



## Bachelorthesis

# Entwicklung einer kamerabasierten Detektion und Verfolgung linienförmiger Objekte am Meeresboden

### **Prüfling:**

Name: Oliver Gruhlke ([ogruhlke@tzi.de](mailto:ogruhlke@tzi.de))

Matrikelnummer: 278736

### **Erstprüfer:**

Prof. Dr. Frank Kirchner ([Frank.Kirchner@dfki.de](mailto:Frank.Kirchner@dfki.de))

### **Zweitprüfer:**

Dr.-Ing. Dipl.-Inform. Thomas Röfer ([Thomas.Roefer@dfki.de](mailto:Thomas.Roefer@dfki.de))

### **Betreuer:**

Christopher Gaudig ([Christopher.Gaudig@dfki.de](mailto:Christopher.Gaudig@dfki.de))

Max Abildgaard ([Max.Abildgaard@atlas-elektronik.com](mailto:Max.Abildgaard@atlas-elektronik.com))

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel verfasst zu haben. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Ich bestätige außerdem, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

---

Ort, Datum

---

Oliver Gruhlke

## Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Grundidee . . . . .	1
1.3. Aufbau der Arbeit . . . . .	2
<b>2. Grundlagen</b>	<b>3</b>
2.1. Simulationsumgebung . . . . .	3
2.2. AUV-Simulation . . . . .	3
2.3. Koordinatensysteme . . . . .	4
2.3.1. Bild und Kamera . . . . .	4
2.3.2. Body . . . . .	5
2.3.3. MATLAB und VRML (Virtual reality modeling language) (World)	7
2.4. Eingesetzte Software . . . . .	7
<b>3. State of the Art</b>	<b>9</b>
3.1. Objekterkennung . . . . .	9
3.1.1. Linienerkennung . . . . .	9
3.1.2. Andere Ansätze . . . . .	12
3.2. Schätzverfahren . . . . .	16
3.2.1. Kalman-Filter . . . . .	16
3.2.2. Regressionsverfahren . . . . .	17
<b>4. Lösungsansatz</b>	<b>20</b>
4.1. Simulationserweiterung . . . . .	22
4.1.1. Steuerung . . . . .	22
4.1.2. Kamerabilder . . . . .	23
4.2. Transformation . . . . .	25
4.2.1. Bild zu Kamera . . . . .	26
4.2.2. Kamera zu Body . . . . .	26
4.2.3. Body zu Welt . . . . .	27
4.2.4. Welt zu VRML . . . . .	28
4.3. Objekterkennung . . . . .	29
4.3.1. Binärbild mit Template . . . . .	29
4.3.2. RANSAC auf Binärbild . . . . .	33
4.4. Schätzverfahren . . . . .	36

---

<b>5. Tests und Evaluation</b>	<b>40</b>
5.1. Tests Objekterkennung . . . . .	41
5.2. Testläufe . . . . .	49
5.2.1. Gerader Verlauf . . . . .	49
5.2.2. Kurve . . . . .	51
5.2.3. Ellipsen . . . . .	57
5.2.4. Schlechte Sichtbedingungen . . . . .	60
5.3. Parametrisierung . . . . .	63
5.3.1. Einpendeln . . . . .	65
5.4. Systematischer Fehler . . . . .	66
5.5. Laufzeittests . . . . .	67
<b>6. Fazit &amp; Ausblick</b>	<b>68</b>
6.1. Fazit . . . . .	68
6.2. Ausblick . . . . .	69
6.2.1. Parametrisierung . . . . .	69
6.2.2. Regressionsverfahren . . . . .	69
6.2.3. Integration in ein Realsystem . . . . .	70
<b>A. Literaturverzeichnis</b>	<b>71</b>
<b>B. Abbildungsverzeichnis</b>	<b>73</b>
<b>C. Gleichungsverzeichnis</b>	<b>75</b>
<b>D. Listingverzeichnis</b>	<b>76</b>
<b>E. Abkürzungsverzeichnis</b>	<b>77</b>
<b>F. Glossar</b>	<b>78</b>
<b>G. Anhang</b>	<b>79</b>
G.1. Objekterkennung weitere Tests . . . . .	79

## 1. Einleitung

Diese Bachelorarbeit behandelt die Entwicklung einer Detektion und Verfolgung von Objekten am Meeresboden. Es wird eine Komponente entwickelt, die einem AUV (Autonomous underwater vehicle) eben dies ermöglicht. Diese Komponente beinhaltet die Erkennung der Objekte, die Schätzung des Objektverlaufs und die Steuerung des AUVs anhand von Wegpunkten.

### 1.1. Motivation

Die Motivation für die Arbeit entspringt der Idee, Kameradaten in einem Robotersystem dezentral zu verarbeiten. So kann die Verarbeitung der Bilddaten auf einem eigenen Prozessor innerhalb der Kamerakomponente umgesetzt werden. Somit würden dem System keine rohen Daten, sondern verwertbare Informationen geliefert.

Ein mögliches Beispiel hierfür ist ein Missionsszenario, in dem ein AUV einem Objekt am Meeresboden autonom folgen soll. So können Strukturen (Kabel, Pipelines etc.) von einem AUV untersucht werden, ohne dass ein Pilot das Fahrzeug steuern und überwachen muss.

Die Kamerakomponente kann hierbei eine Lageposition des Objektes liefern, anstatt eines Bildes, das zentral verarbeitet werden müsste. Die Software für dieses Szenario wird in dieser Arbeit entwickelt.

### 1.2. Grundidee

Die zu entwickelnde Komponente soll dem AUV eine verlässliche Information über die Objektlage liefern. Hierfür gilt es zwei Hauptprobleme zu lösen.

Erstens ist dies die Erkennung der Objekte im Bild. In dieser Arbeit beschränke ich mich auf linienförmige Objekte. Die Bilderkennung soll Position und Ausrichtung des Objektes relativ zum AUV bestimmen können. Das zweite Problem ist die Bestimmung relevanter Daten, wenn die Bilderkennung kein Objekt finden kann, sei es durch zeitweises Versagen der Algorithmik, nicht verwertbare Rohdaten oder durch Unsichtbarkeit der Objekte, wenn diese zum Beispiel von Sand überdeckt sind. In diesem Fall soll ein Schätzverfahren auf Basis der vorherigen Positionsdaten auch weiterhin die ungefähre Objektlage liefern. Als Hilfe für die Algorithmen soll es möglich sein, a-priori Wissen über die Objekte, wie zum Beispiel den Durchmesser des Objektes, angeben zu können.

### **1.3. Aufbau der Arbeit**

Zunächst wird grundlegend die Simulationsumgebung und das AUV beschreiben. Danach werden verschiedene Herangehensweisen ähnlicher Arbeiten vorgestellt und die Eignung für das zu lösende Problem diskutiert. Im dritten Teil wird dann die gewählte und umgesetzte Lösung beschrieben. Zum Schluss folgen dann Ausführungen über die durchgeführten Tests und ein Fazit, sowie ein Ausblick auf weitere Arbeiten.

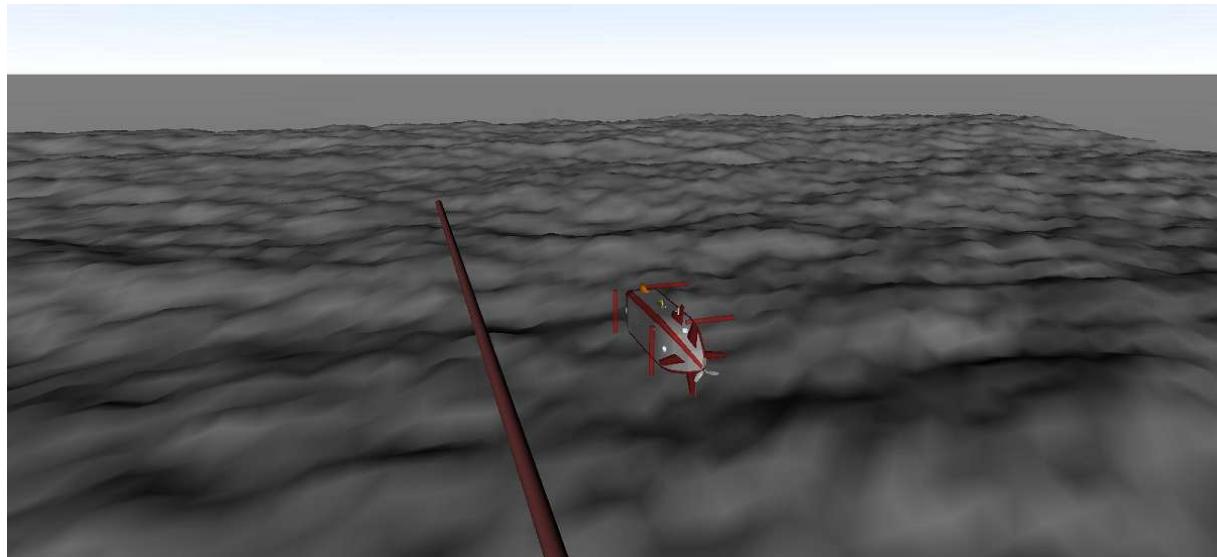
## 2. Grundlagen

### 2.1. Simulationsumgebung

Die Simulationsumgebung wurde von der ATLAS ELEKTRONIK GmbH entwickelt und wird für diese Arbeit zur Verfügung gestellt [Abb. 2.1]. In der Simulation wird die Fahrzeugsensorik und Aktuatorik simuliert (siehe Kapitel 2.2). Außerdem wird noch eine grundlegende Wasserbewegung erzeugt, die durch Parametrisierung verändert werden kann.

Die Umgebung an sich besteht aus *wrl*-Dateien, in denen die Welt mit *VRML* beschrieben wird. Diese einzelnen Dateien werden in der *main.wrl* zusammengefügt. Bereits vorhanden sind *wrl*-Dateien für das AUV, in denen die Realvorlage maßstabsgetreu nachgebildet wird sowie ein Generator zum Erstellen zufallsgenerierter Meeresböden in *wrl*. Die Objekte, die in der Arbeit verfolgt werden sollen, wurden in *VRML* modelliert und dann als Unterknoten in der *main.wrl* hinzugefügt.

Die Simulation basiert auf der MATLAB Simulink 3D Animation Toolbox, somit stehen auch weitere MATLAB -Toolboxen zur Verfügung.



**Abbildung 2.1:** Screenshot der Simulationsumgebung mit dem AUV, einem Testobjekt und dem generierten Meeresboden

### 2.2. AUV-Simulation

Das simulierte AUV wurde von der ATLAS ELEKTRONIK GmbH auf Grundlage eines der eigenen AUVs entwickelt. Zu dieser Simulation gehören die bereits erwähnten *wrl* Dateien sowie eine Simulation der Fahrzeugaktuatorik und -sensorik in MATLAB Simulink.

Es werden die für die Steuerung benötigte Schnittstelle in Form von Wegpunkten ( $x$  und  $y$  Koordinaten) und eine Schnittstelle für die innere Sensorik des AUVs bereitgestellt. Die für diese Arbeit wichtigen Informationen aus der inneren Sensorik bestehen aus der Pose des AUVs in der Welt in Form von geografischen Koordinaten, der Höhe über dem Meeresboden und den Roll-, Pitch- und Yaw-Werten.

Für die Steuerung wird ein **Waypoint Controller** verwendet, der das Fahrzeug auf einer Linie zwischen einem neuen und einem alten Wegpunkten führt. Für die Höhenkontrolle werden zwei Steuerungsmodi zur Verfügung gestellt. Zum einen die Fahrt auf Tiefe unter der Wasseroberfläche oder Höhe über Meeresboden. Für diese Arbeit wird die Fahrt auf Höhe über dem Meeresboden gewählt, da die Transformation von Pixelkoordinaten in Kamerakoordinaten am zuverlässigsten in dem Abstand zum Objekt funktioniert, in dem auch die Kamerakalibrierung durchgeführt wurde. Änderungen der Höhe können zu leichten Fehlern in der Positionsbestimmung führen. Jedoch sind diese Fehler bei realistischen Höhenunterschieden von einigen Metern nicht ausschlaggebend für die Ergebnisse der Arbeit.

Am Bug des AUVs befindet sich eine Kamera, die zentral nach unten ausgerichtet ist. Das Sichtfeld beträgt dabei  $45^\circ$  bei einer Auflösung von 640x480 Pixeln.

## 2.3. Koordinatensysteme

Im Folgenden werde ich die Koordinatensysteme beschreiben, in denen Koordinaten angegeben werden. Hierbei wird von einer Tangentialebene an der WGS84-Kugel<sup>1</sup> verschoben auf den Meeresboden ausgegangen, da nur hinreichend kleine Operationsgebiete des AUVs betrachtet werden, sodass die Erdkrümmung keine Auswirkung hat.

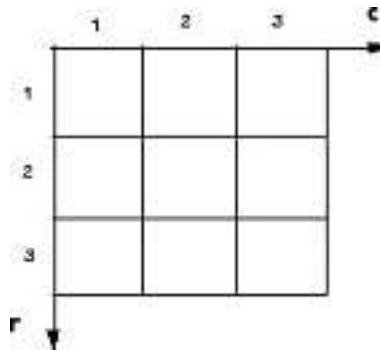
### 2.3.1. Bild und Kamera

Das Bildkoordinatensystem [Abb. 2.2] beschreibt die Anordnung der Pixel im Bild als 2D-Koordinaten. Der Ursprung ist immer die linke obere Bildecke. Ich gehe davon aus, dass das Bildkoordinatensystem immer auf der Meeresbodenebene liegt. Diese Annahme ist wichtig für die Transformationen [Kapitel 4.2].

schönere  
grafik  
evtl  
selber  
machen

---

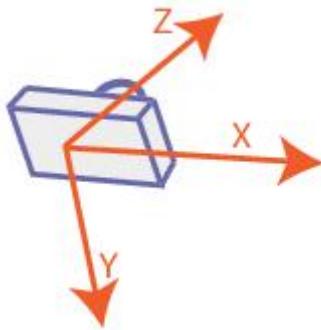
<sup>1</sup> <https://confluence.qps.nl/pages/viewpage.action?pageId=29855173>



**Abbildung 2.2:** Anordnung der 2D-Pixelkoordinaten. Ursprung liegt in der linken oberen Bilddecke. Die X-Achse bildet die Bildspalten und die Y-Achse die Bildzeilen ab.

Das Kamerakoordinatensystem [Abb. 2.3] beschreibt das dreidimensionale Koordinatensystem mit Ursprung im Mittelpunkt der Kameralinse. Die Kamera befindet sich 25 cm unter und 1,3 m vor dem Fahrzeugmittelpunkt.

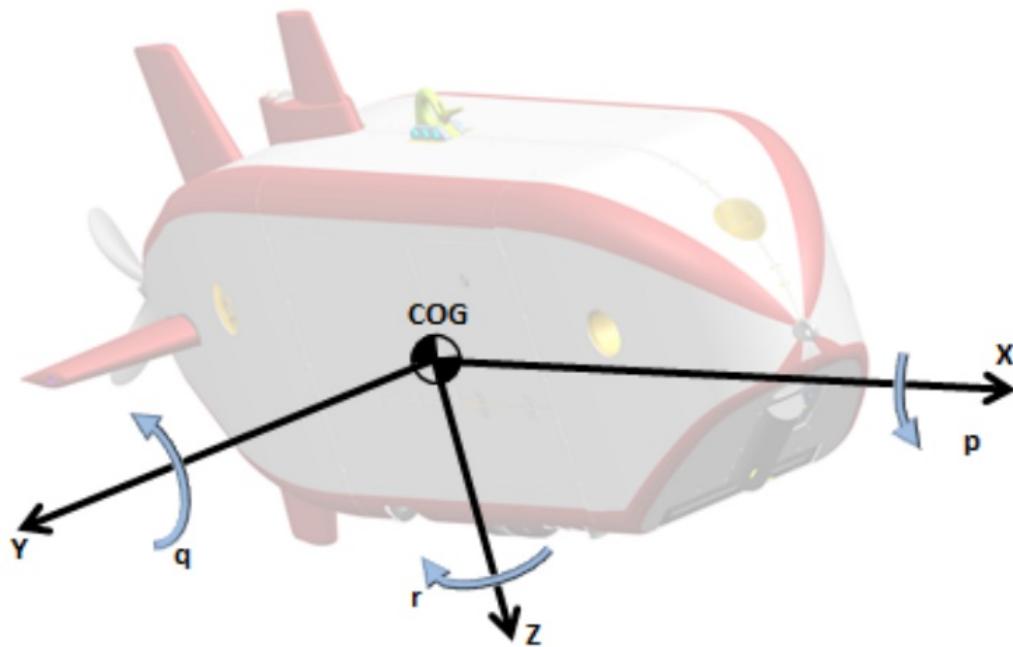
bessere  
qualität



**Abbildung 2.3:** Das Kamerakoordinatensystem. Ursprung des Systems liegt im Mittelpunkt der Kameralinse. Die X-Achse bildet die Bildspalten und die Y-Achse die Bildzeilen im Raum ab. Die Z-Achse zeigt in Blickrichtung der Kamera.

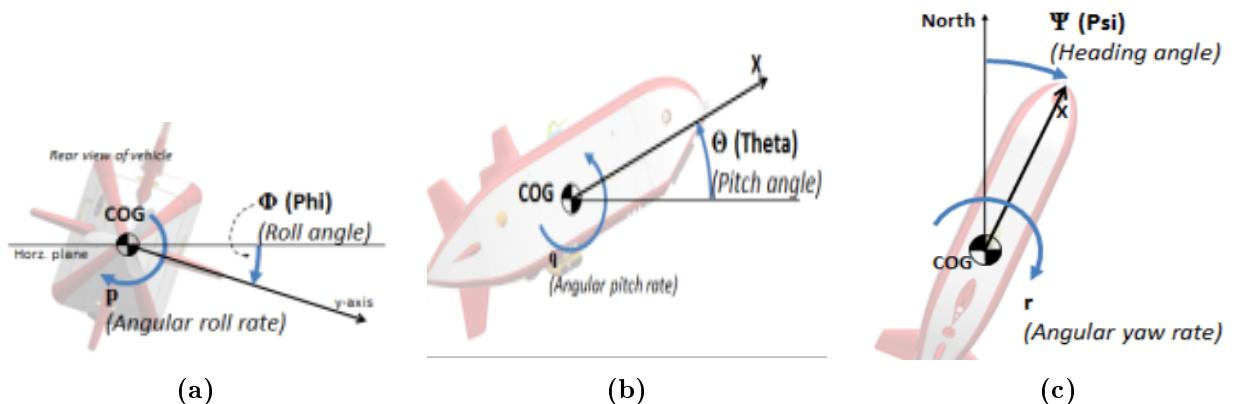
### 2.3.2. Body

Das Body-Koordinatensystem beschreibt das Koordinatensystem relativ zum AUV-Mittelpunkt [Abb. 2.4]. Der Ursprung wird hierbei durch den Massenschwerpunkt des AUVs bestimmt. Das Koordinatensystem entspricht dem klassischen nautischen Koordinatensystem, dem North-East-Down Koordinatensystem (vgl. Kapitel 2 in *Unmanned rotorcraft systems* [5])



**Abbildung 2.4:** Das Body-Koordinatensystem mit Massenschwerpunkt (*cog*) des AUVs. Die X-Achse zeigt frontal voraus, die Y-Achse Richtung Steuerbord und die Z-Achse zeigt nach unten.  $p$   $r$  und  $q$  beschreiben die Neigungswinkel und Rotationsrichtung an den jeweiligen Achsen (vgl. Abb. 2.5)

Die Neigungswinkel (Roll-Pitch-Yaw) werden wie in [Abb. 2.5] im Body Koordinatensystem angegeben.



**Abbildung 2.5:** Die Neigungswinkel am AUV. Phi beschreibt den Roll Winkel um den Massenschwerpunkt (*cog*) und  $p$  die dazugehörige *roll rate*, Theta beschreibt den Pitch Winkel um den Massenschwerpunkt (*cog*) und  $q$  die dazugehörige *roll rate* und Psi beschreibt die Ausrichtung des AUVs im Bezug zur Nordrichtung.

### 2.3.3. MATLAB und VRML (World)

Abbildung 2.6 zeigt die Koordinatensysteme der MATLAB -Grafikbibliothek und der VRML -Bibliothek. Zum Berechnen der Wegpunkte für die Steuerung des AUVs muss eine Pose in das VRML-Koordinatensystem transformiert werden. Der Ursprung beider Systeme liegt im Mittelpunkt der Simulationsumgebung.

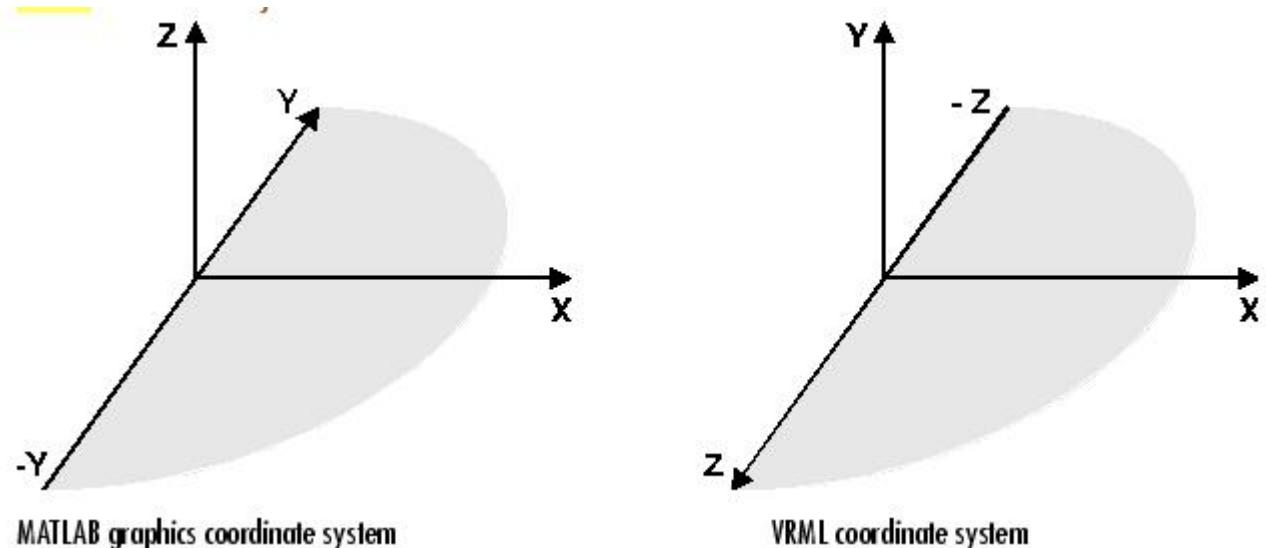


Abbildung 2.6: MATLAB und VRML Koordinatensystem. Der Ursprung beider Systeme definieren den Mittelpunkt der Simulationsumgebung. Die X-Achsen zeigen in beiden Fällen Richtung Norden. Im VRML-System zeigt die Z-Achse Richtung Ost, im MATLAB -System zeigt dementsprechend die negative Y-Achse Richtung Ost. In beiden Systemen liegt die Grundfläche auf dem Meeresboden. Somit zeigt die Z- bzw. Y-Achse vom Meeresboden aufwärts.

## 2.4. Eingesetzte Software

Neben den bereits erwähnten MATLAB -Bibliotheken *Simulink*, *Graphics* und *3D Animation* werden in dieser Arbeit noch weitere Bibliotheken verwendet.

Grundlegende geometrische Berechnungen, zum Beispiel Transformationen in 2D und 3D oder auch Distanzberechnungen von Punkten werden mithilfe der freien Bibliotheken *geom2d*<sup>2</sup> und *geom3d*<sup>3</sup> durchgeführt.

Das Schätzverfahren nutzt die *Optimization-Toolbox*<sup>4</sup>, die Lösungen für verschiedene Minimierungs-, Maximierungs- und Optimierungsprobleme liefert.

Für die Kamerakalibrierung wurde die *Computer Vision System Toolbox* genutzt. Die

schöner  
evtl  
mit tik

<sup>2</sup> <https://de.mathworks.com/matlabcentral/fileexchange/7844-geom2d>

<sup>3</sup> <https://de.mathworks.com/matlabcentral/fileexchange/24484-geom3d>

<sup>4</sup> <https://de.mathworks.com/products/optimization.html>

Toolbox bietet die einfach zu bedienende *Camera Calibration App* mit der die intrinsischen Parameter bestimmt werden können. Hilfestellung lieferte hierbei das dazugehörige Tutorial<sup>5</sup>.

---

<sup>5</sup> <https://de.mathworks.com/help/vision/ug/single-camera-calibrator-app.html>

## 3. State of the Art

In diesem Kapitel werden verwandte Arbeiten zum Thema dieser Arbeit vorgestellt. Außerdem wird erläutert, inwiefern die vorgestellten Arbeiten sich zum Lösen der gestellten Aufgabe eignen.

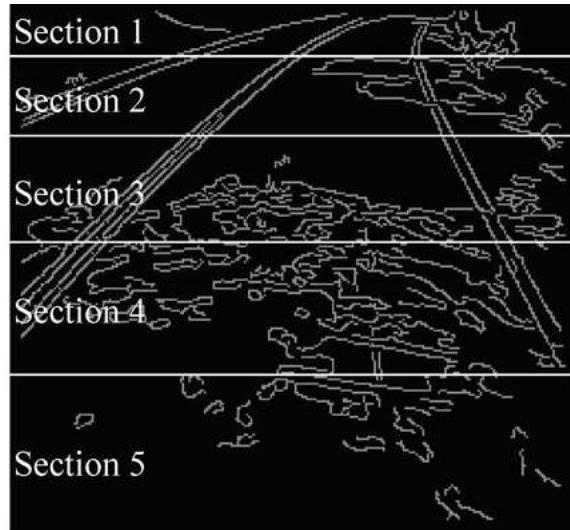
Die Arbeit wird in die Oberpunkte *Objekterkennung*, zum Detektieren der Objekte und Bestimmen der Lage, und das *Schätzverfahren*, zur Bestimmung und Vorhersage des Objektverlaufs unterteilt.

### 3.1. Objekterkennung

Bei der Objekterkennung unterscheide ich grob zwischen Ansätzen, die Objekte aufgrund von Linien und deren Beziehungen detektieren und solchen, die andere Objekteigenschaften nutzen.

#### 3.1.1. Linienerkennung

Linienförmige Objekte heben sich unter anderem durch zwei annähernd parallel verlaufende Kanten vom Hintergrund ab. Ein verwandtes Problem hierzu ist die Detektion eines Straßenverlaufs, der sich ebenso durch zwei fast parallele Linien (den Fahrbahnmarkierungen) auszeichnet [Abb. 3.1].



**Abbildung 3.1:** Ein typischer Straßenverlauf mit entsprechendem Kantenbild und dem An-  
satz der vertikalen Unterteilung in mehrere Segmente aus Lane detection  
and tracking using B-Snake[18].

Viele Arbeiten zum Tracking von Fahrbahnmarkierungen basieren auf der Kantenextrak-

### 3. State of the Art

---

tion ([17],[2], [12],[18]). Aus dem Kantenbild werden dann die Charakteristiken der Straße extrahiert. Hierfür wird oft versucht, mithilfe von LCFs (lane-curve-function) die Markierungen zu finden. Dies eignet sich besonders gut, um den typischen Straßenverlauf in Form von leichten Kurven zu erkennen. Eine LCF stellt ein Modell des gekrümmten Straßenverlaufs dar, das sich durch Parameter bestimmen lässt. Hierbei unterteilt sich die LCF in zwei Breiche. Zum einen einen geraden Beginn, gefolgt von einem gekrümmten Bereich.

Jong Woung Park et al.[12] untersuchen Straßenbilder mithilfe von LCFs und einem Krümmungsindex, der die Richtung und Stärke der Krümmung beeinflusst. Bestimmt wird dabei, ob es sich im Bild um eine gerade, nach links oder nach rechts gebogene Straße handelt.

Die Methode setzt dabei voraus, dass die geraden Anfänge der Straßenmarkierungen nah an der Kamera sowie der *Vanishing point* bereits erkannt wurden. Der *Vanishing point* ist der Punkt, an dem sich parallele Linien projiziert auf die Bildebene schneiden (vgl. Abb. 3.4). Aus diesem Wissen werden zunächst die von der Krümmung unabhängigen Parameter der LCF berechnet. Der Bereich weiter entfernt von der Kamera bestimmt dann die Krümmung der gesuchten LCF. Hierfür wird für die verschiedenen Krümmungen eine *region of interest* (ROI) um die LCF definiert. Innerhalb der ROI wird dann mithilfe von Kantenerkennung evaluiert, welcher Krümmungsgrad am ehesten den Fahrbahnverlauf abbildet. Dieses Verfahren ist in Abbildung 3.2 zu sehen, in dem die ROI der Linkskurve das beste Ergebnis liefert.



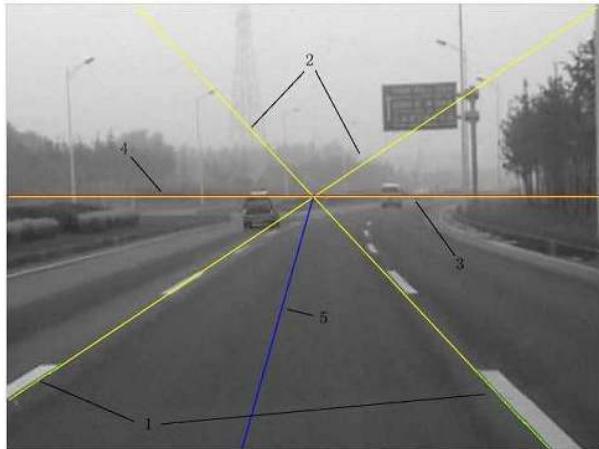
**Abbildung 3.2:** Detektion des Straßenverlaufs mithilfe von LCF und ROI. Die schwarz umrahmte Fläche bestimmt die ROI. Die Kurven der drei Bilder werden durch verschiedene Krümmungindizes der LCF erzeugt. Die von der Krümmung unabhängigen Parameter der LCF sind in allen Bildern gleich.

