Bachelorthesis

Entwicklung einer kamerabasierten Detektion und Verfolgung linienförmiger Objekte am Meeresboden

Prüfling:

Name: Oliver Gruhlke (ogruhlke@tzi.de)

Matrikelnummer: 278736

Erstprüfer:

Prof. Dr. Frank Kirchner (Frank.Kirchner@dfki.de)

Zweitprüfer:

Dr.-Ing. Dipl.-Inform. Thomas Röfer (Thomas.Roefer@dfki.de)

Betreuer:

Christopher Gaudig (Christopher.Gaudig@dfki.de)

Max Abildgaard (Max.Abildgaard@atlas-elektronik.com)

Inhaltsverzeichnis

1. Einleitung

Diese Bachelorarbeit behandelt die Entwicklung einer Detektion und Verfolgung von Objekten am Meeresboden. Es wird eine Komponente entwickelt, die einem AUV 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 Pilotdas Fahrzeug steuern und überwachen müsste.

wirklich Pilot?

Die Kamerakomponente kann hierbei eine Lageposition des Objektes liefern, anstatt einem Bild, das zentral verarbeitet werden müsste. Die Software für diese 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.
Zum ersten ist dies die Erkennung der Objekte im Bild. In dieser Arbeit beschränke ich mich auf Linienförmigen 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 verwertbarer Rohdaten oder durch nicht Sichtbarkeit der Objekte, wenn diese zum Beispiel von Sand überdeckt sind. In diesem Fall soll ein Schätzverfahren aufgrund der vorherigen Positionsdaten auch weiterhin die ungefähre Objektlage bestimmen.
Als Hilfe für die Algorithmen soll es möglich sein a-priori Wissen über die Objekte, wie zum Beispiel der Durchmesser des Objektes angeben zu können.

1.3. Aufbau der Arbeit

Zunächst werde ich grundlegend die Simulationsumgebung und das AUV beschreiben. Danach werde ich verschiedene Herangehensweisen ähnlicher Arbeiten vorstellen und auf ihre Eignung für das von mir zu lösende Problem eingehen. Im dritten Teil gehe ich dann auf die von mir gewählte und umgesetzte Lösung ein. Zum Schluss folgen dann Ausführungen über die durchgeführten Tests und ein Ausblick auf weitere Arbeiten.

2. Grundlagen

2.1. Simulationsumgbeung

Die Simulationsumgebung wurde von der ATLAS ELEKTRONIK GmbH entwickelt und wird für diese Arbeit zur Verfügung gestellt [Abb. 1]. In der Simulation wird die Fahrzeugsensorik und Motorik simuliert (siehe Kapitel 2.2). Außerdem wird noch eine grundlegende Wasserbewegung erzeugt, die durch Parametrisierung verändert werden kann.

umrechnun
lat
lon zu
meter

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 um zufallsgenerierte Meeresböden in *wrl* zu erstellen. Die Objekte, die in der Arbeit verfolgt werden sollen wurden in *VRML* modelliert und dann als Unterknoten in die *main.wrl* hinzugefügt.
Die Simulation basiert auf der MATLAB Simulink 3D Animation Toolbox, somit stehen auch weitere MATLAB Toolboxen zur Verfügung.

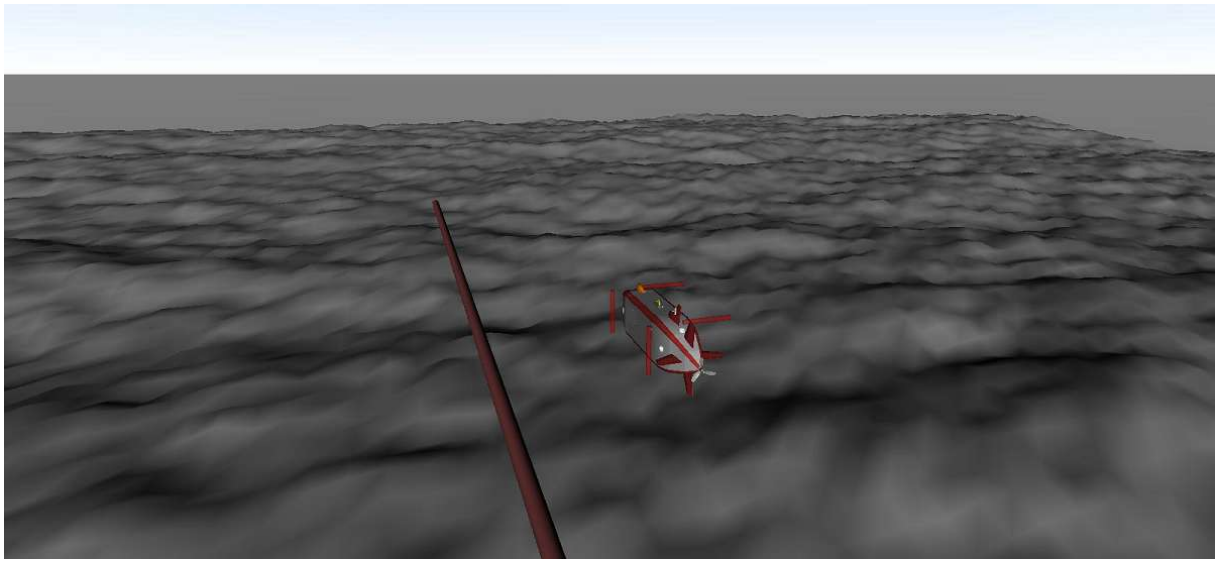


Abbildung 1: Screenshot der Simulationsumgebung

2.2. AUV-Simulation

Das AUV wurde wie bereits beschrieben 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 Fahrzeug Motorik und Sensorik in MATLAB Simulink. Es werden die für die Steuerung benötigte Schnittstelle in Form von Wegpunkten 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 bestehend aus geografischen Koordinaten, der Höhe über dem Meeresboden und den Roll, Pitch und Yaw Werten.
Für die Steuerung wird ein `lane follower controller` verwendet, bei dem eine Linie zwischen einem neuen und altem Wegpunkt gebildet und diese dann verfolgt wird. Für die Höhenkontrolle werden zwei Steuerungsmodi zur Verfügung gestellt. Zum einen die Fahrt auf Tiefe unter der Wasserobefläche oder Höhe über Meeresboden. Für diese Arbeit wird die Fahrt auf Höhe über dem Meeresboden gewählt, da die Objekterkennung und inverse Projektion am besten in einem festen Abstand zum Meeresboden funktioniert.
Am Bug des AUVs befindet sich eine Kamera, die zentral nach unten ausgerichtet ist. Das Sichtfeld beträgt dabei 45° bei einer Auflösung von 640x480 Pixeln.

2.3. Koordinatensysteme

Im folgenden werde ich die Koordinatensysteme beschreiben, in denen Koordinaten angegeben werden. Ich gehe hierbei von einer geraden Ebene aus, da ich von einem hinreichend kleinen Operationsgebiet des AUVs ausgehe, sodass die Erdkrümmung keine Auswirkung hat.

2.3.1. Bild und Kamera

Das Bildkoordinatensystem [Abb. 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].

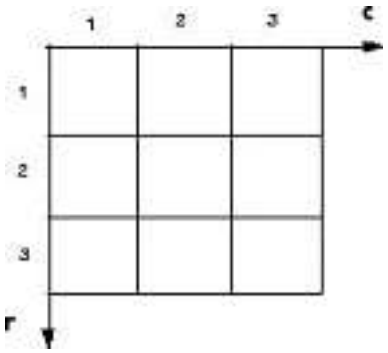


Abbildung 2: Das Bildkoordinatensystem

Das Kamerakoordinatensystem [Abb. 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.

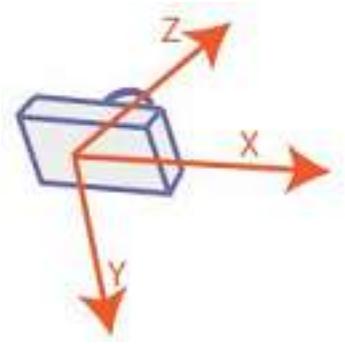


Abbildung 3: Das Kamerakoordinatensystem

2.3.2. Body

Das Body Koordinatensystem beschreibt das Koordinatensystem relativ zum AUV Mitelpunkt (Abb. 4). Der Ursprung wird hierbei durch den Schwerpunkt des AUVs bestimmt. Das Koordinatensystem entspricht dem klassischen nautischen Koordinatensystem, dem North-East-Down Koordinatensystem (vgl. Kapitel 2 in `Unmanned rotorcraft systems` [5])

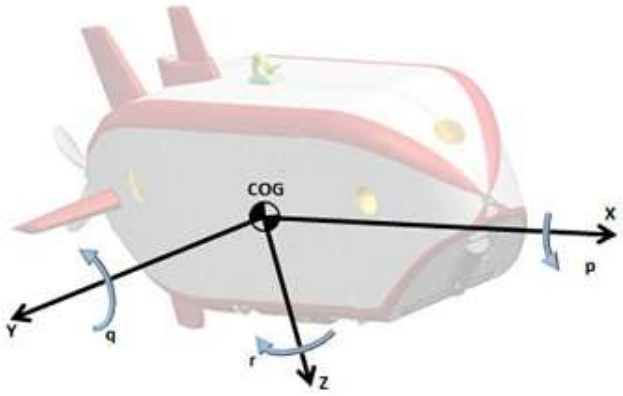


Abbildung 4: Das Body Koordinatensystem mit Mittelpunkt des AUVs

Die Neigungswinkel (Roll-Pitch-Yaw) werden wie in [Abb. 5] im Body Koordinatensystem angegeben.

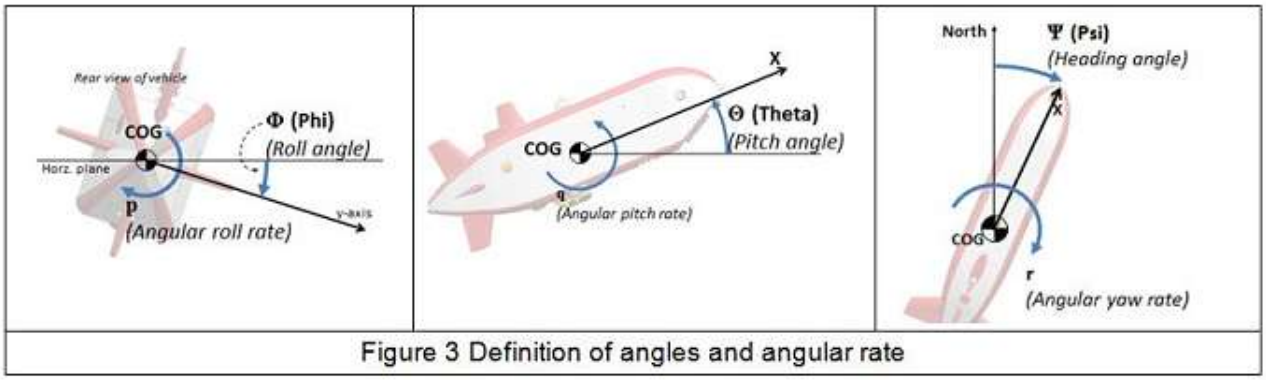


Abbildung 5: Die Neigungswinkel am AUV.

2.3.3. MATLAB und Vrml (World)

Abbildung 6 zeigt die Koordinatensysteme der MATLAB Grafik Bibliothek 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.

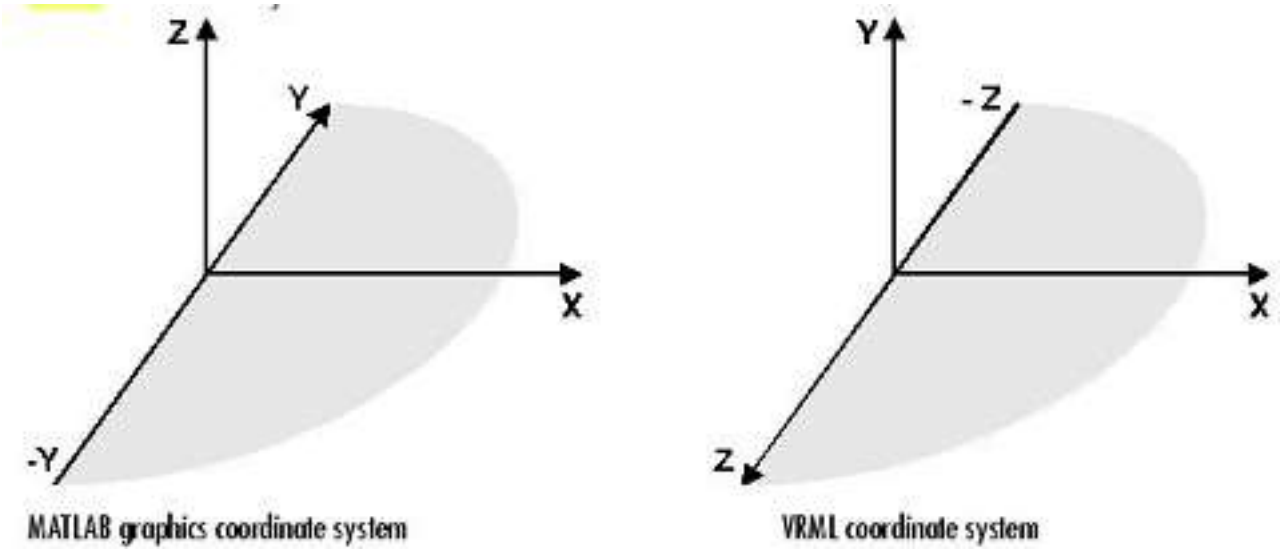


Abbildung 6: MATLAB und Vrml Koordinatensystem.

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 Bibliothe-

ken *geom2d*[1] und *geom3d*[2] durchgeführt.
Das Schätzverfahren nutzt die *Optimization* Toolbox [3], die Lösungen für verschiedene Minimierungs-, Maximierungs- und Optimierungsprobleme liefert.
Für die Kamera Kalibrierung wurde die MATLAB *Computer Vision System Toolbox* genutzt. Die Toolbox bietet die einfach zu bedienende *Camera Calibration App* mit der die intrinsischen Parameter bestimmt werden können. Hilfestellung lieferte hierbei das dazugehörige Tutorial [4]

[1] https://de.mathworks.com/matlabcentral/fileexchange/7844-geom2d
[2] https://de.mathworks.com/matlabcentral/fileexchange/24484-geom3d
[3] https://de.mathworks.com/products/optimization.html
[4] 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.
Ich unterteile meine Arbeit hierbei in die Oberpunkte *Objekterkennung*, zum Detektieren der Objekte und Bestimmung der Lage, und das *Schätzverfahren*, zur Bestimmung und Vorhersage des Objektverlaufs.

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 (7).

