code>Data</code> Klasse.	20
3.9	Definition der <code>removeOverlaps</code> Funktion der <code>PointCloud</code> Klasse.	22
3.10	Implementation der <code>covariance</code> Funktion in der <code>DataXY</code> Klasse.	23
3.11	Erstellung von <code>Line</code> Objekten durch die orthogonale Regressionsgerade für Datenpunkte	24
4.1	Implementation der <code>step</code> Funktion der <code>LucasKanade</code> Klasse.	30
5.1	Definition der <code>momentanpol</code> Funktion, welche in der <code>group</code> Klasse definiert ist Die Ermittlung der Punkte <code>p1</code> und <code>p2</code> sowie der Winkel <code>w1</code> und <code>w2</code> wurde hier aus Platzgründen weggelassen.	37
6.1	Im <code>Dijkstra</code> Konstruktor werden die Punkte des übergebenen <code>PointCloud</code> Objektes in einen Graphen überführt.	47

6.2	Berechnung der <code>dist</code> Eigenschaft der einzelnen Objekte aus dem Dijkstra-Graphen.	48
6.3	Bestimmung der äußersten Knoten zur Bestimmung von Geraden durch den Korrelationskoeffizienten in der <code>groupsByCorrelation</code> Funktion innerhalb der <code>Dijkstra</code> Klasse.	49
6.4	Bestimmung der äußersten Knoten zur Bestimmung von Geraden durch den Korrelationskoeffizienten in der <code>groupsByCorrelation</code> Funktion innerhalb der <code>Dijkstra</code> Klasse.	50
6.5	Bestimmung des nächsten Centroids in der <code>kMeansDijkstra</code> Funktion. .	51

Linkverzeichnis - <https://aka.klawr.de/ba>

#1: https://klawr.github.io/ba	2
#2: https://developer.mozilla.org/en-US/docs/Web/API/EventListener . . .	4
#3: https://github.com/klawr/ba/blob/master/index.html	5
#4: https://klawr.github.io/ba/src/bilder/bilder2_1.html	6
#5: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now . .	7
#6: https://github.com/klawr/ba/blob/master/src/scripts/simulation.js .	8
#7: https://github.com/nayuki/Nayuki-web-published-code/blob/master/smallest-enclosing-circle/smallest-enclosing-circle-demo.js	13
#8: https://www.nayuki.io/page/smallest-enclosing-circle	13
#9: https://github.com/klawr/ba/commit/7209dfee5ab0a70470e22862f26f7c45046cd98f	14
#10: https://github.com/klawr/ba/blob/master/src/third_party/smallestEnclosingCircle.js	16
#11: https://github.com/klawr/ba/blob/master/src/scripts/pointCloud.js#L63	16
#12: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map	18
#13: https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga3dbce297c1feb859ee36707e1003e0a8	29
#14: https://klawr.github.io/ba/src/bilder/bilder1_3.html	58

Colophon

This thesis was typeset with L^AT_EX 2_<. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at <http://cleanthesis.der-ric.de/>.

Erklärung über selbständig erbrachte Leistungen

Hiermit versichere ich, dass die von mir vorgelegte Prüfungsleistung selbständig und ohne unzulässige fremde Hilfe erstellt wurde. Alle verwendeten Quellen sind in der Arbeit so aufgeführt, dass Art und Umfang der Verwendung nachvollziehbar sind.

Dortmund, 30. August 2021

Kai Lawrence