Alternativ können auch gerade Linien im Kantenbild gesucht werden. In A Real-time Lane Detection Algorithm Based on a Hyperbola-Pair Model [6] suchen Chen et al. mithilfe des RANSAC-Algorithmus auf einem Kantenbild zwei gerade Linien. Auf Grundlage der gefundenen Linien wird anschließend die Mitte der Straße und in weiteren Schrit-

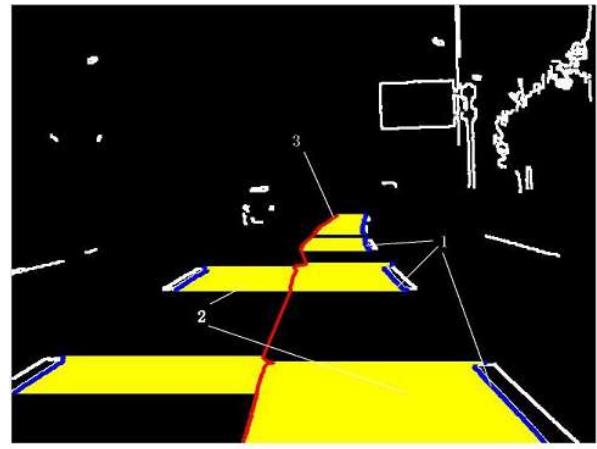
3. State of the Art

---

ten der Straßenverlauf bestimmt [Abb. 3.3].



(a) Ursprungsbild mit Linien- und Horizontmarkierungen

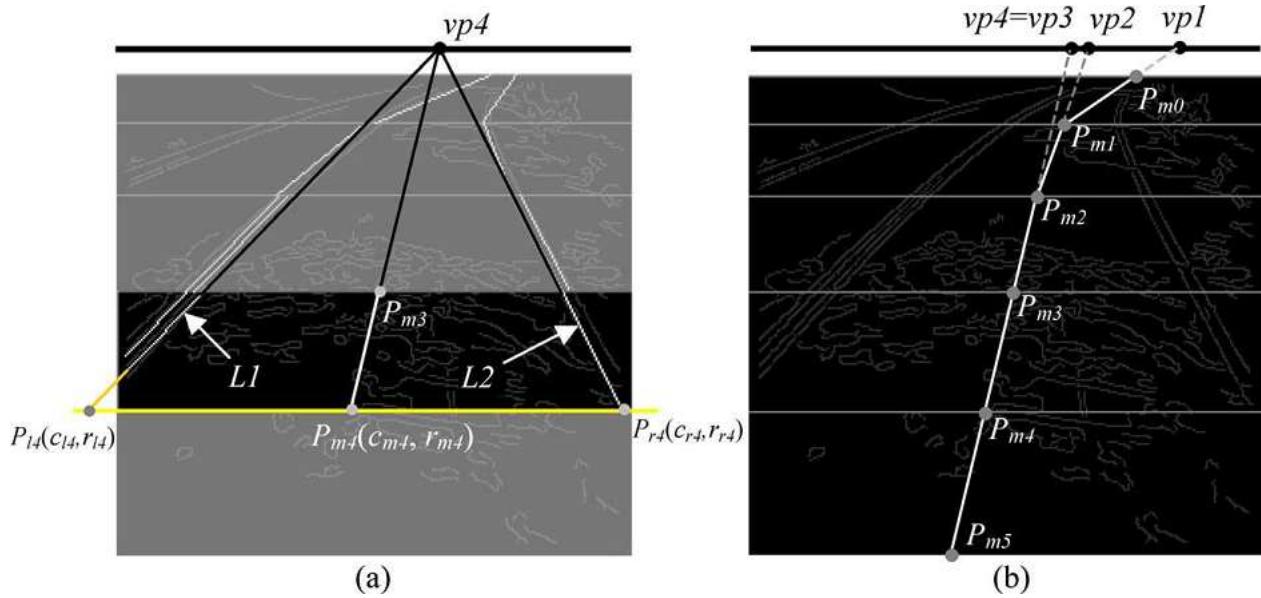


(b) Kantenbild mit Bereichen zwischen zwei Linien und Mitte der Straße

**Abbildung 3.3:** Erkennungsprozess aus *A Real-time Lane Detection Algorithm Based on a Hyperbola-Pair Model*. Im rechten Bild ist in rot die detektierte Straßenmitte gekennzeichnet.

Einen ähnlichen Ansatz verfolgen auch Wang et al. für ihre Detektion in *Lane detection and tracking using B-Snake*[18]. Besonders hierbei ist, dass das Eingabebild vertikal in fünf Segmente eingeteilt wird (siehe Ab. 3.1). In jedem Segment werden mithilfe der *Hough Transformation* Linien erkannt und wie bei Chen die Mitte der Straße bestimmt. Außerdem wird für jedes Segment auch der *Vanishing point* bestimmt. Der *Vanishing point* eines Segmentes bestimmt die Richtung der detektierten Straße (siehe Abb. 3.4). Die Arbeit bietet eine Möglichkeit innerhalb eines Bildes eine Kurve zu detektieren, obwohl nur gerade Linien gesucht wurden.

### 3. State of the Art



**Abbildung 3.4:** Ergebnis der Straßenverlaufsdetektion mithilfe von *Hough Transformation* nach Wang et al.. Links werden die erkannten Straßenmarkierungen der Segmente angedeutet und der *Vanishing point* des vierten Segmentes. Rechts ist die final detektierte Mittellinie der Straße über alle Segmente.

Aus den von mir betrachteten Arbeiten geht hervor, dass mithilfe von LCFs kurvige Straßenverläufe gut erkannt werden können. Da Kameras in den betrachteten Arbeiten jedoch nach vorne ausgerichtet sind, um den Straßenverlauf möglichst weit zu erkennen, unterscheidet sich dieser Ansatz in einem wesentlichen Punkt vom Unterwasserszenario. In den meisten Einsatzgebieten würde eine nach vorn ausgerichtete Kamera keine Objekte am Meeresboden sehen können. Der Höhenunterschied von der Kamera zum Boden wäre zu groß um Objekte nah am Fahrzeug zu sehen und in der Entfernung sind oftmals, bedingt durch schlechte Sichtverhältnisse, keine Objekte erkennbar. Aus diesen Gründen sollte die Kamera gerade nach unten oder leicht nach vorne geneigt ausgerichtet sein. Aus dieser Ausrichtung resultiert jedoch, dass der betrachtete Bereich weitaus kleiner ist und die meisten Objekte nur eine sehr leichte Krümmung im Bild aufweisen. In solch kleinen Bereichen reicht eine Linienerkennung aus.

#### 3.1.2. Andere Ansätze

Im CSurvey-Projekt [1] beleuchten Albize et al. eine Pipeline mit einem Linienlaser und erkennen die Linie im Kamerabild über den Helligkeitswert. Für die Detektion wird für jede Bildzeile ein Helligkeitsmaximum gesucht. Hierfür wird ein Template, das den typischen Helligkeitsunterschied der beleuchteten Pipeline zum Hintergrund abbildet, auf die Zeile angewandt [Abb. 3.5].

Diese Detektion wurde vor allem in verschiedenen Entfernungen zum Boden und verschiedenen Trübungsgraden des Wassers getestet. Ein Ergebnis der Arbeit ist, dass mit dem

Template auch bei schlechter Sicht die Pipeline noch erkennbar ist. Jedoch wird ab einem bestimmten Trübheitsgrad das gesamte Bild zu hell für eine Erkennung, da die Reflexion des Laserlichts im trüben Wasser viel zu groß ist.

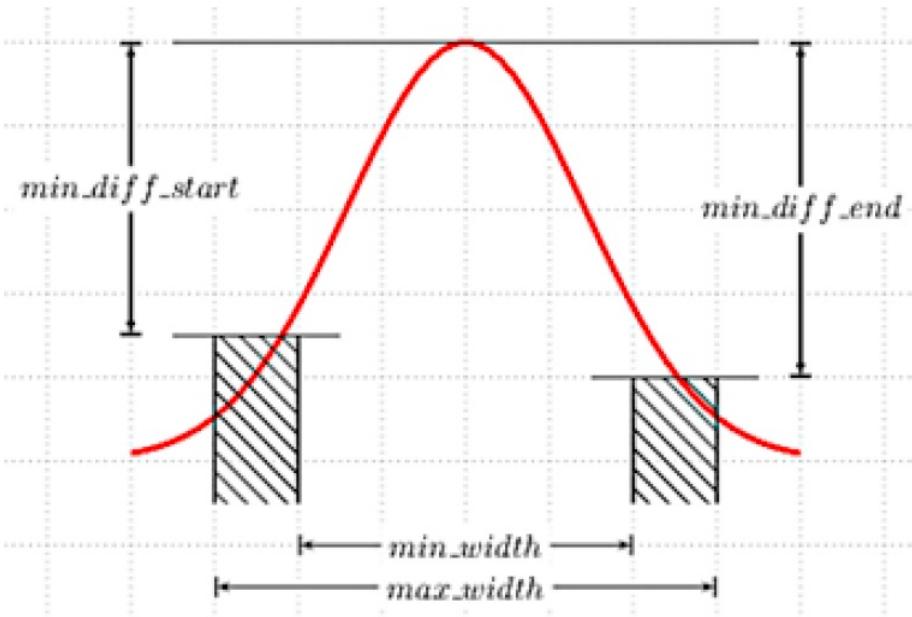


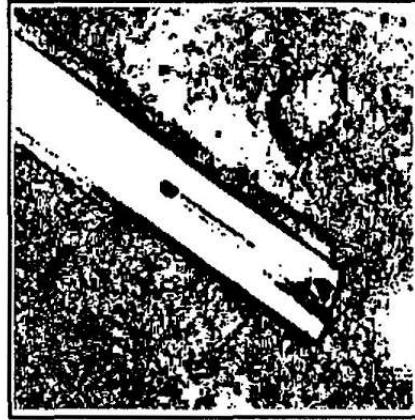
Abbildung 3.5: Template zur Detektion auf Helligkeitsdaten aus *CSurvey*.

Im Avalon-Projekt [7] wird eine Pipeline mit einem Farbfilter im HSV-Farbraum detektiert. Dieser Filter erzeugt zuerst ein Binärbild. Auf diesem Binärbild wird dann mit einem Canny Edge Detector ein Kantenbild generiert. Im Kantenbild wird mithilfe der Hough-Transformation nach Linien gesucht.

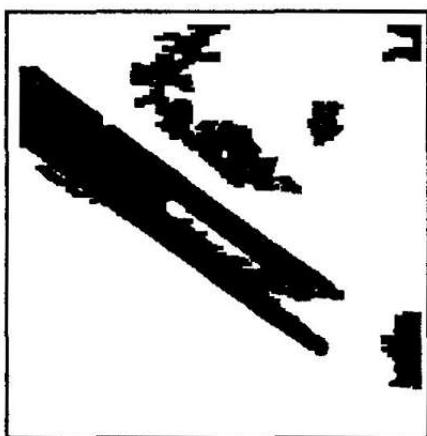
In Simple vision tracking of pipelines for an autonomous underwater vehicle[9] wird ähnlich wie bei den Linienerkennungsansätzen eine Kantenerkennung durchgeführt. Im Kantenbild werden dann durch Segmentierung Regionen definiert, auf welche dann umschließende Rechtecke gelegt werden. Aus diesen Rechtecken lässt sich dann die gesuchte Pipeline bestimmen.



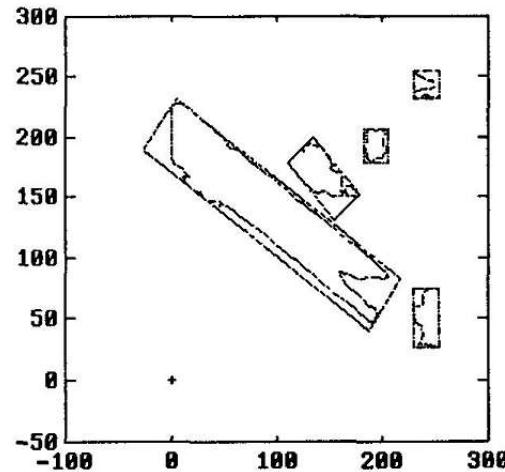
(a) Originalbild nach Kontrastverstärkung



(b) Kantenbild durch Sobel-Filter



(c) Segmentiertes Bild



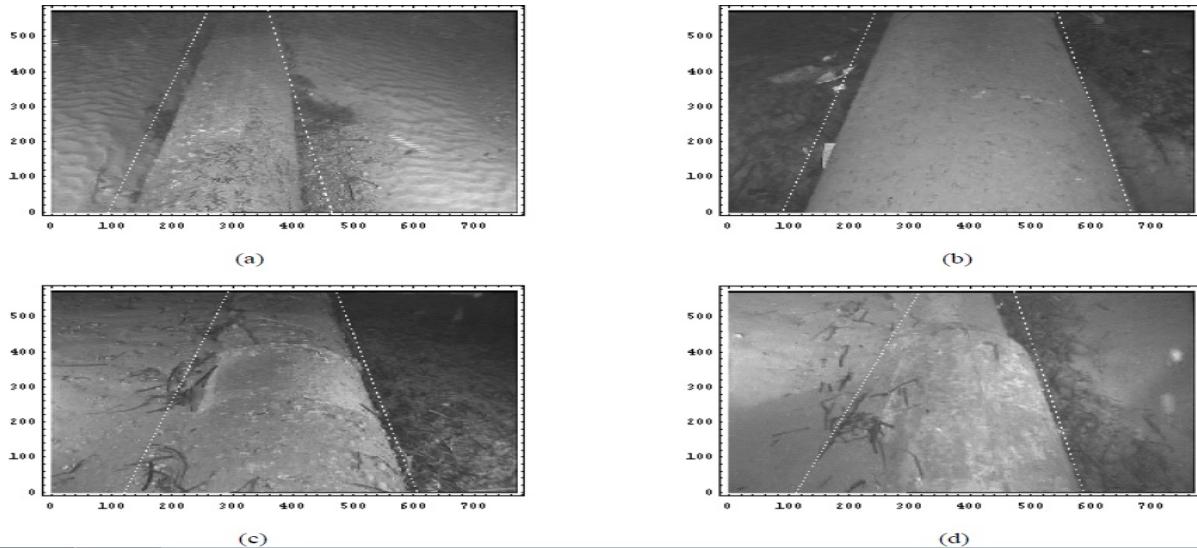
(d) Erkannte Rechtecke

**Abbildung 3.6:** Einzelschritte der Erkennung aus *Simple vision tracking of pipelines for an autonomous underwater vehicle*. Die Pipeline wird hierbei durch das Untersuchen des Kantenbildes auf rechteckige Strukturen detektiert.

Foresti et al. detektieren in *A Vision Based System for Object Detection in Underwater Images*[8] Unterwasserpipelines mithilfe eines neuronalen Netzes. In Abbildung 3.7 ist zu sehen, dass die Pipelines sehr gut erkannt wurden. Selbst bei vom Sand verdeckten Pipelines (3.7c und 3.7d) werden die Kanten der Pipelines noch richtig bestimmt.

3. State of the Art

---



**Abbildung 3.7:** Pipelines erkannt mithilfe eines neuronalen Netzwerks aus *A Vision Based System for Object Detection in Underwater Images*[8]. Es werden selbst verdeckte Pipelines sehr gut erkannt.

Trotz dieser guten Ergebnisse habe ich mich gegen ein neuronales Netz entschieden. Zum einen gab es zu Beginn der Arbeit keine geeigneten Trainingsbilder aus der Simulationsumgebung, um das Netz zu trainieren. Außerdem ist ein neuronales Netz stets als *black box* zu betrachten und die Detektion ist nicht eindeutig nachvollziehbar. Neben diesen Faktoren ist auch der Berechnungsaufwand für ein neuronales Netz höher einzuschätzen, als bei anderen Ansätzen.

Aus diesen Gründen habe ich mich für einen leichter zu implementierenden, klassischeren Ansatz der Bildverarbeitung entschieden.

Die auf Linienerkennung basierenden Ansätze eignen sich sehr gut, solange klare Kanten im Bild erkennbar sind. Im Unterwasserszenario setzt dies gute Sicht- und Lichtverhältnisse voraus. Außerdem würden vom Meeresboden verdeckte Objekte keine oder sehr kurvige Kanten ergeben, in denen die vorgestellten Ansätze keine Ergebnisse liefern würde. Beide Voraussetzungen sind im Unterwasserbereich nicht erfüllbar, weswegen in dieser Arbeit ein anderer Ansatz gewählt wurde.

Sehr gut eignet sich ein Helligkeitsbasierter Ansatz. Wie im *CSurvey*-Projekt[1] gezeigt, kann ein solcher Ansatz selbst unter schlechten Sichtbedingungen noch gute Ergebnisse liefern. Es ist zu erwarten, dass auch in der verwendeten Simulationsumgebung gute Resultate erzielt werden können.

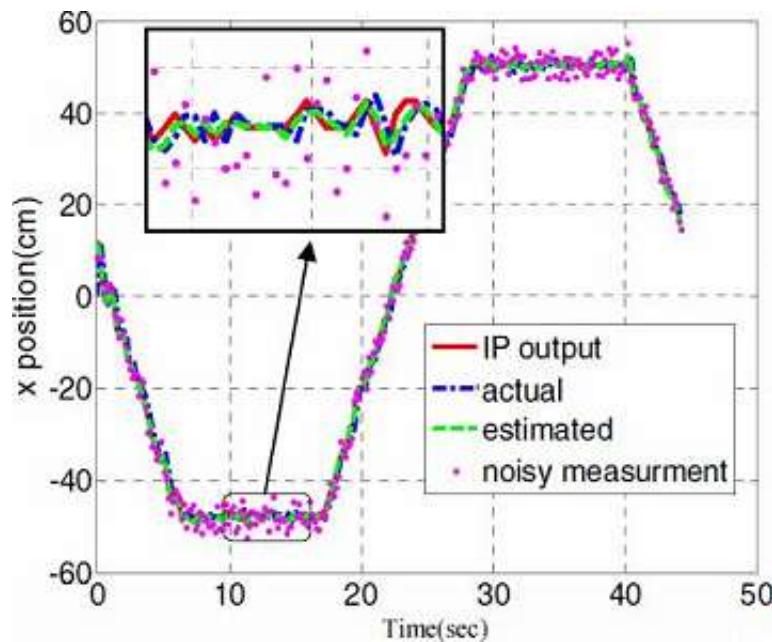
## 3.2. Schätzverfahren

Wie in der Einleitung beschrieben muss ein Schätzverfahren entwickelt werden, um dem Objektverlauf optimal folgen zu können. Das Verfahren muss auf der Ausgabe der Objekterkennung aufsetzen.

### 3.2.1. Kalman-Filter

Ein Kalman-Filter ist ein Ansatz, um verrauschte Messwerte zu verbessern und auch ausbleibende Messungen auszugleichen. Der Kalman-Filter basiert auf einem linearen Zustand wie zum Beispiel der Pose und Posenänderung eines Roboters. In jedem Zeitschritt des Filters wird aufgrund des vorherigen Zustands und dem Weltmodell ein Folgezustand berechnet und dieser dann mit den aktuellen Messwerten verglichen.(vgl. *Optimal State Estimation, Chapter 5*[15]). In *Real-Time Visual Tracking of a Moving Object Using Pan and Tilt Platform: A Kalman Filter Approach*[16] nutzen Bahare Torkaman und Mohammad Farrokhi einen Kalman-Filter, um die Bewegung eines Objektes, das mit einer Kamera detektiert wird, zu verfolgen (siehe Abb. 3.8). Der Zustand des Kalman-Filters ist dabei die  $x$ - und  $y$ -Position des Objektes in der Ebene, sowie dessen aktuelle Positionsänderung. Der Zustandsübergang wird durch die Objektbewegung durchgeführt. Das Update der Messung besteht aus der erkannten Position der Objekterkennung.

In der Arbeit ist sehr gut zu sehen, wie der Kalman-Filter den Fehler der Objekterkennung nahezu halbiert.



**Abbildung 3.8:** Eine Objektbewegung wird mit einem Kalman-Filter verfolgt. Lila sind hierbei die Positionsbestimmungen aus der Bilderkennung, blau der wirkliche Objektverlauf, grün der vom Zustandsübergang des Kalman-Filters erwartete Verlauf und rot der vom Kalman-Filter korrigierte Verlauf.

Der Kalman-Filter eignet sich jedoch nicht gut für das gestellte Szenario. Um den Messfehler der Objekterkennung auszugleichen, müsste der Zustand des Kalman-Filters das zu verfolgende Objekt abbilden. Dieser Zustand sowie der benötigte Zustandsübergang sind jedoch nicht problemlos zu definieren.

Das Problem beim Definieren des Zustands liegt darin, dass das Objekt selbst keine wechselnden Zustände hat, sondern stets fest bleibt. Ein Ansatz wäre, die Position und Ausrichtung des Objektes im Zustand abzubilden. Bei einem Zustandsübergang müsste dann eine neue Position und Ausrichtung berechnet werden. Da sich das Objekt selbst nicht bewegt, müsste dieser Übergang in Abhängigkeit der Bewegung des AUVs umgesetzt werden. Es gibt jedoch keinen direkten Zusammenhang zwischen der Objektlage und der AUV Bewegung und somit kann kein Folgezustand berechnet werden. Da das Objekt keine Eigenbewegung durchführt ist die Lage stets unabhängig vom AUV. Eine von dem AUV abhängige Position wäre die nächste Position vom AUV auf dem Objektverlauf. Hierfür müsste jedoch aus den vorhandenen Daten zuerst der Objektverlauf berechnet werden, was direkt zum nächsten vorgestellten Ansatz führt.

nochmal  
checken  
was  
hier  
was ist

### 3.2.2. Regressionsverfahren

Ein weiterer Lösungsansatz für das Schätzverfahren ist die Regression. Bei der Regression wird versucht, ein parametrisierbares Modell an gegebene Daten anzupassen. Dabei wird versucht, den Fehler der Daten im Bezug zur aus dem Model generierten Kurve zu

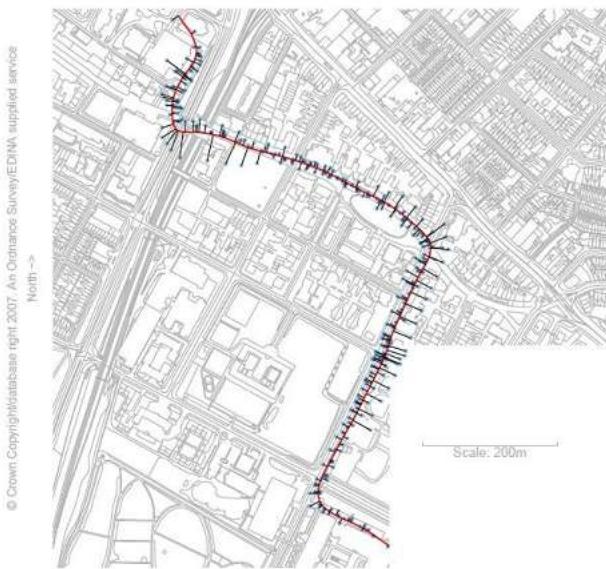
### 3. State of the Art

---

minimieren. Brundson nutzt in [Path estimation from GPS tracks\[4\]](#) einen Regressionsansatz, um aus fehlerbehafteten GPS-Daten einen Pfad durch ein Stadtgebiet genauer zu bestimmen. Die GPS-Daten ähneln den zu erwartenden Daten der Objekterkennung. Auch hier gibt es einen reellen Verlauf und fehlerbehaftete Messdaten zu beiden Seiten dieses Verlaufs.

Brundson berechnet die optimalen Modellparameter durch das Minimieren des quadrierten Fehlers jedes Punktes zur Kurve.

In der Abbildung 3.9 ist gut zu sehen, wie die Kurve nahezu den echten Pfad abbildet.



**Abbildung 3.9:** Regression angewandt auf Positionsangaben eines GPS-Empfängers aus [Path estimation from GPS tracks\[4\]](#). Die Fehler zu beiden Seiten des realen Fehlers werden gut durch die berechnete Kurve ausgeglichen

Eine sehr ähnliche Arbeit findet sich in [Autonomous Searching and Tracking of a River using an UAV\[14\]](#) von Rathinam et al.. Ziel der Arbeit ist es mithilfe eines UAVs (Unmanned Aerial Vehicle) den Verlauf eines Flusses zu bestimmen. Das Flugzeug ist mit einer Kamera zur Detektion des Flusses ausgestattet.

Aus der Objekterkennung werden GPS-Positionen des Flusses berechnet. Durch diese Daten wird dann durch Regression eine Kurve gelegt (siehe Abb. 3.10).

Im gelb gekennzeichneten Bereich des rechten Bildes ist zu sehen, dass die Kurve an dieser Stelle nicht den Flussverlauf abbildet. Im selben Bereich ist im linken Bild zu sehen, dass die Objekterkennung keine Ergebnisse lieferte. Dies liegt daran, dass das Flugzeug dem engen Flussverlauf aufgrund der Trägheit der Steuerung nicht folgen konnte. Diese Trägheit ist in ähnlichen Rahmen auch im Unterwasserszenario zu erwarten.

Bemerkenswert ist, dass im roten Bereich ebenfalls keine Ergebnisse der Objekterkennung vorhanden sind, die Kurve jedoch trotzdem genau dem Flussverlauf folgt. Dies lässt sich

3. State of the Art

---

darauf zurückzuführen, dass die letzten GPS-Daten vor der Lücke die Abzweigung der Kurve andeuten, was im gelben Bereich nicht der Fall ist.



(a) Flusspositionen nach Objekterkennung



(b) Flussverlauf nach Curve Fitting

**Abbildung 3.10:** Regression angewendet auf GPS-Daten eines Flussverlaufs aus Autonomous Searching and Tracking of a River using an UAV[14]. Im gelben Bereich wird ein fehlender Sichtkontakt zum Fluss nicht optimal abgedeckt. Im roten Bereich ist die Kurve trotz fehlender Detektion nahezu optimal abgebildet.

Das Curve Fitting Verfahren eignet sich gut für die Problemstellung der Arbeit. Die zwei vorgestellten Arbeiten zeigen, dass sowohl Fehler durch Ausreißer abgefangen werden, als auch Bereiche ohne Ergebnisse der Objekterkennung gut überbrückt werden können. Die genaue Beschreibung der eingesetzten Lösung und des von mir eingesetzten Modells folgt im nächsten Kapitel.

## 4. Lösungsansatz

In diesem Kapitel wird der von mir implementierte Lösungsansatz vorgestellt. Grundle-  
gend gehe ich zuerst auf die Erweiterung der Simulationsumgebung und die implementierte  
Transformationskette ein.

Danach folgen genauere Ausführungen zu den Kernelementen der Arbeit, der Objekter-  
kennung und dem Schätzverfahren.

Ein grundlegender Datentyp, der sich über alle Teile der Arbeit erstreckt, ist durch die  
Struktur *pointInFrame* [Listing 1] definiert. Diese Struktur bildet einen von der Objek-  
terkennung detektierten Punkt des Objektes mit seiner Position und Orientierung ab.  
Zudem wird gespeichert, in welchem Referenzframe der Punkt angegeben ist. Neben die-  
sen Positionsdaten sind zudem noch die Gütefaktoren der Objekterkennung angegeben  
(siehe hierfür Kapitel 6.2.2).

```
1 Point_In_Frame = struct;
2 Point_In_Frame.point = [0 0 0];
3 Point_In_Frame.direction = 0;
4 Point_In_Frame.peakheight = 0;
5 Point_In_Frame.area = 0;
6 Point_In_Frame.frame = frames.image
7 Point_In_Frame.numParts = 0;
8 Point_In_Frame.fitsBorder = false;
9 Point_In_Frame.relativeCount = 0;
10 Point_In_Frame.valid = false;
11 Point_In_Frame.theta = 0;
12 Point_In_Frame.phi = 0;
```

**Listing 1:** Initialisierung der *pointInFrame*-Struktur, die erkannte Punkte in verschiedenen  
Referenzkoordinatensystemen abbildet.

