Mobile Roboterplattform

Benjamin Ranft, Jacques Mayländer, Markus Schreiber und Marc Sons

Karlsruher Institut für Technologie (KIT)
Institut für Mess- und Regelungstechnik

Lernziele

1. Erkennung einer roten Linie in Kamerabildern
2. Least-Squares-Schätzung von Modell-Parametern
3. Robuste Schätzung von Modell-Parametern (M-Estimator, RANSAC)
4. Programmierung in C++ mit *ROS* und *OpenCV*

Inhaltsverzeichnis

1 Einleitung

In den letzten Jahren wurden Quadrokopter immer populärer. Dank der rasanten Entwicklung von Prozessoren und Sensoren für Smartphones können auch diese kleinen unbemannten Fluggeräte mittlerweile für private Nutzer erschwinglich hergestellt werden. In Verbindung mit einem anwendungsfreundlichen Steuerungs-Interface, z. B. Gamepad oder Tablet-PC, können sogar Kinder den Flug problemlos steuern.

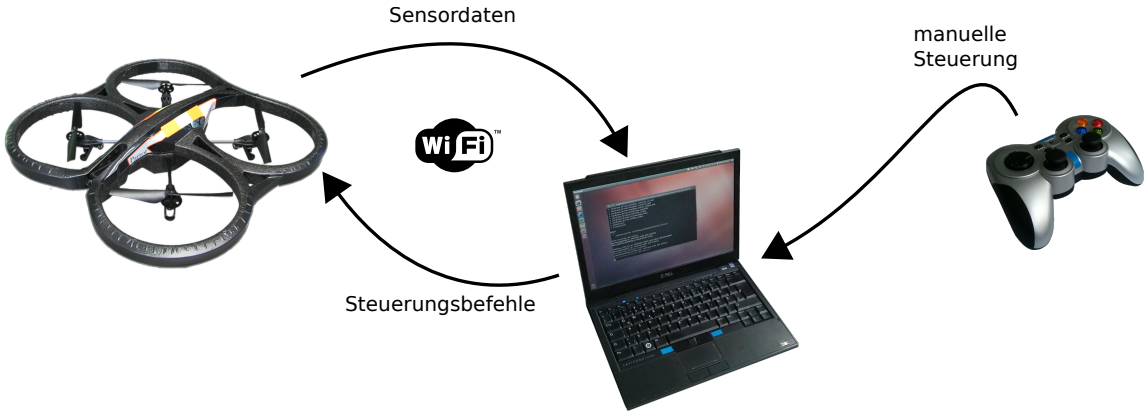
Leider gehen in diesen gekapselten Systemen die technischen Funktionalitäten unter. Man sieht als "Anwender" nicht, wie und warum etwas funktioniert. In diesem Praktikumsversuch soll zunächst die Funktionsweise eines Quadrokopters veranschaulicht werden. Beim nachfolgenden Parametrieren einer Höhen- und Gierwinkelregelung sowie beim abschließenden automatischen Nachfliegen einer Linie auf dem Boden soll Fachwissen im Bereich der Mess- und Regelungstechnik vertieft und angewendet werden.

1.1 Versuchsaufbau

Für diesen Versuch wird das Quadrokopter-Modell *ARDrone 2.0* der Firma *Parrot SA* verwendet. Dieses ist mit den folgenden Sensoren ausgestattet:

- ein Ultraschall-Abstandssensor zur Höhenmessung
- eine nach unten gerichtete Kamera
- eine nach vorne gerichtete Kamera (in diesem Versuch nicht genutzt)
- ein Inertial-Messsystem mit Kompass zur Bewegungs- und Orientierungsschätzung