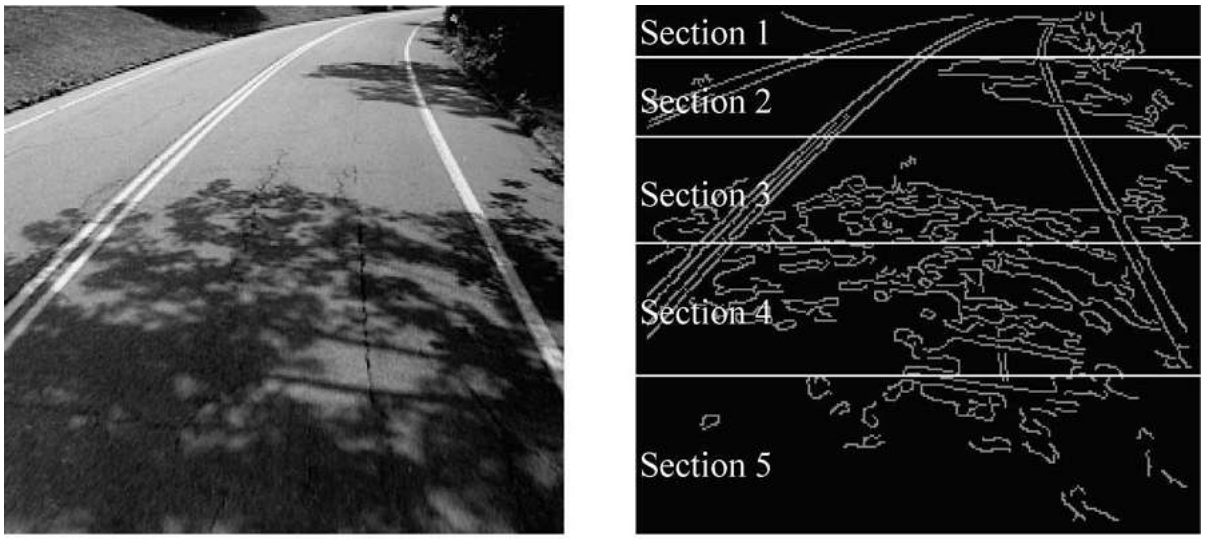


Abbildung 7: Ein typischer Straßenverlauf mit entsprechendem Kantenbild

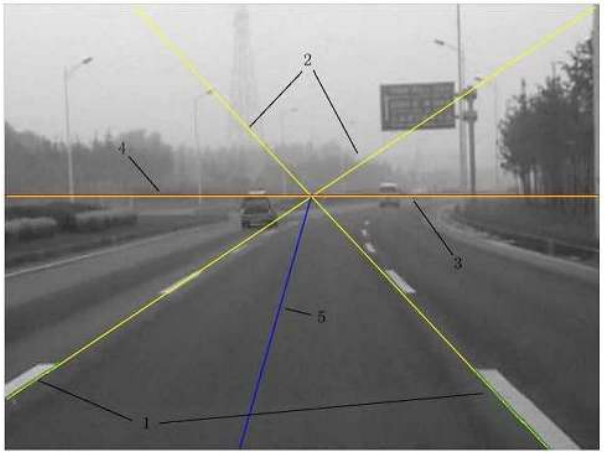
Viele Arbeiten zum Tracking von Fahrbahnmarkierungen basieren auf der Kantenextraktion ([17],[2], [12],[18]). Aus dem Kantenbild werden dann die Charakteristiken der Straße

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.
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. 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 geschaut, welcher Krümmungsgrad am ehesten die Fahrbahnverlauf abbildet. Dieses Verfahren ist in Abbildung 8 zu sehen, in dem die ROI der Linkskurve das beste Ergebnis liefert.

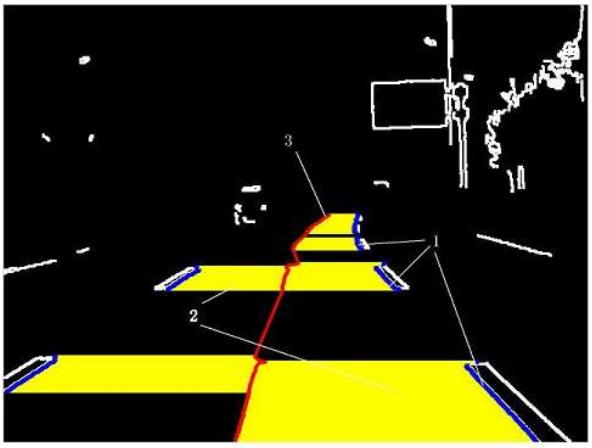
glossar?

Abbildung 8: Detektion des Straßenverlaufs mithilfe von LCF und ROI

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 dann die Mitte der Straße und in weiteren Schritten dann auch der Straßenverlauf bestimmt [9].



(a) Ursprungsbild mit Linien- und Horizontmarkierungen



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

Abbildung 9: Erkennungsprozess aus *A Real-time Lane Detection Algorithm Based on a Hyperbola-Pair Model*

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 horizontal in fünf Segmente eingeteilt wird (siehe 7). 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 *vp* bestimmt. Der *vanishing point* eines Segmentes bestimmt die Richtung der detektierten Straße (siehe Abbildung 10).
Die Arbeit bietet eine Möglichkeit innerhalb eines Bildes eine Kurve zu detektieren, obwohl nur gerade Linien gesucht wurden.

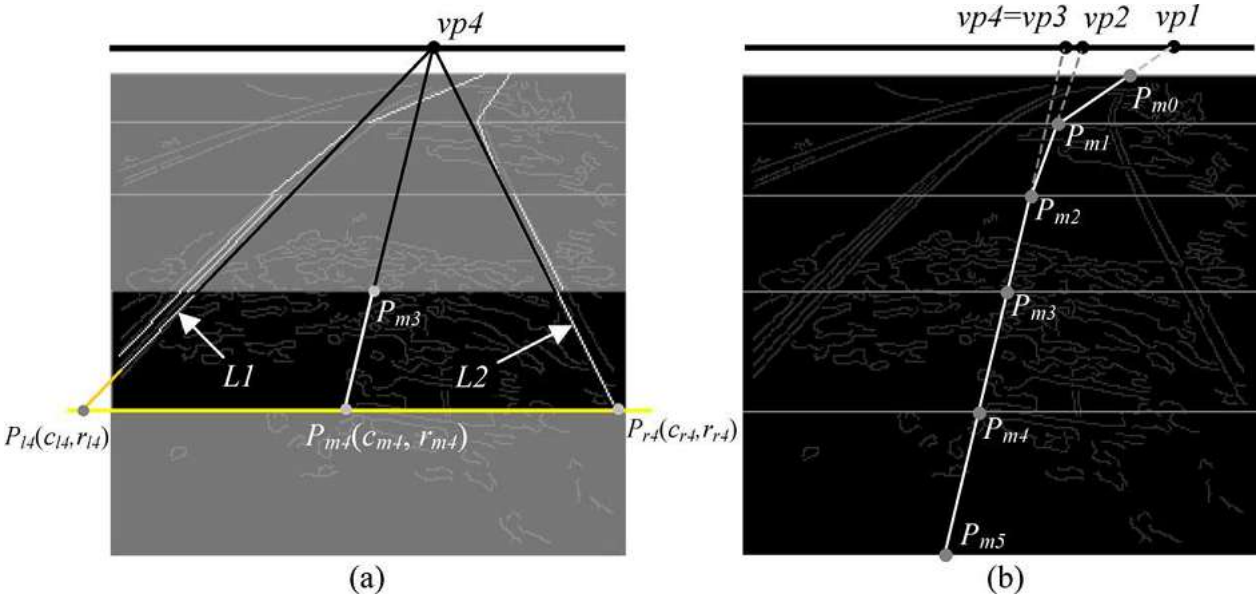


Abbildung 10: Ergebnis der Straßenverlaufsdetektion mithilfe von Hough Transformation

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 zum 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. Hier reicht eine Linienerkennung aus.

3.1.2. Andere Ansätze

Im `CSurvey`-Projekt [1] beleuchten Albiez 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 [11].
Diese Detektion wurde vor allem in verschiedenen Entfernungen zum Boden und verschiedenen Trübheitsgraden 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.

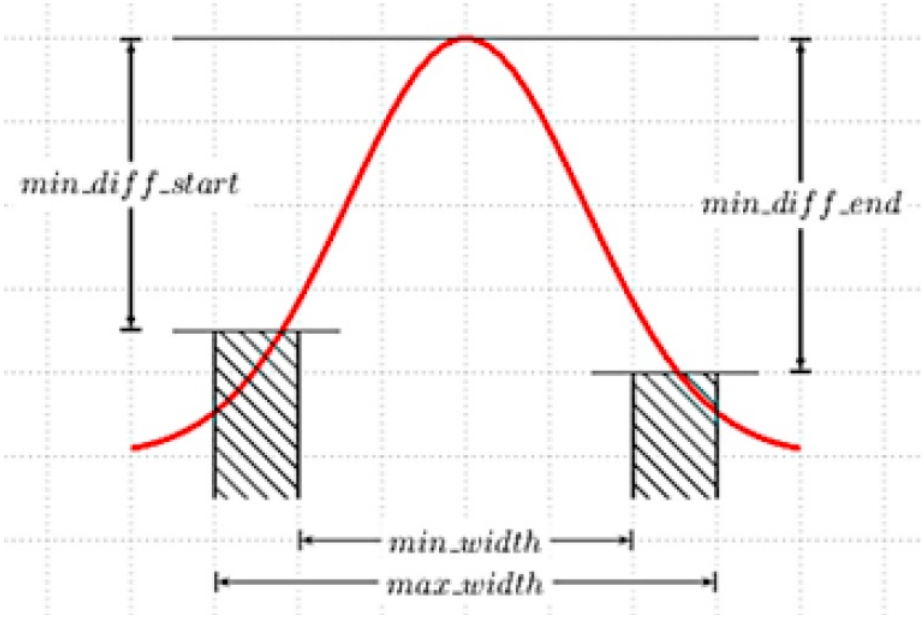
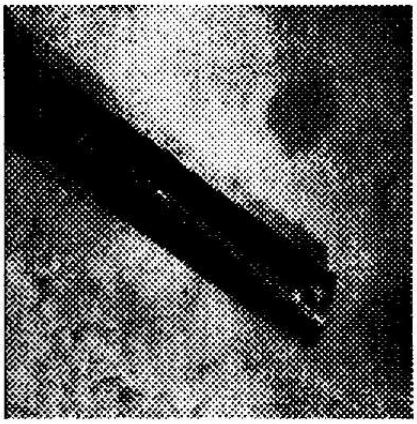
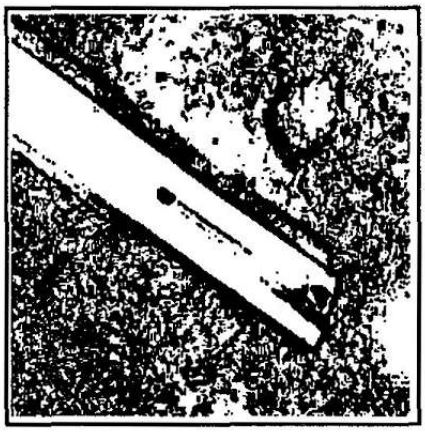


Abbildung 11: Template zur Helligkeitsdetektion aus *CSurvey*

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 mithilfe von Segmentierung Regionen definiert. Auf diese Regionen werden dann umschließende Rechtecke gelegt. Aus diesen Rechtecken lässt sich dann die gesuchte Pipeline bestimmen.

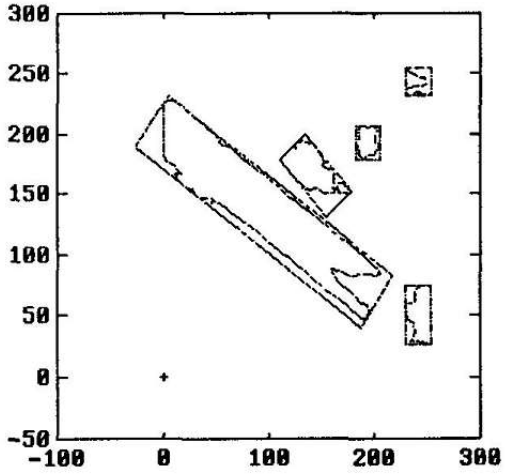


(a) Originalbild nach Kontrastverstärkung



(b) Kantenbild durch Sobel-Filter

(c) Segmentiertes Bild



(d) Erkannte Rechtecke

Abbildung 12: Einzelschritte der Erkennung aus *Simple vision tracking of pipelines for an autonomous underwater vehicle*

Foresti et al. detektieren in *A Vision Based System for Object Detection in Underwater Images* Unterwasserpipelines mithilfe eines neuronalen Netz. In Abbildung 13 ist zu sehen, dass die Pipelines sehr gut erkannt wurden. Selbst bei vom Sand verdeckte Pipeline (c und d) werden die Kanten der Pipelines noch richtig bestimmt wurden.

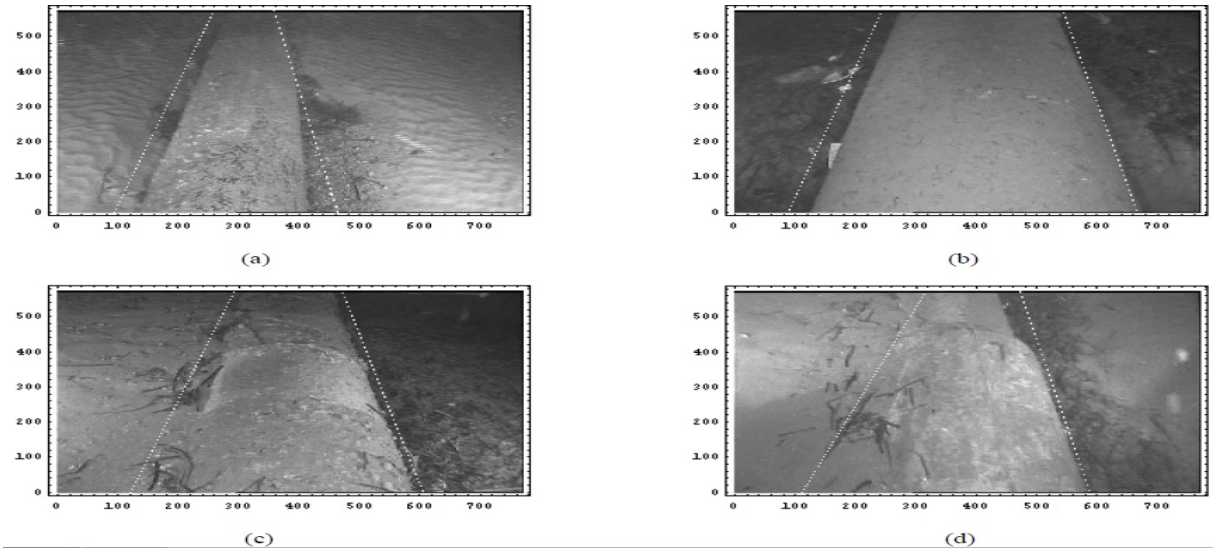


Abbildung 13: Pipelines erkannt mithilfe eines neuronalen Netzwerks

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 Vorratssetzungen 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 verwendet 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 eine Möglichkeit um verrauschte Messwerte zu verbessern und auch ausbleibende Messungen auszugleichen. Der Kalman Filter basiert auf einem linearen State 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 14). Der State des Kalman Filters ist dabei die x- und y-Position des Objektes in der Ebene, sowie dessen aktuelle Änderung der Position. Der Zustandsübergang wird durch die aktuelle Positionsänderung des Objektes 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.