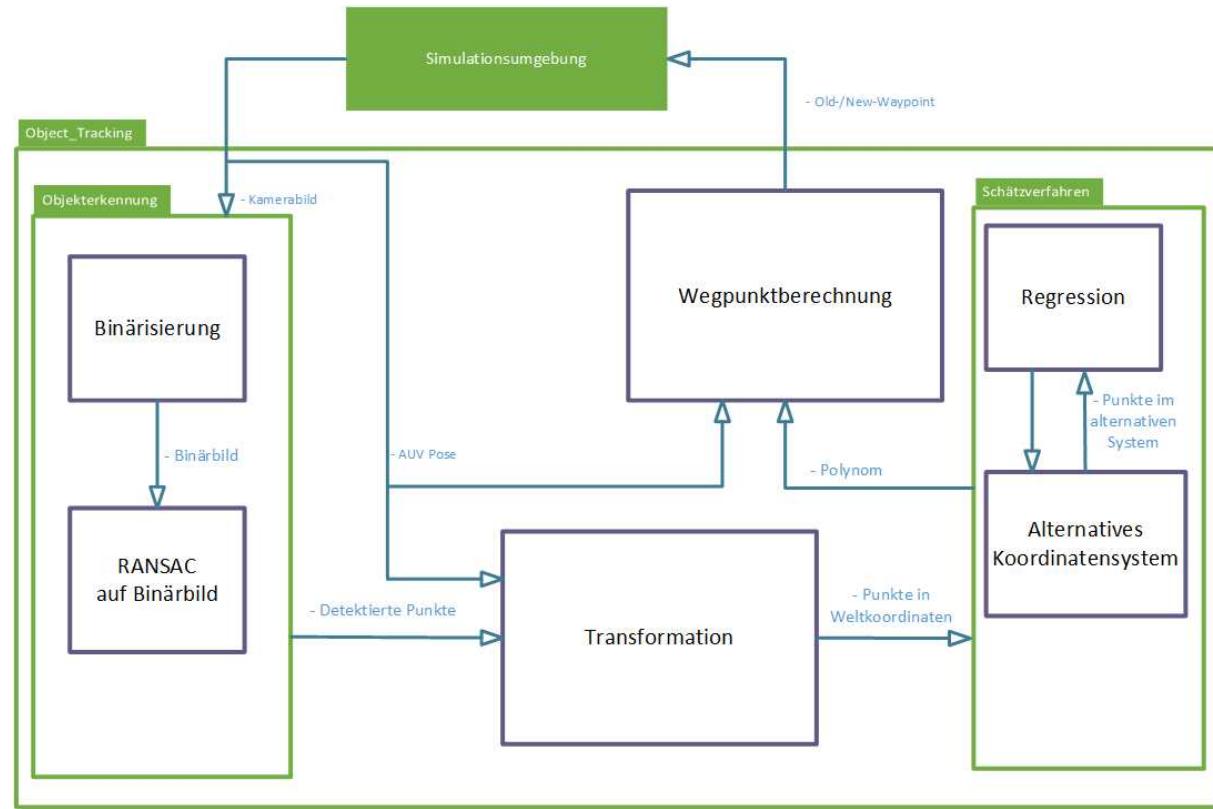


Abbildung 4.1: Systementwurf der eingesetzten Lösung. Die Simulationsumgebung

## 4.1. Simulationserweiterung

### 4.1.1. Steuerung

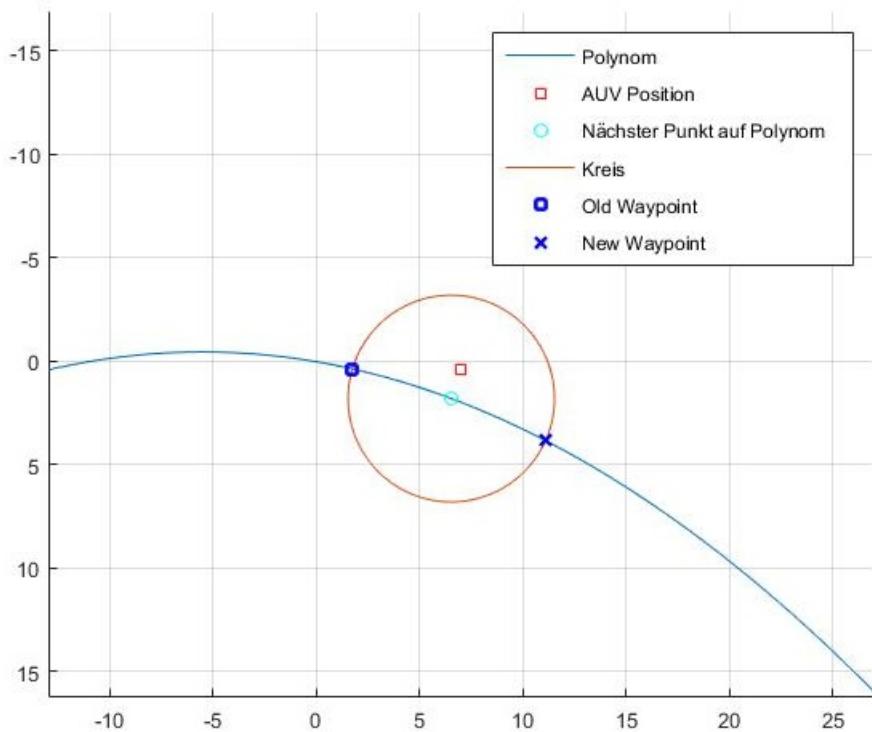
Wie im Kapitel 2.2 Grundlagen beschrieben, wird in der bestehenden Simulation ein **Waypoint Controller**, der eine Linie zwischen einem *old\_waypoint* und einem *new\_waypoint* bildet, verwendet. Die Schnittstelle zur Steuerung bildet somit die Kombination aus den beiden Wegpunkten. Die Berechnung der Wegpunkte wird auf Basis des Polynoms aus dem Schätzverfahren generiert.

Zunächst wird die Position des AUVs durch die aktuelle Transformationsmatrix in das alternative Weltkoordinatensystem (siehe Absatz 4.4) transformiert. Es wird der transformierten Position des AUVs nächstgelegene Punkt auf dem Polynom zur berechnet. Dieser Punkt dient als Zentrum für einen Kreis zur Bestimmung der Wegpunkte. Mithilfe der Kreisgleichung [Gleichung 1] werden die zwei Schnittpunkte des Polynoms mit dem Kreis berechnet. Da durch die Transformationsmatrix sichergestellt wird, dass das AUV in Richtung der *X – Achse* fährt, kann problemlos der Schnittpunkt mit höherem *x*-Wert als *next\_waypoint* und dementsprechend der zweite als *old\_waypoint* verwendet werden. Es wird davon ausgegangen, dass bei einem solch kleinen Kreisradius (zwischen 5 und 10 Metern) nicht mehr als zwei Schnittpunkte zwischen Polynom und Kreis vorhanden sind. Sollte dies der Fall sein, wäre das Polynom viel zu stark gekrümmmt, um noch verfolgt zu werden. Im Szenario dieser Arbeit gibt es auch keine Objekte, die eine solch starke Krümmung aufweisen.

Der letzte Schritt besteht aus der Transformation der Wegpunkte in das reale VRML-Koordinatensystem mithilfe der inversen Transformationsmatrix. Das Verfahren ist in Abbildung 4.2 grafisch dargestellt.

$$0 = (X_{test} - Center_X)^2 + (Y_{test} - Center_Y)^2 - r^2 \quad (1)$$

**Gleichung 1:** Kreisgleichung zum Test ob ein Punkt  $X_{test}$ ,  $Y_{test}$  auf einem Kreis liegt.  $Center_X$  und  $Center_Y$  bilden hierbei den Mittelpunkt eines Kreises mit Durchmesser  $r$ .

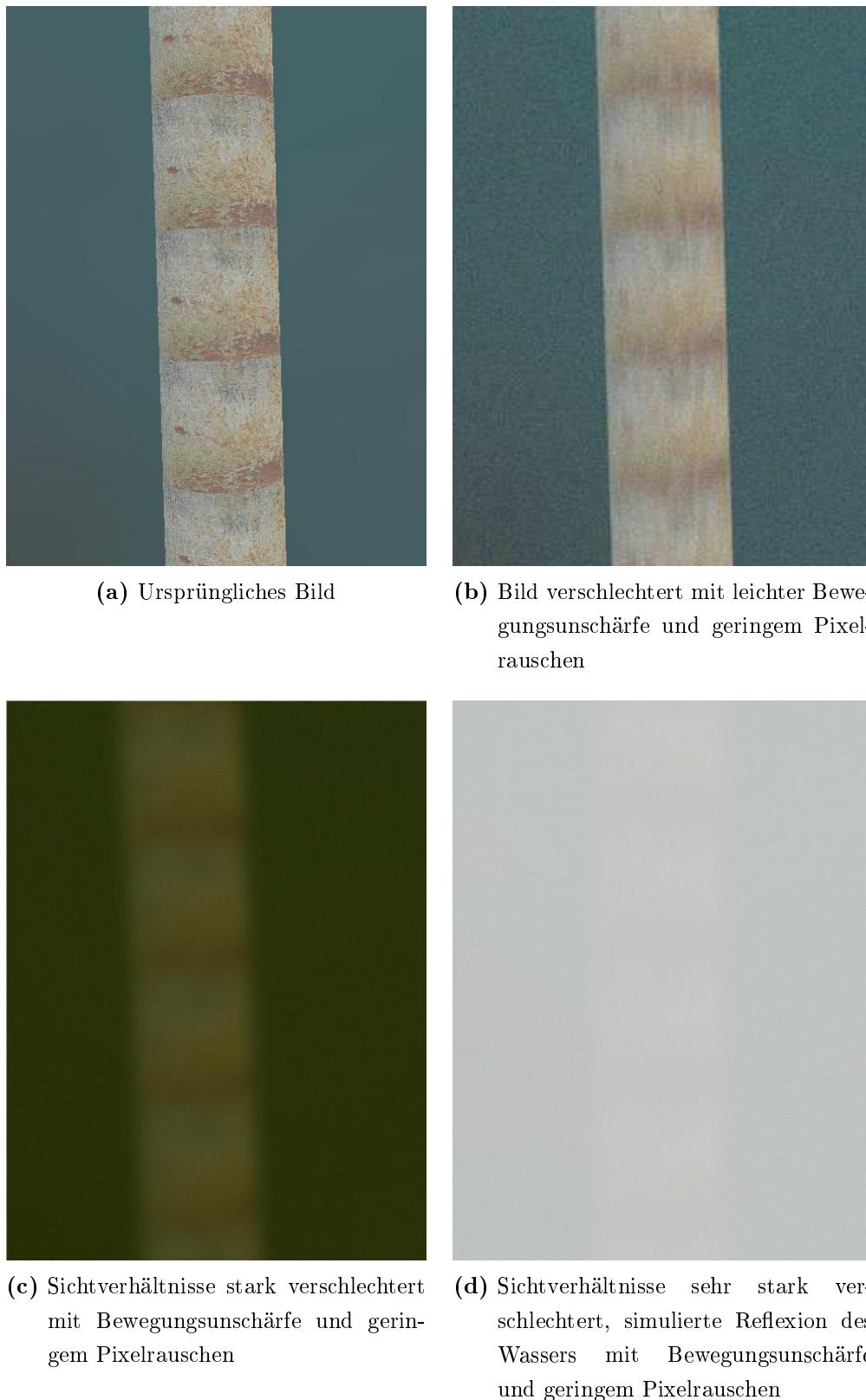


**Abbildung 4.2:** Verfahren zur Bestimmung der Wegpunkte. Der Wegpunkt wird im alternativen Weltkoordinatensystem bestimmt. Hierbei wird ein Kreis um den nächsten Punkt vom AUV auf dem Polygon bestimmt. Die Schnittpunkte des Kreises bilden die Wegpunkte für die AUV-Steuerung.

#### 4.1.2. Kamerabilder

Da die Simulation in der ursprünglichen Form noch sehr *klinische* Bilder generierte, mussten diese Bilder künstlich verschlechtert und die Sichtverhältnisse eingeschränkt werden, um realistische Eingangsbilder zu erzeugen. In Abbildung 4.3 ist von links nach rechts ein ursprüngliches Kamerabild, ein verschlechtertes Bild und ein sehr stark verschlechtertes Bild zu sehen. Die Testläufe der Arbeit wurden mit dem Verschlechterungsgrad des mittleren Bildes durchgeführt. Die Objekterkennung wurde zudem noch mit Bildern, wie dem rechten Bild getestet.

nochmal  
mit  
grö-  
ßerer  
schrift



**Abbildung 4.3:** Simulationsbilder. a) zeigt das ursprüngliche Bild. In b) bis e) wird das Bild auf verschiedenen Arten verschlechtert. Die meisten Testläufe wurden mit der Verschlechterung von b) und d) durchgeführt. Tests unter sehr schlechten Bedingungen mit einem Grad der Verschlechterung aus e)

## 4.2. Transformation

Wie bereits in der Einleitung beschrieben, werden mehrere Koordinatensysteme genutzt. Zur sicheren Verwendung der Koordinatensysteme sind Transformationen unter den Systemen zwingend nötig. Eine Transformation besteht aus einer Rotation und einer Translation, die sich aus den Beziehungen der Systeme zueinander ergibt.

Im `enum frames` [Listing 2] sind die verschiedenen Frames definiert, zwischen denen eine Transformation möglich ist.

```
1 classdef frames < uint16
2 %FRAMES Summary of this class goes here
3 % Detailed explanation goes here
4 enumeration
5   image(0),cam(1),body(2),world(3),vrml(4)
6 end
7
8 end
```

**Listing 2:** Enumeration der Frames, die die verschiedenen Koordinatensystem bezeichnen

Umgesetzt wird eine Transformation aus einem *source* Frame in einen *target* Frame durch die Funktion `transform` [Listing 3]. Die Transformation ist nur in eine Richtung möglich, da die inverse Transformation für diese Arbeit nicht benötigt wurde.

```
1 function transformed = transform(toTransform,targetFrame,
2   height,PosEast_m,PosNorth_m,psi,phi,theta,cameraParameters
3 )
4 %TRANSFORM Summary of this function goes here
5 % Detailed explanation goes here
6 while(toTransform.frame~=targetFrame)
7   switch toTransform.frame
8     case frames.image
9       toTransform = pic2cam(toTransform,height,phi,
10                           theta,cameraParameters);
11     case frames.cam
12       toTransform = cam2body(toTransform);
13     case frames.body
14       toTransform = body2world(toTransform,psi,
15                               PosEast_m,PosNorth_m,height);
16     case frames.world
17       toTransform = world2vrml(toTransform);
```

```
14     end
15
16     end
17     transformed = toTransform;
end
```

**Listing 3:** Transformation von *source* in *target* Frame. Die verschiedenen Transformation werden hier verwaltet und solange die nächste Transformation ausgeführt, bis der *target* Frame erreicht wird.

#### 4.2.1. Bild zu Kamera

Die verlustfreie Transformation von 2D-Pixelkoordinaten in 3D-Kamerakoordinaten ist mit einer Kamera nicht möglich, da die Tiefeninformation nicht vorhanden ist. Jedoch lässt sich mit dem Wissen über die Entfernung zur Bildebene, der *flat world assumption* (siehe Kapitel 2.3) und den intrinsischen Kameraparametern eine ausreichend gute Transformation durchführen. Da die Kamera gerade nach unten gerichtet ist, entspricht die Entfernung zur Bildebene der Höhe des AUVs über dem Meeresboden, welche über die Sensorik bestimmt wird. Die intrinsischen Kameraparameter lassen sich über eine Kamerakalibrierung bestimmen, welche mithilfe der MATLAB *Computer Vision System Toolbox* durchgeführt.

Da die resultierende Transformation am besten im Abstand der Kalibrierung funktioniert wurde die Kalibrierung in einem Abstand von 6 Metern durchgeführt, was im späteren Verlauf auch der gewünschte Abstand zum Boden ist.

Aus der Kamerakalibrierung wird ein *CameraParameter*<sup>6</sup> Objekt erzeugt, welches die Methode *pointsToWorld* bietet. Die Methode berechnet eine Projektionsmatrix aus den Kamera-Parametern und dem bekannten Abstand der Kamera zum Objekt. Mithilfe der Inversen dieser Matrix können dann Pixel in Kamerakoordinaten umgerechnet werden. Leichte Neigungswinkel, die während der Fahrt auftreten, können durch die Multiplikation mit der entsprechenden Rotationsmatrix herausgerechnet werden. Jedoch ist dabei zu beachten, dass durch die Neigungswinkel die Fläche, die die Kamera sieht, vergrößert wird. Dadurch bilden einzelne Pixel mehr Fläche ab und die Transformation wird ungenauer. Die z-Koordinate ergibt sich aus dem Wissen, Objekte am Meeresboden zu betrachten und der Tatsache, dass die Höhe der Kamera über dem Meeresboden bekannt ist.

#### 4.2.2. Kamera zu Body

Die Transformation vom Kamerakoordinatensystem zum Bodykoordinatensystem besteht aus einer Translation und einer Rotation, die durch die Montageposition der Kamera am

<sup>6</sup> <https://de.mathworks.com/help/vision/ref/cameraparameters-class.html>

AUV bestimmt wird [Kapitel 2.3.1].

Aufgrund der Verschiebung der Kamera zum Bodykoordinatenursprung (Schwerpunkt des AUVs) ergibt sich eine Translation um 1,3m in x-Richtung und 0,25m in z-Richtung.

Die Rotation beträgt dabei 90° um die Z-Achse.

Somit ergibt sich die Transformationsmatrix aus Gleichung 2

$$\begin{pmatrix} x_{body} \\ y_{body} \\ z_{body} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 & 1,3 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0,25 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{pmatrix} \quad (2)$$

**Gleichung 2:** Transformation der Kamerakoordinaten zu Bodykoordinaten. Die Kamerakoordinaten werden um 1,3m auf der x-Achse und 0,25m auf der z-Achse verschoben. Außerdem wird eine Rotation um 90° um die Z-Achse durchgeführt.

#### 4.2.3. Body zu Welt

Für die Transformation vom Bodykoordinatensystem in das Weltkoordinatensystem ist wieder eine Translation und eine Rotation nötig. Aus der Definition der Koordinatensysteme folgt zunächst eine Rotation um 180° um die X-Achse nötig. Die Translation ergibt sich aus der Position des AUVs (Position Nord/Ost in Metern).

Die Rotation wird durch die Ausrichtung des AUVs in der Welt (Yaw [Abb. 2.5]) bestimmt. Somit ergibt sich die Transformationsmatrix aus Gleichung 3

$$\begin{pmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(yaw) & -\sin(yaw) & 0 & Pos_{north} \\ \sin(yaw) & \cos(yaw) & 0 & Pos_{east} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \left( \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \right) \cdot \begin{pmatrix} x_{body} \\ y_{body} \\ z_{body} \\ 1 \end{pmatrix} \end{pmatrix} \quad (3)$$

**Gleichung 3:** Transformation der Bodykoordinaten zu Weltkoordinaten. Zunächst werden die Body-Koordinaten um 180° um die X-Achse rotiert. Im Anschluss findet eine Translation zu der Position des AUVs in der Welt und eine Rotation um die Z-Achse, die die Ausrichtung des AUVs abbildet, statt.

#### 4.2.4. Welt zu VRML

Für die Transformation von Weltkoordinaten in VRML Koordinaten ist nur eine Rotation um  $-90^\circ$  um die X-Achse nötig [Abb. 4].

$$\begin{pmatrix} x_{vrm} \\ y_{vrm} \\ z_{vrm} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{body} \\ y_{body} \\ z_{body} \\ 1 \end{pmatrix} \quad (4)$$

**Gleichung 4:** Transformation von Weltkoordinaten in VRML Koordinaten. Hierfür ist nur eine Rotation um  $-90^\circ$  um die X-Achse nötig.

## 4.3. Objekterkennung

In diesem Kapitel wird die umgesetzte Objekterkennung beschrieben. Dabei wird aus einem Rohbild der Kamera ein *pointInFrame*-Objekt erzeugt. Grob besteht die Objekterkennung aus zwei Schritten. Zuerst wird aus dem Rohbild ein Binärbild erzeugt und im Anschluss im Binärbild das gesuchte Objekt detektiert.

### 4.3.1. Binärbild mit Template

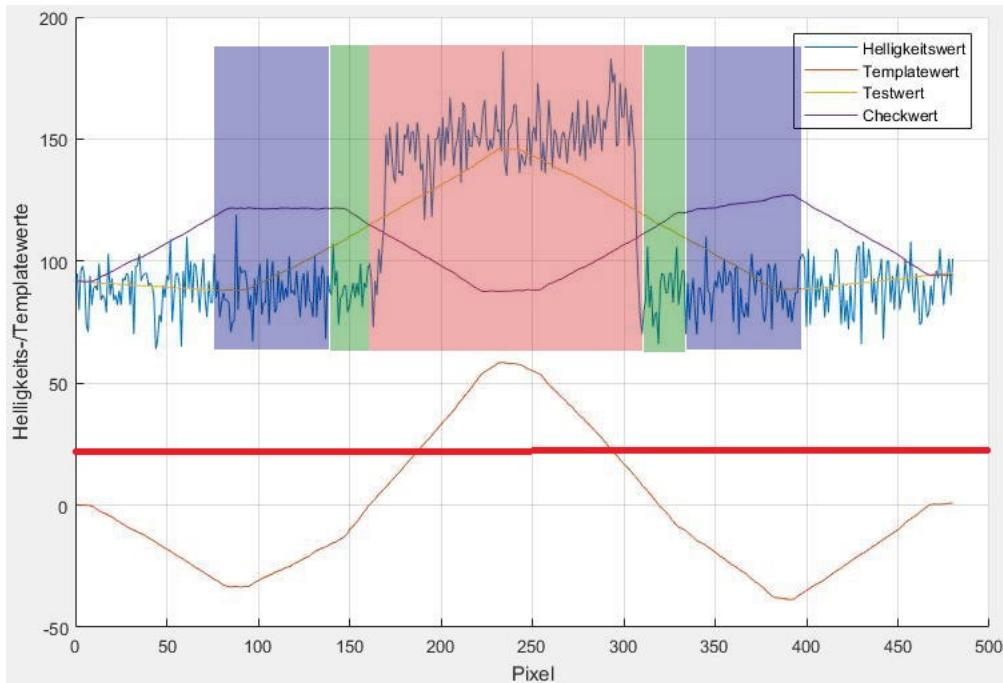
Die Objekterkennung basiert auf einem ähnlichen Verfahren wie das vorgestellte CSurvey Projekt[1].

Da eine Farberkennung aufgrund der Sichtbedingungen nicht in Frage kommt, wird im ersten Schritt das RGB-Bild in ein Graustufenbild umgewandelt. Der erste Ansatz bestand darin, das Helligkeitsbild zu betrachten, da ein gesuchtes Objekt einen höheren Helligkeitswert besitzt als der Meeresboden (siehe Abb. 4.5b und 4.6b).

Aus Erfahrungswerten früherer Projekte riet Christopher Gaudig mir, die Rotwerte der Bilder zu betrachten, da oftmals sowohl der Meeresboden als auch trübes Wasser geringe Rotwerte haben. In den Abbildungen 4.5c und 4.6c ist dies zu beobachten. Die Kurven sehen denen der Helligkeitswerte sehr ähnlich, jedoch sind die Ausschläge des Objektes in den Rotwerten höher.

Im nächsten Schritt wird mithilfe eines Templates [Abb. 4.4] ein Binärbild erzeugt. Das Template zeichnet sich durch drei Pixelangaben aus. Die *Testpixel* (rot) geben einen Bereich an, der im aktuellen Schritt geprüft wird. Die *Checkpixel* (blau) geben den Bereich rechts und links neben dem Testbereich an und bilden den Referenzwert. Die *Borderpixel* (grün) geben einen Bereich zwischen Test- und Checkbereich an, der ignoriert wird. Durch das Ignorieren des Bereichs kann ein langsamer regelmäßiger Übergang, der bei Betrachtung der direkten Nachbarpixeln des Testbereichs als Checkbereich zu einem geringen Templatewert führt, trotzdem noch einen hohen Templatewert ergeben. Jedes Pixel dient einmal als Mittelpunkt des Testbereichs, um zu entscheiden, ob das betrachtete Pixel Teil des Objektes sein kann. Dies ist der Fall, wenn der Wert des Pixels [Gleichung 5] einen Schwellenwert (rote Linie) übersteigt.

Kann man die grauen mit legends gut genug erkennen?



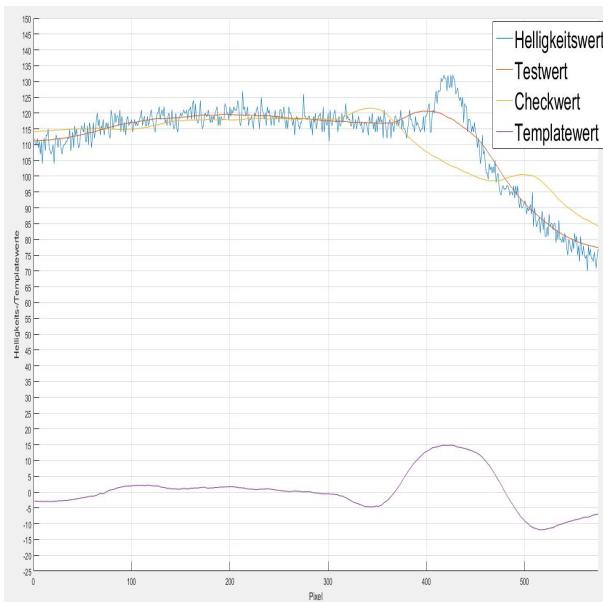
**Abbildung 4.4:** Template zum Bestimmen des Binärbilds. Getestet wird das Pixel im Zentrum des Testbereichs (rot). Der Templatewert ergibt sich aus der Subtraktion des Durchschnitts im Checkbereich (blau) vom Durchschnitt des Testbereichs (rot). Der Borderbereich (grün) wird dabei nicht beachtet.

$$TV = \frac{\text{sum}(Testpixel)}{\#TP} - \frac{\text{sum}(Checkpixel)}{\#CP} \quad (5)$$

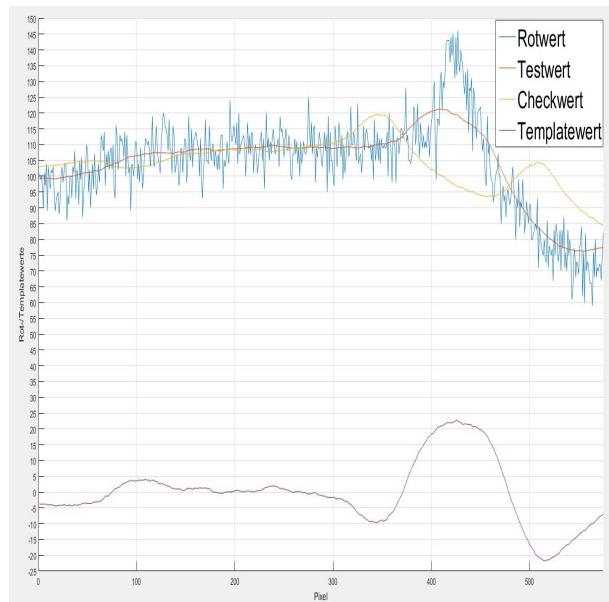
**Gleichung 5:** Templatewertberechnung für ein Pixel als Formelausdruck. Der berechnete Wert  $TV$  ist der Unterschied zwischen Test- und Checkbereich.  $\#TP$  bezeichnet die Anzahl der Testpixel und  $\#CP$  die Anzahl der Checkpixel.



(a) Originalbild. Das Testbild stammt aus Aufnahmen eines Testlaufs im Unisee (siehe Abschnitt 5.1).

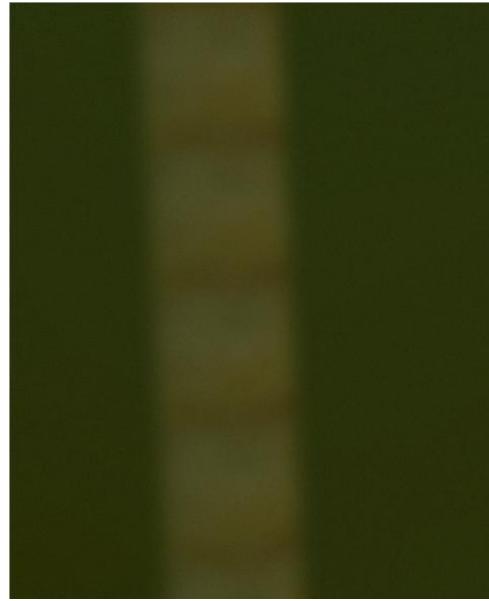


(b) Auswertung des Helligkeitsverlauf einer Bildzeile im oberen Drittel des Bildes

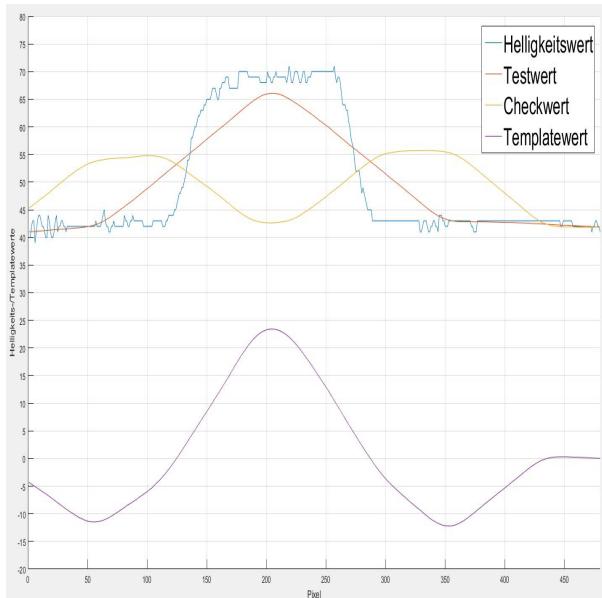


(c) Auswertung des Rotwertverlauf einer Bildzeile im oberen Drittel des Bildes

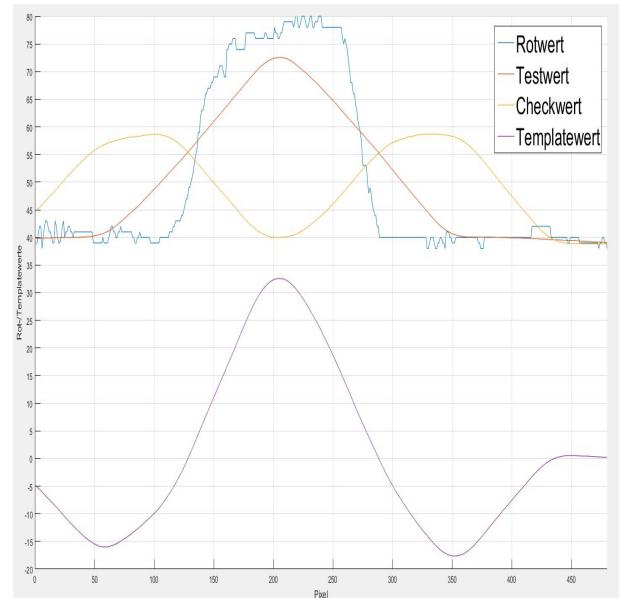
**Abbildung 4.5:** Helligkeit und Rotwert im echten Testbild. In beiden Grafiken zeigt die blaue Linie die jeweiligen Pixeldaten, die gelbe Linie den Wert des Testbereichs, die lila Linie den Durchschnitt des Checkbereichs und die weiter unten gelegene rote Linie den Templatewert für das Pixel. In beiden Templatewerten ist die Pipeline eindeutig zu erkennen, wobei der Ausschlag im Rotwert weitaus höher ist.