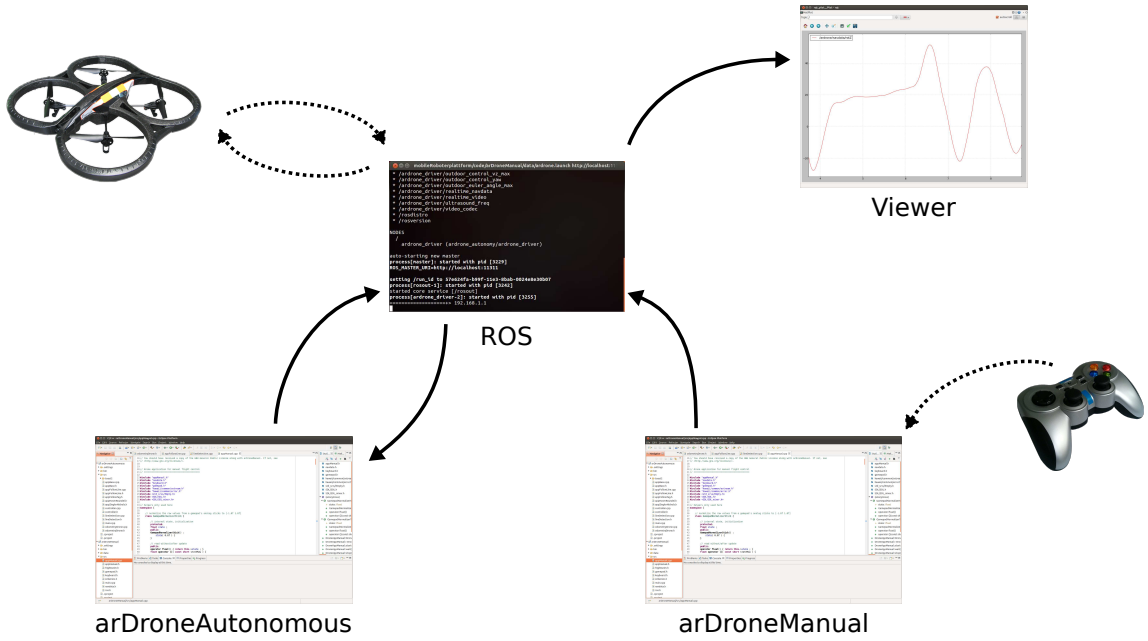
Der Quadrokopter selbst ist ein geschlossenes System, auf dem nicht ohne Weiteres eigener Programmcode ausgeführt werden kann. Stattdessen überträgt er Messwerte und Bilddaten über WLAN an einen Rechner. Dieser verarbeitet die Daten und sendet Steuerungsbefehle an den Quadrokopter zurück. Befehle der autonomen Flugsteuerung können jederzeit mit manuellen Eingaben per Tastatur oder Gamepad überlagert werden.



1.2 Softwarestruktur

Ein Grundbaustein ist das Software-Framework *Robot Operating System (ROS)*: `ros.org`. Es ist im Bereich der Robotik weit verbreitet, so dass viele Sensoren und Roboter – u. a. die *AR.Drone* – mit bestehenden Software-Paketen direkt angesprochen werden können. Eine zentrale Datenbank erlaubt die Kommunikation zwischen Quadrokopter und verschiedenen Programmen.

Die Datenverarbeitung auf dem Rechner findet in zwei unabhängigen Programmen statt: Eines bestimmt laufend die Kommandos für den autonomen Flug, ein weiteres implementiert die manuelle Steuerung. Hierdurch ist es möglich, bei einer eventuellen Fehlfunktion oder einem Absturz der autonomen Steuerung jederzeit per Gamepad oder Tastatur die Kontrolle über den Quadrokopter zu übernehmen. In diesem Versuch wird nur das Programm zum autonomen Fliegen bearbeitet, während das zur manuellen Steuerung unverändert genutzt wird.



Die Software ist in der Programmiersprache C++ realisiert. Zur Bildverarbeitung wird die *OpenCV*-Bibliothek verwendet. Als Entwicklungsumgebung kommt *Eclipse CDT* unter dem Linux-Betriebssystem *Ubuntu* zum Einsatz.