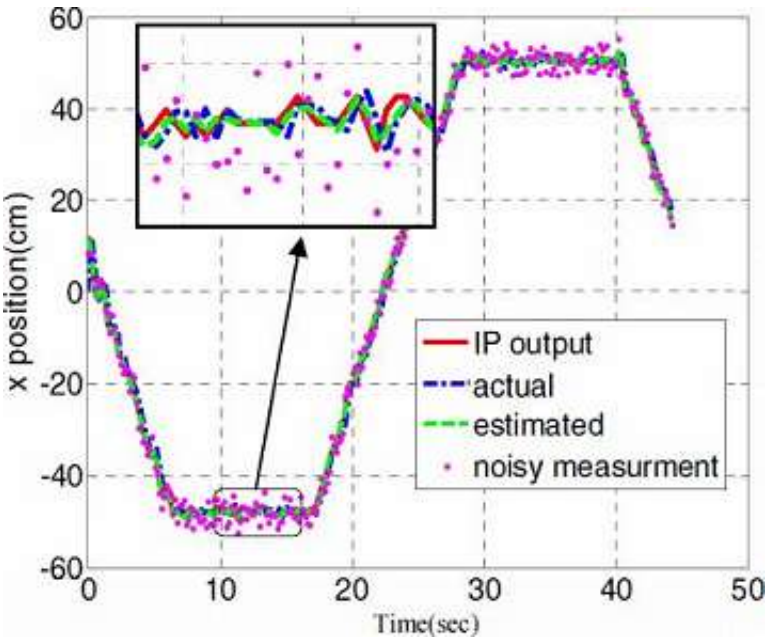


Abbildung 14: Eine Objektbewegung wird mit einem Kalman-Filter verfolgt

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

Das Problem beim definieren des States liegt darin, dass das Objekt selbst keine wechselnden Zustände hat, sondern stets fest bleibt. Ein Ansatz wäre es die Position und Ausrichtung des Objektes im State 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.

3.2.2. Regressionsverfahren

Ein weiterer Lösungsansatz für das Schätzverfahren wäre die Regression. Bei der Regression wird versucht ein parametrisierbares Modell an gegebene Daten anzupassen. Dabei wird versucht der Fehler der Daten im Bezug zur aus dem Model generierten Kurve zu 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 erwarteten 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 15 ist gut zu sehen, wie die Kurve nahezu den echten Pfad abbildet.

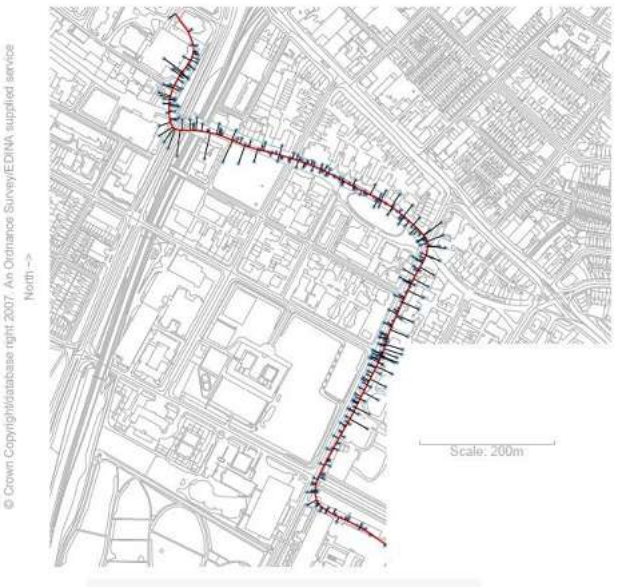
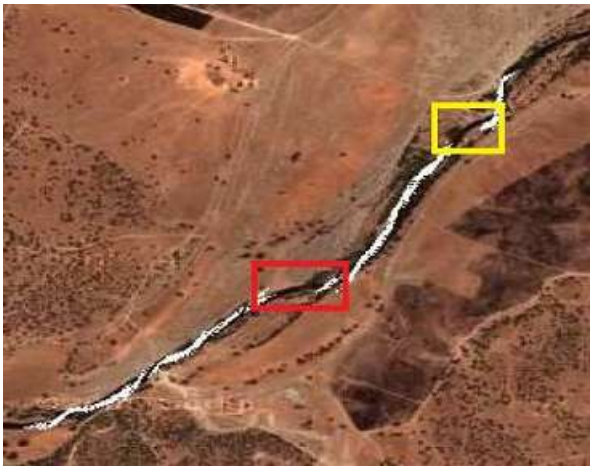


Abbildung 15: Curve Fitting durch GPS Daten

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 war es mithilfe eines UAVs

(Unmaned 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 mithilfe von Regression eine Kurve gelegt (siehe Abbildung **??**).
Im gelb gekennzeichneten Bereich im rechten Bild 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 im ähnlichen Rahmen auch im Unterwasserszenario zu erwarten.
Bemerkenswert ist, dass im roten Bereich ebenfalls keine Ergebnisse der Objekterkennung vorhanden sind, die Kurve jedoch trotzdem annähernd genau dem Flussverlauf folgt. Dies lässt sich darauf zurückführen, 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

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

4. Lösungsansatz

In diesem Kapitel wird der von mir implementierte Lösungsansatz vorgestellt. Grundlegend gehe ich zuerst auf die Erweiterung der Simulationsumgebung und die implementierte Transformationskette ein.
Danach folgen genauere Ausführungen zu den Kernelementen der Arbeit, der Objekterkennung und dem Schätzverfahren.

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

ref auf ausblick

```
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: *pointInFrame* Struktur Initialisierung

4.1. Simulationserweiterung

4.1.1. Steuerung

Wie im Kapitel 2.2 Grundlagen beschrieben wird in der bestehenden Simulation ein `lane follower controller`, der eine Linie zwischen einem *old_ waypoint* und einem *new_ waypoint* bildet verwendet. Die Schnittstelle zur Steuerung bildet also 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 transformiert. Es wird der nächste Punkt auf dem Polynom zur transformierten Position des AUVs 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* genommen 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 gekrümmt, 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 16 grafisch dargestellt.

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

Gleichung 1: Kreisgleichung zum Test ob ein Punkt X_{test}, Y_{test} auf einem Kreis liegt

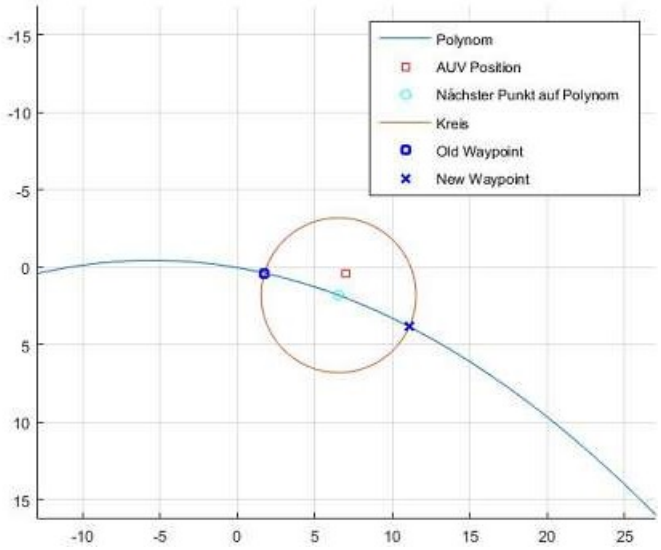
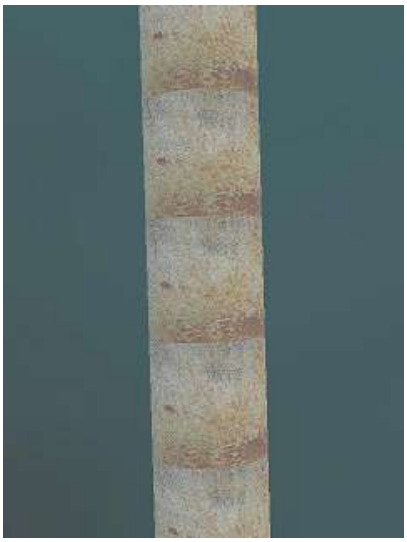


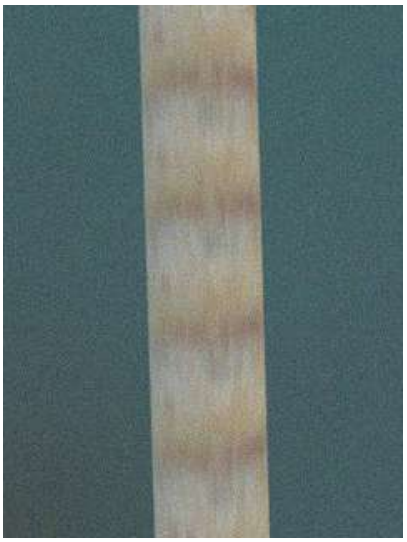
Abbildung 16: Bestimmung der Wegpunkte

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 17 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.

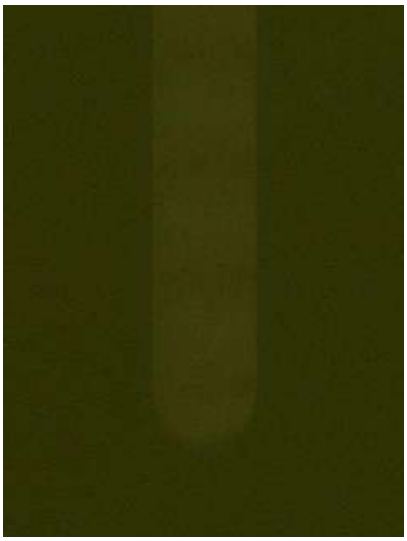


(a) Ursprüngliches Bild

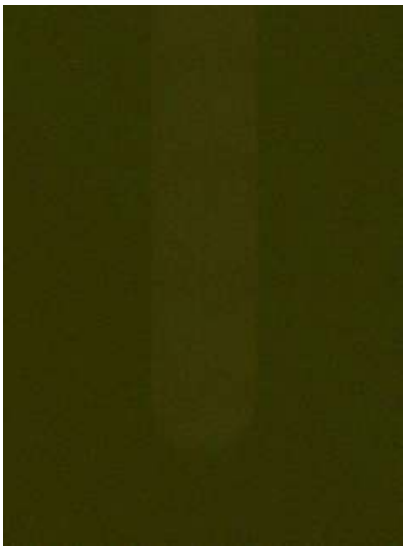


(b) Bild verschlechtert mit leichtem *blur* und geringem Pixelrauschen

(c) Bild verschlechtert mit starkem *blur* und starkem Pixelrauschen



(d) Sichtverhältnisse verschlechtert



(e) Sichtverhältnisse stark verschlechtert

(f) Sichtverhältnisse sehr stark verschlechtert und simulierte Reflexion des Wassers

Abbildung 17: Simulationsbilder

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.

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

end
```

Listing 2: Enumeration der Frames

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.

```
function transformed =transform(toTransform,targetFrame,
   height,PosEast_m,PosNorth_m,psi,phi,theta,cameraParameters
   )
%TRANSFORM Summary of this function goes here
%   Detailed explanation goes here
    while(toTransform.frame~=targetFrame)
        switch toTransform.frame
            case frames.image
                toTransform = pic2cam(toTransform,height,phi,
                   theta,cameraParameters);
            case frames.cam
                toTransform = cam2body(toTransform);
            case frames.body
                toTransform = body2world(toTransform,psi,
                   PosEast_m,PosNorth_m,height);
            case frames.world
                toTransform = world2vrml(toTransform);
```

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

Listing 3: Transformation von *source* in *target* Frame

4.2.1. Bild zu Kamera

Die verlustfreie Transformation von 2D-Pixelkoordinaten in 3D-Kamerakoordinaten ist mit einer Kamera nicht möglich. Jedoch lässt sich über das Wissen über die Entfernung zur Bildebene und die intrinsischen Kameraparameter 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. Die Kamerakalibrierung wurde 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 Kamera Kalibrierung wird ein *CameraParameter*[5] 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 AUV bestimmt wird [Kapitel 2.3.1].
Aufgrund der Position der Kamera zum Bodykoordinatenursprung (Schwerpunkt des AUVs) ergibt sich eine Translation um 1.3 in X Richtung und 0.25 in Z Richtung.

[5] https://de.mathworks.com/help/vision/ref/cameraparameters-class.html

Die Rotation beträgt dabei 90° um die Z-Achse.
Somit ergibt sich die Tranformationsmatrix Gleichung 2

$$\begin{pmatrix} x_{body} \\ y_{body} \\ z_{body} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 & 1.3 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.25 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{pmatrix} \tag{2}$$

Gleichung 2: Transformation der Kamerakoordinaten zu Bodykoordinaten

4.2.3. Body zu Welt

Die Transformation vom Bodykoordinatensystem in das Weltkoordinatensystem ist wieder eine Translation und eine Rotation nötig. Aus der Definition der Koordinatensysteme ist 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 [Abbildung 5]) bestimmt. Somit ergibt sich die Tranformationsmatrix 3

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

Gleichung 3: Transformation der Bodykoordinaten zu Weltkoordinaten

4.2.4. Welt zu VRML

Für die Transformation von Weltkoordinaten in VRML Koordinaten ist nur eine Rotation um −90° um die X-Achse nötig [Abbildung 4].

$$\begin{pmatrix} x_{vrml} \\ y_{vrml} \\ z_{vrml} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{body} \\ y_{body} \\ z_{body} \\ 1 \end{pmatrix} \tag{4}$$

Gleichung 4: Transformation von Weltkoordinaten in VRML Koordinaten

4.3. Objekterkennung

4.3.1. Binärbild mit Template

Die Objekterkennung basiert auf einem ähnlichen Verfahren wie das vorgestellte CSurvey Projekt[1].
Da ein Farberkennung aufgrund der Sichtbedingungen nicht in Frage kommt wird im ersten Schritt das RGB-Bild in ein Graustufenbild umgewandelt. Die erste Idee war es hier das Helligkeitsbild zu betrachten, da ein gesuchtes Objekt einen höheren Helligkeitswert besitzt, als der Meeresboden (siehe Abbildung 19b und 20b).
Aus Erfahrungswerten früherer Projekte riet Christopher Gaudig mir, die Rotwerte der Bilder zu betrachten, da oftmals der Meeresboden und trübes Wasser geringe Rotwerte haben. In den Abbildungen 19c und 20c 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 (Abbildung 18) 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. Jedes Pixel dient einmal als Mittelpunkt des Testbereichs, um zu entscheiden, ob das betrachte Pixel Teil des Objektes sein kann. Dies ist der Fall, wenn der Wert des Pixels [Gleichung 5] einen Schwellwert (rote Linie) übersteigt.

Kann man die graphen mit legends gut genug erkennen?