(a) Originalbild der Simulation



(b) Auswertung des Helligkeitsverlaufs einer Bildzeile im oberen Drittel des Bildes



(c) Auswertung des Rotwertverlaufs einer Bildzeile im oberen Drittel des Bildes

**Abbildung 4.6:** Helligkeit und Rotwert im Simulationsbild. In beiden Grafiken zeigt die blaue Linie die jeweiligen Pixeldaten, die gelbe Linie den Wert des Testbereichs, die lila Linie den Durchschnitt des Checkbereichs und die weiter unten gelegene rote Linie den Templatewert für das Pixel. In beiden Templatewerten ist die Pipeline eindeutig zu erkennen, wobei der Ausschlag im Rotwert weitaus höher ist.

### 4.3.2. RANSAC auf Binärbild

Im weiteren Verlauf wird auf dem Binärbild gearbeitet. Nach dem Vorbild von Wang et al. [18] wird das Bild in drei Segmente unterteilt.

In jedem Segment wird dann mithilfe des RANSAC-Algorithmus ein Rechteck gesucht [Listing 4]. Der Algorithmus sampled verschiedene Rechtecke im Segment. Jedes Rechteck wird durch einen Mittelpunkt, eine Orientierung, die Breite und die Höhe definiert. Die Höhe ergibt sich aus der Höhe des Segmentes und die Breite wird durch die erwartete Breite des Objektes festgelegt. Mittelpunkt und Orientierung werden in jedem Iterationsschritt zufällig gewählt.

Für jeden Punkt des Binärbilds wird dann geprüft, ob er im Rechteck liegt (ein *Inlier* ist). Gemäß des RANSAC wird das Rechteck mit den meisten Inliern gewählt.

```
1 function ransac(segment, height, width, minInlier, iterNum)
2     maxInlier = 0;
3     orientations = -pi/4:0.05:pi/4;
4     bestCenter = None;
5     bestOrientation = None;
6     for i = 1:iterNum
7         boxCenter = selectRandomPoint(segment);
8         boxOrientation = selectRandomValue(
9             orientations);
10        inliers = findPointsInBox(segment, box=[
11            boxCenter, boxOrientation, height, width]);
12
13        if(len(inliers) > minInlier && len(inliers) >
14            maxInlier)
15            maxInlier = len(inliers)
16            bestCenter = boxCenter;
17            bestOrientation = boxOrientation;
18        end
19    end
20end
```

Listing 4: Eingesetzter RANSAC als Pseudocode.

Somit gibt es für jedes Bild bis zu drei Objektposen. Durch das Unterteilen in Segmente lässt sich zum einen bestimmen, in wie vielen Segmenten ein Objekt erkannt wurde (entspricht der *Länge* des Objektes im Bild). Des weiteren kann ein gebogener Verlauf oder

ein abgeknicktes Objekt im Bild sinnvoll erkannt werden.

In den ersten Tests dieses Verfahrens ist ersichtlich geworden, dass es einen Tradeoff zwischen Geschwindigkeit und Erkennungsgüte gab. Die Erkennung wurde besser, je größer das Maximum der Iteration für RANSAC gewählt wurden. Da der RANSAC jedoch auf jedes der drei Segmente separat angewendet wird, reduziert sich die Geschwindigkeit bei steigender Iterationsanzahl deutlich. Für eine zuverlässige Erkennung waren zu viele Iterationen nötig, sodass das Verfahren nicht einsetzbar wäre.

Als Lösung für dieses Probleme werden die möglichen erzeugten Rechtecke für den RANSAC begrenzt. Da das Template nur in horizontaler Richtung auf das Bild angewendet wird, sind horizontal liegende Objekte im Binärbild nicht sichtbar. Aufgrund dieser Tatsache lassen sich die Orientierungen auf einen Bereich begrenzen, anstatt diese komplett zufällig zu wählen. Durch diese Maßnahme wurden die benötigten Iterationen für ein zuverlässiges Ergebnis drastisch reduziert. Jedoch steigt auch die Gefahr Orientierungen nicht mehr richtig zu erkennen, wie in Abbildung 4.7 gezeigt.

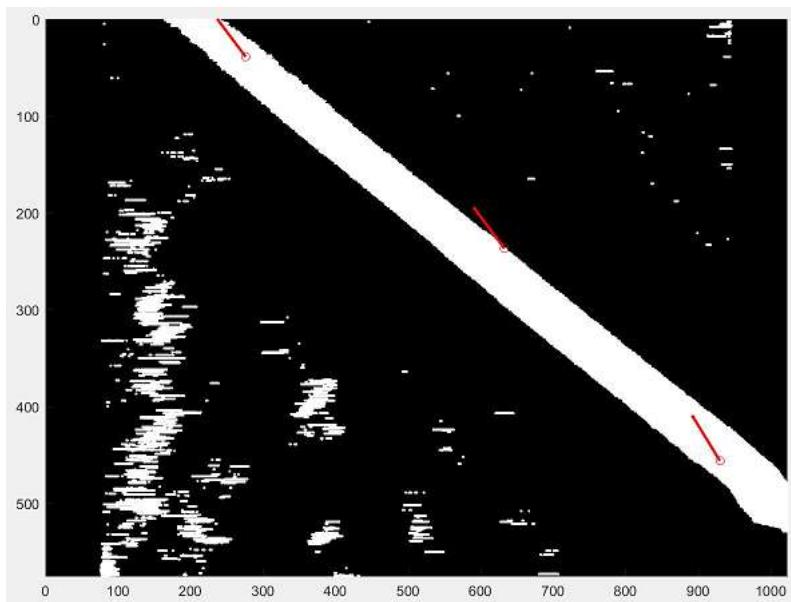


Abbildung 4.7: Falsch detektierte Objektorientierung aufgrund der Beschränkung der Ausrichtungen für den RANSAC

Der zweite Faktor, der die Geschwindigkeit der Objekterkennung verringerte, ist die Menge der Punkte im Binärbild. So musste für jede Iteration des RANSAC für jeden Punkt geprüft werden, ob der Punkt im Rechteck liegt. In den Testbildern der Simulation lag die Anzahl der Punkte teilweise bei weit über 10000, was in Kombination mit 200 Iterationen zu einer inakzeptablen Laufzeit von ca. 5 Sekunden pro Bild führte.

Zum Lösen dieses Problems wurde vor dem Einsatz des RANSAC die Punktanzahl verringert, indem nur jedes dritte Pixel betrachtet wird und dieses den Mittelwert aller

richtige  
wort-  
wahl?

evtl  
box  
vom  
ransac  
zeigen

seiner Nachbarpixel erhält. Somit konnte die Punktanzahl zuverlässig auf unter 2000 verringert werden, was zu einer deutlichen Beschleunigung (ca. 1 Sekunde pro Bild) ohne nennenswerte Verschlechterung der Ergebnisse führte. Die Laufzeit wurde dabei mit dem MATLAB *Stopwatch Timer*<sup>7</sup> gemessen.

Grafik?

---

<sup>7</sup> <https://de.mathworks.com/help/matlab/ref/tic.html>

#### 4.4. Schätzverfahren

In diesem Abschnitt wird das implementierte Schätzverfahren erläutert. Das Verfahren nutzt die Ergebnisse der Bilderkennung und versucht mithilfe der Regression ein Polynom  $f$  zweiten Grades durch alle erkannten Punkte unter Betrachtung ihrer Orientierung zu fitten. Der Ansatz basiert auf dem *Least-Squares* Verfahren[15], wobei versucht wird, die Gleichung 6 zu minimieren.

$x_i$  und  $y_i$  sind hierbei die Koordinaten der erkannten Punkte. Es wird über alle Punkte der quadratische Fehler vom Funktionswert für ein  $x$  zu gegebenen Modellparametern  $p$  zum  $y$  aus der Bilderkennung summiert.  $f(p, x)$  ist eine beliebige Funktion, die  $x$  in Abhängigkeit von  $p$  auf eine reelle Zahl abbildet.

$$err = \sum_i (f(p, x_i) - y_i)^2 \quad (6)$$

**Gleichung 6:** Least-Squares-Ansatz.  $x_i$  und  $y_i$  sind die erkannten Objektpositionen.

Über die Zeit gesehen wird die Menge an detektierten Punkten immer größer. Da das Ziel des Verfahrens die Vorhersage des Objektverlaufs ist, ist eine gute Extrapolation wichtiger als das richtige Abbilden aller Punkte der Vergangenheit. Für die Extrapolation ist anzunehmen, dass neuere Punkte für den Verlauf wichtiger sind als ältere. Aus diesem Grund werden die Punkte über die Zeit exponentiell aufsteigend gewichtet. Somit erhalten aktuelle Punkte einen höheren Stellenwert als ältere, ohne jedoch alte Punkte komplett zu verwerfen.

Um diese Anforderungen umzusetzen wird eine Erweiterung des *Least-Squares* verwendet, den *Weighted-Least-Squares*[Gleichung 7]

$$err = \sum_i w_i \cdot (f(p, x_i) - y_i)^2 \quad (7)$$

**Gleichung 7:** Weighted-Least-Squares Verfahren. Erweitert das Least-Squares Verfahren um eine Gewichtung der Punkte.

Das *Weighted-Least-Squares* Verfahren bietet eine gute Grundlage für die Regression. Es bleiben jedoch noch einige Probleme, die das Verfahren in der Form nicht lösen kann.

1. Beachtung der Orientierung erkannter Punkte
2. Bedingungen für die Kurve (z.B. maximale Steigung)

3. Schätzungen für Punktverläufe, die sich nicht durch ein einzelnes Polynom darstellen lassen

Zum Lösen der ersten zwei Probleme bietet die MATLAB -Funktion *fmincon*<sup>8</sup> aus der Optimization Toolbox eine geeignete Lösung. Die Funktion bietet die Möglichkeit eine Funktion  $F(p)$  zu minimieren, wobei mit  $c(p) \leq 0$  eine Bedingung erfüllt werden muss. Die Funktion  $c(p, x_i)$  [Gleichung 8] berechnet über den Funktionsverlauf von  $f(p, x)$  mithilfe der Ableitung  $f'(p, x)$  die Steigung in jedem Punkt  $x_i$ . Da *fmincon* prüft, ob die Bedingungsfunktion kleiner 0 ist wird von der Steigung ein Maximalwert ( $\max_{slope}$ ) abgezogen (*Erfüllt 2.*).

$$c(p, x_i) = f'(p, x_i) - \max_{slope} \quad (8)$$

**Gleichung 8:** Funktion zum Überprüfen, ob die Steigung einen Maximalwert nicht übersteigt.  
 $\max_{slope}$  gibt die maximal erlaubte Steigung des Polynoms an, die mit der Ableitung der Funktion überprüft wird.

Die Funktion  $F(p)$  wird als  $F(p, x, y, s, w, n, m, tau)$  [Gleichung 9] definiert, wobei  $x$  und  $y$  erneut die Punkte der Bilderkennung darstellen,  $s$  die erkannte Orientierung im Punkt und  $w$  das Gewicht darstellt. Die Funktion  $F$  besteht aus einer Linearkombination der Funktionen  $g$  und  $h$ , wobei  $g$  den summierten Fehler der Position [Gleichung 10] ( $x, y$  Koordinaten) und  $h$  den summierten Fehler der Orientierung [Gleichung 11] mithilfe des *Weighted-Least-Squares* Verfahren berechnen (*Erfüllt 1.*).  $n$  und  $m$  gewichten, wie stark die einzelnen Fehlerarten (Position und Orientierung) in den Gesamtfehler für die gegebenen Funktionsparameter  $p$  eingehen.

Um die erhaltenen Polynome einschränken zu können, wurde  $F$  noch gemäß der *Tikhonov Regularisierung* [10] angepasst. Durch die *Tikhonov-Regularisierung* können wenig gekrümmte Kurven bevorzugt werden, was für einen ruhigeren Fahrtverlauf sorgen kann.

$$F(p) = F(p, x, y, s, w, n, m, tau) = n \cdot g(p, x, y, w) + m \cdot h(p, x, s, w) + tau \cdot p \quad (9)$$

$$g(p, x, y, w) = \sum_i w_i \cdot (f(p, x_i) - y_i)^2 \quad (10)$$

$$h(p, x, s, w) = \sum_i w_i \cdot (f'(p, x_i) - s_i)^2 \quad (11)$$

**Gleichung 9:** Zusammensetzung der Funktion  $F$ , die minimiert wird. In (10) wird der *Weighted-Least-Squares* auf den Fehler der Position und in (11) auf den Fehler der Steigung angewendet.

---

<sup>8</sup> <https://de.mathworks.com/help/optim/ug/fmincon.html>

Um das Problem 3. zu lösen, betrachten wir Abbildung ???. Das Objekt ist hierbei so gelegen, dass kein Polynom zweiten Grades sinnvoll durch die Daten gelegt werden kann und außerdem ein Teilabschnitt parallel zur *Y – Achse* verläuft. Der letzte Fall ist zu beachten, da ein solcher Verlauf durch eine unendliche Steigung im Polynom abgebildet werden müsste.

Als Lösung für dieses Problem wird ein alternatives Weltkoordinatensystem eingeführt. Dieses unterscheidet sich durch eine Transformation vom echten Weltkoordinatensystem. Nach jeder Regression wird das berechnete Polynom in den aktuellsten Punkten getestet. Sollte dabei ein gewisser Fehlerschwellenwert überschritten werden, wird eine neue Transformation berechnet [Listing 5]. Diese neue Transformation besteht aus einer Translation zum Punkt mit dem größten *x*-Wert und einer Rotation um die durchschnittliche Ausrichtung der neuesten Punkte. Neben der Transformationsmatrix wird auch die Inverse der Matrix gespeichert, die für die Wegpunktberechnung [Kapitel 4.1.1] wichtig ist. Da die Transformation ausgelöst wird, sobald das Polynom in den neuesten Punkten einen zu großen Fehler ergibt, werden nach dem Speichern der Matrizen alle Punkte, bis auf die neuesten verworfen, um ein potentiell besseres Polynom für die Extrapolation zu ermöglichen.

Sobald eine beschriebene Transformation gespeichert wurde, werden alle Punkte vor der Regression in das alternative Koordinatensystem transformiert. Durch diese Transformation sind die erkannten Punkte entlang der *X – Achse* gelegen und somit ist es möglich, stets ein geeignetes Polynom für die Punkte zu finden. Durch die Translation liegen die Punkte stets nah am Ursprung, was den Parameterraum für die Regression verringert und somit zu schnelleren Ergebnissen führt.

```
1 function polynomFit(points ,maxError)
2     actualTransform = loadActualTransformation();
3     points_T = transformPoints(points ,actualTransform);
4
5     polynom = regression(points_T);
6     error = calculateError(points_T ,polynom);
7     if (error >= maxError)
8         translation = findGreatestXValue(points_T);
9         rotation = averageDirectionOfLastPoints(
10             points_T);
11         newTransform = createTransMatrix(-translation
12             ,-rotation) * actualTransform;
13         saveNewTransform();
14
15 end
```

hier  
noch  
gra-  
phen  
zur  
trans-  
forma-  
tion

13 | end

**Listing 5:** Pseudocode des Schätzverfahrens

## 5. Tests und Evaluation

In diesem Kapitel werden die Tests der Arbeit zusammengefasst. Zuerst werden spezifische Tests für die Güte der Objekterkennung beschrieben. Im zweiten Teil werden verschiedene Testläufe der Simulation mit verschiedenen Objektverläufen angeführt.

## 5.1. Tests Objekterkennung

Für diese Arbeit ist eine verlässliche Erkennung auf Simulationsbildern wichtig. Aus diesem Grund beziehen sich die Tests auf Simulationsbilder. Außerdem werden noch Tests auf Echtdaten durchgeführt, um das Verfahren zu evaluieren.

Für die Tests werden verschiedene Szenarien verwendet. Zunächst wird ein gerades Objekt, das komplett zu sehen ist, getestet [Abb. 5.1]. Dieser Test repräsentiert das einfachste Szenario zur Detektion. Im zweiten Test wird dieses gerade Objekt vom Meeresboden verdeckt [Abb. 5.2], um eine Detektion zu erschweren.

Im dritten Test wird ein Objekt betrachtet, das innerhalb des Bildes abgeknickt ist [Abb. 5.3]. Dabei wird getestet, wie sich die verschiedenen Ausrichtungen innerhalb eines Bildes auf die Detektion auswirken. Ein vierter Test überprüft das Verhalten der Objekterkennung, wenn kein Objekt im Bild zu sehen ist. Hier sollte keine erfolgreiche Detektion entstehen. Alle Tests werden unter verschiedensten Sichtbedingungen durchgeführt. Die ursprünglichen Simulationsbilder als einfachsten Test, die Bildqualität, in der die meisten Testläufe des AUVs durchgeführt wurden und Bilder unter sehr schlechten Sichtbedingungen, in denen es schwer ist überhaupt ein Objekt zu entdecken.

Auf den nachfolgenden Seiten sind die Ergebnisse dieser Tests aufgeführt. Wie zu erwarten war, werden die Objekte und ihre Ausrichtungen im ursprünglichen Simulationsbild am besten erkannt. Hier sind die Grenzen zwischen Meeresboden und Objekt am klarsten und das Binärbild bildet das Objekt nahezu optimal ab. Ebenso kann der Templateschwellenwert so hoch gesetzt werden, dass keine Störpunkte vorhanden sind.

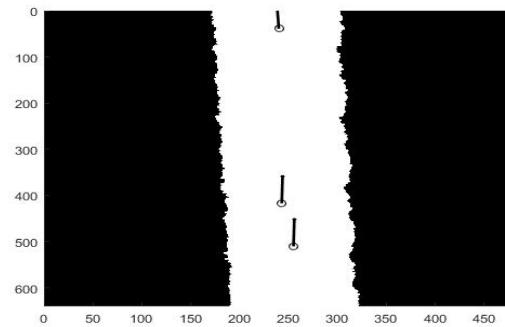
Die Tests mit schlechteren Sichtbedingungen führen ebenfalls zu guten Ergebnissen. In den Binärbildern sind oftmals *wellenförmige* Kanten zu beobachten, die teilweise zu leicht fehlerhaften Detektionen der Ausrichtung führen. Zurückzuführen sind diese Kanten darauf, dass das Objekt aufgrund der Textur nicht überall gleich hohe Pixelwerte hat und so die *dunkleren* Bereiche am Rand des Objektes den Schwellenwert nicht mehr überschreiten. Da der Unterschied zwischen Meeresboden und Objekt bei diesen Sichtbedingungen nicht so groß ist wie im ursprünglichen Bild, kann der Schwellenwert nicht weiter gesenkt werden, ohne zu viele Störpunkte zu erhalten.

Unter sehr schlechten Sichtbedingungen werden die Objekte noch immer gut detektiert. Jedoch sind in einigen Binärbildern trotz der einfachen Umgebung bereits viele Störpunkte sichtbar. Auch komplett ohne Objekt sind Punkte vorhanden. Ebenso wird das Objekt im Binärbild nicht mehr so gut abgebildet, wie unter besseren Bedingungen. Die bereits erwähnten *wellenförmigen* Kanten sind auch hier zu beobachten.

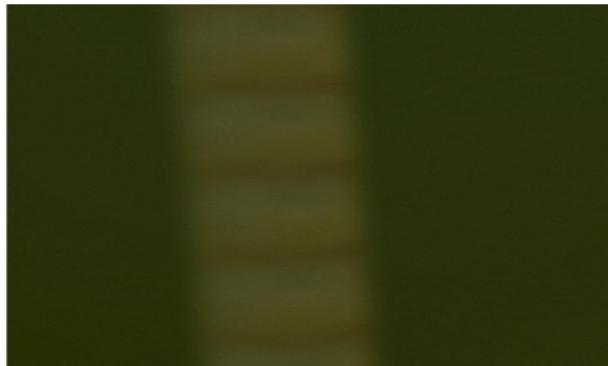
## Tests auf Simulationsbildern



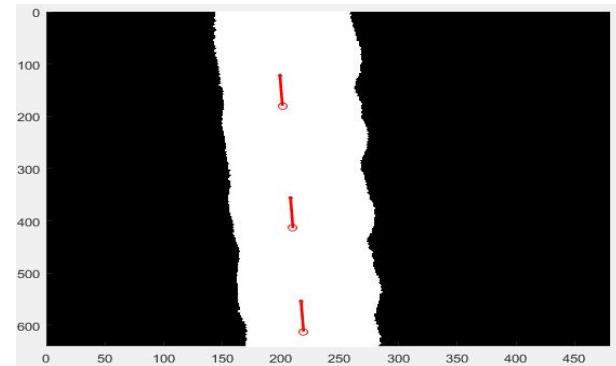
(a) Objekt in ursprünglichem Simulationsbild



(b) Detektiertes Objekt im ursprünglichen Simulationsbild



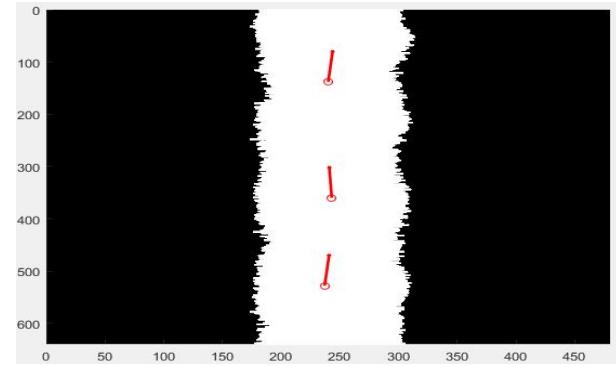
(c) Objekt unter schlechteren Sichtbedingungen



(d) Detektiertes Objekt unter schlechteren Sichtbedingungen



(e) Objekt unter sehr schlechten Sichtbedingungen

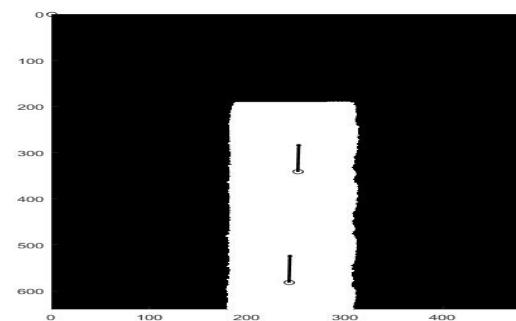


(f) Detektiertes Objekt unter sehr schlechten Sichtbedingungen

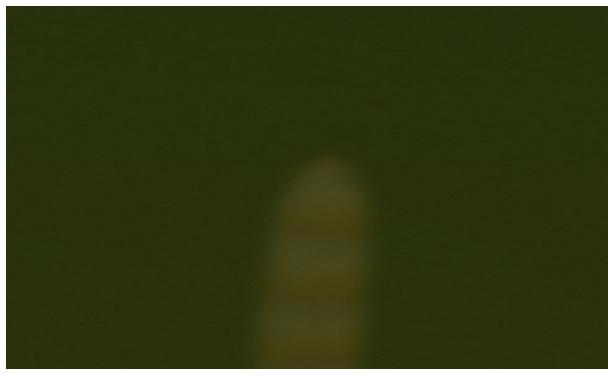
**Abbildung 5.1:** Ein gerades Objekt, dass ohne Einschränkungen zu sehen ist wird unter verschiedenen Sichtbedingungen und Bildqualität getestet. Die Gerade wird unter allen Bedingungen gut detektiert. In f) ist zu sehen, dass unter sehr schlechten Sichtbedingungen ein *gezackter* Kantenverlauf entsteht.



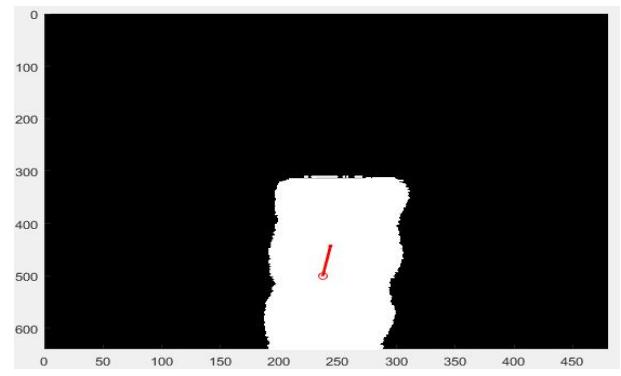
(a) Objekt in ursprünglichem Simulationsbild



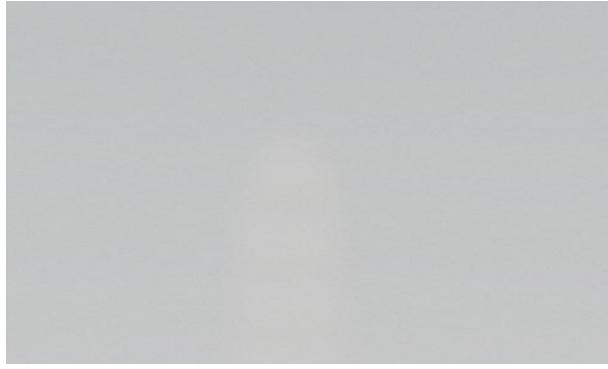
(b) Detektiertes Objekt im ursprünglichen Simulationsbild



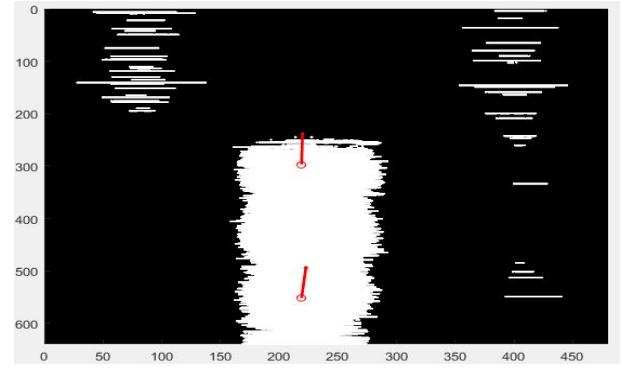
(c) Objekt unter schlechteren Sichtbedingungen



(d) Detektiertes Objekt unter schlechteren Sichtbedingungen



(e) Objekt unter sehr schlechten Sichtbedingungen

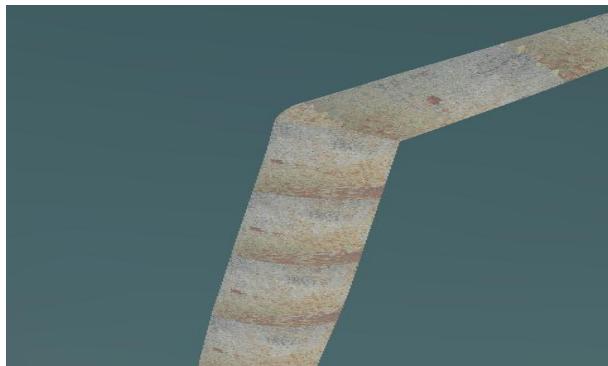


(f) Detektiertes Objekt unter sehr schlechten Sichtbedingungen

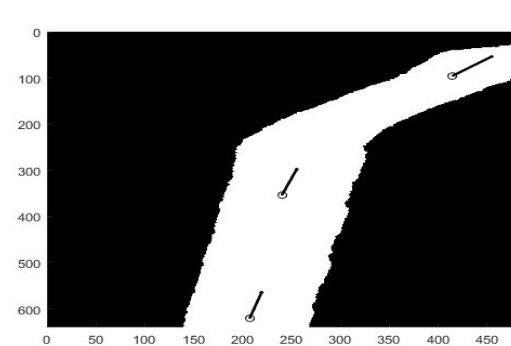
**Abbildung 5.2:** Das verdeckte Objekt wird unter allen Sichtbedingungen erkannt. Hier ist der Effekt der drei Segmente zu sehen. Während in b) und f) das Objekt noch in zwei Segmenten detektiert wurde, ist das Objekt in d) zu kurz um im zweiten Segment noch detektiert zu werden.