2 Zielsetzung

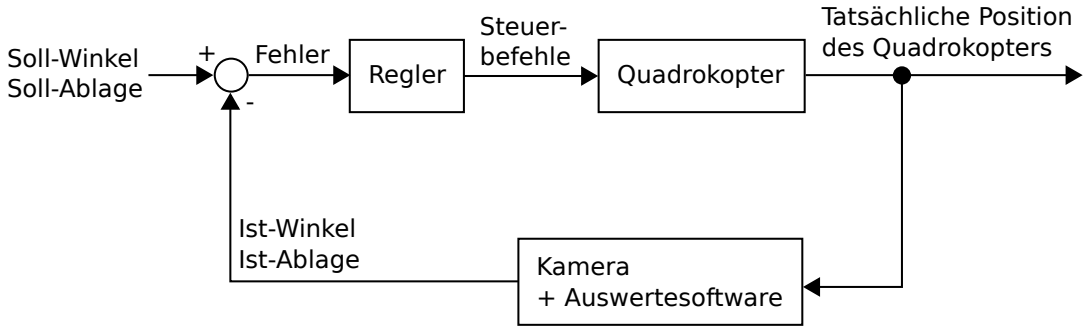
Ziel des Versuchs ist das autonome Nachfliegen einer roten Linie auf dem Boden, welche durch ein flexibles Seil dargestellt wird. Die Erkennung und Modellierung dieser Line ist dabei durch Teilverdeckungen und in der Nähe befindlichen Objekten gestört. Für eine ausreichend gute Linienschätzung müssen daher im Vorfeld zwei renormierte robuste Schätzverfahren programmiert, evaluiert und miteinander verglichen werden. Die Regelung

des Quadrokopters sowie die Regelung für das Nachfliegen der Linie sind als gegeben angenommen.

3 Grundlagen

3.1 Regelung des Quadrokopters

Der Quadrokopter besitzt vier Freiheitsgrade: Er kann sich translatorisch vor-/rückwärts, seitwärts und auf-/abwärts bewegen sowie um seine Hochachse rotieren ("gieren"). Für jede Achse wird als Stellgröße eine normierte (Winkel-)Geschwindigkeit $\in [-1, 1]$ vorgegeben, welche intern durch einen unterlagerten Regler eingeregelt wird. Es müssen also nicht einzelne Motordrehzahlen o. ä. bestimmt werden. Beim Folgen der Linie werden von den vier Freiheitsgraden nur zwei geregelt, nämlich der Gierwinkel und die seitliche Ablage. Flughöhe und Vorwärtsgeschwindigkeit sind dagegen konstant.



3.2 Erkennung der Linie

Im Folgenden wird zwischen Bildkoordinatensystem und Quadrokopter-Koordinatensystem unterschieden. Wie in der Bildverarbeitung üblich wird eine Pixelposition im Bild mit (u, v) angegeben. Der Ursprung ist die linke obere Ecke. Das Koordinatensystem des Quadrokopters ist so orientiert, dass die X-Achse nach vorne (in Richtung der vorderen Kamera), die Y-Achse nach links und die Z-Achse nach oben zeigen (Abbildung 1).

3.2.1 Bildvorverarbeitung

Nach der Bildvorverarbeitung soll die Linie deutlich hervortreten und Störungen im Bild möglichst verschwinden. Hierzu wird für jedes Pixel unabhänigig ein Wert im Ergebnisbild L berechnet, der umso höher ist, je eher das Aussehen des Pixel der Linie entspricht.

HSV-Farbraum Hilfreich zur Erkennung der roten Linie ist die Transformation des Bildes in den HSV-Farbraum. Beim klassischen RGB-Farbraum wird eine Farbe durch die additive Überlagerung der Farbintensitäten der Grundfarben Rot, Grün und Blau definiert. Dagegen beschreibt der HSV-Farbraum eine Farbe mit dem Farbwert $H \in [0°, 360°]$ (*hue*), der Sättigung $S \in [0, 1]$ (*saturation*) und dem Hellwert $V \in [0, 1]$ (*value*). Jede Farbangabe mit $V = 0$ entspricht dabei Schwarz.