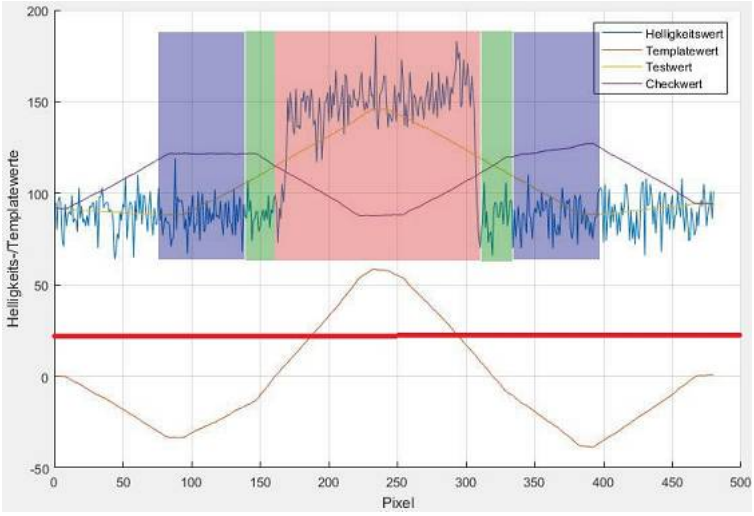
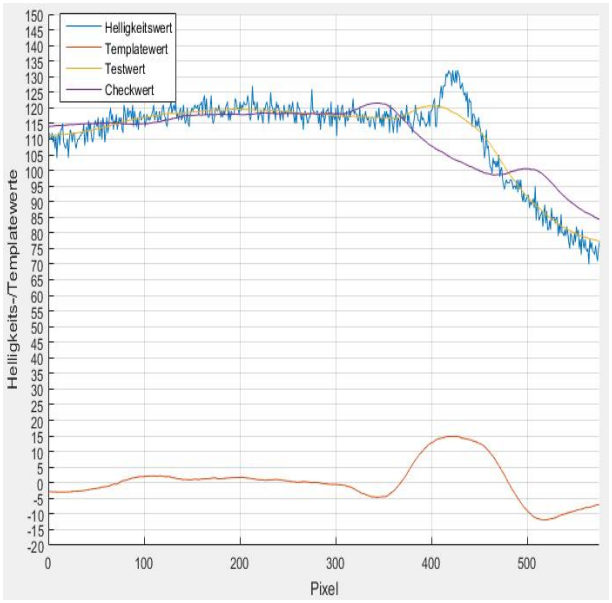


Abbildung 18: Template zum Bestimmen des Binärbilds

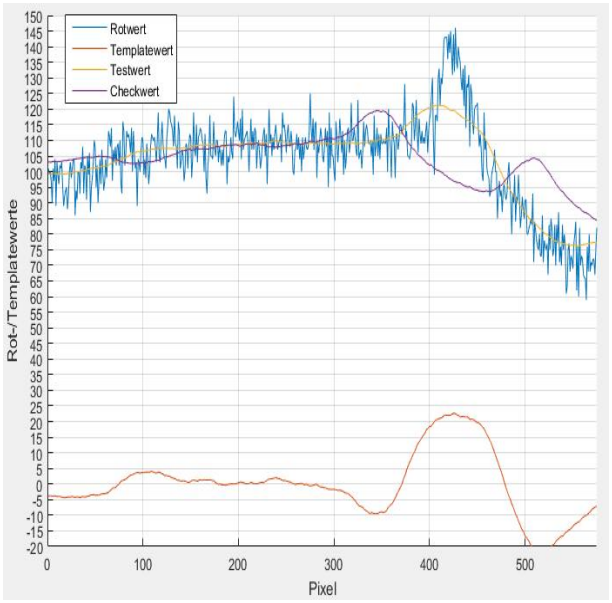
$$TV = \frac{sum(Testpixel)}{\#TP} - \frac{sum(Checkpixel)}{2 \cdot \#CP} \tag{5}$$

Gleichung 5: Templatewertberechnung für ein Pixel

(a) Originalbild

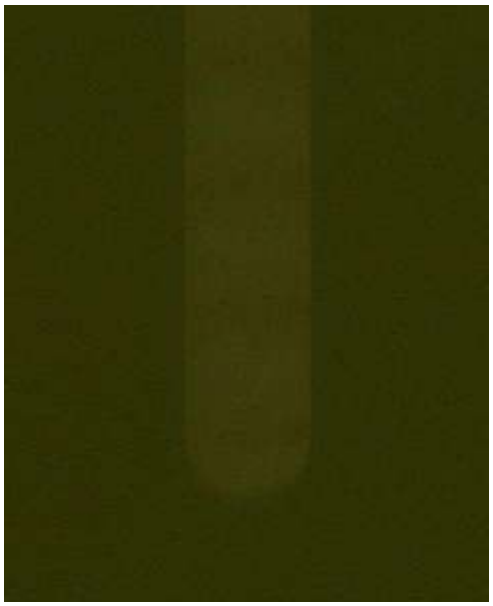


(b) Helligkeitsverlauf einer Bildzeile im oberen Drittel des Bildes

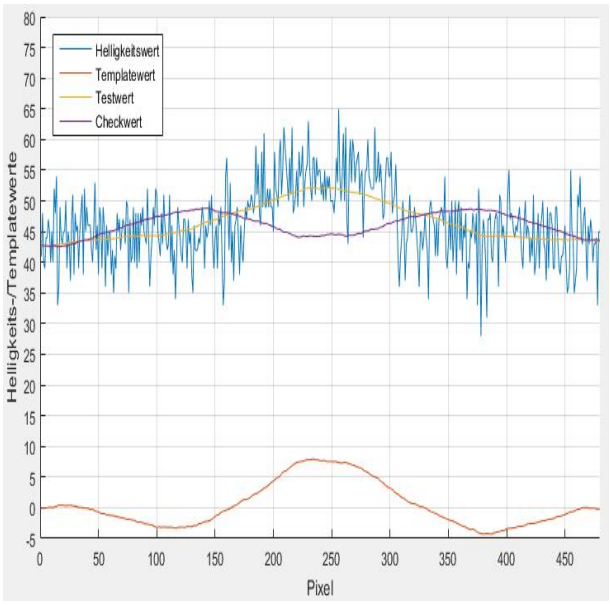


(c) Rotwertverlauf einer Bildzeile im oberen Drittel des Bildes

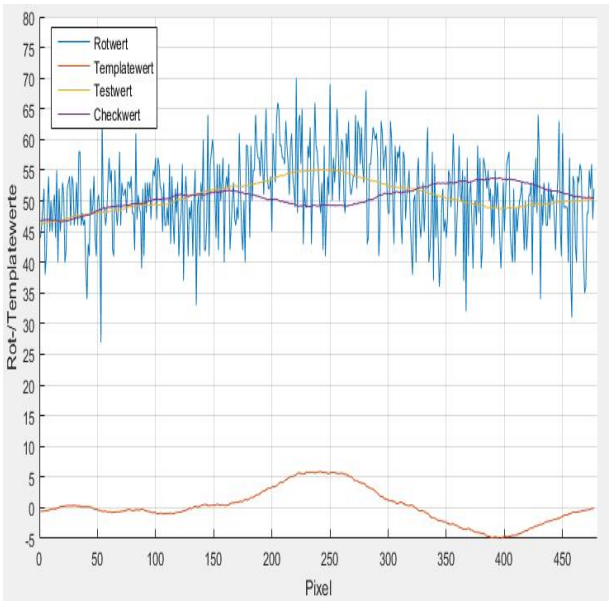
Abbildung 19: Helligkeit und Rotwert im echten Testbild



(a) Originalbild der Simulation



(b) Helligkeitsverlauf einer Bildzeile im oberen Drittel des Bildes



(c) Rotwertverlauf einer Bildzeile im oberen Drittel des Bildes

Abbildung 20: Helligkeit und Rotwert im Simulationsbild

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. Dieses Rechteck wird durch einen Mittelpunkt, eine Orientierung, der Breite und der 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.

```
function ransac(segment,height,width,minInlier,iterNum)
        maxInlier = 0;
        orientations = -pi/4:0.05:pi/4;
        bestCenter = None;
        bestOrientation = None;
        for i = 1:iterNum
                boxCenter = selectRandomPoint(segment);
                boxOrientation = selectRandomValue(
                   orientations);
                inliers = findPointsInBox(segment, box=[
                   boxCenter,boxOrientation,height,width]);

                if(len(inliers) > minInlier && len(inliers) >
                    maxInlier)
                        maxInlier = len(inliers)
                        bestCenter = boxCenter;
                        bestOrientation = boxOrientation;
                end
        end
end
```

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 mehr maximale Iterationen dem RANSAC erlaubt wurden. Da der RANSAC jedoch auf jedes der drei Segmente separat angewendet wird, wird bei steigender Iterationsanzahl die Geschwindigkeit deutlich reduziert. Für eine zuverlässige Erkennung waren zu viele Iterationen nötig, sodass das Verfahren nicht einsetzbar wäre.

richtige wortwahl?

Als Lösung für dieses Probleme habe ich 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 21 gezeigt.

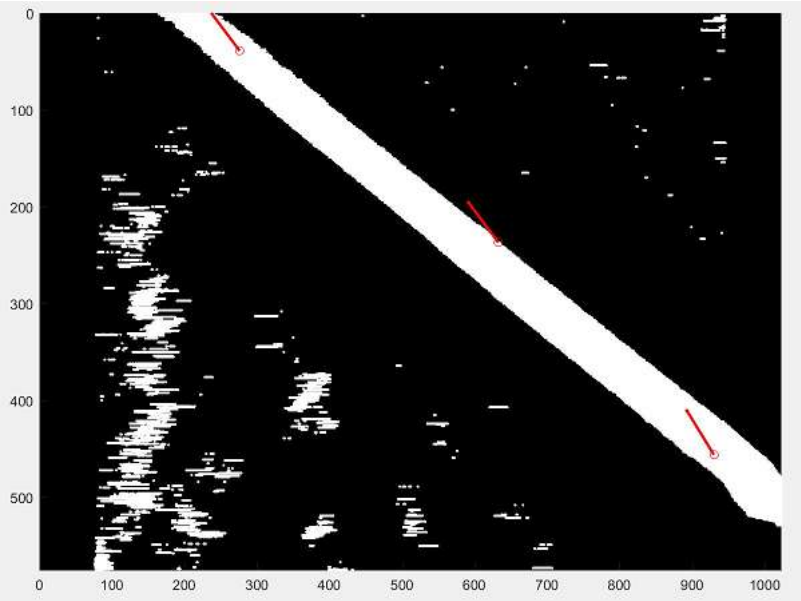


Abbildung 21: Falsche Erkennung aufgrund der Beschränkung der Ausrichtungen für den RANSAC

Der zweite Faktor, der die Geschwindigkeit der Objekterkennung verringerte war 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.
Zum Lösen dieses Problems wurden vorm Einsatz des RANSAC die Punktanzahl verringert, indem nur jedes dritte Pixel betrachtet wird und diese den Wert aller 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.

Grafik?

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 in Betrachtung ihrer Orientierung zu fitten. Das Verfahren 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 summiert der quadratische Fehler vom Funktionswert zu gegebenen Parametern zum y aus der Bilderkennung berechnet. $f(p, x)$ ist eine beliebige Funktion, die x in Abhängigkeit von p auf eine reelle Zahl abbildet.

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

Gleichung 6: Least Squares Ansatz

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 abnehmend gewichtet. Somit erhalten aktuelle Punkte einen höheren Stellenwert als ältere, ohne jedoch alte Punkte komplett zu verwerfen.
Um diese Anforderungen umzusetzen habe ich eine Erweiterung des *Least-Squares* genutzt, den *Weighted-Least-Squares*[Gleichung 7]

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

Gleichung 7: Weighted Least Squares Verfahren

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* [6] aus der Optimization Toolbox eine geeignete Lösung. Die Funktion bietet die 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.*).

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

Gleichung 8: Funktion zum überprüfen, ob die Steigung einen Maximalwert nicht übersteigt.

Die Funktion $F(p)$ wird als $F(p, x, y, s, w, n, m, tau)$ [Gleichung 9] definiert, wobei x und y wieder die Punkte der Bilderkennung darstellen, s die erkannte Orientierung im Punkt und w das Gewicht. 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.

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

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

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

Gleichung 9: Zusammensetzung der Funktion F, die minimiert wird.

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

[6] https://de.mathworks.com/help/optim/ug/fmincon.html

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 führe ich ein alternatives Weltkoordinatensystem. 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 Fehlerschwellwert überschritten 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 beschrieben Transformation gespeichert wurde werden alle Punkte vor der Regression in das alternative Koordinatensystem transformiert. Durch diese Transformation sind die erkannten Punkte stets 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.

hier noch graphen zur transformation

```
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(
                      points_T);
10                 newTransform = createTransMatrix(-translation
                      ,-rotation) * actualTransform;
11                 saveNewTransform();
12         end
13 end
```