5. Tests und Evaluation

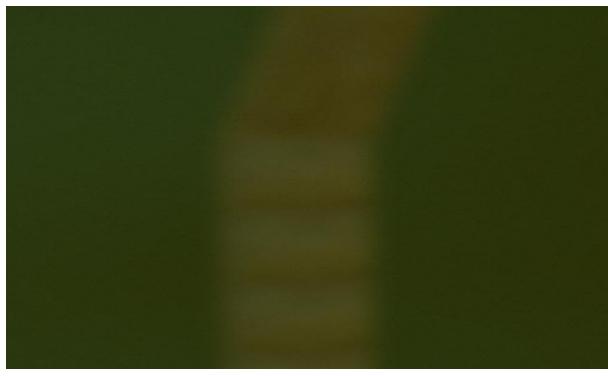
---



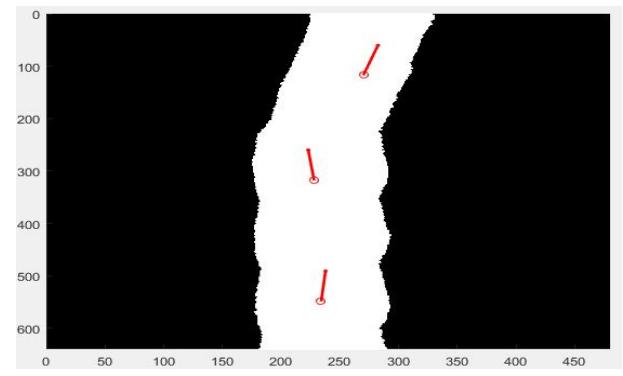
(a) Objekt in ursprünglichem Simulationsbild



(b) Detektiertes Objekt im ursprünglichen Simulationsbild



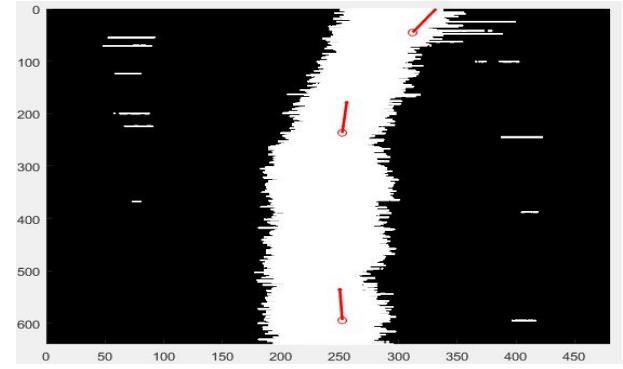
(c) Objekt unter schlechteren Sichtbedingungen



(d) Detektiertes Objekt unter schlechteren Sichtbedingungen



(e) Objekt unter sehr schlechten Sichtbedingungen



(f) Detektiertes Objekt unter sehr schlechten Sichtbedingungen

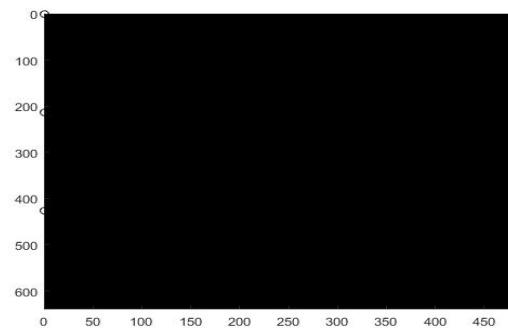
**Abbildung 5.3:** Das abgeknickte Objekt wird unter allen Sichtbedingungen erkannt. In d) und f) ist ein *wellenförmiger* Kantenverlauf zu beobachten, der in d) zu einer leicht falschen Bestimmung der Orientierung im geraden Bereich führt. In f) ist zu sehen, dass die Störpixel im abgeknickten Bereich zu einer Fehldetektion der Orientierung führt.

5. Tests und Evaluation

---



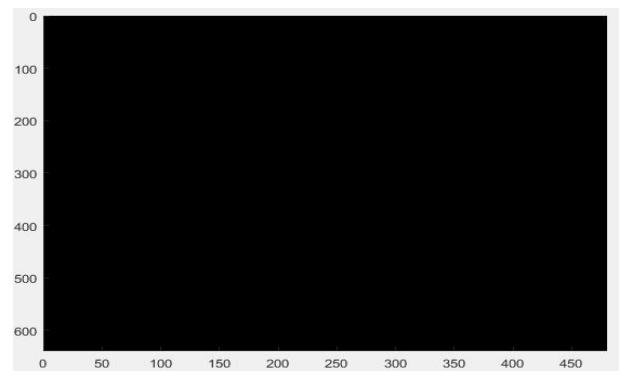
(a) Leeres ursprünglichem Simulationsbild



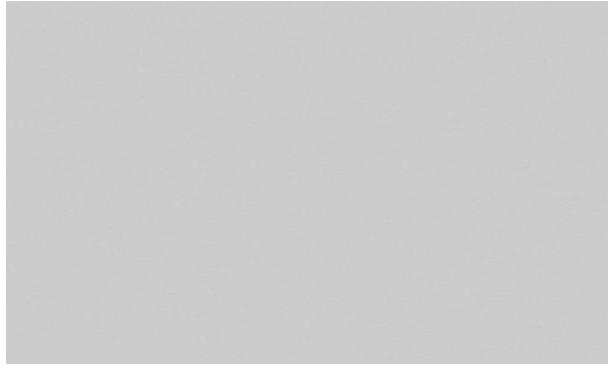
(b) Keine Fehldetektion im ursprünglichen Simulationsbild



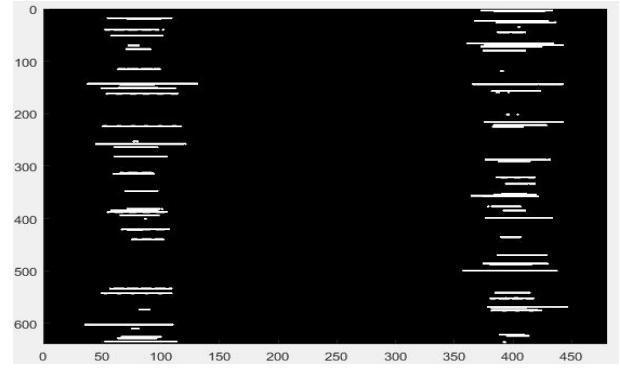
(c) Leeres Bild unter schlechten Sichtbedingungen



(d) Keine Fehldetektion unter schlechteren Sichtbedingungen



(e) Leeres Bild unter sehr schlechten Sichtbedingungen



(f) Einige Störpunkte aus dem Bild ohne Objekt unter sehr schlechten Sichtbedingungen

**Abbildung 5.4:** In den Vergleichsbildern ohne Objekt wird auch unter allen Sichtbedingungen kein Objekt detektiert. Lediglich in f) sind einige Störpunkte im Binärbild vorhanden, die auf den niedrig gewählten Templateschwellenwert [Kapitel 4.3.1] zurückzuführen sind.

## Tests auf Realbildern

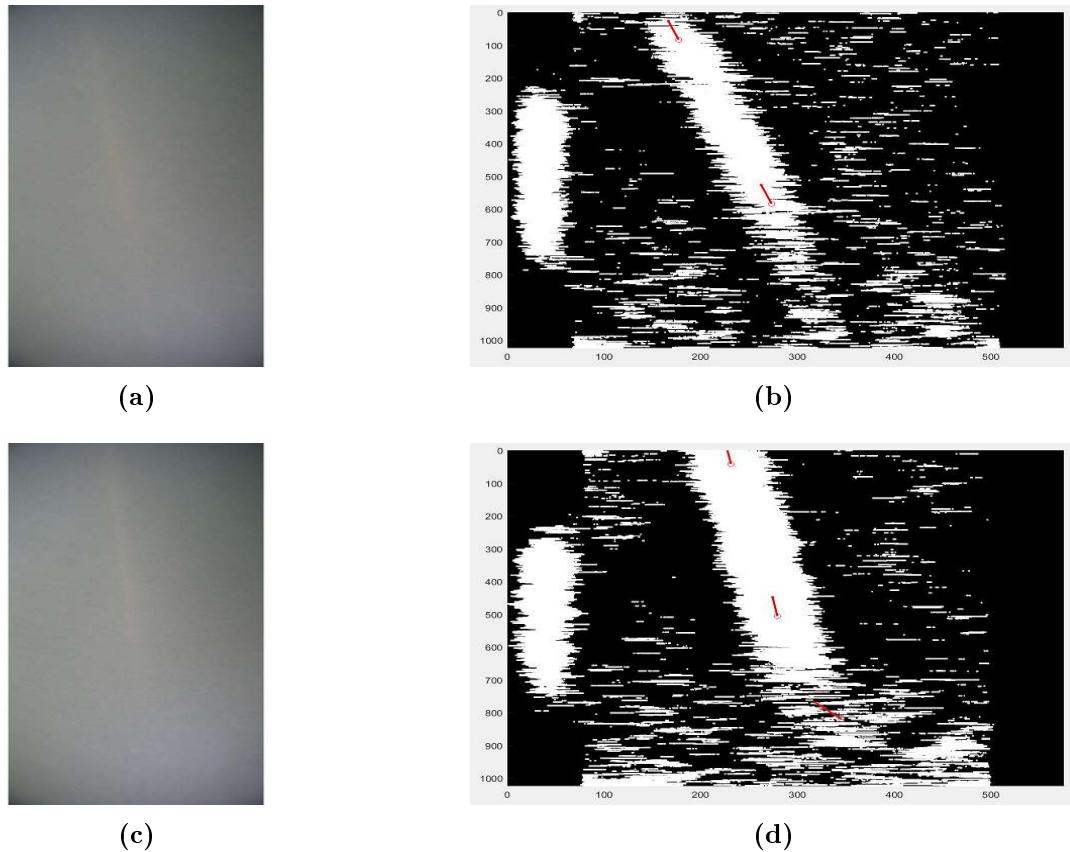
Um zu testen, ob das Verfahren auch bei echten Bildern funktioniert, wurden insgesamt 12 Bilder [Anhang G.1] aus einem Testlauf des AUVs *DAGON*<sup>9</sup> während des Projektes *CUSLAM*<sup>10</sup> im Unisee getestet. Diese Testbilder wurden so ausgewählt, dass die relevanten Fälle abgedeckt sind. Es werden Bilder mit sehr schlechten Sichtbedingungen, in denen die Pipeline sehr schwer zu erkennen ist, in anderen ist die Pipeline sehr gut sichtbar und hebt sich deutlich vom Hintergrund ab. Zudem werden noch Bilder gewählt, in denen die Pipeline zuerst gut und im Bildverlauf immer schlechter sichtbar ist. Außerdem sind in der Auswahl noch Bilder, in denen die Pipeline direkt angestrahlt wird und dadurch sehr stark reflektiert.

In den Bildern, in denen die Pipeline kaum zu erkennen ist, wie in Abbildung 5.5b oder 5.5d, muss der Schwellenwert für das Template entsprechend niedrig gesetzt werden, was zu vielen Punkten im Binärbild führt, die nicht zum Objekt gehören. In Bildern, in denen die Pipeline direkt angestrahlt wird und klar heller ist als der Hintergrund, wie in Abbildung 5.6, kann der Schwellenwert höher angesetzt werden, um weniger Störpunkte zu erhalten.

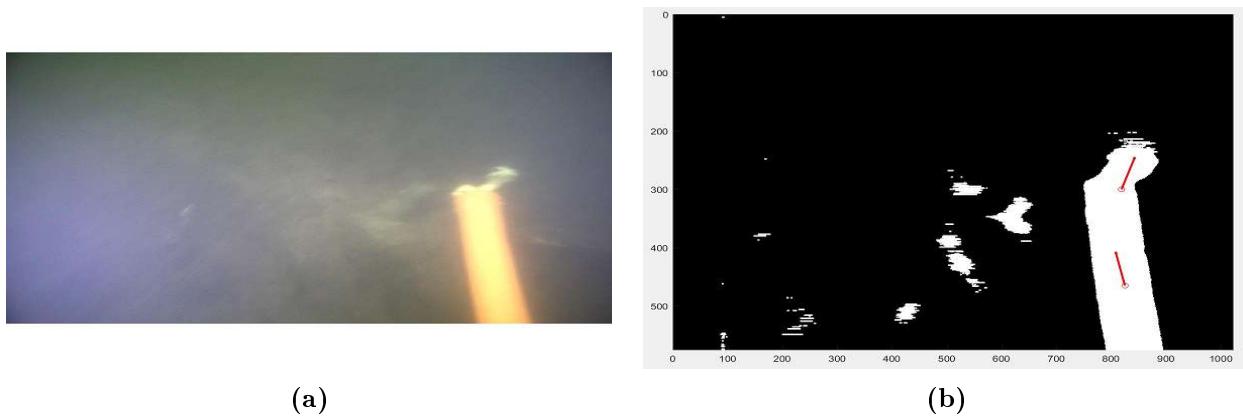
---

<sup>9</sup> <http://robotik.dfki-bremen.de/de/forschung/robotersysteme/dagon.html>

<sup>10</sup> <http://robotik.dfki-bremen.de/de/forschung/projekte/cuslam.html>



**Abbildung 5.5:** Tests der Objekterkennung auf realen Bildern aufgenommen im Unisee vom AUV *DAGON*. Trotz sehr schlechter Sichtbedingungen und vieler Störpunkten im Binärbild wird die Pipeline in den oberen zwei Segmenten gut erkannt. In d) ist im unteren Bereich eine Fehldetektion aufgrund der hohen Störpunktdichte in diesem Bereich. Die Randbereiche werden während des Binärisierungsprozess am Bild hinzugefügt, um das Template auch auf den ersten und letzten Pixeln anwenden zu können. Diese Bereiche werden bei der Detektion nicht betrachtet.



**Abbildung 5.6:** Test der Objekterkennung auf einem realen Bild aufgenommen im Unisee vom AUV *DAGON*. Die Pipeline reflektiert sehr stark und hebt sich dadurch deutlich vom Hintergrund ab. Jedoch gibt es eine starke Reflexion nah an der Pipeline, was zu einer Fehldetektion im zweiten Segment führt.

In den durchgeföhrten Tests wird deutlich, dass die Objekterkennung unter verschiedensten Sichtbedingungen gute Ergebnisse liefert. Die Sichtbedingungen haben dabei einen Einfluss auf die Binärisierung des Eingabebildes. Je schlechter die Sichtbedingungen sind, desto geringer muss der Templateschwellenwert gewählt werden. Der Schwellenwert ist jedoch ausschlaggebend für die Qualität der Abbildung vom Objekt im Binärbild. Umso geringer der Schwellenwert gewählt wird, umso mehr Störpunkte sind auch im Binärbild vorhanden (vgl. Abb. 5.5 mit 5.6). Bei schlechten Bedingungen muss der Schwellenwert jedoch gering gewählt werden, um das Objekt abbilden zu können (vgl. Kapitel 4.3.1). Somit steigt bei schlechteren Sichtbedingungen die Gefahr falsche Detektionen aufgrund von Störpunkten zu erhalten oder auch Objekte zu detektieren, wo keine vorhanden sind.

## **5.2. Testläufe**

Im folgenden werden die verschiedenen Testläufe genau beschrieben. Für alle Testläufe gilt, dass das AUV zuerst 30 Meter geradeaus fährt, bevor es auf das Objekt trifft. Dadurch wird eine stabile Fahrt erreicht und die Schwankungen beim anfänglichen Beschleunigen verfälschen die Ergebnisse nicht. Ebenso wird auch gewährleistet, dass das AUV stets direkt auf das Objekt trifft, da das Explorieren und Auffinden des Objektes nicht Teil der Arbeit ist.

Zu jedem Testlauf befindet sich auf der CD ein Video, in dem das AUV von oben, sowie die Rohbilder der Kamera, die Ausgabe der Objekterkennung und das berechnete Polynom zu sehen sind.

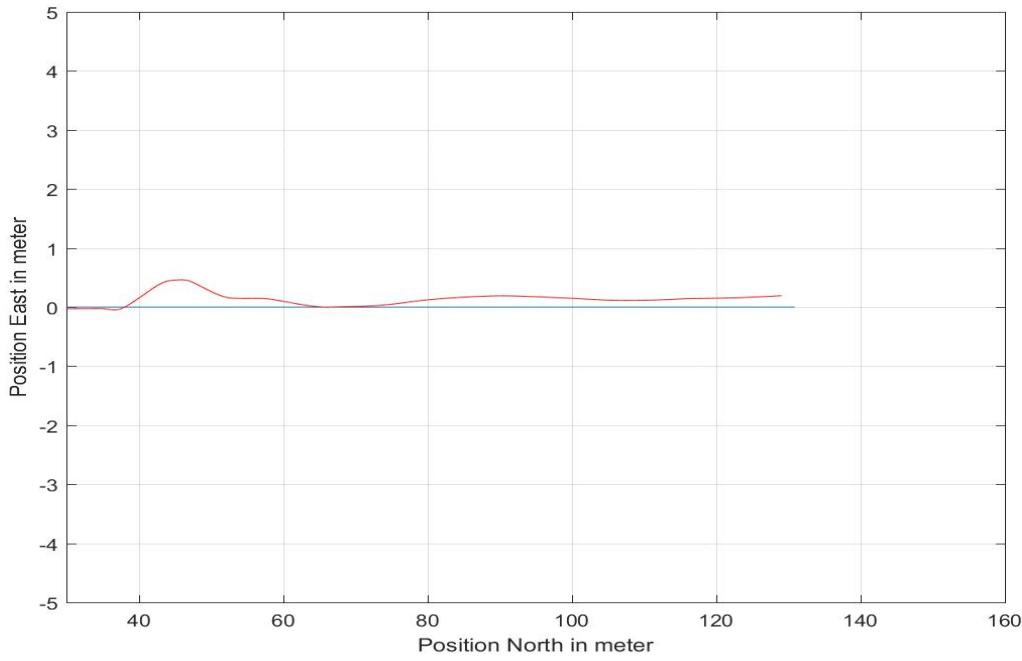
### **5.2.1. Gerader Verlauf**

Für die ersten Tests wurde ein 100 Meter langes Objekt gerade in die Simulationsumgebung eingefügt. Dieses Objekt ist in mehreren Bereichen vom Meeresboden leicht bis komplett bedeckt.

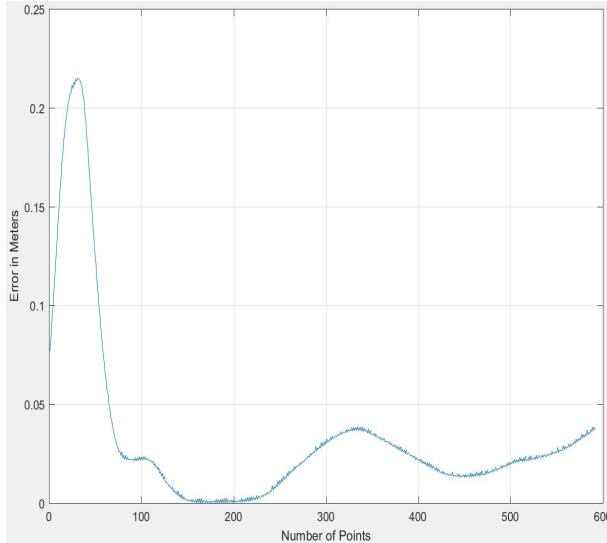
Die Testläufe mit geraden Objekten zeigten, dass mit steigender Anzahl an detektierten Punkten die Gerade immer besser vom Schätzverfahren abgebildet wurde. Es sind nach einer längeren geraden Strecke, in der sich das Fahrzeug einpendeln kann, auch weitere Strecken ohne Sichtkontakt kein Problem.

5. Tests und Evaluation

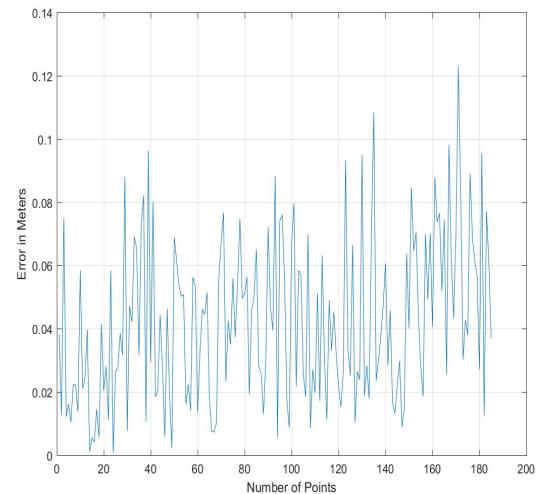
---



(a) Fahrtverlauf des AUVs (rot) an einem geraden Objekt (blau). Nach erstem Sichtkontakt zum Objekt ist ein Einpendeln auf die gerade Linie zu beobachten.



(b) Absoluter Fehler der AUV-Position zur echten Position des Objektes. Auch hier ist zu beobachten, dass ein großer Fehler zu Beginn des Objektes auftritt, der beim Fahrtverlauf weiter verrin-gert wird.



(c) Absoluter Fehler der detektierten Objektposi-tion zur echten Objektposition. In Betrachtung von b) ist zu beobachten, dass der Fehler der de-tektierten Objektposition größer ist als der Feh-ler im daraus resultierenden Fahrtverlauf. Dies ist darauf zurückzuführen, dass es Fehleraus-schläge zu beiden Seiten des Objektes gibt, die durch die Regression ausgeglichen werden.

**Abbildung 5.7:** Testlauf an einem geraden Objekt. Nach anfänglich größerem Fehler folgt das AUV dem Objekt mit nur sehr geringem Fehler. Das Einpendeln ist auf die Be-rechnung des Polynoms zurückzuführen, da bei wenigen Punkten zu Beginn der Verlauf noch nicht eindeutig als Gerade bestimmbar ist. Siehe hierfür Kapitel 5.3.1.

### **5.2.2. Kurve**

Nach den Tests mit geraden Objekten wurden kurvige Objekte mithilfe von Polynomial- und Exponentialfunktionen in die Simulation eingefügt. Hierbei wurde darauf geachtet kein Polynom zweiten Grades zu verwenden, um dem Regressionsverfahren keine perfekte Lösung zu bieten.

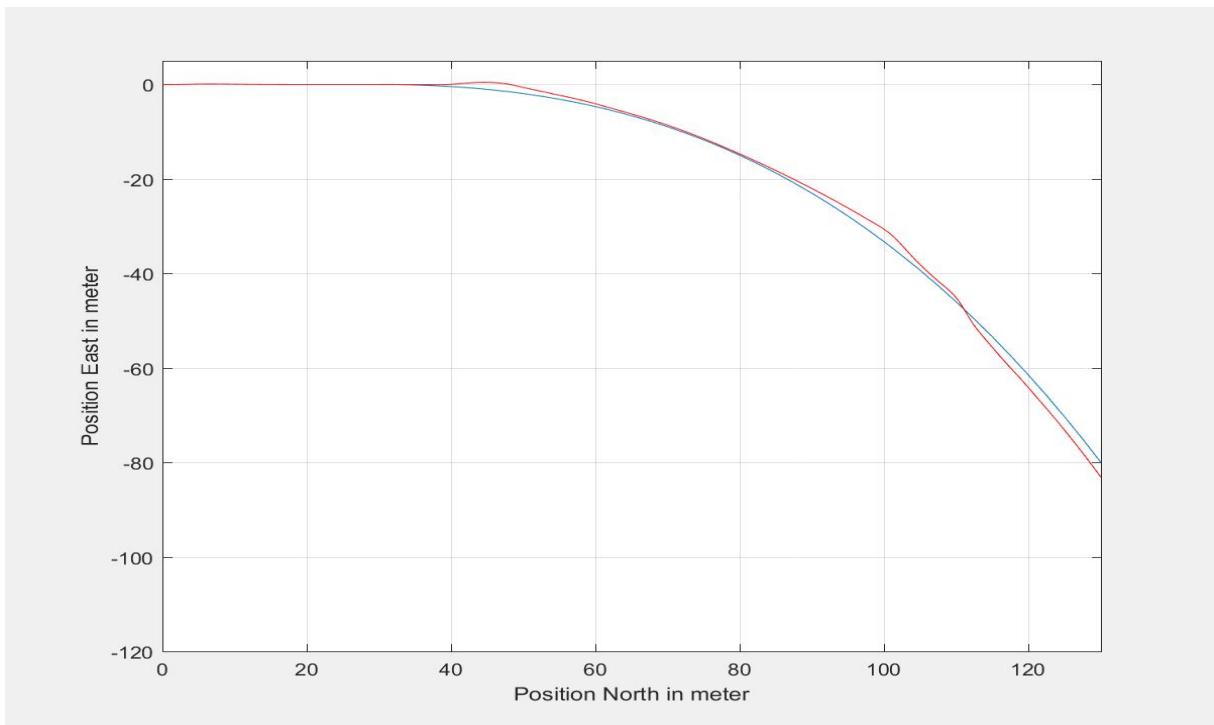
Ziel dieser Tests ist es zu zeigen, dass das Folgen einer Links- sowie Rechtskurve und auch ein Wechseln zwischen beiden Kurvenarten möglich ist. Für letzteres wurde eine Sinuskurve genutzt, um ein entsprechendes Objekt zu erzeugen.

Außerdem wurde noch eine Kurve nach einer langen Geraden erzeugt. Hiermit wird getestet, ob auch wechselnden geometrischen Strukturen gefolgt werden kann.

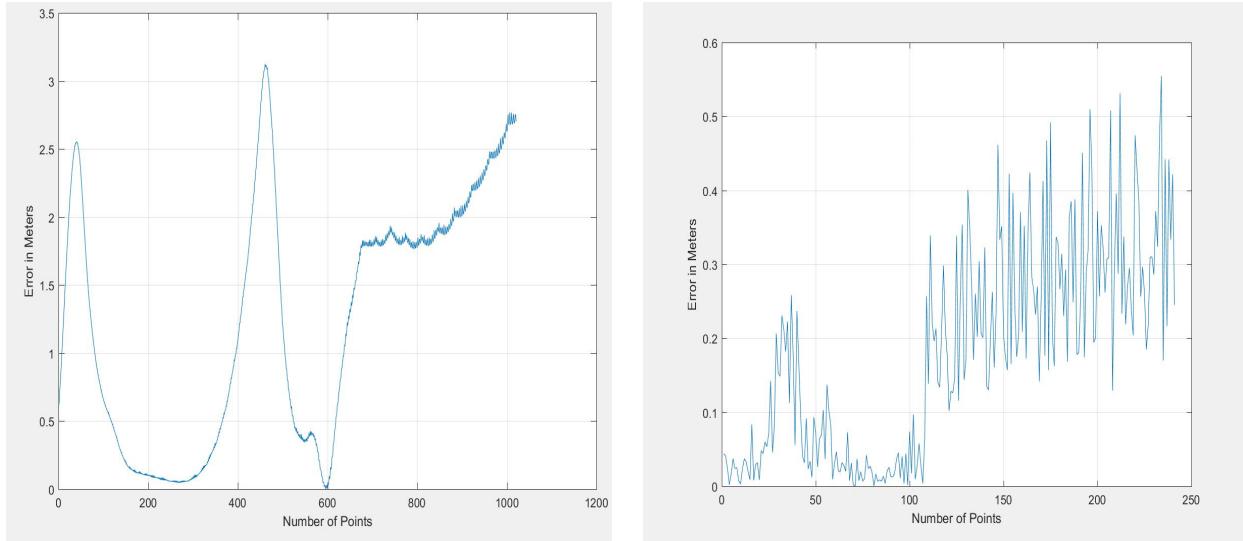
Bei den Tests mit kurvigen Objekten ist zu beobachten, dass zu Beginn jeder Kurve zunächst ein größerer Fehler zur Objektlage entsteht. Bei lang gezogenen Kurven wird dieser Fehler schnell wieder ausgeglichen. Auch bei mehreren Kurven ist dieses Verhalten zu beobachten (siehe Abb. 5.12).

5. Tests und Evaluation

---



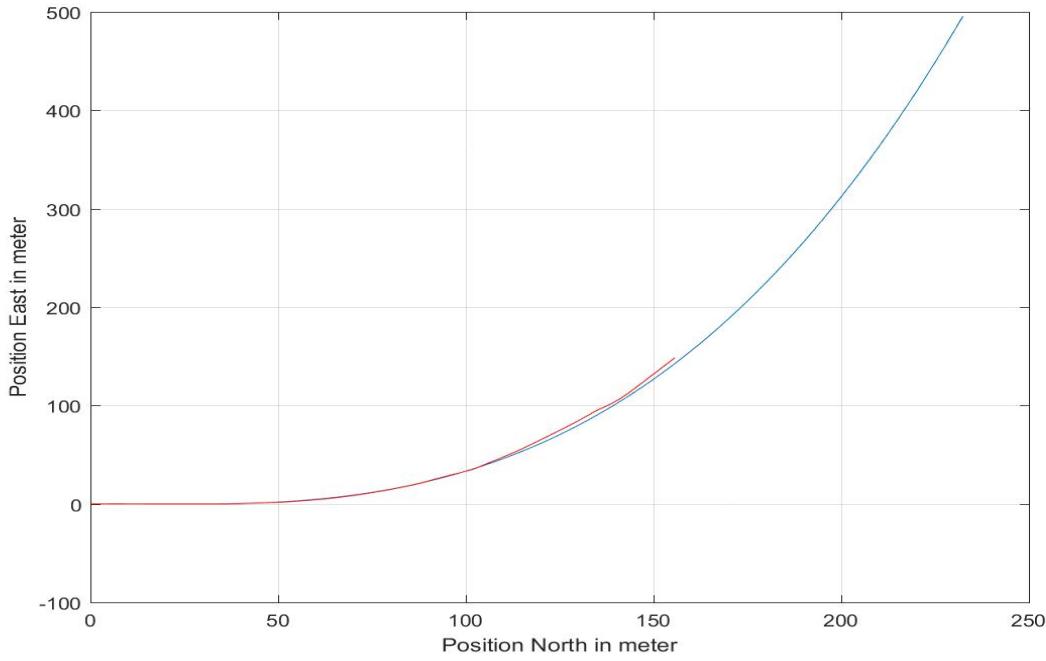
(a) Fahrtverlauf (rot) bei einer Kurve (blau).