[3] basierend auf Grafik von Eric Pierce (http://en.wikipedia.org/wiki/File:HSV_cone.jpg), lizensiert unter Creative Commons Attribution-Share Alike 3.0 Unported (http://creativecommons.org/licenses/by-sa/3.0)

Binarisierung Eine einfache Möglichkeit zur Unterscheidung zwischen “Linie” und “nicht Linie” ist die Binarisierung anhand der Farbwerte. So kann beispielsweise ein minimaler und maximaler Farbwert H_{min} bzw. H_{max}, sowie eine Schwelle für die Sättigung S_{min} und den Hellwert V_{min} festgelegt werden. Ein Pixel (u, v) mit dem Farbwert H_{uv}, der Sättigung S_{uv} und dem Hellwert V_{uv} gehört genau dann zur Linie, wenn gilt:

$$(H_{min} < H_{uv} < H_{max}) \quad \wedge \quad (S_{min} < S_{uv}) \quad \wedge \quad (V_{min} < V_{uv}) \tag{1}$$

Für jedes Pixel (u, v) kann entsprechend als Ergebniswert definiert werden:

$$L_{uv} = \sigma(H_{uv} - H_{min}) \cdot \sigma(H_{max} - H_{uv}) \cdot \sigma(S_{uv} - S_{min}) \cdot \sigma(V_{uv} - V_{min}) \tag{2}$$

Linienevidenz Statt binär zwischen “Linie” und “nicht Linie” zu unterscheiden, können auch Zwischenwerte im Übergangsbereich genutzt werden. Pro Kanal $K \in \{H, S, V\}$ im Farbraum wird eine Bewertungsfunktion w_K durch zwei Schwellwerte K_1 und K_2 beschrieben. Innerhalb dieses Übergangsbereichs zwischen K_1 und K_2 wird linear interpoliert. Das Gesamtergebnis jedes Pixels erhält man durch Multiplikation der verschiedenen Bewertungen:

$$L_{ges} = L_H \cdot L_S \cdot L_V \tag{3}$$

3.2.2 Schätzung der Linie

Modellierung Die Linie wird als Polynom dritten Grades modelliert. Damit sind s-förmig gekrümmte Linien mit zwei Wendepunkten im Bild möglich. Da der Quadrokopter in Längsrichtung der Linie nachfliegen soll, verläuft diese vornehmlich von oben nach unten durch das Bild. Es bietet sich daher eine Funktion $u = f(v)$ an, wobei v der Bild-Zeile und $u = f(v)$ der Bild-Spalte entspricht:

$$f(v) = av^3 + bv^2 + cv + d \tag{4}$$

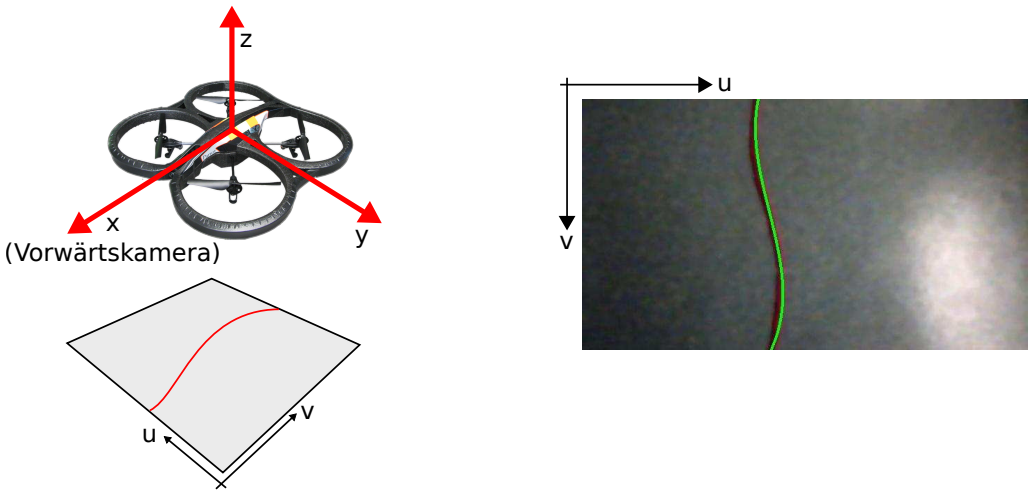


Abbildung 1: Koordinatensystem des Quadrokopters und des Kamerabildes

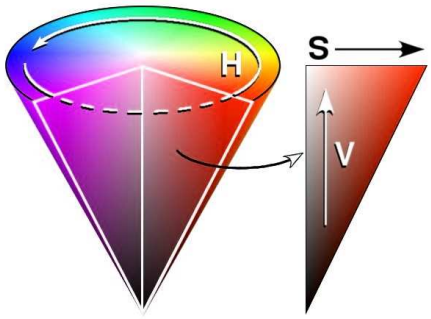


Abbildung 2: Veranschaulichung des HSV-Farbraums[3]

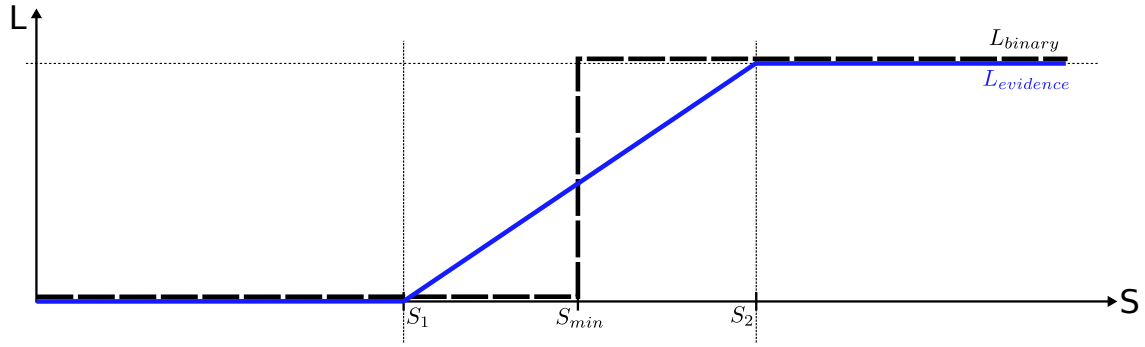


Abbildung 3: Vergleich der Bewertungsfunktion für die Sättigung S eines Pixels bei Binarisierung bzw. mit einem Übergangsbereich.

Lokalisierung von zeilenweisen Maxima Die Linie wird aus mehreren Punkten geschätzt, von denen erwartet wird, dass sie auf der Linie liegen. Mit der bisherigen Bildvorverarbeitung haben Pixel auf der Linie einen hohen Ergebnis-Wert. Daher ist es naheliegend, für jede Zeile v_i des Bildes diejenige Spalte $u_i(v_i)$ mit dem maximalen Wert zu bestimmen.

In einem binarisierten Bild haben jedoch alle Pixel auf der Linie einen gleichhohen Wert, so dass kein eindeutiges Maximum vorliegt. Abhilfe schafft hier eine Glättung entlang jeder Zeile, etwa mit einem Gauß-Filter. Dagegen treten in einem Linienevidenz-Bild bei geeigneten Parametern K_1 und K_2 auch ohne Filterung eindeutige Maxima hervor.