Listing 5: Pseudocode des Schätzverfahrens

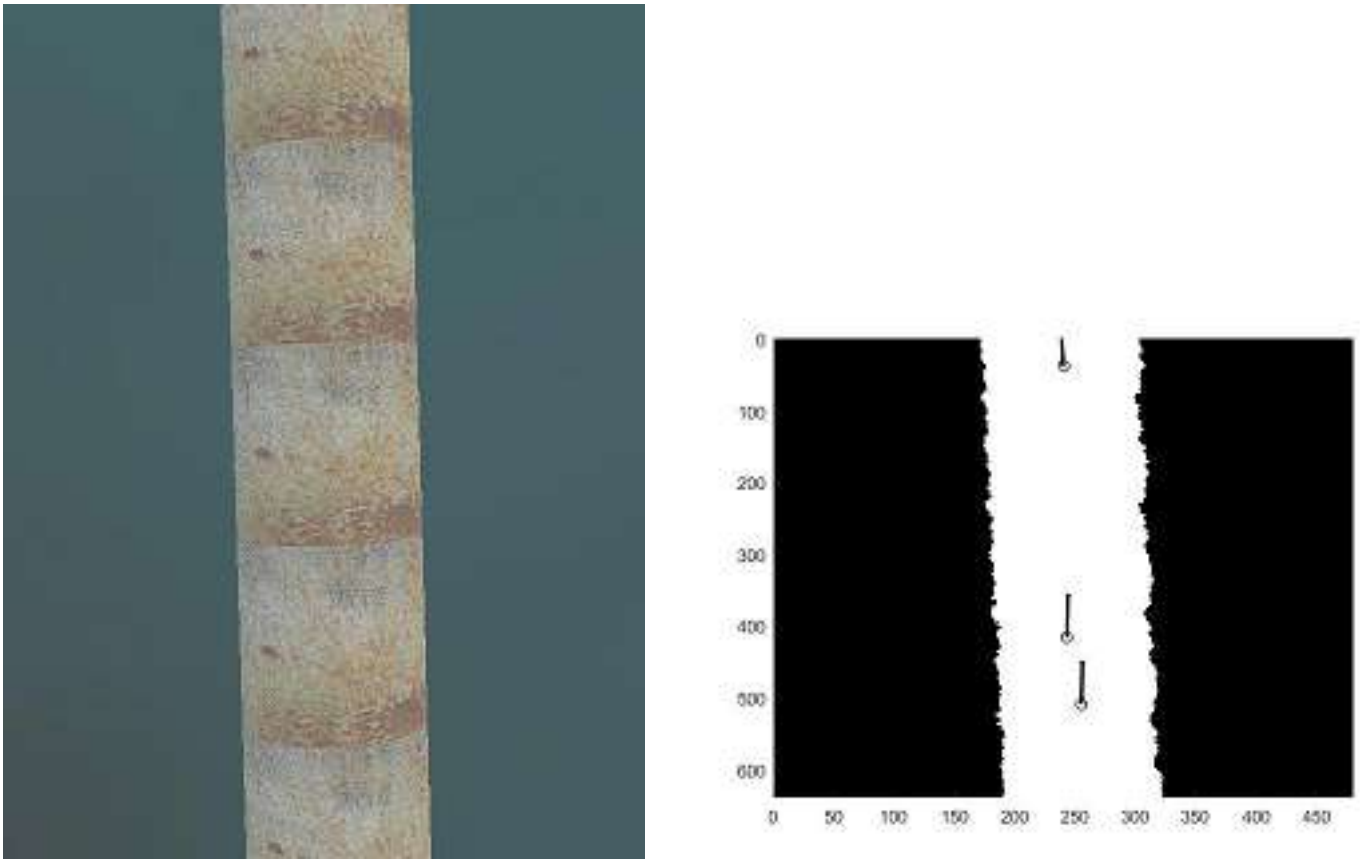
5. Tests und Evaluation

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

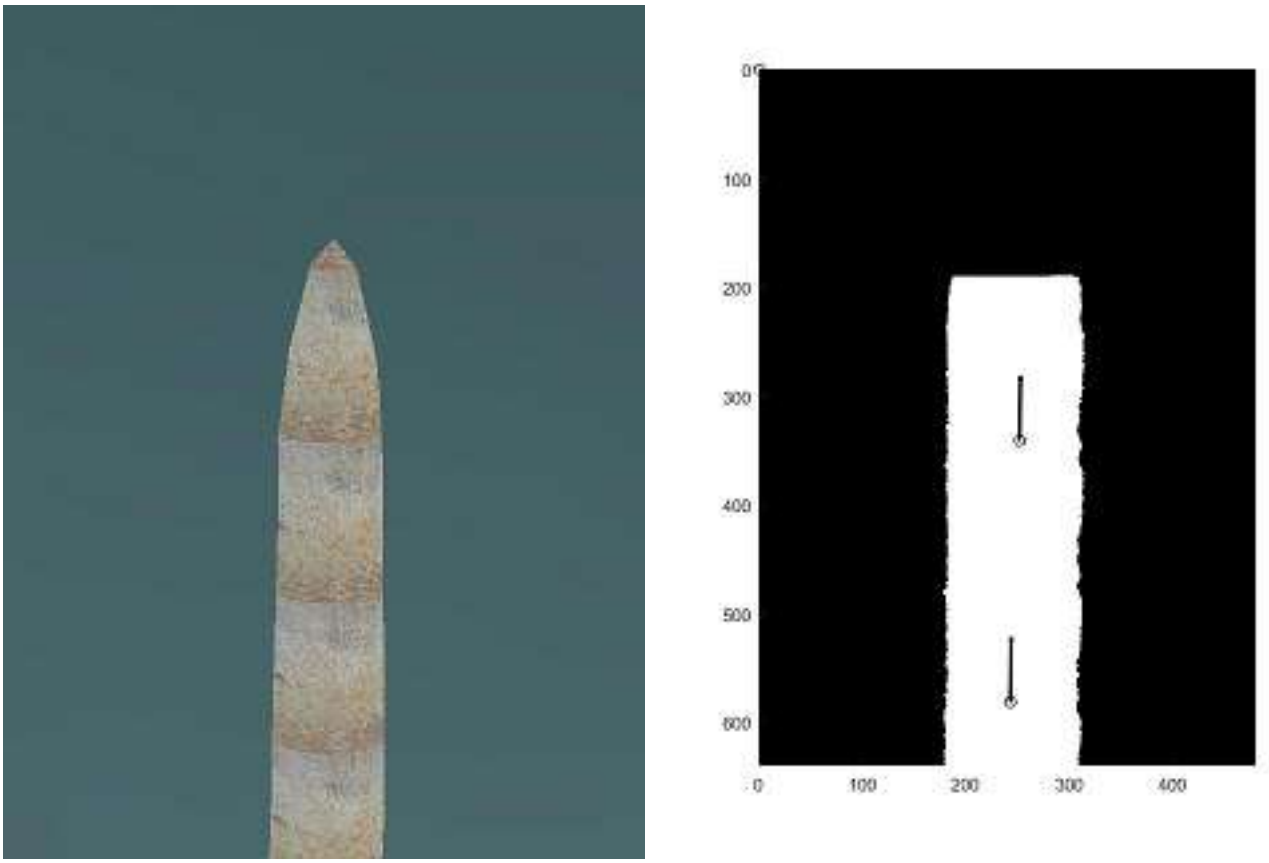
Die Objekterkennung wurde auf drei Arten getestet. Für diese Arbeit ist eine verlässliche Erkennung auf Simulationsbildern wichtig. Aus diesem Grund beziehen sich die ersten Tests auf Simulationsbildern. Der dritte Test dient der Evaluation der genutzten Methode in Bezug auf Echtdaten.
Die erste Testreihe besteht aus Bildern, auf denen das Objekt gut zu sehen ist und nur leicht verdeckt ist. Außerdem werden die Bilder so gewählt, dass teilweise in einzelnen Segmenten kein Objekt zu sehen ist und dass das Objekt über in verschiedenen Segmenten unterschiedlich Orientiert sind.
In der zweiten Testreihe werden die gleichen Bilder künstlich verschlechtert, um Bewegungsunschärfe und Rauschen der Kamera zu simulieren und die Grenzen der Objekterkennung im Bezug zur Bildqualität ermittelt. Für beide Testreihen wurden auch Vergleichsbilder ohne Objekt genommen.

Erste Testreihe



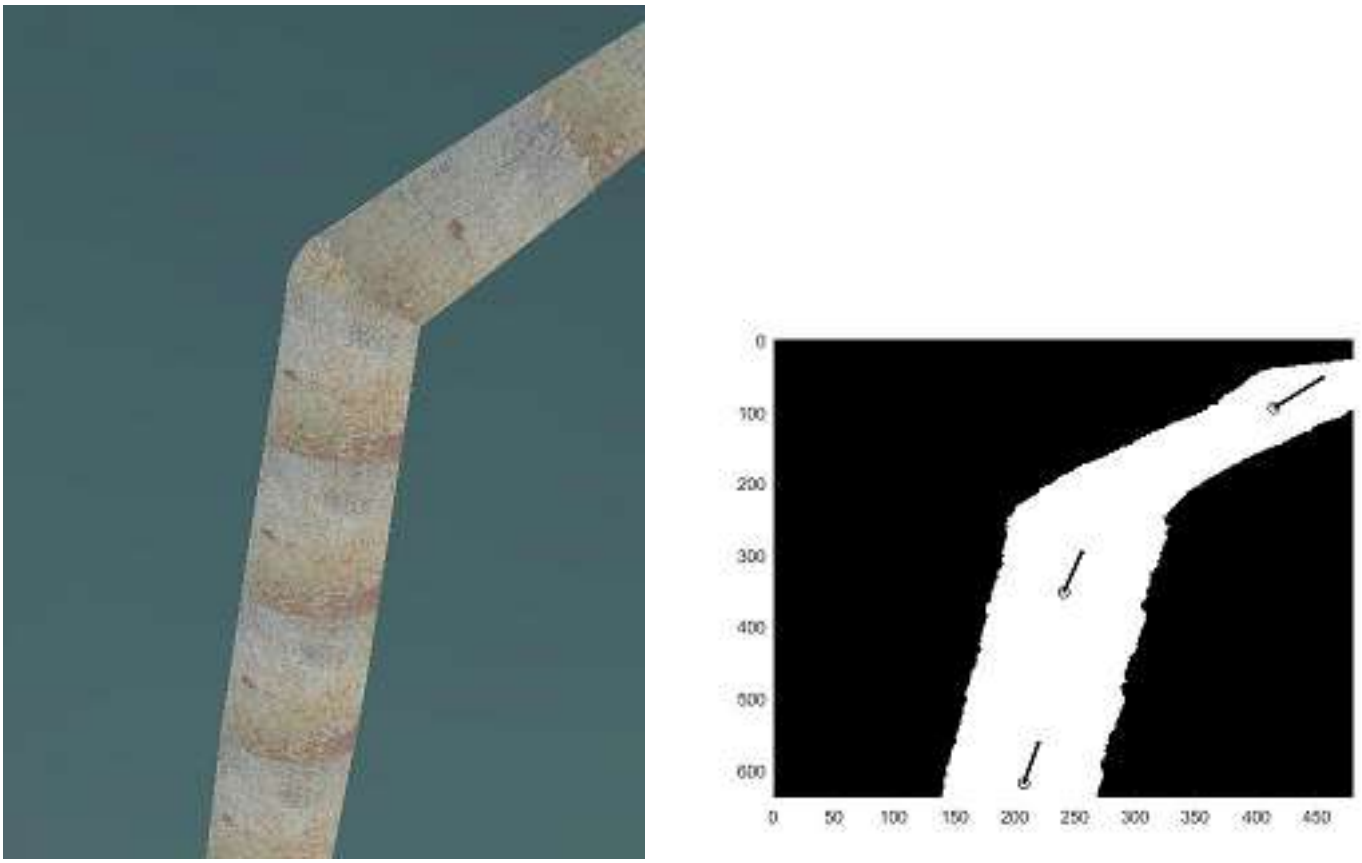
(a) **(b)**

Abbildung 22: Grades Objekt mit optimaler Qualität



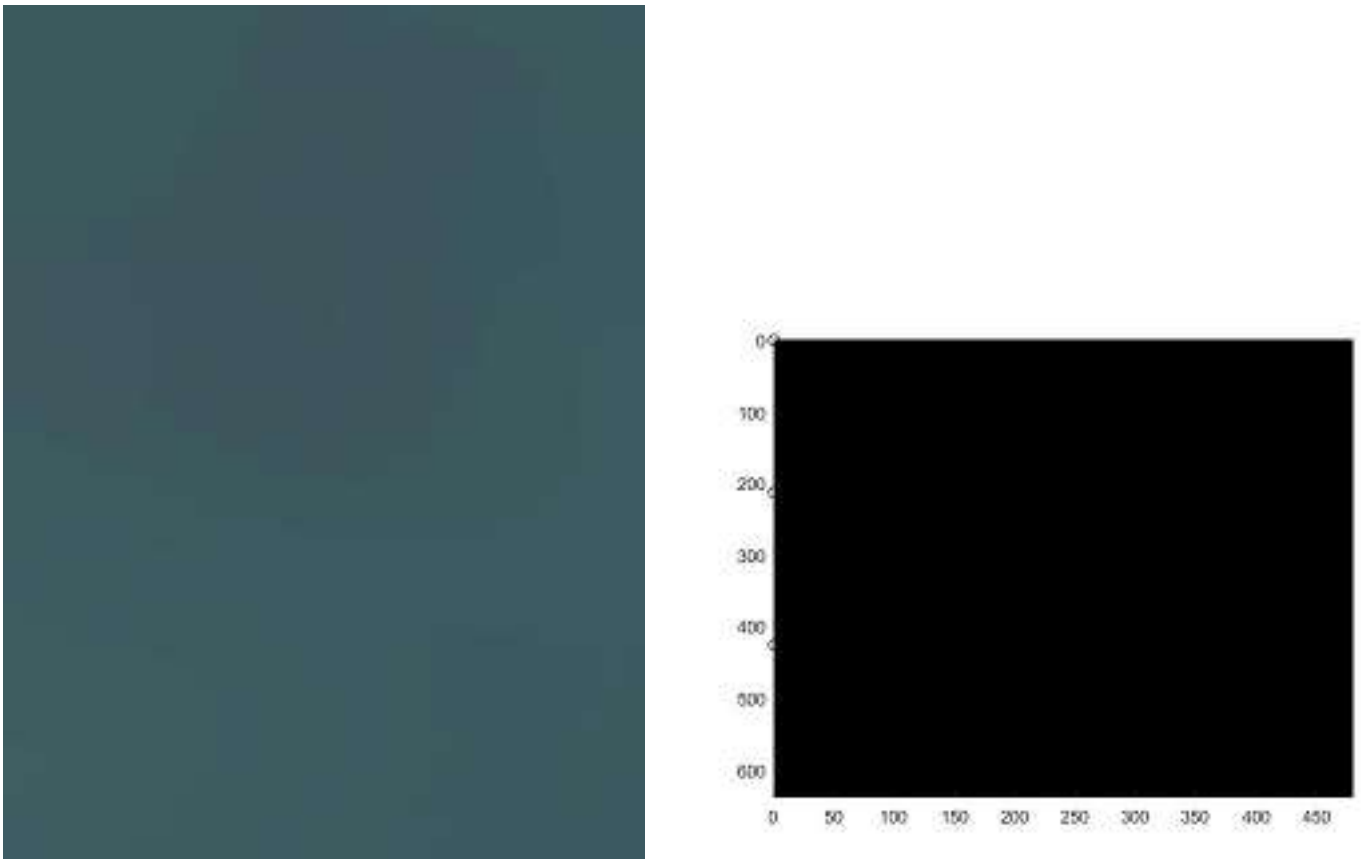
(a) **(b)**

Abbildung 23: Verdecktes Objekt mit optimaler Qualität



(a) **(b)**

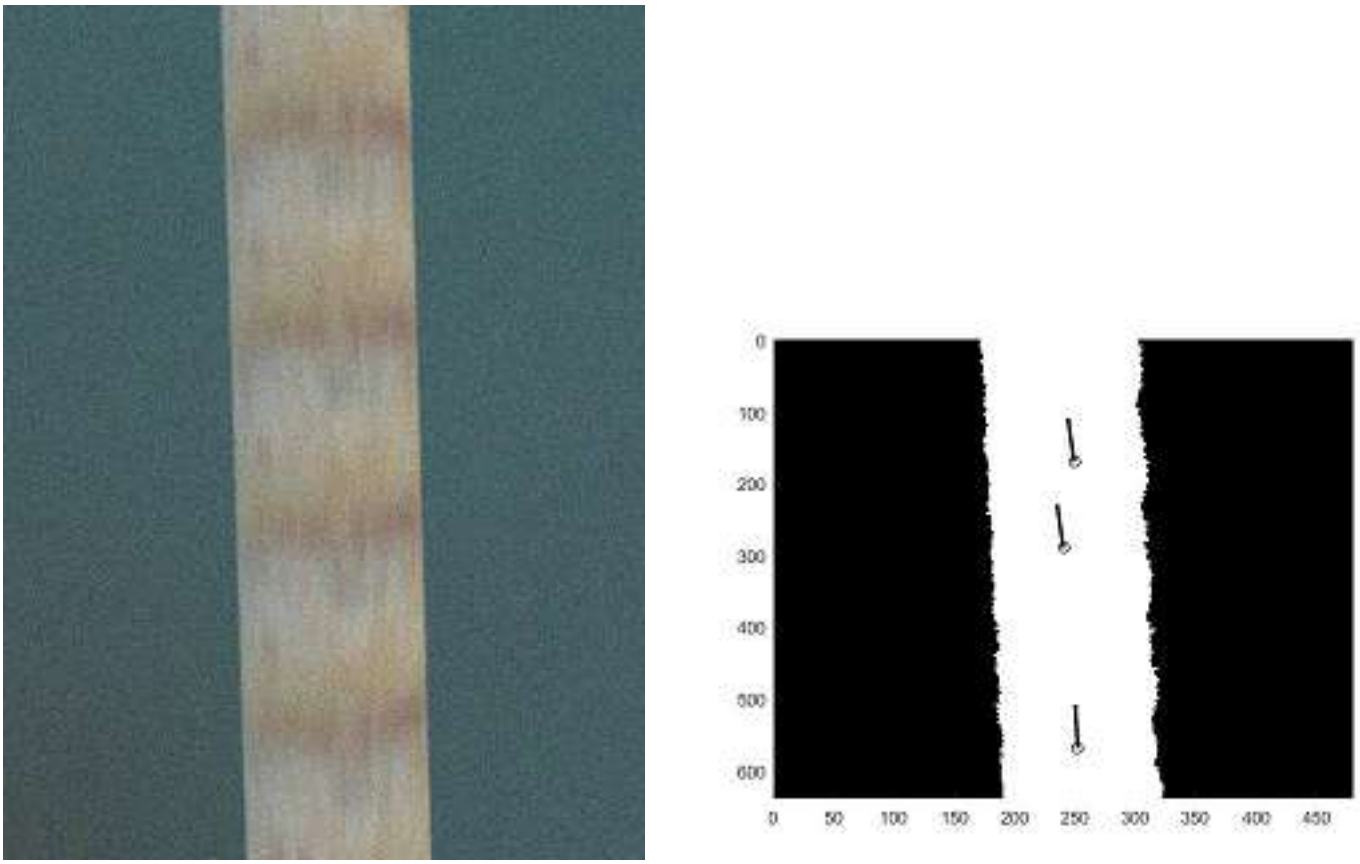
Abbildung 24: Geknicktes Objekt mit optimaler Qualität