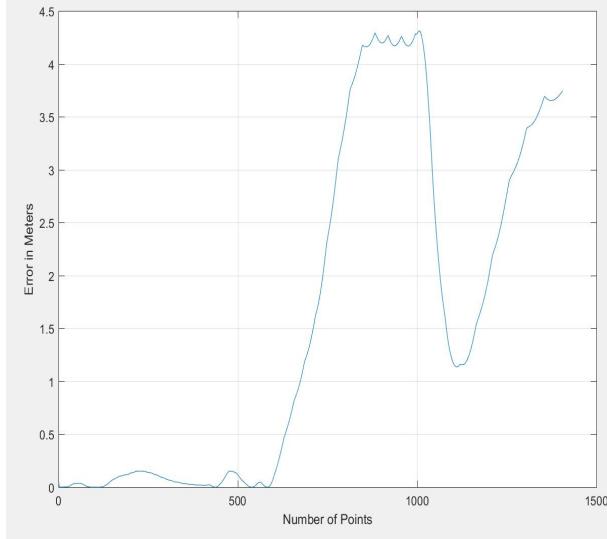
(b) Fehler der AUV Position zur echten Position des Objektes. Am Ende ist zu beobachten, wie sich der systematische Fehler aus c) in einem beständigen Fehler der Fahrt resultiert.

(c) Fehler der detektierten Objektposition zur echten Objektposition. Es scheint, dass die zweite Hälfte der Punkte einen systematischen Fehler hat. Siehe hierfür Kapitel 5.4.

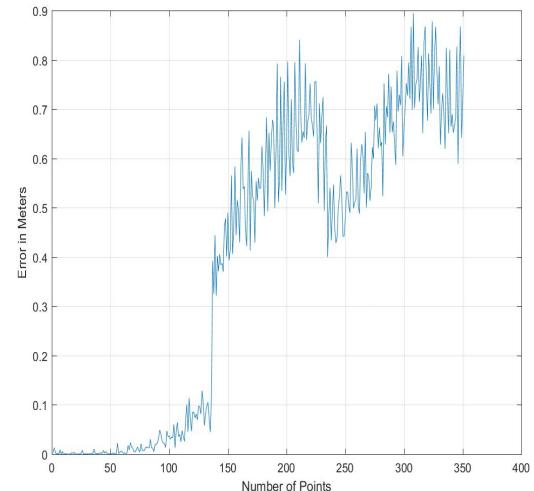
**Abbildung 5.8:** Testlauf mit einer Kurve. In a) und b) ist zu erkennen, dass einige Meter benötigt werden, um auf die Kurve zu reagieren. Der zweite größere Fehlerausschlag ist durch eine teilweise komplett Verdeckung des Objektes zu erklären. In a) ist sehr gut zu beobachten, dass der Fehler zuerst ansteigt, sobald das Objekt nicht sichtbar ist, bei erneuter Detektion des Objektes aber sehr schnell korrigiert wird.



(a) Fahrtverlauf (rot) bei einer Kurve (blau).

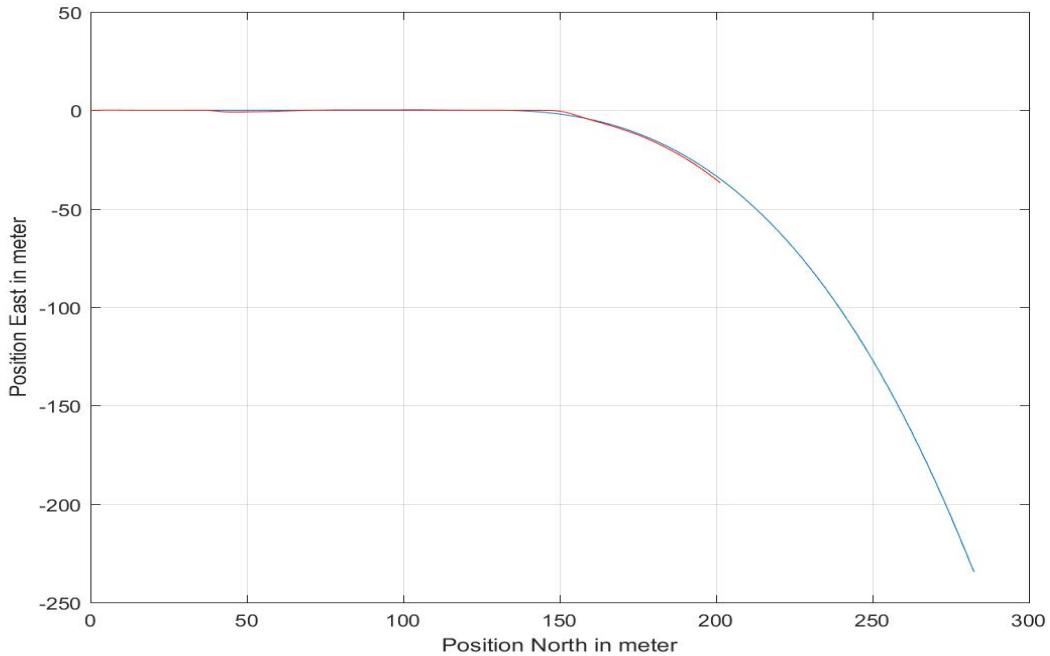


(b) Fehler der AUV Position zur echten Position des Objektes.

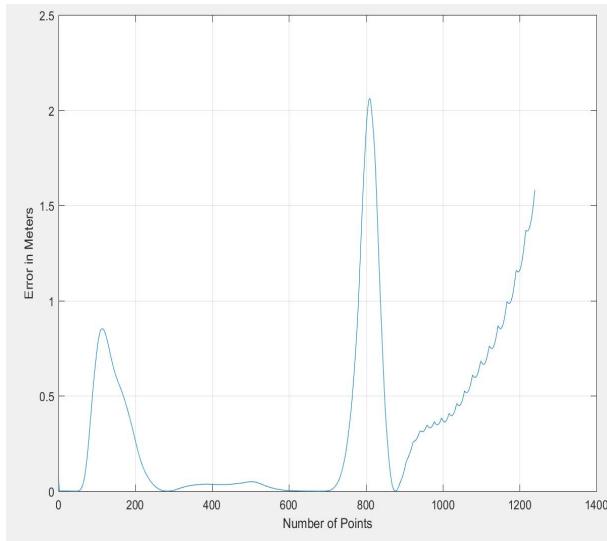


(c) Fehler der detektierten Objektposition zur echten Objektposition.

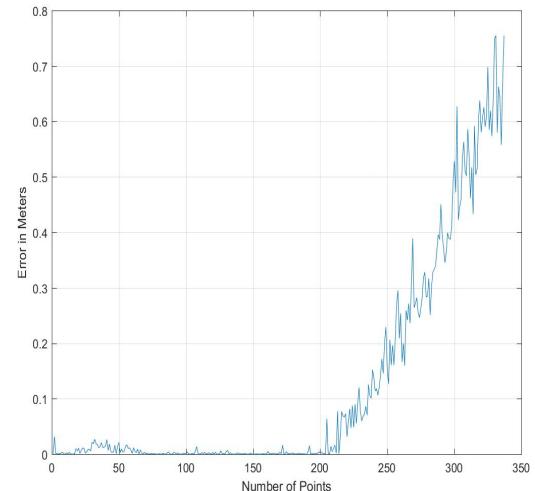
**Abbildung 5.9:** Testlauf mit einer Kurve. In diesem Lauf wurde die Kurve, anders als in Abb. 5.8, in die andere Richtung erzeugt. Wie zu erwarten sind die Ergebnisse in diesem Lauf analog zur anderen Kurve.



(a) Fahrtverlauf (rot) bei einer Kurve nach einer längeren Gerade (blau).

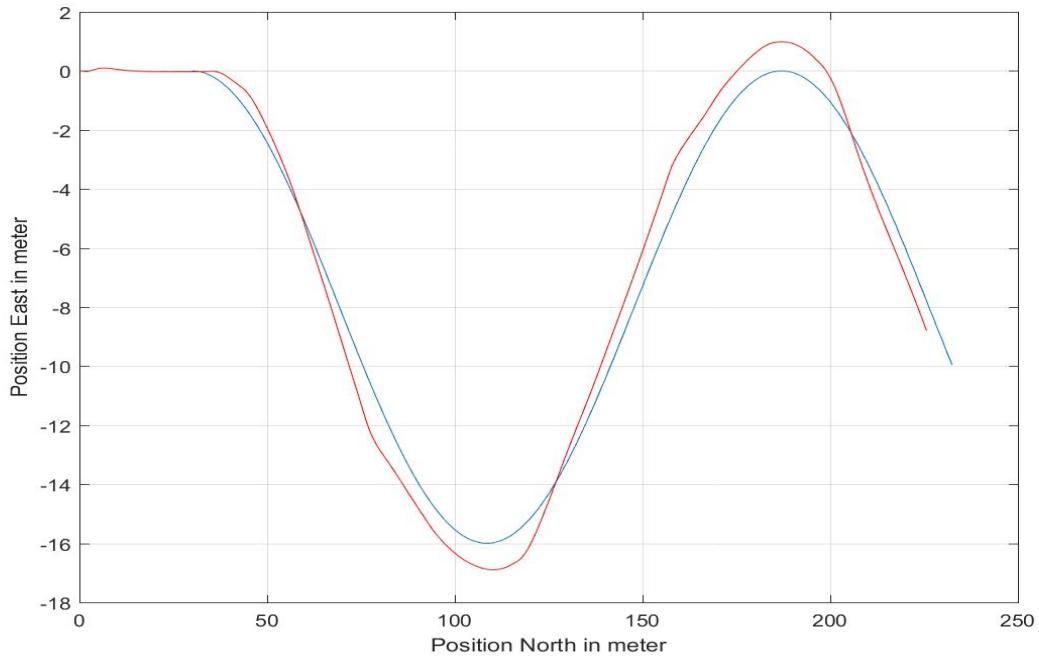


(b) Fehler der AUV Position zur echten Position des Objektes.

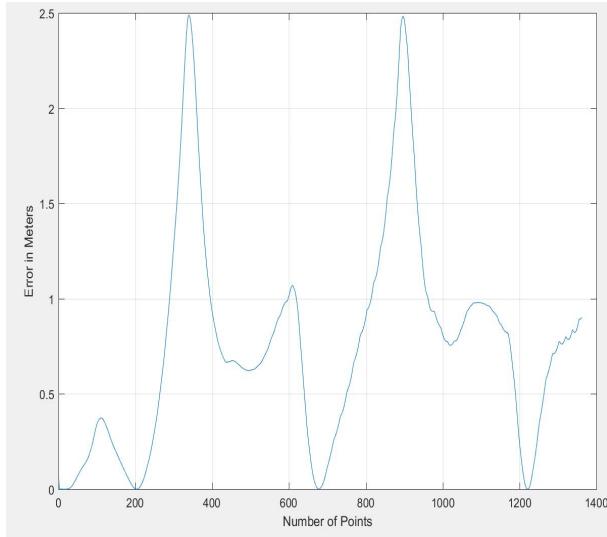


(c) Fehler der detektierten Objektposition zur echten Objektposition.

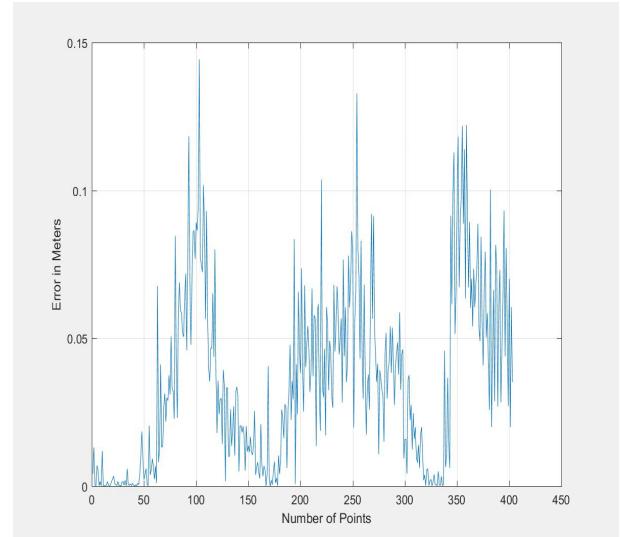
**Abbildung 5.10:** Testlauf mit einer Kurve nach einer längeren Gerade. In diesem Lauf wird der Wechsel zwischen Gerade und Kurve getestet. Hierbei ist zu sehen, dass die gerade Strecke zunächst gut verfolgt wird. Der Wechsel zur Kurve resultiert in einem größeren Fehler und einem Einpendeln. Dieses Einpendeln fällt stärker aus, da die Parameter so gewählt werden mussten, dass schnell auf Kurven reagiert werden kann (vgl. 5.3)



(a) Fahrtverlauf (rot) bei einer Sinuskurve (blau).

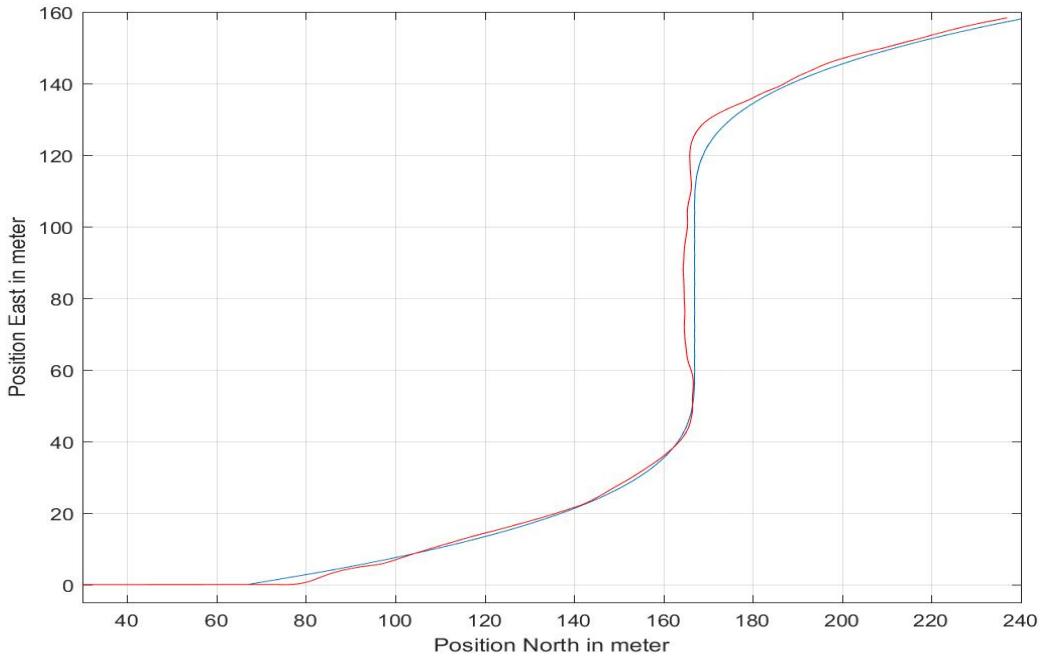


(b) Fehler der AUV Position zur echten Position des Objektes. Eine interessante Beobachtung in dieser Grafik ist der sehr ähnliche Fehlerausschlag

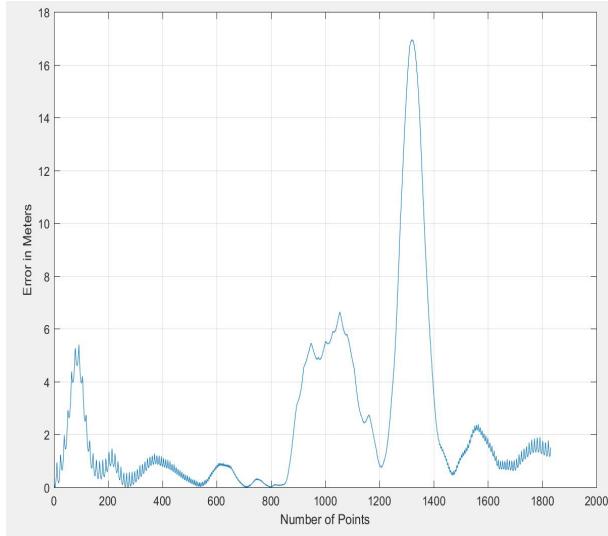


(c) Fehler der detektierten Objektposition zur echten Objektposition.

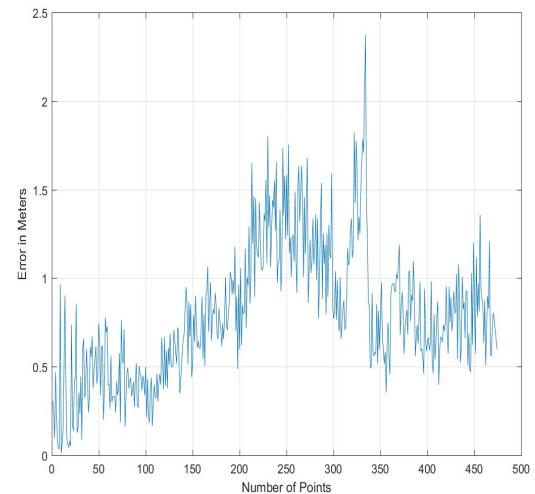
**Abbildung 5.11:** Beim Testlauf mit der Sinuskurve ist zu beobachten, dass es innerhalb der Kurven aufgrund der Richtungsänderung des Verlaufs einen größeren Fehler der Verfolgung gibt. Nach der Kurve wird dem Objekt jedoch schnell wieder gut gefolgt.



(a) Fahrtverlauf (rot) bei einem kurvigen Objektverlauf(blau). Da die Kurve zu Beginn einen starken Knick macht, tritt dort ein größerer Fehler auf, bis richtig reagiert wird.



(b) Fehler der AUV Position zur echten Position des Objektes. Trotz des Fehlers im geraden Bereich und dem sehr großen Fehler innerhalb der Rechtskurve wird das Objekt nach dem Auschlag wieder gut verfolgt.



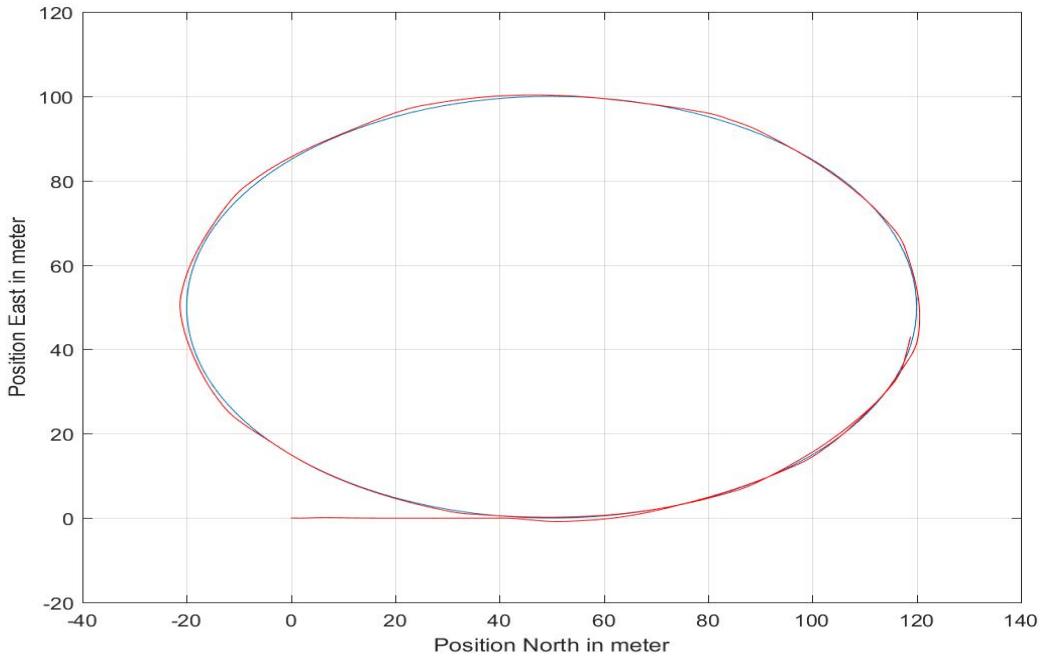
(c) Fehler der detektierten Objektposition zur echten Objektposition. Der hier zu beobachtende Fehler ist im gesamten Bereich hoch.

**Abbildung 5.12:** Den Kurven in diesem Lauf ist für das AUV aufgrund der starken Krümmung schwer zu folgen. Im mittleren Bereich tritt auf längerer Strecke ein größerer Fehler auf. Aufgrund dieses Fehlers wird die Rechtskurve fast *verpasst* jedoch aufgrund des Schätzverfahrens trotzdem noch verfolgt. In diesem Lauf ist der Zusammenhang zwischen Positions- und Detektionsfehler deutlich zu erkennen.

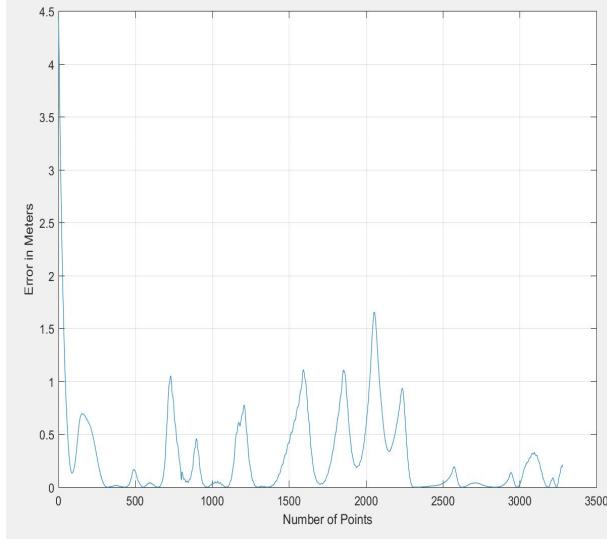
### 5.2.3. Ellipsen

Für die finalen Tests wurden Ellipsen verwendet. Eine Ellipse erfüllt einige Eigenschaften, die für die Arbeit zu nicht trivial zu lösenden Problemen führen. Zum einen gibt es verschieden stark gebogene Kurven und fast gerade Abschnitte. Zum anderen gibt es kontinuierliche Abschnitte, die parallel zur *Y – Achse* verlaufen.

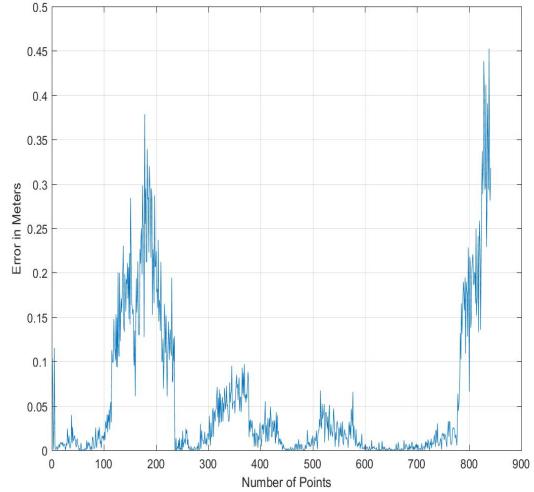
In diesen Tests ist zu sehen, dass sowohl Ellipsen, als auch der Wechsel zwischen Geraden und Ellipsenabschnitten gefolgt werden kann.



(a) Fahrtverlauf (rot) bei einer Ellipse (blau). Es wurde anderthalb mal um die Ellipse gefahren.

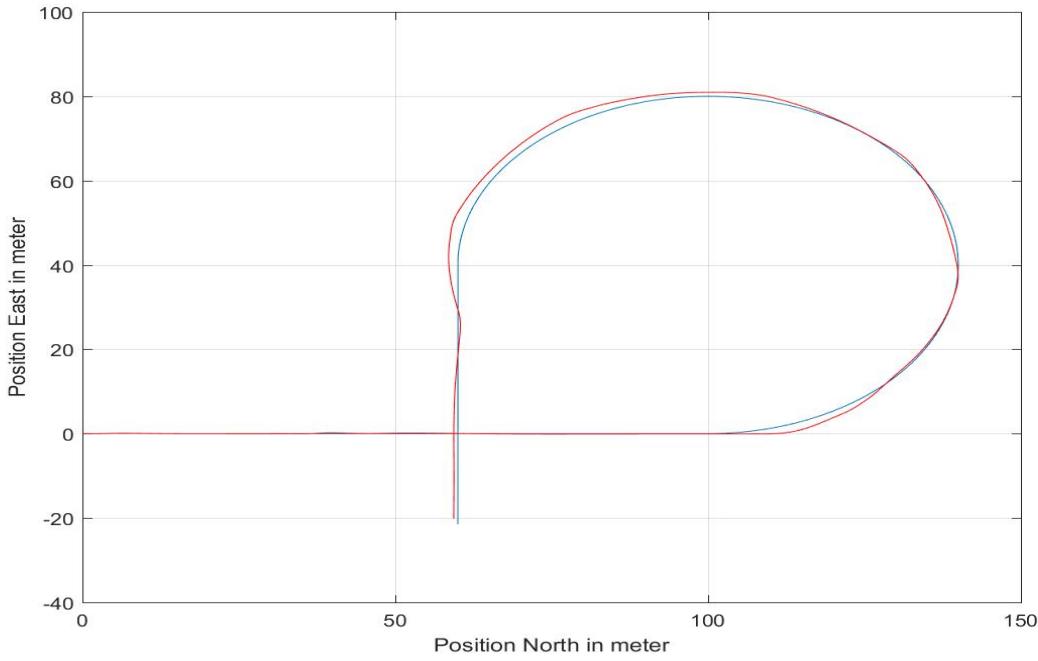


(b) Fehler der AUV Position zur echten Position des Objektes. Es ist ein gleichmäßiges Auftreten von Fehlerspitzen zu beobachten. Der größte Ausschlag ist einer Unsichtbarkeit des Objektes innerhalb des rechten oberen Ellipsenabschnitts zuzuschreiben.

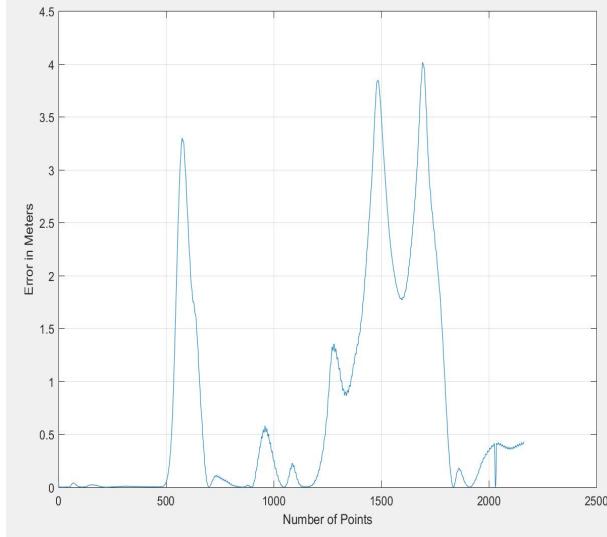


(c) Fehler der detektierten Objektposition zur echten Objektposition. Es sind zwei Bereiche mit größerem Fehler zu beobachten. Diese liegen beide im unteren linken Bereich der Ellipse, in dem das Objekt teilweise vom Meeresboden bedeckt ist.

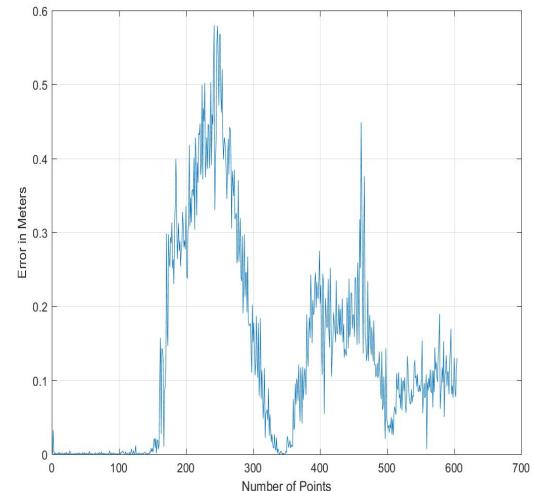
**Abbildung 5.13:** Im Testlauf der Ellipse ist zu beobachten, wie die ständig ändernde Krümmung der Bahn zu Fehlerspitzen führt. Bei jeder Spitze ist der Fehler der Regression so hoch, dass eine Transformation der detektierten Punkte stattfindet (siehe Kapitel 4.4), die zu einer direkten Abnahme des Fehlers führt.



(a) Fahrtverlauf (rot) bei einer Ellipse mit geraden Abschnitten (blau).



(b) Fehler der AUV Position zur echten Position des Objektes.