Aus Zeile v_i und geschätzer zugehöriger Spalte $\hat{u}_i$ können Punktpaare $(\hat{u}_i, v_i)$ für den Schätzer gebildet werden. Einen unsicherheitsbehafteten Schätzwert stellt in diesem Fall nur die Spalte $\hat{u}_i$ dar, denn die Zeile v_i ist exakt bekannt.

Least Squares Schätzung Für n gefundene Punkte auf der Linie gilt der folgende lineare Zusammenhang zwischen den Beobachtungen $\hat{u}_i$, der Designmatrix $\mathbf{H}$ und dem geschätzten Parametervektor $\hat{\mathbf{p}} = (\hat{a}, \hat{b}, \hat{c}, \hat{d})^T$ der Polynom-Koeffizienten:

$$\begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_n \end{pmatrix} + \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \vdots \\ \hat{e}_n \end{pmatrix} = \begin{bmatrix} v_1^3 & v_1^2 & v_1 & 1 \\ v_2^3 & v_2^2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ v_n^3 & v_n^2 & v_n & 1 \end{bmatrix} \begin{pmatrix} \hat{a} \\ \hat{b} \\ \hat{c} \\ \hat{d} \end{pmatrix} \quad \Longleftrightarrow \quad \hat{\mathbf{u}} + \hat{\mathbf{e}} = \mathbf{H}\hat{\mathbf{p}} \tag{5}$$

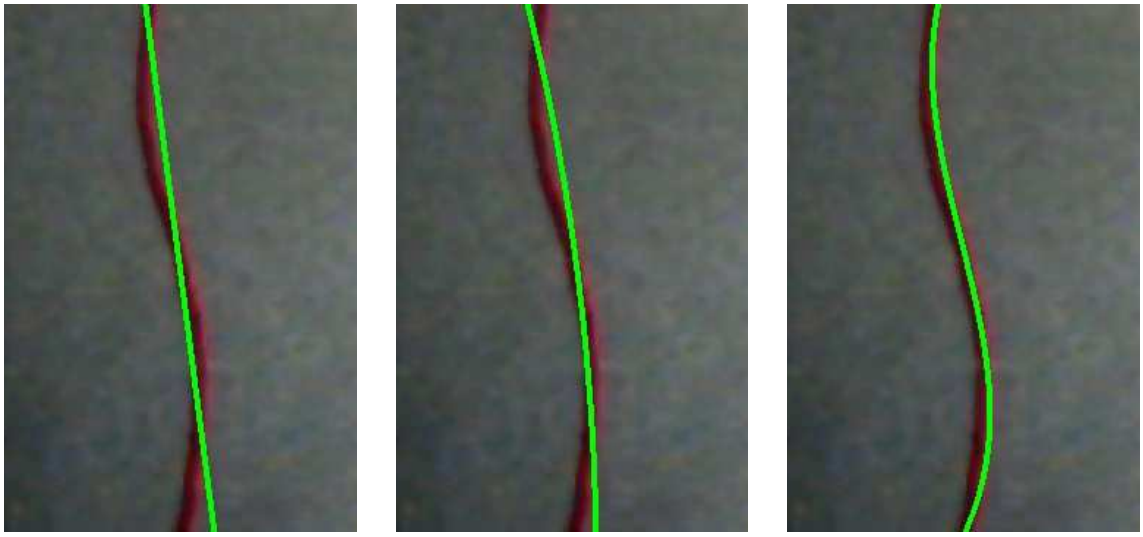
Dabei ist $\hat{\mathbf{e}}$ der Residuen-Vektor, welcher die Abweichungen zwischen den Beobachtungen und der geschätzten Linie beschreibt. Der Ausdruck $\hat{\mathbf{e}}^T\hat{\mathbf{e}}$ ergibt als skalaren Wert die Summe der Quadrate aller Fehler. Mit der folgenden Formel kann ein optimaler Parametervektor $\hat{\mathbf{p}}$ bestimmt werden, welcher dieses Fehlermaß minimiert:

$$\hat{\mathbf{p}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\hat{\mathbf{u}} \tag{6}$$

Eine Herleitung dieser Formel finden Sie ebenfalls im Skript zur Vorlesung *Grundlagen der Mess- und Regelungstechnik*.