(a) **(b)**

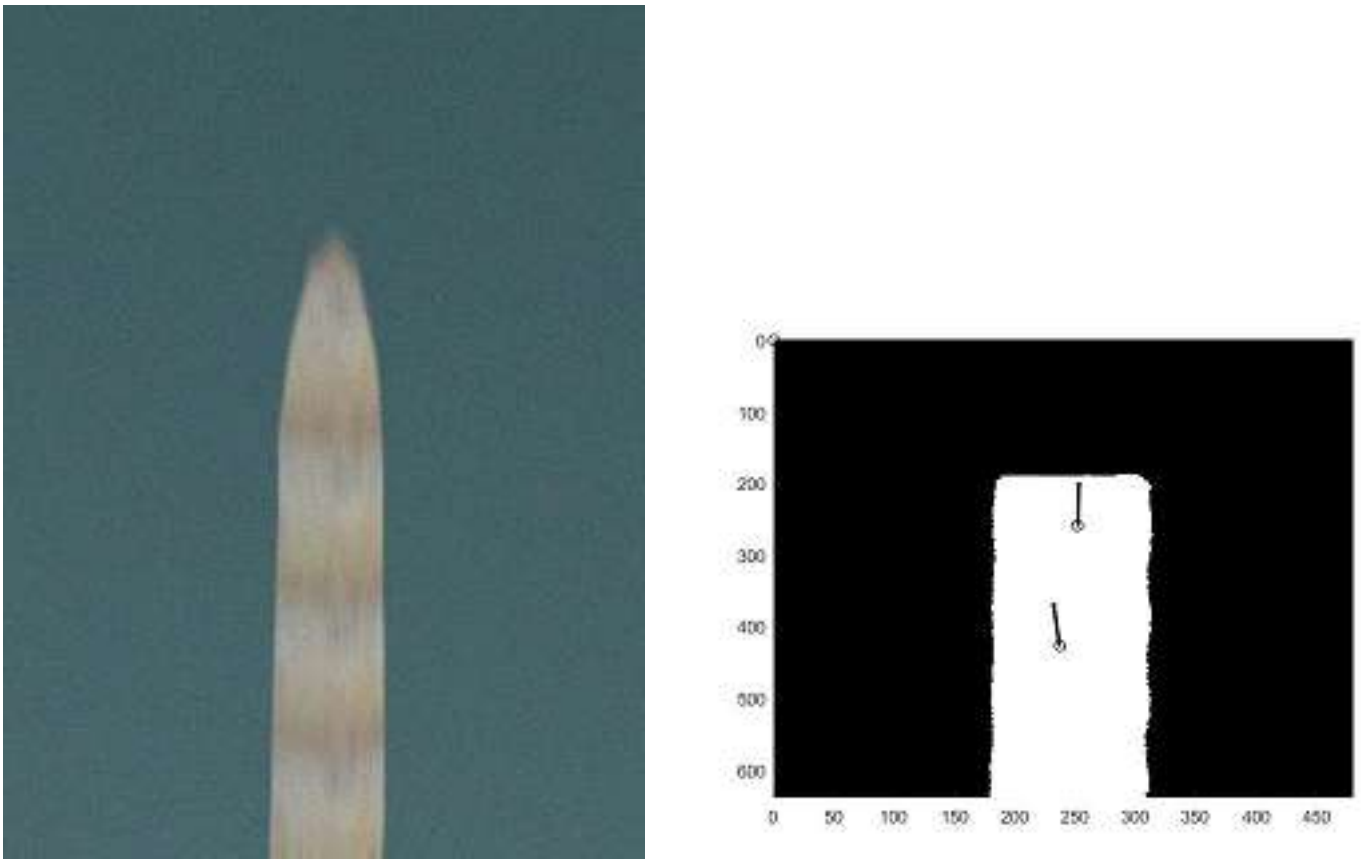
Abbildung 25: Kein Objekt mit optimaler Qualität

Zweite Testreihe



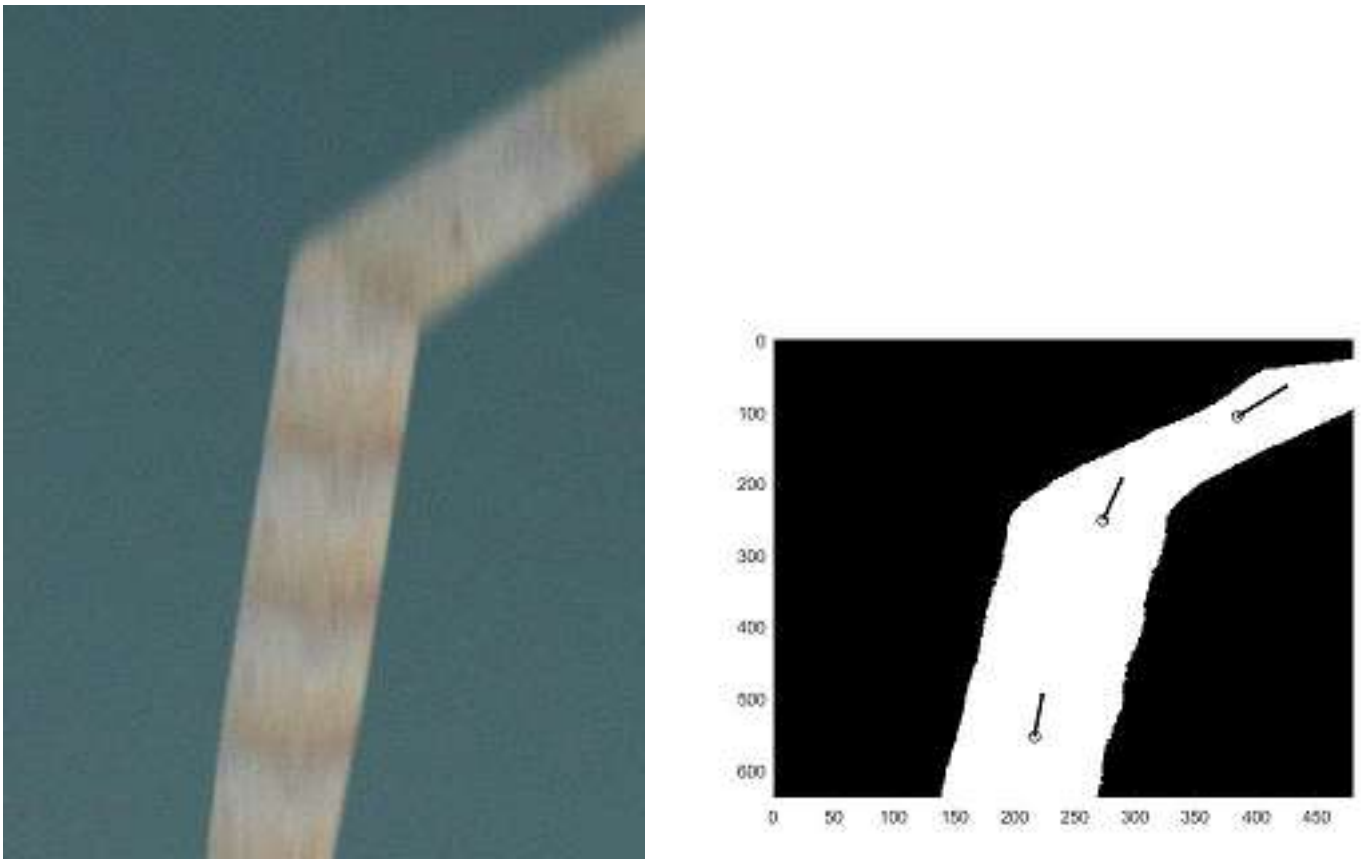
(a) (b)