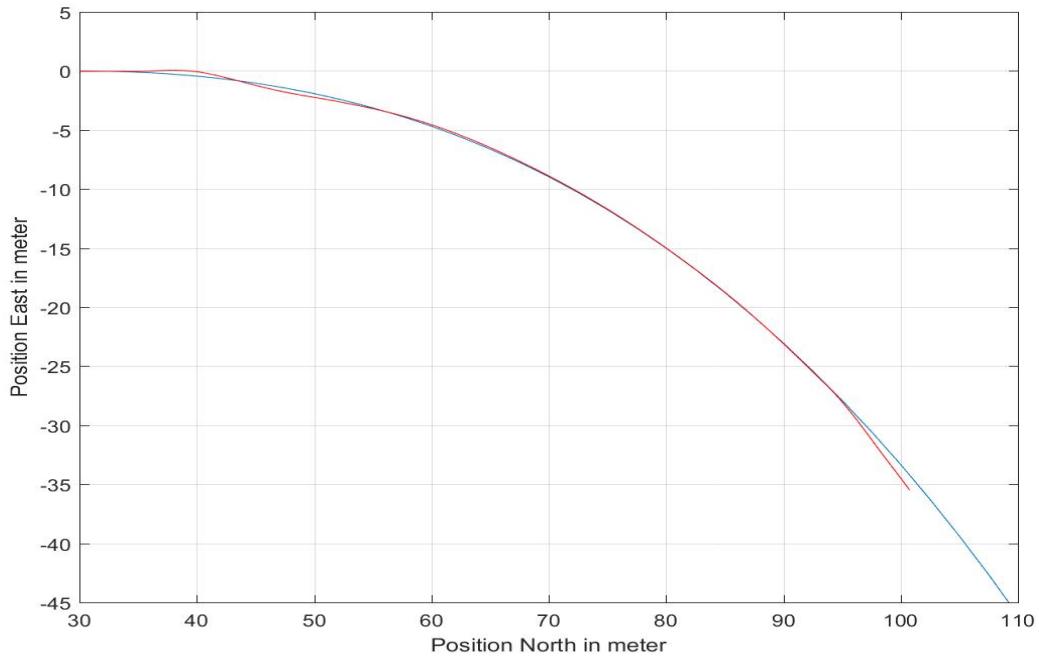


(c) Fehler der detektierten Objektposition zur echten Objektposition.

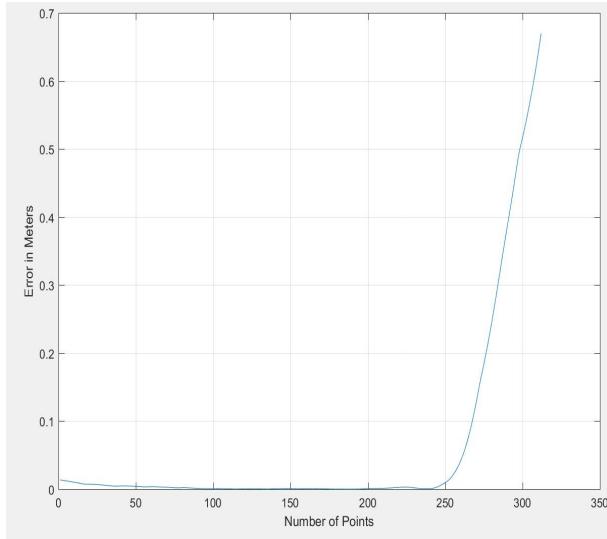
**Abbildung 5.14:** Testlauf von einer Ellipse gemischt mit geraden Abschnitten. Beim Übergang in die Ellipse ist ein erwarteter hoher Fehler zu beobachten, der durch den Wechsel der Form zu erklären ist. Innerhalb der Ellipse sind einige verdeckte Bereiche, was die Fehler in b) und c) erklärt.

#### **5.2.4. Schlechte Sichtbedingungen**

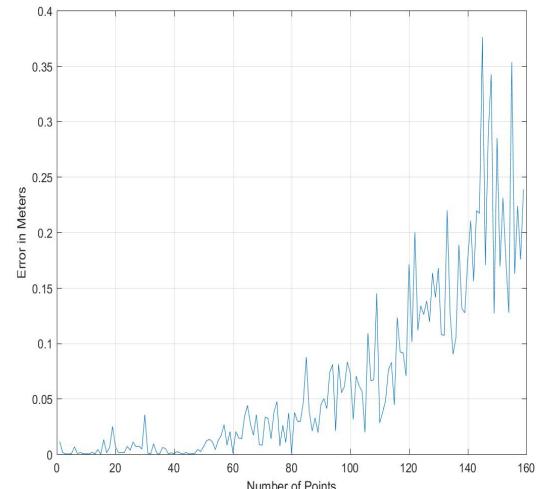
Einige Tests wurden mit sehr schlechten Sichtbedingungen (siehe Abb. ??) wiederholt. In den Tests ist zu sehen, dass selbst unter diesen Bedingungen dem Objekt gefolgt werden kann. Jedoch ist bei diesen Tests zu beachten, dass der Templateschwellenwert sehr gering gewählt wurde und dadurch eine sehr sensitive Objekterkennung genutzt wurde. Da jedoch in der Simulation nur Meeresboden und Zielobjekt existieren führte dies nicht zu Problemen. Schon kleinste Objekte, wie kleinere Felsen oder Pflanzen, würden bei einem solch geringen Schwellenwert zu schwerwiegenden Fehldetections führen. Tests mit anderen Objekten in der Simulationsumgebung wurden jedoch nicht durchgeführt, da es keine einfache Methode gibt Steine oder Pflanzen hinzuzufügen. Die Tatsache, dass aber selbst der Meeresboden schon zu Störpunkten innerhalb des Binärbilds führt unterstützt diese Aussage.



(a) Fahrtverlauf des AUVs (rot) bei einer Kurve (blau) unter schlechten Sichtbedingungen.

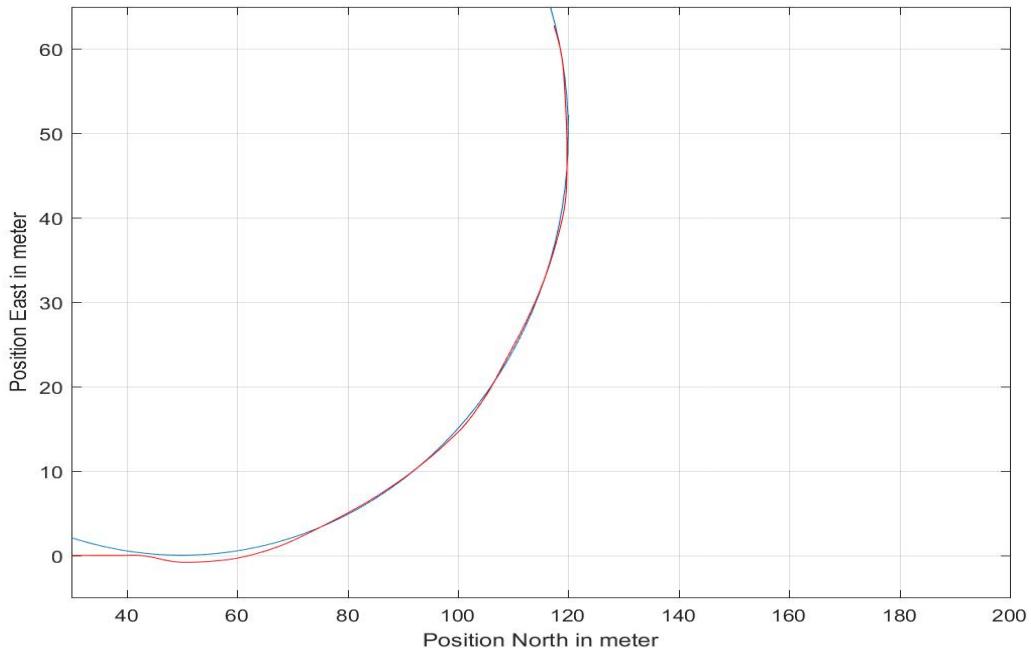


(b) Fehler der AUV Position zur echten Position des Objektes.

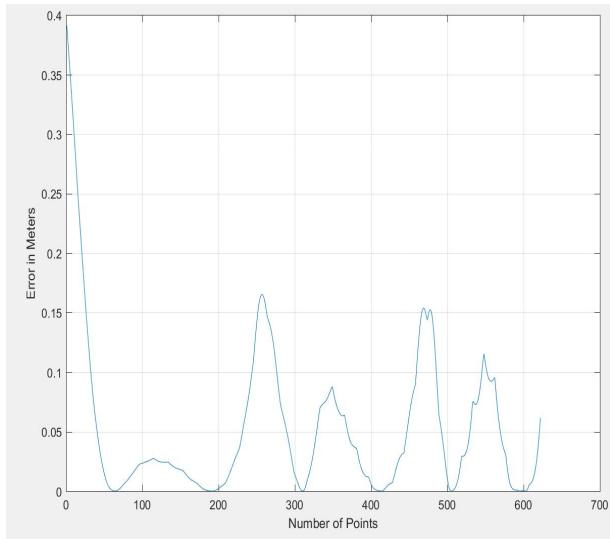


(c) Fehler der detektierten Objektposition zur echten Objektposition. Der Anstieg zum Ende ist auf leichte Verdeckung des Objektes zurückzuführen.

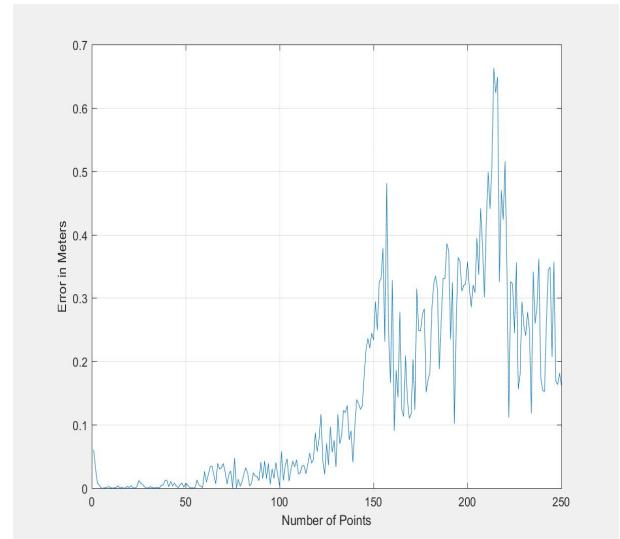
**Abbildung 5.15:** Das Folgen der Kurve funktioniert auch bei schlechten Sichtbedingungen. Es ist jedoch zu beobachten, dass eine nur leichte Verdeckung des Objektes einen starken Einfluss auf den Fehler hat.



(a) Fahrtverlauf (rot) bei einer Ellipse (blau) unter schlechten Sichtbedingungen.



(b) Fehler der AUV Position zur echten Position des Objektes.



(c) Fehler der detektierten Objektposition zur echten Objektposition.

**Abbildung 5.16:** Ähnlich zu Abb. 5.15 wird dem Objekt gut gefolgt und leichte Verdeckungen haben einen starken Einfluss auf den Fehler.

### 5.3. Parametrisierung

Während der Tests ist stark aufgefallen, dass das Regressionsverfahren sehr parameterabhängig ist. Wie im vorherigen Kapitel zu sehen ist, liefert die implementierte Lösung mit der richtigen Parametrisierung ein gutes Verhalten. Bei falscher Parametrisierung wird die Verfolgung sehr viel schlechter bis unmöglich.

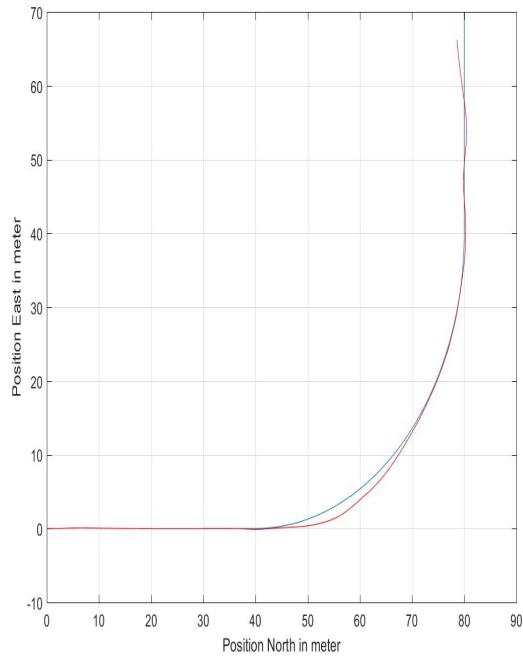
Große Unterschiede gibt es bei geraden und kurvigen Objekten. Als ausschlaggebende Parameter stellten sich die Gewichtungen der einzelnen Fehlerarten, die Stärke der *Tikhonov Regularisierung* (siehe Gleichung 9) und der maximale Gesamtfehler der Regression bis zu einer Transformation (siehe Abschnitt 4.4) heraus. Bei geraden Objekten führt eine Gleichgewichtung der Fehlerarten, eine starke *Tikhonov Regularisierung* und ein hoher erlaubter Maximalfehler zu sehr guten Ergebnissen (vgl. Kapitel 5.3.1).

Kurvige Objekte, wie in Abb. 5.12 oder 5.14, führen bei einer höheren Gewichtung des Orientierungsfehlers, keiner *Tikhonov Regularisierung* und einem geringeren erlaubten Maximalfehler zu guten Ergebnissen. Problematisch ist, dass die *falsche* Parametrisierung (z.B. hoher Maximalfehler für kurvige Objekte) im schlimmsten Fall zu einem großen Fehler, damit einhergehendem Sichtverlust zum Objekt und einem schlecht geschätzten Polynom führt, dass keine erneute Annäherung zum Objekt mehr gelingt.

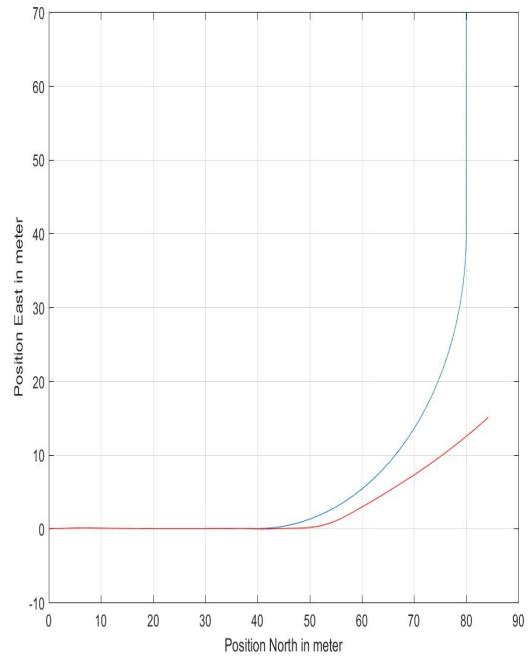
Dieses Verhalten ist im Testlauf Abb. 5.17 zu beobachten. Je schärfer die Kurven und je mehr Wechsel zwischen geraden und kurvigen Bereichen im Objekt vorhanden sind, desto schwieriger ist es die richtigen Parameter zu finden. Vor allem für den Test der Geraden gefolgt von einer Kreisbahn [Abb. 5.14] war es in der Testphase schwierig die richtige Parametrisierung zu finden ohne entweder den Wechsel zwischen Kurve und Gerade (oder zurück) zu verpassen oder aber während den Geraden aufgrund des schlechten Einpendelns [Kapitel 5.3.1] das Objekt zu verlieren.

5. Tests und Evaluation

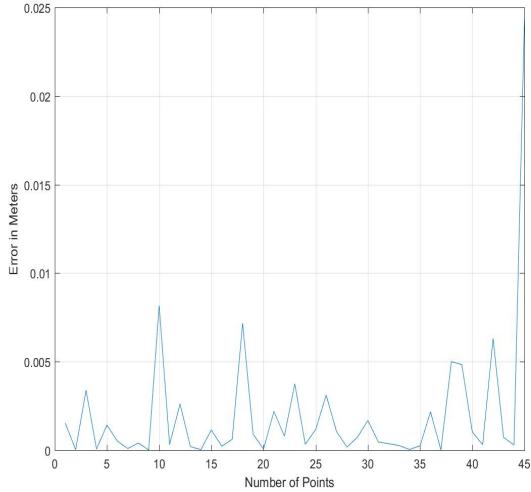
---



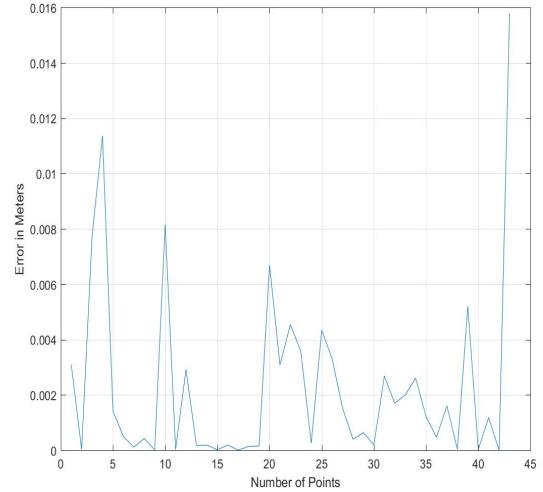
(a) Fahrtverlauf des AUVs (rot) an einer scharfen Kurve (blau) mit guten Parametern für den Objektverlauf.



(b) Fahrtverlauf des AUVs (rot) an einer scharfen Kurve (blau) mit schlechteren Parametern für den Objektverlauf.



(c) Fehler der detektierten Objektposition zur echten Objektposition.



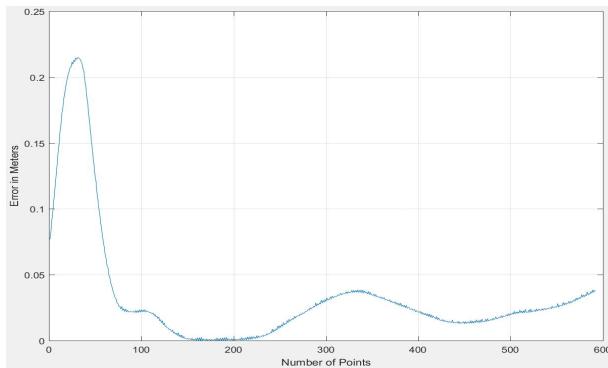
(d) Fehler der detektierten Objektposition zur echten Objektposition.

**Abbildung 5.17:** Hier wurden zwei Testläufe in der identischen Umgebung (Sichtbedingungen und Objektlage) durchgeführt. Nur die Parametrisierung der Regression wurden geändert. Es ist zu beobachten, dass trotz gleich guter Detektion des Objektes (der Fehlerausschlag in c) und d) ist sehr ähnlich) das Objekt nur mit der richtigen Parametrisierung verfolgt werden kann.

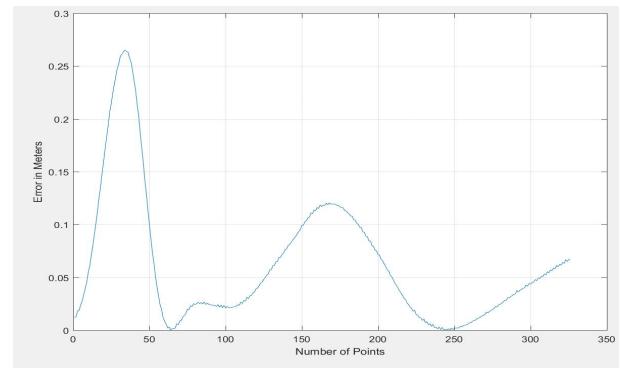
### 5.3.1. Einpendeln

In jedem Testlauf fiel auf, dass bei erster Sicht des Objektes eine Art **Einpendeln** stattfindet, also ein zuerst großer Fehler, der dann über mehrere Meter beständig abnimmt, bis eine stabile Fahrt über dem Objekt erreicht wird. Diese Beobachtung konnte auch beim Wechsel von geraden Objektverläufen auf kurvige Verläufe gemacht werden.

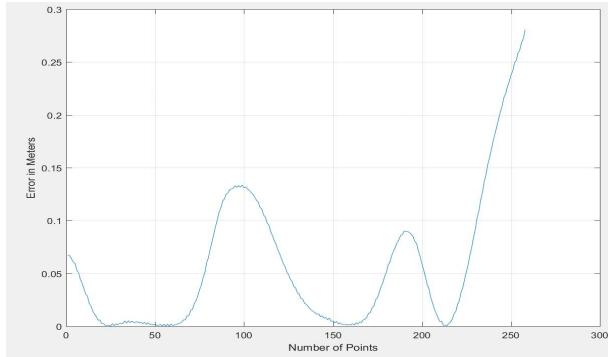
In Abbildung 5.18 ist diese Beobachtung mit verschiedenen Parametrisierungen dargestellt.



(a) Sehr gute Parametrisierung mit Gleichgewichtung der Fehlerarten, *Tikhonov Regularisierung* und hohem Maximalfehler.



(b) Gute Parametrisierung mit Gleichgewichtung der Fehlerarten, geringer *Tikhonov Regularisierung* und weder hohem, noch geringem Maximalfehler.



(c) Schlechte Parametrisierung mit höherer Gewichtung des Orientierungsfehlers, keiner *Tikhonov Regularisierung* und geringem Maximalfehler.

**Abbildung 5.18:** Testlauf mit einem geraden Objekt [Abb. 5.7] mit unterschiedlicher Parametrisierung. Im Fehler der AUV-Position zur Objektposition ist zu erkennen, dass bei den guten Parametrisierungen der Fehler über die Zeit abfällt. Bei der sehr guten Parametrisierung (a)) sogar noch weitaus schneller. Bei der schlechten Parametrisierung (c)) findet keine Verringerung des Fehlers statt. Die Höhe der Ausschläge nimmt auch nicht beständig ab, wie bei den guten Parametern.

## 5.4. Systematischer Fehler

Während einiger Testläufe gab es längere Bereiche, in denen stets mit einem leichten Versatz zum Objekt gefahren wurde. Sehr deutlich zu sehen ist dieses Verhalten in den Abbildungen 5.8, 5.10 und 5.12. Betrachtet man bei diesen drei Abbildungen jeweils den Fehler der detektierten Objektposition zur realen Objektposition, fällt auf, dass der Fehler um einen bestimmten Wert verteilt ist und systematisch erscheint.

Für das Auftreten dieses Fehlers gibt es einige mögliche Erklärungen. Zum einen fällt auf, dass der systematische Fehler meist nach scharfen Kurven oder teilweiser Unsichtbarkeit des Objektes auftritt. Da die implementierte Lösung bei ausbleibenden Ergebnissen der Objekterkennung die Fahrthöhe erhöht, um den Sichtbereich zu vergrößern und bei erneuter Sichtung wieder auf die Zielhöhe korrigiert, wird nach einem Bereich ohne Detektion stets abwärts gefahren, was zu einem erhöhten Pitch-Wert führt.

Bei scharfen Kurven wird durch die Fahrteigenschaften ein höherer Roll-Wert erreicht, als bei gerader Fahrt. Der systematische Fehler tritt also oft direkt nach Situationen auf, die eine unruhige Fahrt verursachen können. Dies lässt darauf schließen, dass die Transformation von 2D- in 3D-Koordinaten (vgl. 4.2.1) für den Fehler verantwortlich ist.

Eine andere mögliche Erklärung ist darin zu finden, dass das Objekt in den Bereichen mit systematischem Fehler oft leicht verdeckt ist. Verdeckte Objekte erscheinen im Binärbild schmäler. Da die Detektion im Binärbild über eine Box mit der erwarteten Breite des Objektes im Bild durchgeführt wird, führt ein schmäler erscheinendes Objekt zu Problemen. Bei optimal sichtbaren Objekten *passt* diese Box am besten (die meisten Inlier), wenn der Mittelpunkt der Box in horizontaler Richtung genau in der Mitte des Objektes liegt. Bei schmäler erscheinenden Objekten *passt* die Box auch an weiteren Stellen des Objektes genau so gut, wie in der Mitte. Durch den Zufallsaspekt des RANSAC-Algorithmus kann hier ein breiter Bereich als Objektmitte detektiert werden.

Gestützt wird diese Möglichkeit auch durch die Tatsache, dass die Streuung des Fehlers in Bereichen mit systematischen Fehler weitaus größer ist.

hier  
grafik

## 5.5. Laufzeittests

Tests zur Laufzeit wurden mit dem **Simulink Profiler**<sup>11</sup> durchgeführt. Der **Simulink Profiler** erstellt nach einem Simulationsdurchlauf einen Report über Laufzeiten der einzelnen Komponenten. Die wichtigen Daten, für Aussagen über Laufzeit, aus diesem Report sind die Anzahl der Aufrufe und der *self time* (Laufzeit) der Komponente.

Die Analyse der aufgezeichneten Daten während mehrerer Testläufe zeigte, dass die Objekterkennung und das Schätzverfahren die einzigen für die Laufzeitbetrachtung relevanten Komponenten sind. Andere Komponenten, wie die Berechnung der Wegpunkte oder die Transformationen fallen hingegen nicht ins Gewicht und benötigen bei einer Gesamtaufzeit der Simulation von 1000 Sekunden lediglich ca. 2 Sekunden.

Die Daten stammen aus Testläufen zwischen 1000 und 2500 Sekunden GesamtSimulationzeit. Dies entspricht zwischen 180 und 350 Komponentenaufrufen.

Die Objekterkennung benötigte hierbei im Durchschnitt ( $\frac{\text{Komponentenlaufzeit}}{\text{AnzahlAufrufe}}$ ) zwischen 1,3 und 1,56 Sekunden.

Das Schätzverfahren benötigte im Schnitt zwischen 0,65 und 0,8 Sekunden.

Zu diesen Angaben muss noch beachtet werden, dass Simulink an einigen Stellen die Arbeit limitierte und auf verlangsamende Alternativen zurückgegriffen werden musste. Zum Beispiel ist das Schätzverfahren in einem *Interpreted MATLAB Function Block*<sup>12</sup> umgesetzt. Bereits in der Dokumentation des Blocks steht die Anmerkung: "*This block is slower than the Fcn block because it calls the MATLAB parser during each integration step. Consider using built-in blocks (such as the Fcn block or the Math Function block) instead*". Dieser Block ist jedoch die einzige einfache Möglichkeit anonyme Funktionen (*function pointer*) zu verwenden, die für die Optimierungsfunktionen der *Optimization Toolbox* zwingend notwendig sind, um selbst entworfene Probleme zu lösen.

---

<sup>11</sup> <https://de.mathworks.com/help/simulink/ug/how-profiler-captures-performance-data.html>

<sup>12</sup> <https://de.mathworks.com/help/simulink/slref/interpretedmatlabfunction.html>

## 6. Fazit & Ausblick

In diesem Kapitel wird ein Fazit zu den Ergebnissen der Arbeit gezogen. Hierbei wird auch auf die Einsatzfähigkeit der Lösung auf realen Testsystemen eingegangen. Zuletzt wird noch ein Ausblick auf mögliche Folgearbeiten gegeben.

### 6.1. Fazit

Die in der Arbeit vorgestellte Lösung erzielt, wie die Tests zeigen, gute Ergebnisse. Gemäß der Zielsetzung ist es möglich, verschiedensten Objektverläufen zuverlässig zu folgen. Wie die Tests zur Objektdetektion zeigen, ist die Detektion auch unter schlechten Sichtbedingungen möglich. Auch die Verfolgung der Objekte funktioniert dabei zufriedenstellend. Einzig die Störanfälligkeit ist hierbei größer. Es hat sich gezeigt, dass die Fahrzeugregelung bei stetig wechselnden Wegpunkten nicht optimal ist, was in einigen Fällen zu einem schlechten Abfahren des eigentlich guten Polynoms führt. Trotzdem kann das verwendete Verfahren die nicht optimale Polynomverfolgung durch gute Extrapolation in den meisten Fällen ausgleichen. Da der Fokus der Arbeit nicht auf der Fahrzeugregelung liegt und auch kein voller Zugriff darauf besteht, ist der Ausgleich durch die Extrapolation ein gewünschtes Verhalten.

Weniger optimal ist jedoch die Abhängigkeit des Verfahrens von der Parametrisierung. Der erste gut zu wählende Parameter ist der Templateschwellenwert für die Objekterkennung. Ist dieser zu gering, ist das Binärbild zu sehr von Störpunkten gefüllt, ist er zu hoch, wird das Objekt nicht mehr gut im Binärbild abgebildet. Beides führt zu einer schlechteren Detektion und somit zu einem schlechteren Gesamtverhalten.

Für die Parametrisierung des Schätzverfahrens wird viel Vorwissen über die Objekte benötigt. Nur wenn bekannt ist, ob das Objekt stark gekrümmmt ist oder nicht, kann eine gute Verfolgung gesichert werden. Bei sich oft ändernden geometrischen Formen der Objekte kann nur sehr schwer eine geeignete Parametrisierung für alle Abschnitte gefunden werden.

Die naheliegendsten Anwendungen liegen in der Untersuchung von menschlich erzeugten Strukturen, wie Ölpipelines oder Unterwasserkabeln. Betrachtet man diese zwei Szenarien, ist das benötigte Vorwissen über die Strukturen durchaus gegeben. Bei Pipelines ist alleine durch den Durchmesser kein hoher Krümmungsgrad gegeben. Eine Parametrisierung ist also möglich.

Für einen Einsatz auf einem Livesystem ist die Laufzeit der Komponenten ebenfalls richtungsweisend. Mit den Angaben aus Kapitel 5.5 lässt sich eine ungefähre maximale Fahrtgeschwindigkeit abschätzen. Rechnet man mit einer Höhe von 6m über Grund und einem

Sichtwinkel von  $45^\circ$  voraus kann ein Objekt in bis zu 6m Entfernung erkannt werden. Da ein AUV mit zum Halten der Zielhöhe stets leicht nach vorne gebeugt fahren muss, sind die 6m nicht ganz zu erreichen. Um eine Kurve rechtzeitig zu entdecken und schnell genug einen geeigneten Wegpunkt zu Berechnen sollten zwei Folgebilder sich überschneiden. Bei einer Gesamtlaufzeit der Komponenten von ca. 2 Sekunden kombiniert mit den vorangegangenen Überlegungen ergibt dies eine maximale Fahrgeschwindigkeit von ungefähr  $1,5 \frac{\text{m}}{\text{sek}}$  bzw. ca. 3 Knoten. In einem Szenario, in dem Pipelines oder Kabel inspiziert werden sollen ist auch keine höhere Geschwindigkeit erforderlich. Somit ist das implementierte Verfahren durchaus für einen Einsatz im Realsystem geeignet (siehe Kapitel 6.2.3).