3.2.3 Robuste Schätzung mittels M-Estimator

Die in Abschnitt 3.2.2 beschriebene Least-Squares-Schätzung stößt in der Praxis oftmals an ihre Grenzen, da einzelne Messwerte, die nicht dem Schätzmodell (also in diesem Fall der Polynomgleichung) entsprechen, das Ergebnis verfälschen. Solche Messwerte werden allgemein als Ausreißer (Englisch: Outliers) bezeichnet. Unter Umständen werden viele gute Datenpunkte durch wenige Ausreißer überlagert, was beim Least-Squares-Schätzer zu einer starken Verzerrung des zu schätzenden Polynoms führt. Im vorliegenden Anwendungsfall der bildbasierten Sensorik können als Ursache für Ausreißer in der Messreihe z. B. andere rote Objekte im Sichtbereich der Kamera sein.
Im weiteren Verlauf des Versuchs soll nun ein M-Estimator als Schätzverfahren implementiert werden, das robust gegenüber Ausreißern in der Messreihe ist. Abbildung 5 zeigt die Ergebnisse der Polynomschätzung für einen Least-Squares-Schätzer und einen M-Estimator. Dabei ist zu sehen, dass die Schätzung mit dem M-Estimator deutlich besser ist als die Schätzung mit dem Least-Squares-Schätzer.



(a) Gerade (b) Parabel (c) Polynom 3. Ordnung

Abbildung 4: vom Quadrokopter aus aufgenommene Bilder mit geschätzter Linie

Abbildung 5: Vergleich der Polynomschätzung mit Least-Squares-Schätzer und M-Estimator bei Ausreißern in den Beobachtungen. Orange: Beobachtungen. Gelb: Geschätztes Polynom. Links: Least-Squares-Schätzer. Rechts: M-Estimator.

Grundgedanke des M-Estimators ist es – wie beim Least-Squares-Schätzer – eine Funktion von $\mathbf{e}$ zu minimieren, die den Unterschied $\mathbf{e}$ zwischen Beobachtungen und Schätzungen bewertend beschreibt (vgl. Gl. (5))[4]. Bei einem Least-Squares-Schätzer ist diese Funktion quadratisch und berücksichtigt alle Messwerte gleichermaßen, was sich bei Ausreißern in der Messreihe negativ auswirkt. Durch die Wahl einer geeigneten Gewichtung der Beobachtungen in Abhängigkeit der Residuen e_i verringert der M-Estimator den Einfluss von Ausreißern. Dazu wird die herkömmliche Least-Squares-Schätzer Formulierung um eine positiv semidefinite Gewichtungsmatrix $\mathbf{W}$ erweitert

$$\hat{\mathbf{p}} = (\mathbf{H}^T\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{W}\hat{\mathbf{y}}. \tag{7}$$

Die Gewichtungsmatrix $\mathbf{W}$ ist beim M-Estimator eine Diagonalmatrix mit den Gewichten w_i auf der Hauptdiagonalen. Diese Gewichte werden dann iterativ als Funktion $w_i = w_i(e_i)$ des korrespondieren Residuums e_i angepasst. Als Startpunkt für den Iterationsprozess wird der Parametersatz p^0 eines Least-Squares-Schätzers gewählt. Dies kann durch das Setzen aller Gewichte $w_i^0 = 1$ erreicht werden. Als Einflussfunktion soll die Huber-Gewichtsfunktion

$$w(e) \quad = \quad \begin{cases} 2 & \text{, wenn } \mid e \mid \leq c \\ 2c/\mid e \mid & \text{, wenn } \mid e \mid > c \end{cases} \tag{8}$$

gewählt werden. Abbildung 6 zeigt die Gewichtsfunktion sowohl für den M-Estimator mit Huber-Funktion als auch für den Least-Squares-Schätzer. Beim M-Estimator werden Beobachtungen mit hohem Residuum e_i geringer gewichtet als Beobachtungen mit geringem Residuum. Beim Least-Squares-Schätzer werden alle Beobachtungen gleich gewichtet.