Abbildung 26: Grades Objekt mit verschlechterter Qualität



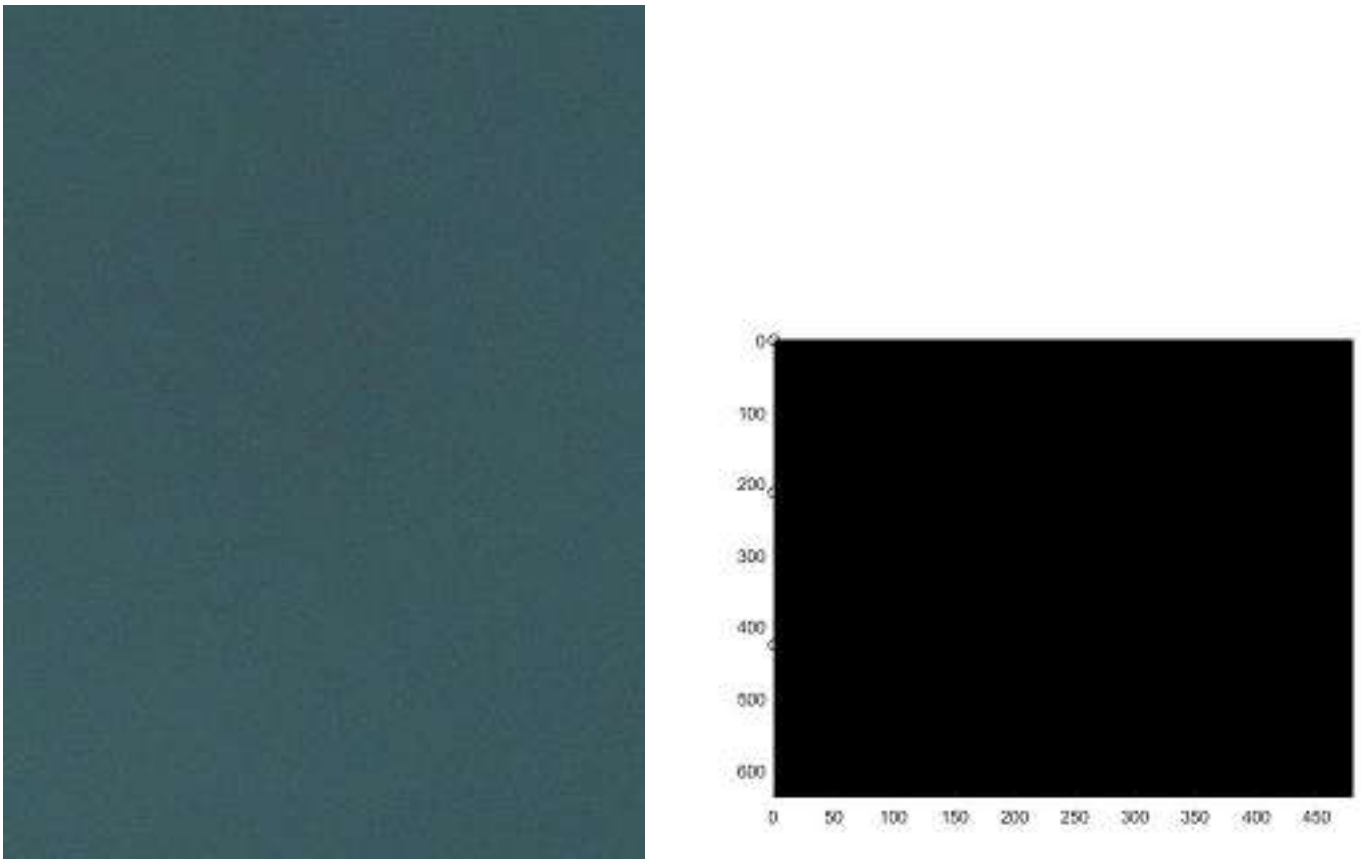
(a) **(b)**

Abbildung 27: Verdecktes Objekt mit verschlechterter Qualität



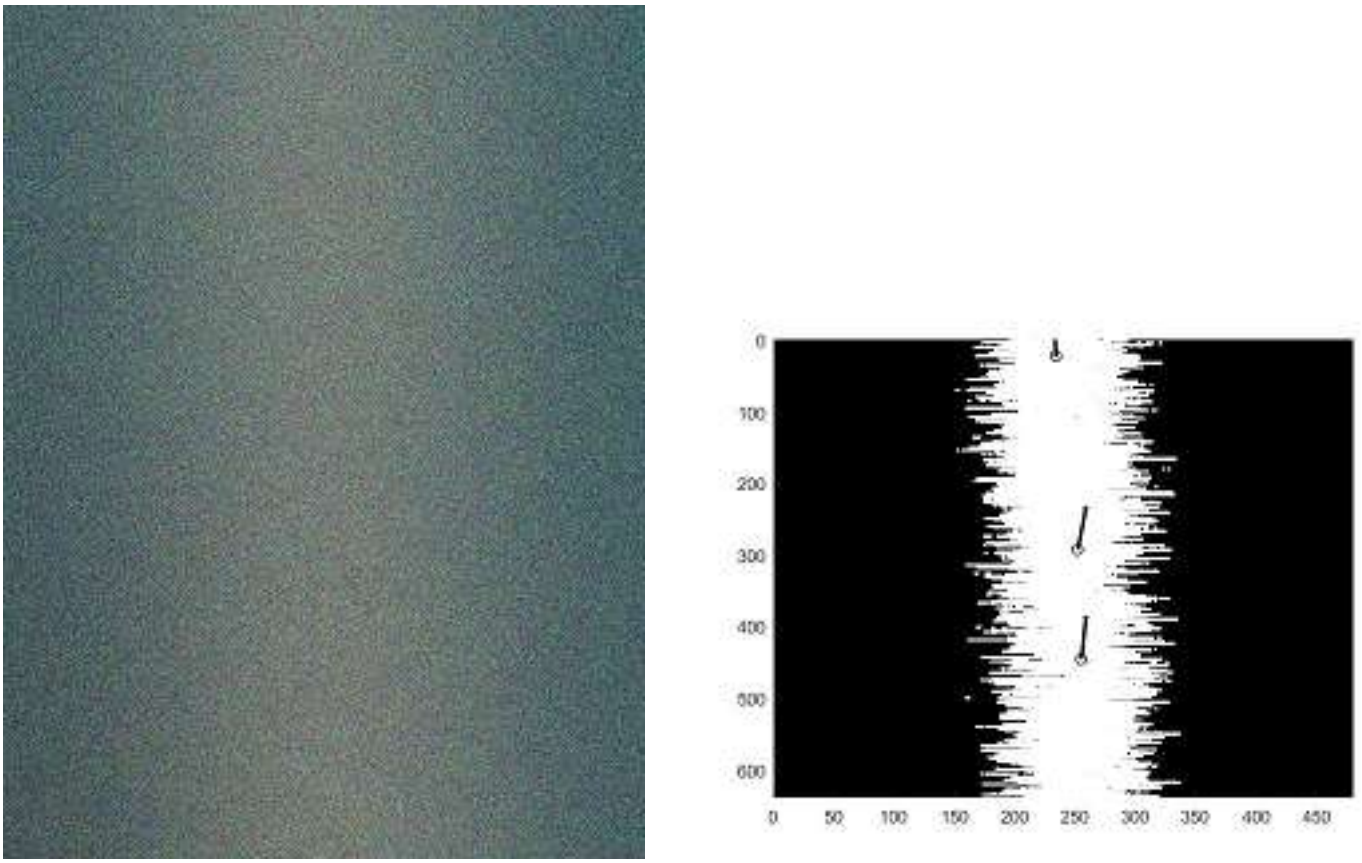
(a) **(b)**

Abbildung 28: Geknicktes Objekt mit verschlechterter Qualität



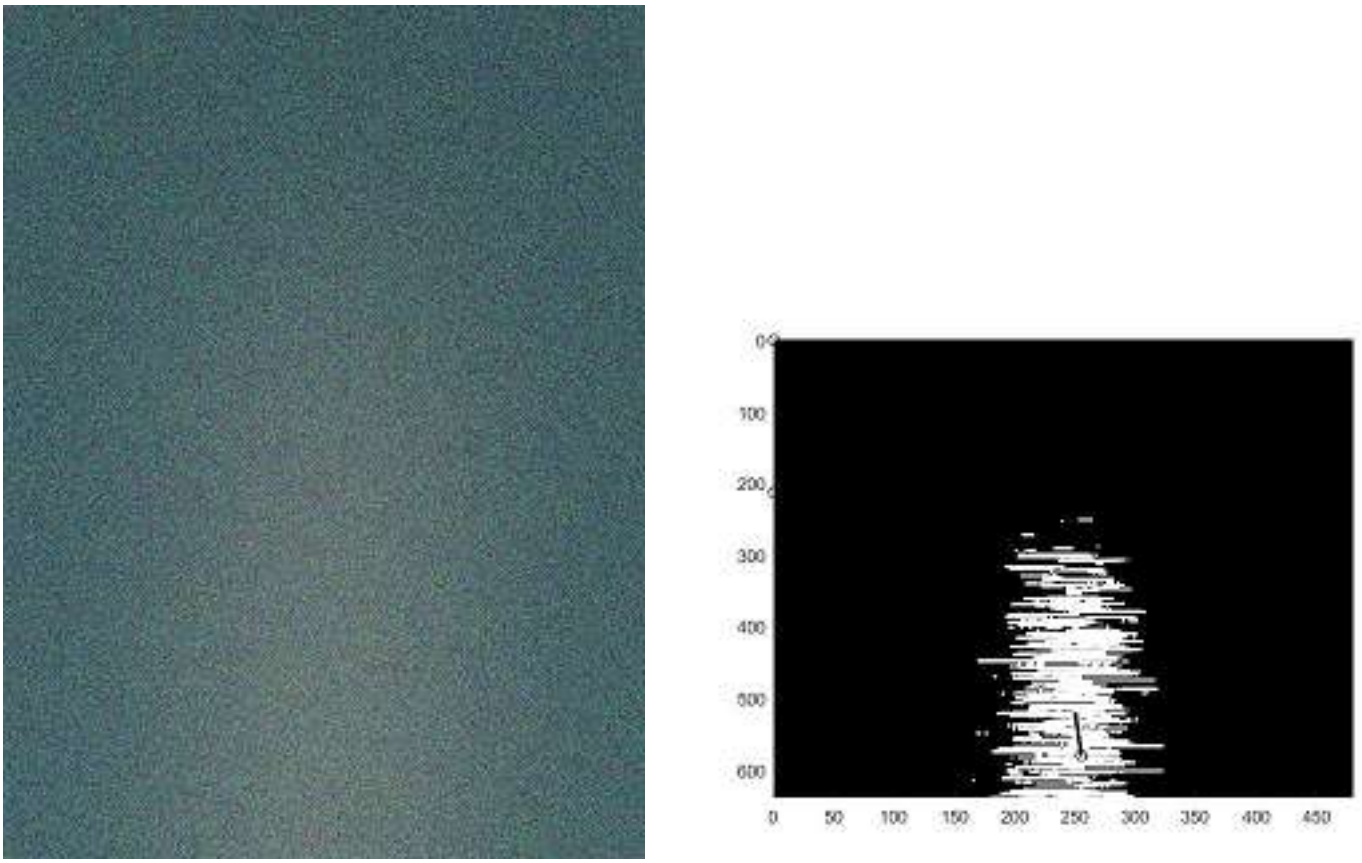
(a) **(b)**

Abbildung 29: Kein Objekt mit verschlechterter Qualität



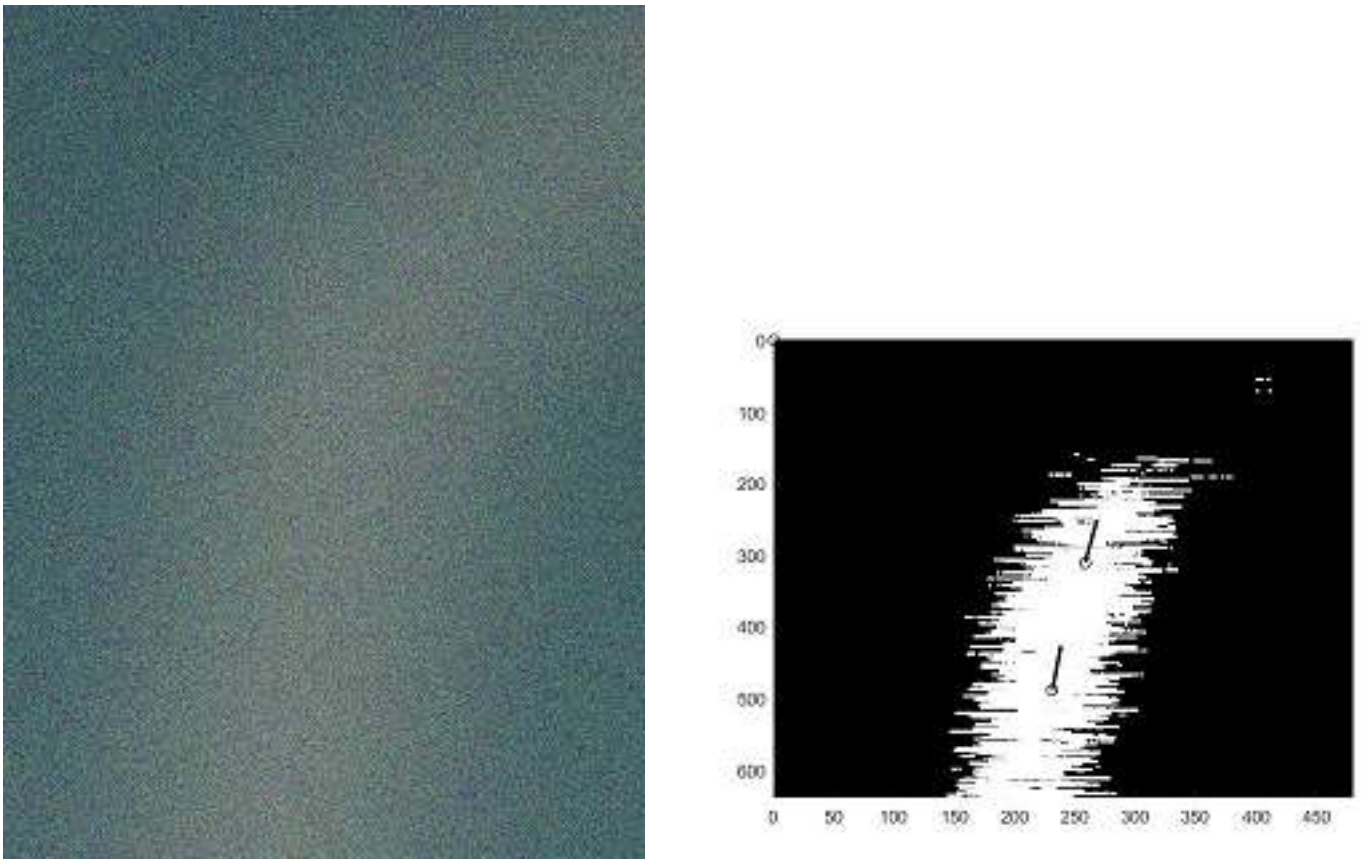
(a) **(b)**

Abbildung 30: Grades Objekt mit sehr schlechter Qualität



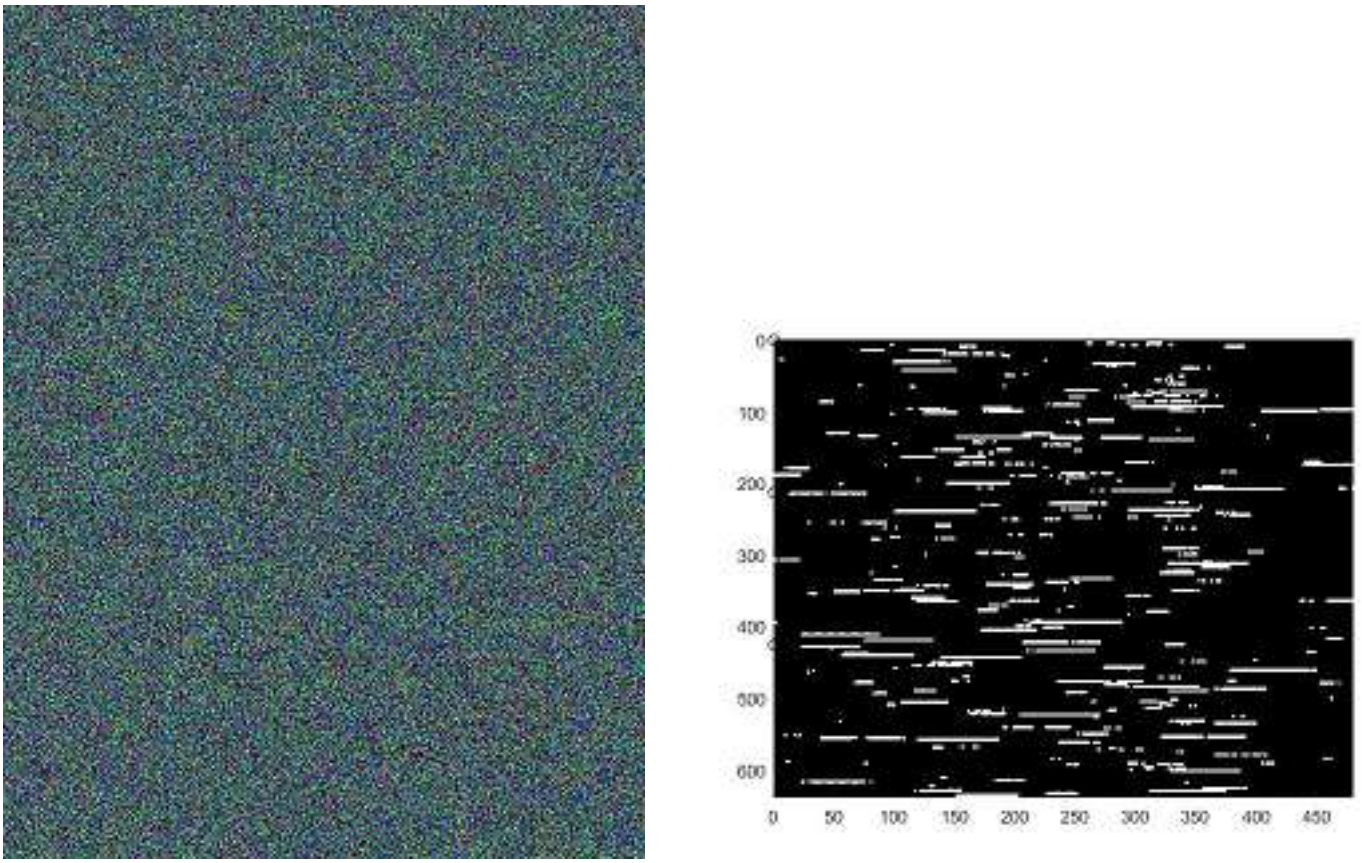
(a) **(b)**

Abbildung 31: Verdecktes Objekt mit sehr schlechter Qualität



(a) **(b)**

Abbildung 32: Geknicktes Objekt mit sehr schlechter Qualität



(a) **(b)**

Abbildung 33: Kein Objekt mit sehr schlechter Qualität

Dritte Testreihe

Um zu zeigen, dass das Verfahren auch bei echten Bildern funktioniert wurden insgesamt 12 Bilder [E.1] aus einem Testlauf des AUVs *DAGON*[7] während des Projektes *CUSLAM*[8] im Unisee getestet. In einige Bilder, wie [??] oder [??] ist die Pipeline schwer zu erkennen. Dadurch muss der Schwellwert für das Template entsprechend niedrig gesetzt werden, was in vielen Punkten im Binärbild führt, die nicht zum Objekt gehören. In Bildern, in denen die Pipeline direkt angestrahlt wird und klar heller ist, wie in [??] oder [??] ist das Objekt wieder klar heller als der Hintergrund, der Schwellwert kann höher angesetzt werden und die Erkennung hat weniger Punkte im Binärbild.

[7] http://robotik.dfki-bremen.de/de/forschung/robotersysteme/dagon.html
[8] http://robotik.dfki-bremen.de/de/forschung/projekte/cuslam.html

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, damit eine stabile Fahrt erreicht wird und die Schwankungen beim anfänglichen Beschleunigen die Ergebnisse nicht verfälschen. Ebenso wird auch gewährleistet, dass das AUV stets direkt auf das Objekt trifft, da dass 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, die Rohbilder der Kamera, die Ausgabe der Objekterkennung und das berechnetet Polynom zu sehen ist.

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.

5.2.2. Kurve

Nach den Tests zum geraden Verlauf 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 sind. Für letzteres wurde eine Sinuskurve genutzt, um ein entsprechendes Objekt zu erzeugen. hier grafiken
Außerdem wurde noch eine Kurve nach einer langen Geraden erzeugt. Hiermit wird getestet, ob auch wechselnden geometrischen Strukturen gefolgt werden kann.

5.3. Kreisbahn

Für die finalen Tests wurden Kreisbahnen in Form von Ellipsen verwendet. Eine Ellipse erfüllt einige Eigenschaften, die für die Arbeit nicht trivial zu lösen sind. Zum einen gibt es verschieden stark gebogene Kurven und fast gerade Abschnitte. Zum anderen gibt es ständig Abschnitte, die parallel zur $Y - Achse$ verlaufen.

5.4. Parametrisierung

Während der Tests ist stark aufgefallen, dass das Regressionsverfahren sehr parameterabhängig ist.

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. 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. URL: 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.

B. Abbildungsverzeichnis

C. Gleichungsverzeichnis

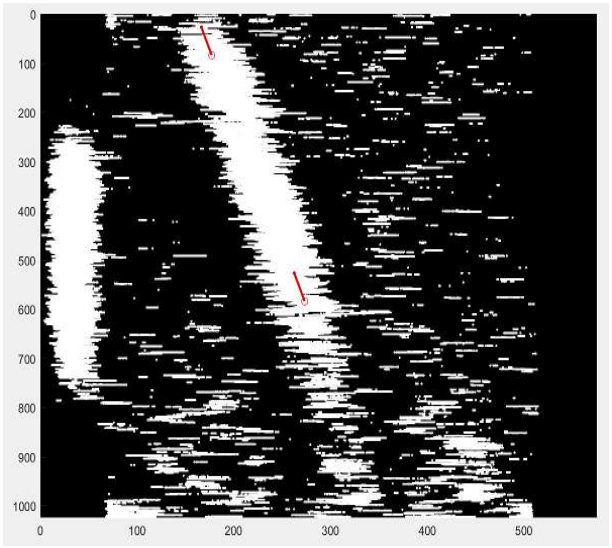
D. Listingverzeichnis

E. Anhang

E.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.