## 6.2. Ausblick

### 6.2.1. Parametrisierung

Da die Parametrisierung einen entscheidenden Einfluss auf das Verfahren hat, könnten folgende Arbeiten an dieser Stelle mehr Flexibilität bringen. Zum Beispiel kann ein Verfahren entwickelt werden, dass den Templateschwellenwert zur Binärisierung während der Objektdetektion dynamisch anpasst. In einer echten Mission können sich im Verlauf die Sichtbedingungen ändern. Ein fest gesetzter Schwellenwert wird in so einem Fall keine Ergebnisse mehr liefern können. Der Schwellenwert könnte bei einem über dem gesamten Bild niedrigen Rotwert gesenkt, bei vielen Störpunkten erhöht oder bei längerem Ausbleiben von Detektionsergebnissen wiederum gesenkt werden.

Ein ähnliches Prinzip kann auch für die Parameter des Schätzverfahrens angewendet werden. Ein Verfahren kann unabhängig von der Regression den aktuellen Verlauf im Bezug auf die Parametrisierung zu schätzen. Dieser Ansatz kann beispielsweise als Klassifizierungsproblem definiert werden und je nach Objektklasse ein definiertes Parameterset genutzt werden.

### 6.2.2. Regressionsverfahren

In der implementierten Lösung werden die Daten für die Regression nur quantitativ gewichtet. Sollte es also viele Fehldetections in einem Bereich geben, in dem das Objekt unsichtbar ist, würden diese Fehler das Schätzverfahren stark beeinflussen und die Verfolgung gefährden.

Hier ist eine qualitative Bewertung der Daten eine gute Lösung. Es kann bewertet werden, wie *sicher* die Objekterkennung eine Detektion als Teil des Objektes erkennt. Eine Grundlage für eine solche Bewertung wird von der implementierten Objekterkennung ausgetragen. In Listing 1 sind die Gütefaktoren angegeben. Die `peakheight` gibt an, wie groß

der Unterschied der Rotwerte der Pixel des Objektes im Vergleich zum Durchschnitt des gesamten Bildes ist. `Area` bestimmt, wie viel der erwarteten Objektfläche das detektierte Objekt ausfüllt. `relativeCount` gibt das Verhältnis der Inlier nach dem RANSAC zur Gesamtanzahl der Punkte im Binärbild an. `fitsBorder` zeigt als Boolean, ob das detektierte Objekt zur erwarteten Breite passt. `numParts` gibt an in wie vielen Bildsegmenten das Objekt erkannt wurde. `theta` und `phi` geben die Neigungswinkel des AUVs zum Zeitpunkt der Detektion an.

Mit diesen Eigenschaften als Grundlage kann ein *machine learning* Verfahren trainiert werden. Hierfür können Trainingsdaten bestehend aus den Eigenschaften und dem Fehler von jedem Punkt zum echten Objekt herangezogen werden. Nach dem Training kann dann aus der Objekterkennung ein Fehlerausschlag erwartet und ebenso eine qualitative Bewertung für die Regression durchgeführt werden.

### 6.2.3. Integration in ein Realsystem

Ein weiterer Schritt ist die Integration der Lösung auf dem Realsystem. Hierfür kann die komplette Lösung inklusive Fahrzeugregelung aus Simulink exportiert werden. Dabei ist auch eine Generierung von C/C++-Code möglich, was zu einer Verbesserung der Laufzeit führen kann.

Bevor jedoch die Lösung exportiert werden kann, muss das Regressionsverfahren angepasst werden. Da die Regression auf der *Optimization-Toolbox* basiert, welche aber nicht exportiert werden kann, muss hier eine gleichwertige Alternative entweder in MATLAB oder aber auch in Form einer C oder C++ Bibliothek gefunden werden. Ebenfalls kann in diesem Schritt die Regelung angepasst werden, sodass ein allgemein ruhiger Fahrtverlauf ermöglicht werden kann. Gerade, wenn das AUV als Sensorträger verwendet wird, ist eine ruhige Fahrt fördernd für verlässliche Sensordaten während der Mission.

werden...  
werden  
...  
werden

- ## A. Literaturverzeichnis
- [1] Jan Christian Albiez et al. „CSurvey - An autonomous optical inspection head for AUVs“. In: *Robotics and Autonomous Systems* 67 (2015), S. 72–79.
  - [2] Li Bai, Yan Wang und Michael Fairhurst. „Multiple Condensation filters for road detection and tracking“. In: *Pattern Analysis and Applications* 13.3 (2010), S. 251–262.
  - [3] Illes Balears. „Visual underwater cable/pipeline tracking“. In: (2007).
  - [4] Chris Brunsdon. „Path estimation from GPS tracks“. In: *Proceedings of the 9th International Conference on GeoComputation*. National Centre for Geocomputation, Maynooth University. 2007.
  - [5] Guowei Cai, Ben M Chen und Tong Heng Lee. *Unmanned rotorcraft systems*. Springer Science & Business Media, 2011.
  - [6] Qiang Chen und Hong Wang. „A real-time lane detection algorithm based on a hyperbola-pair model“. In: *2006 IEEE Intelligent Vehicles Symposium*. IEEE. 2006, S. 510–515.
  - [7] M Dabrowska. „Avalon by DFKI RIC and University of Bremen“. In: (2011).
  - [8] Gian Luca Foresti und Stefania Gentili. „A vision based system for object detection in underwater images“. In: *International Journal of Pattern Recognition and Artificial Intelligence* 14.02 (2000), S. 167–188.
  - [9] JO Hallset. „Simple vision tracking of pipelines for an autonomous underwater vehicle“. In: *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE. 1991, S. 2767–2772.
  - [10] Jari Kaipio und Erkki Somersalo. *Statistical and computational inverse problems*. Bd. 160. Springer Science & Business Media, 2006.
  - [11] Joel C McCall und Mohan M Trivedi. „An integrated, robust approach to lane marking detection and lane tracking“. In: *Intelligent Vehicles Symposium, 2004 IEEE*. IEEE. 2004, S. 533–537.
  - [12] Jong Woung Park, Joon Woong Lee und Kyung Young Jhang. „A lane-curve detection based on an {LCF}“. In: *Pattern Recognition Letters* 24.14 (2003), S. 2301–2313. ISSN: 0167-8655. DOI: [http://dx.doi.org/10.1016/S0167-8655\(03\)00056-4](http://dx.doi.org/10.1016/S0167-8655(03)00056-4). URL: <http://www.sciencedirect.com/science/article/pii/S0167865503000564>.
  - [13] Christopher Rasmussen. „Texture-Based Vanishing Point Voting for Road Shape Estimation.“ In: *BMVC*. Citeseer. 2004, S. 1–10.

- [14] Sivakumar Rathinam et al. „Autonomous searching and tracking of a river using an UAV“. In: *2007 American Control Conference*. IEEE. 2007, S. 359–364.
- [15] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.
- [16] Bahare Torkaman und Mohammad Farrokhi. „Real-time visual tracking of a moving object using pan and tilt platform: A Kalman filter approach“. In: *20th Iranian Conference on Electrical Engineering (ICEE2012)*. IEEE. 2012, S. 928–933.
- [17] Vincent Voisin et al. „Road Markings Detection and Tracking Using Hough Transform and Kalman Filter“. In: *Advanced Concepts for Intelligent Vision Systems: 7th International Conference, ACIVS 2005, Antwerp, Belgium, September 20-23, 2005. Proceedings*. Hrsg. von Jacques Blanc-Talon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 76–83. ISBN: 978-3-540-32046-3. DOI: [10.1007/11558484\\_10](https://doi.org/10.1007/11558484_10). URL: [http://dx.doi.org/10.1007/11558484\\_10](http://dx.doi.org/10.1007/11558484_10).
- [18] Yue Wang, Eam Khwang Teoh und Dinggang Shen. „Lane detection and tracking using B-Snake“. In: *Image and Vision computing* 22.4 (2004), S. 269–280.

2.1.	Screenshot der Simulationsumgebung . . . . .	3
2.2.	Das Bildkoordinatensystem . . . . .	5
2.3.	Das Kamerakoordinatensystem . . . . .	5
2.4.	Das Body-Koordinatensystem . . . . .	6
2.5.	Die Neigungswinkel im Body-Koordinatensystem . . . . .	6
2.6.	MATLAB und VRML Koordinatensystem . . . . .	7
3.1.	Typischer Straßenverlauf mit entsprechendem Kantenbild . . . . .	9
3.2.	Detektion eines Straßenverlaufs über eine LCF . . . . .	10
3.3.	Detektion eines Straßenverlaufs mit dem RANSAC-Algorithmus . . . . .	11
3.4.	Detektion eines Straßenverlaufs mit der <i>Hough Transformation</i> . . . . .	12
3.5.	Template zur Pipelinedetektion aus <i>CSurvey</i> . . . . .	13
3.6.	Detektion einer Pipeline im Kantenbild . . . . .	14
3.7.	Pipelinedetektion mit neuronalem Netzwerk. . . . .	15
3.8.	Schätzung einer Objektbewegung mit Kalman-Filter. . . . .	17
3.9.	Abbildung eines Pfadverlaufs auf Basis von GPS-Daten. . . . .	18
3.10.	Abbildung eines Flusses mit Regression . . . . .	19
4.1.	Systementwurf . . . . .	21
4.2.	Verfahren zur Bestimmung der Wegpunkte . . . . .	23
4.3.	Simulationsbilder . . . . .	24
4.4.	Template zum Bestimmen des Binärbilds . . . . .	30
4.5.	Helligkeit und Rotwert im echten Testbild . . . . .	31
4.6.	Helligkeit und Rotwert im Simulationsbild . . . . .	32
4.7.	Orientierung aufgrund von Beschränkung des RANSAC falsch detektiert . . . . .	34
5.1.	Test der Objekterkennung auf geraden Objekt . . . . .	42
5.2.	Test der Objekterkennung auf teilweise verdeckten Objekt . . . . .	43
5.3.	Test der Objekterkennung auf abgeknickten Objekt . . . . .	44
5.4.	Test der Objekterkennung leerem Bild . . . . .	45
5.5.	Test der Objekterkennung auf Realbildern mit schlechten Sichtverhältnissen . . . . .	47
5.6.	Test der Objekterkennung auf Realbildern mit starker Reflexion . . . . .	48
5.7.	Testlauf an einem geraden Objekt . . . . .	50
5.8.	Testlauf mit einer Kurve . . . . .	52
5.9.	Zweiter Testlauf mit einer Kurve . . . . .	53
5.10.	Testlauf mit einer Kurve nach längerer Gerade . . . . .	54
5.11.	Testlauf mit Sinuskurve . . . . .	55
5.12.	Testlauf mit einer S-Kurve . . . . .	56
5.13.	Testlauf mit einer Ellipse . . . . .	58

5.14. Testlauf mit einem Wechsel zwischen Geraden und Ellipsenabschnitten . . .	59
5.15. Testlauf mit einer Kurve bei schlechten Sichtbedingungen . . . . .	61
5.16. Testlauf mit einem Ellipsenabschnitt unter schlechten Sichtbedingungen . .	62
5.17. Testläufe am identischen Objekt mit verschiedener Parametrisierung . . . .	64
5.18. Einpendelverhalten am geraden Objekt verdeutlicht . . . . .	65

1.	Kreisgleichung zum Test, ob ein Punkt auf einem Kreis liegt. . . . .	22
2.	Transformation der Kamerakoordinaten zu Bodykoordinaten . . . . .	27
3.	Transformation der Bodykoordinaten zu Weltkoordinaten . . . . .	27
4.	Transformation von Weltkoordinaten in VRML Koordinaten . . . . .	28
5.	Templatewertberechnung für ein Pixel als Formelausdruck . . . . .	30
6.	Least-Squares-Ansatz . . . . .	36
7.	Weighted-Least-Squares-Verfahren . . . . .	36
8.	Funktion zum Überprüfen, ob die Steigung einen Maximalwert nicht übersteigt. . . . .	37
9.	Zusammensetzung der Funktion F, die minimiert wird. In (10) wird der <i>Weighted-Least-Squares</i> auf den Fehler der Position und in (11) auf den Fehler der Steigung angewendet. . . . .	37

1.	Initialisierung der <i>pointInFrame</i> -Struktur, die erkannte Punkte in verschiedenen Referenzkoordinatensystemen abbildet. . . . .	20
2.	Enumeration der Frames, die die verschiedenen Koordinatensystem bezeichnen . . . . .	25
3.	Transformation von <i>source</i> in <i>target</i> Frame. Die verschiedenen Transformation werden hier verwaltet und solange die nächste Transformation ausgeführt, bis der <i>target</i> Frame erreicht wird. . . . .	25
4.	Eingesetzter RANSAC als Pseudocode. . . . .	33
5.	Pseudocode des Schätzverfahrens . . . . .	38

## E. Abkürzungsverzeichnis

**AUV** Autonomous underwater vehicle.

**GPS** Global positioning System.

**VRML** Virtual reality modeling language.

## F. Glossar

**Bewegungsunschärfe** Unschärfe eines Bildes erzeugt durch Bewegung während der Aufnahme..

**LCF** *Lane-curve-function.* Ein parametrisierbares Modell, dass einen geraden Beginn mit einer anschließenden Krümmung darstellt..

**Pitch** *Nicken* des Fahrzeugs. Gibt eine Rotation um die *Y – Achse* (Querachse) des Bodykoordinatensystems an..

**Pose** Gibt die räumliche Lage eines Objektes an. Die Lage wird als Kombination aus Position und Orientierung angegeben..

**RANSAC** *Random sample consensus.* Iterativer Algorithmus, der durch zufälliges auswählen von Parametern ein geeignetes Modell für gegebene Daten schätzt..

**Roll** *Rollen* des Fahrzeugs. Gibt eine Rotation um die *X – Achse* (Längsachse) des Bodykoordinatensystems an..

**Transformation** Umrechnung von Koordinaten eines Punktes, in dieser Arbeit bestehend aus Translation (Verschiebung) und Rotation..

**Vanishing point** Punkt, an dem sich parallele Linien projiziert auf die Bildebene schneiden.

**Waypoint Controller** Controller zur Steuerung anhand von Wegpunkten..

**Yaw** *Heading* des Fahrzeugs. Gibt eine Rotation um die *Z – Achse* (horizontale Achse) des Bodykoordinatensystems an..

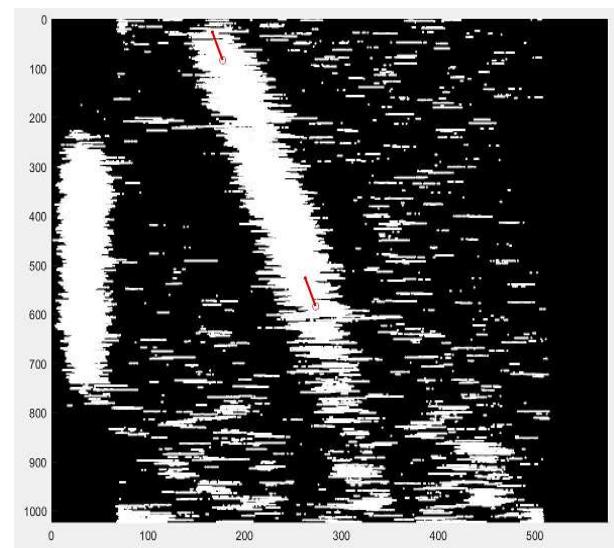
## G. Anhang

### G.1. Objekterkennung weitere Tests

Im folgenden folgt eine Übersicht Objekterkennung auf verschiedenen Testbildern aus dem Unisee. Die Objekterkennung arbeitet natürlich wie in der Arbeit beschrieben. Die Kennzeichnung Original- und im Binärbild dienen nur der verschiedenen Darstellung.



(a) Originalbild

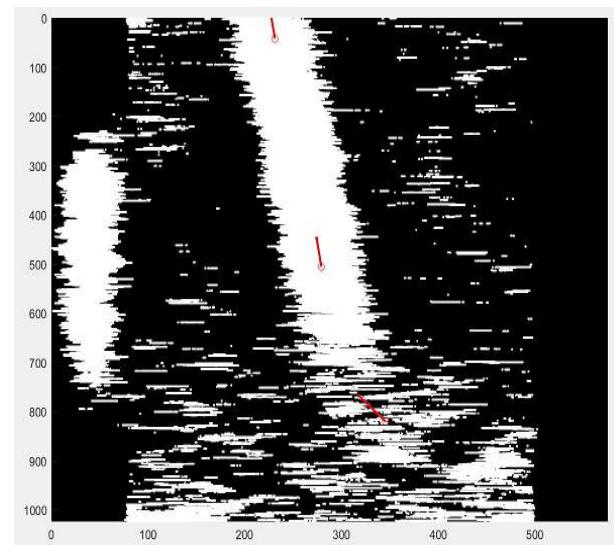


(b) Erkannte Objekte

Abbildung G.1



(a) Originalbild

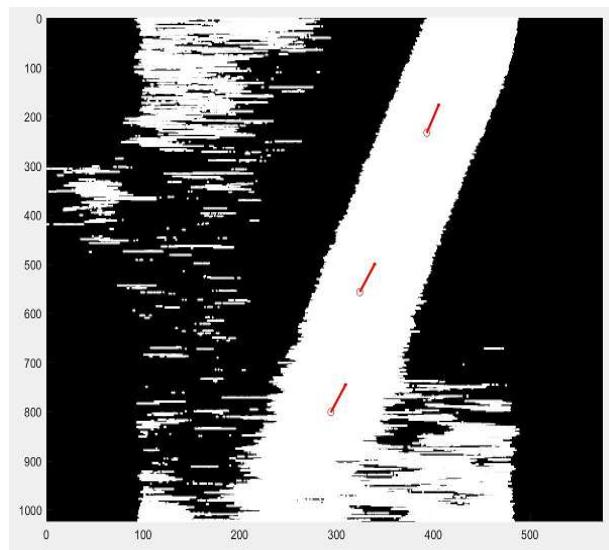


(b) Erkannte Objekte

Abbildung G.2



(a) Originalbild

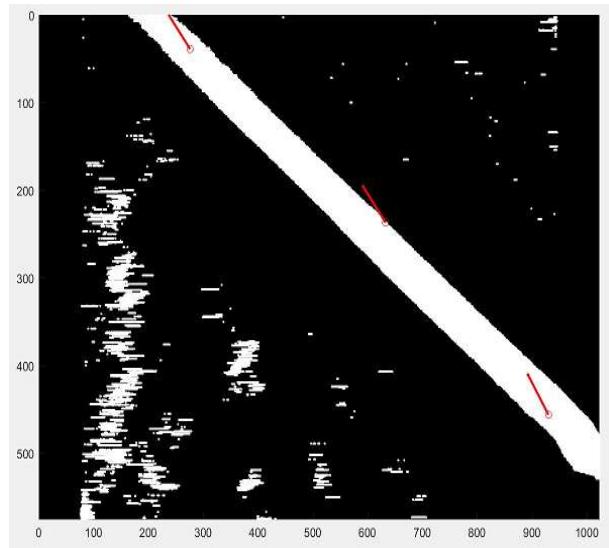


(b) Erkannte Objekte

Abbildung G.3



(a) Originalbild

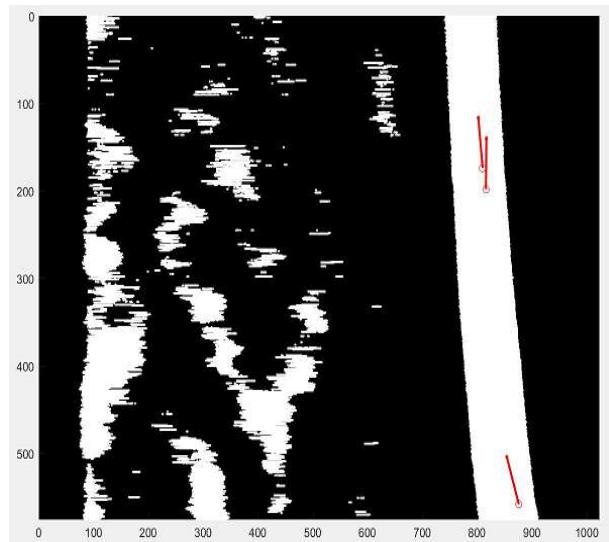


(b) Erkannte Objekte

Abbildung G.4



(a) Originalbild

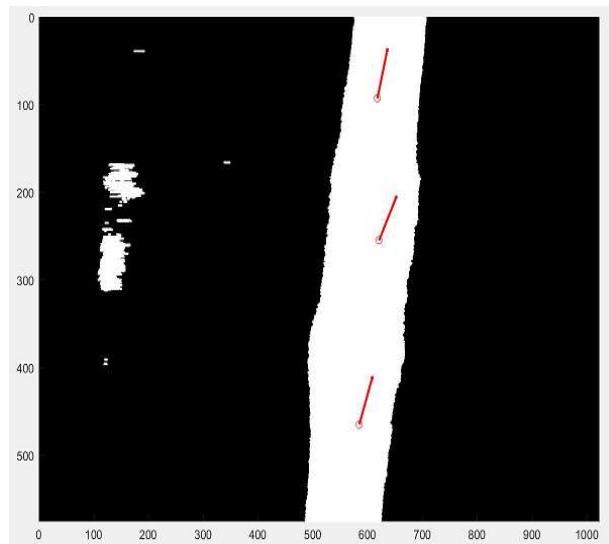


(b) Erkannte Objekte

Abbildung G.5



(a) Originalbild

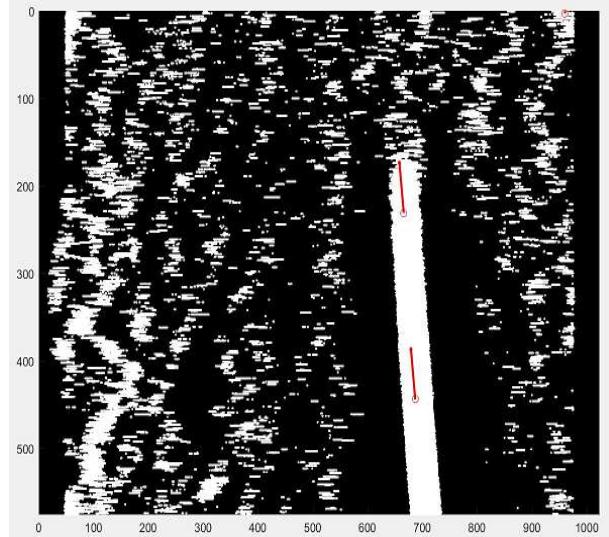


(b) Erkannte Objekte

Abbildung G.6

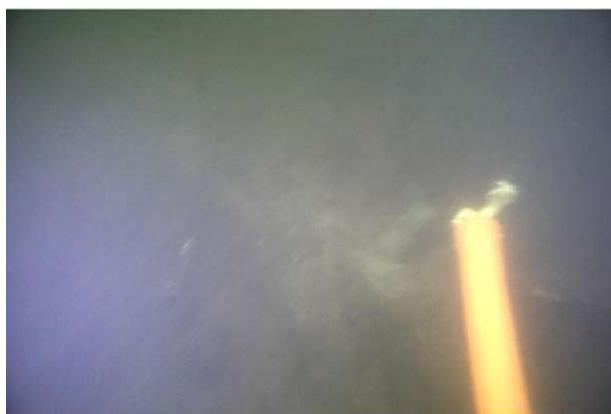


(a) Originalbild

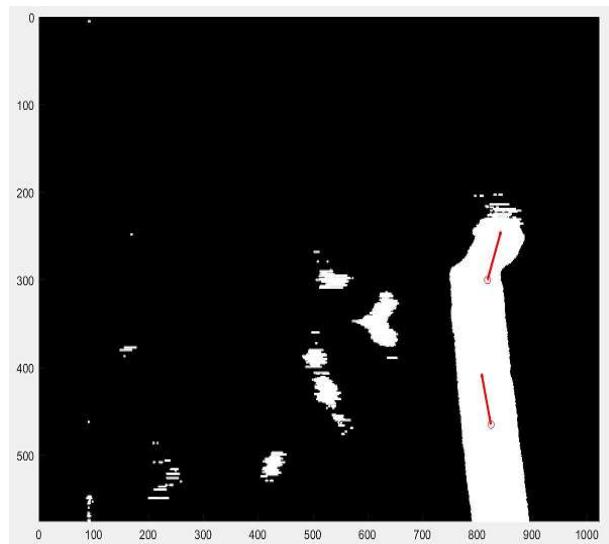


(b) Erkannte Objekte

Abbildung G.7



(a) Originalbild

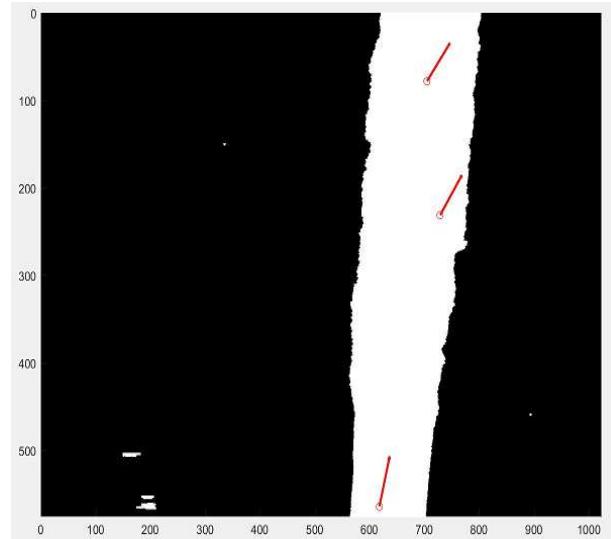


(b) Erkannte Objekte

Abbildung G.8



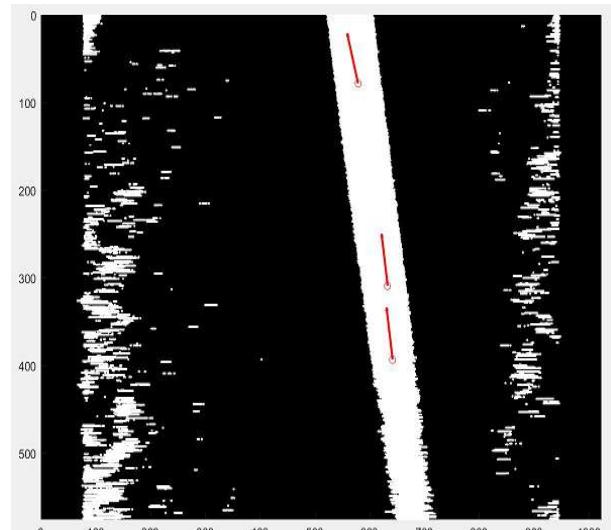
(a) Originalbild



(b) Erkannte Objekte



(c) Originalbild

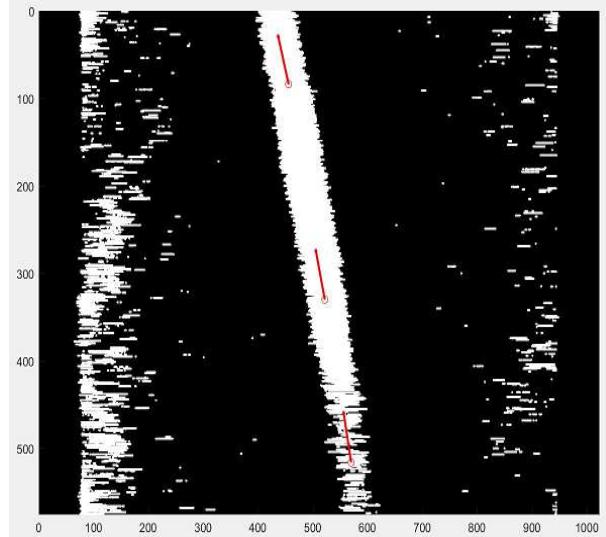


(d) Erkannte Objekte

Abbildung G.9



(a) Originalbild

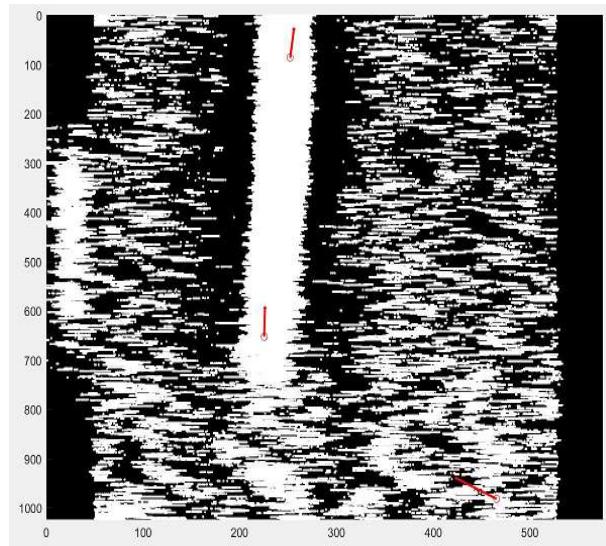


(b) Erkannte Objekte

Abbildung G.10



(a) Originalbild



(b) Erkannte Objekte

Abbildung G.11