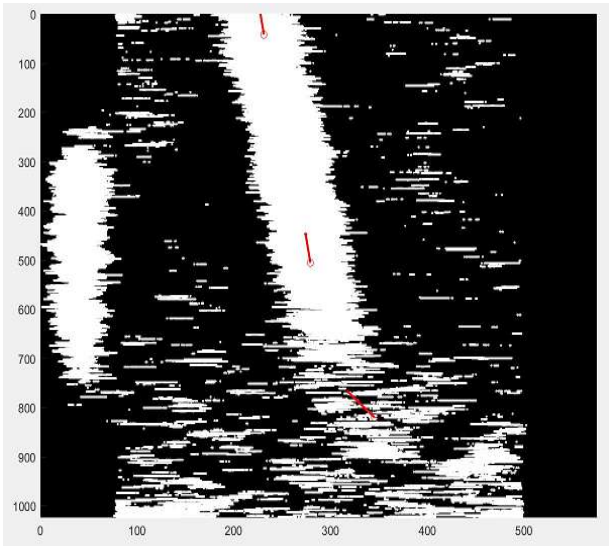
(a) Originalbild



(b) Erkannte Objekte

Abbildung 34

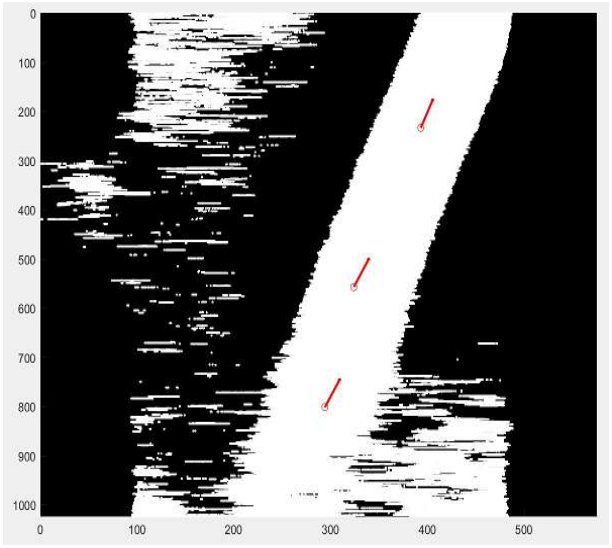
(a) Originalbild



(b) Erkannte Objekte

Abbildung 35

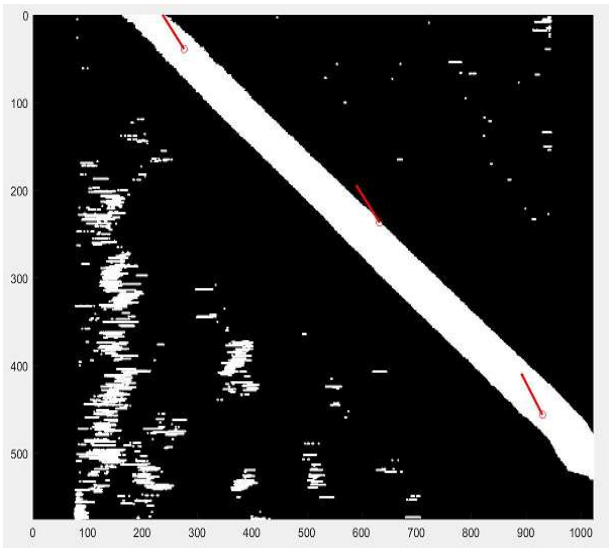
(a) Originalbild



(b) Erkannte Objekte

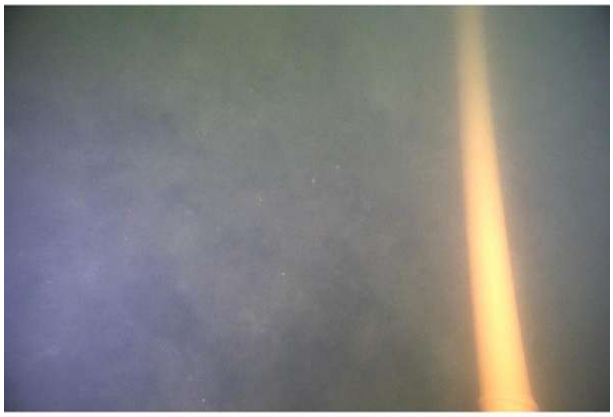
Abbildung 36

(a) Originalbild

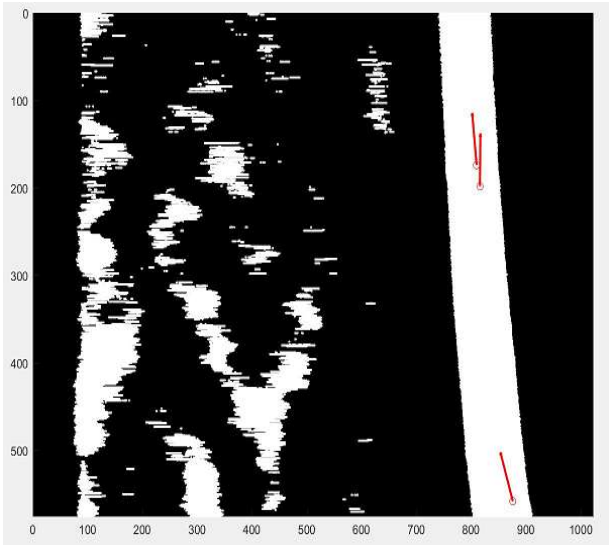


(b) Erkannte Objekte

Abbildung 37

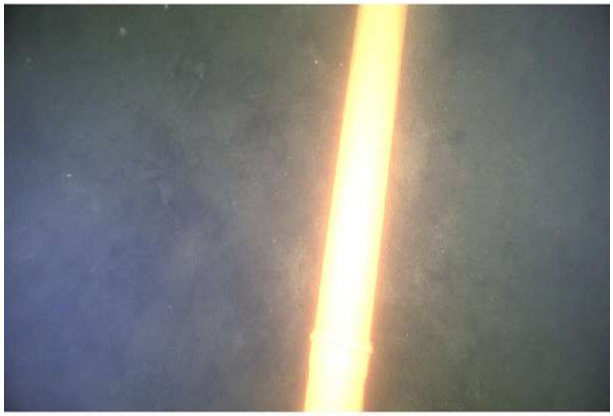


(a) Originalbild

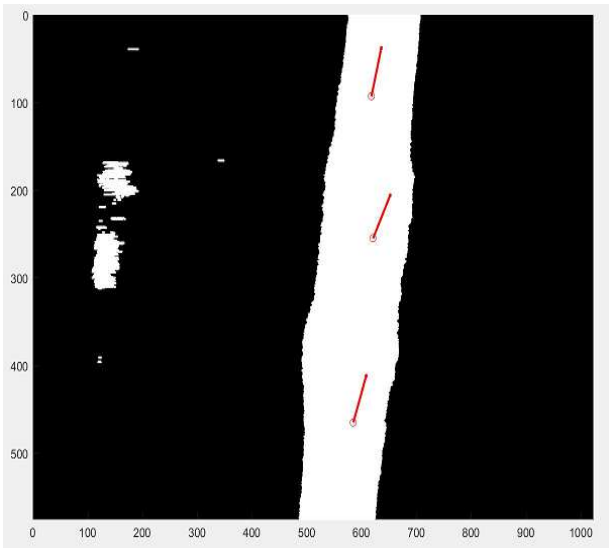


(b) Erkannte Objekte

Abbildung 38



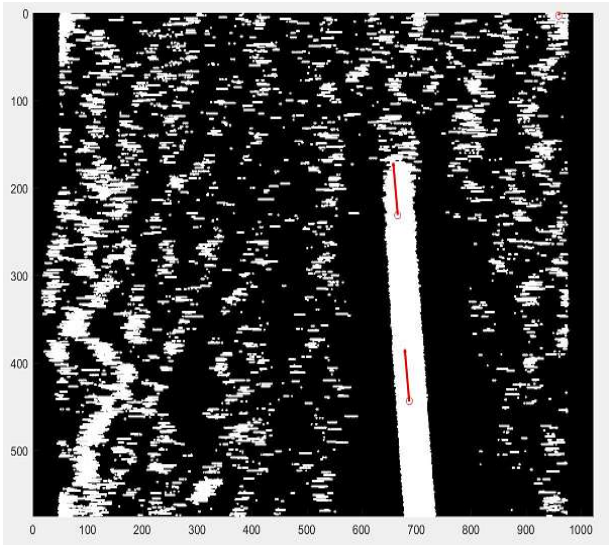
(a) Originalbild



(b) Erkannte Objekte

Abbildung 39

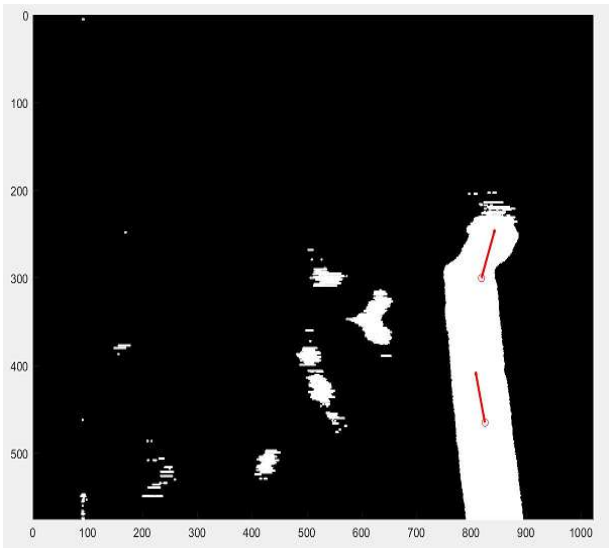
(a) Originalbild



(b) Erkannte Objekte

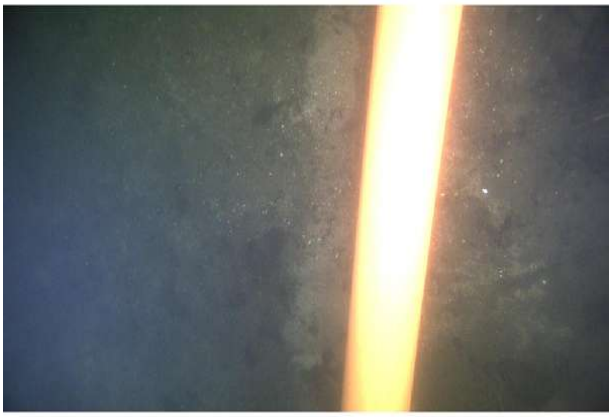
Abbildung 40

(a) Originalbild

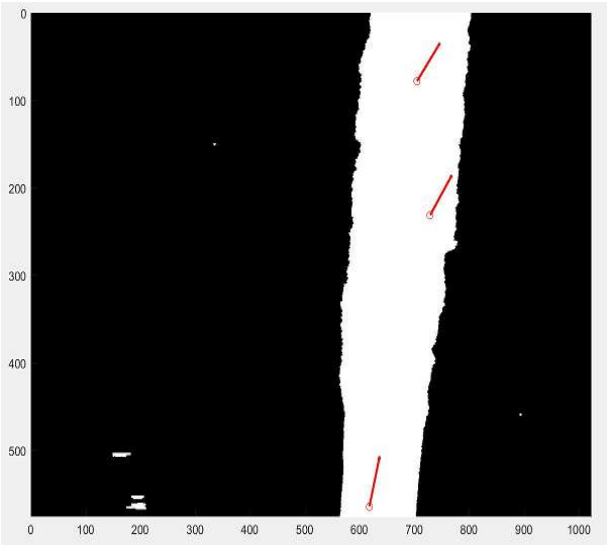


(b) Erkannte Objekte

Abbildung 41

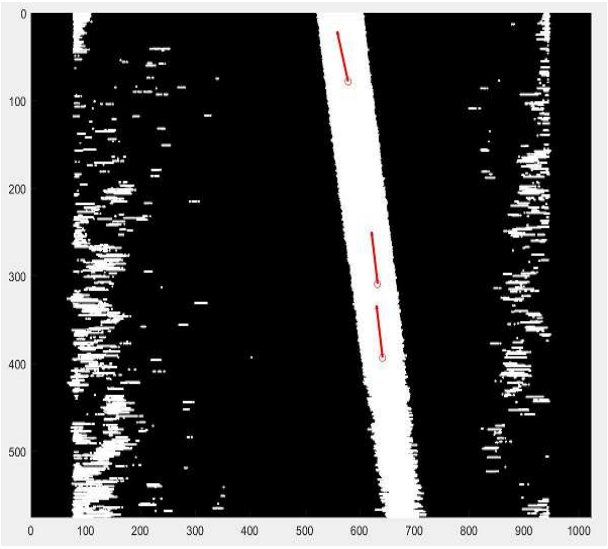


(a) Originalbild



(b) Erkannte Objekte

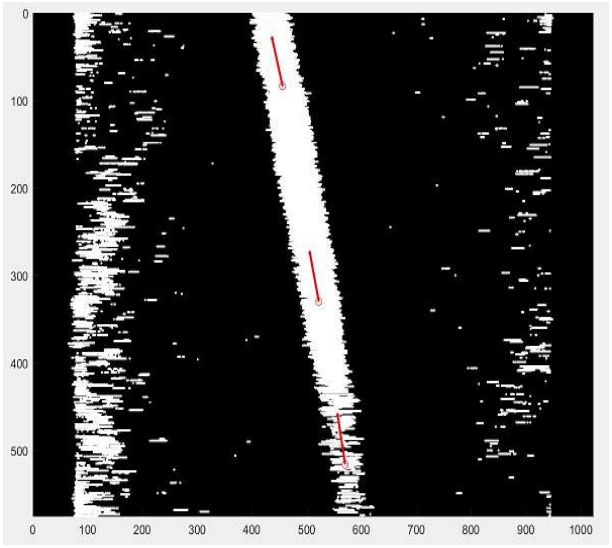
(c) Originalbild



(d) Erkannte Objekte

Abbildung 42

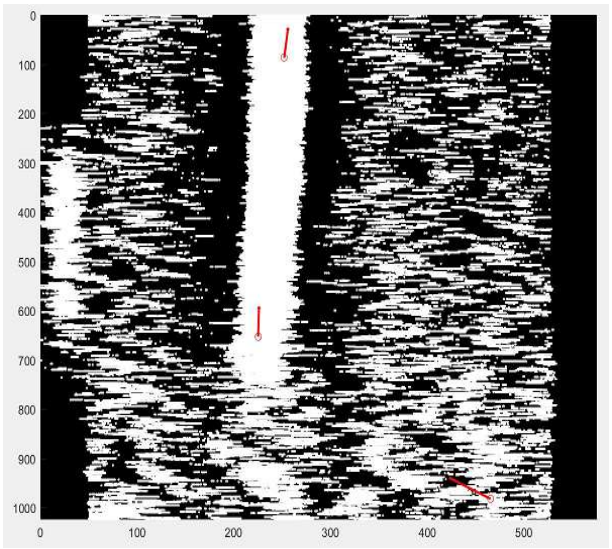
(a) Originalbild



(b) Erkannte Objekte

Abbildung 43

(a) Originalbild



(b) Erkannte Objekte

Abbildung 44