[4]Die Notation der Residuen mit einem ˆ-Zeichen wird im Folgenden ausgelassen.

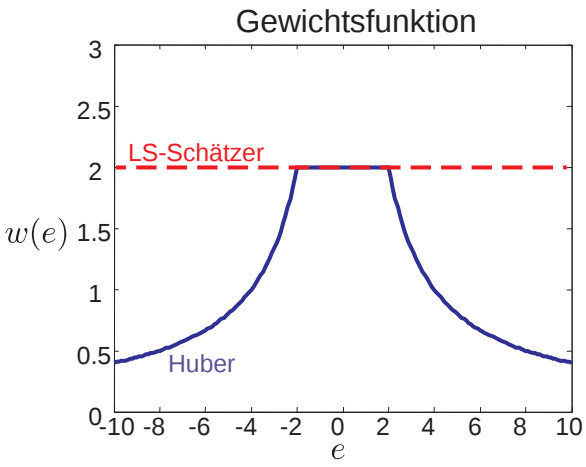


Abbildung 6: Links: Huber-Einflussfunktion (blau) und Einflussfunktion beim Least-Squares-Schätzer (rot). Rechts: Huber-Gewichtsfunktion (blau) und Gewichtsfunktion beim Least-Squares-Schätzer (rot).

Als Hilfestellung zur Implementierung des M-Estimators kann folgender Pseudocode verwendet werden:

Pseudocode M-Estimator

- **Initialisierung**: Schätze Geradenparameter $\hat{p}^0$ mit Least-Squares-Schätzer (LS) (Hinweis: setze dazu alle Gewichte w_i^0 des gewichteten LS-Schätzers zu eins)
  - **DO**
  - a) Berechne die Gewichte w_i^k, $i = 1 \ldots n$ mit Hilfe der Huber-Funktion (Gl. 8)
  - b) $k = k + 1$
  - c) Ermittle die Geradenparameter $\hat{p}^k$ mit Hilfe von Gleichung (7)
  - **WHILE** $(k < 20)$ **AND** $\exists \hat{p}_i : (\frac{\hat{p}_i^{k-1}}{\hat{p}_i^k} > \varepsilon_1 \vee \frac{\hat{p}_i^{k-1}}{\hat{p}_i^k} < \varepsilon_2)$
- **END**

3.2.4 Robuste Schätzung mittels RANSAC

Der M-Estimator liefert gute Ergebnisse für einen begrenzten Anteil an Ausreißern in den Beobachtungen. Wird der Anteil an Ausreißern in der Messreihe zu groß, sind auch die Ergebnisse dieses Schätzers fehlerhaft. Auch das zeigt sich im rechten Bild von Abbildung 5. Die Anzahl der Messpunkte auf den roten Ausreißer-Flecken ist ähnlich der Anzahl von Messpunkten auf der wahren roten Linie, was im Falle des M-Estimators ebenfalls in einer nicht optimalen Schätzung des Linienpolynoms resultiert. In solchen Fällen ist der RANSAC-Schätzer die bessere Wahl.

Abbildung 7: Vergleich der Polynomschätzung mit M-Estimator und RANSAC bei Ausreißern in den Beobachtungen. Orange: Beobachtungen. Gelb: Geschätztes Polynom. Links: M-Estimator. Rechts:RANSAC.

Abbildung 7 zeigt den direkten Vergleich zwischen M-Estimator und RANSAC. Offensichtlich ist die deutlich bessere Schätzung des Linienpolynoms mittels RANSAC auch bei einer großen Anzahl von Ausreißerpunkten.
RANSAC steht für "RANdom SAmple Consensus" und bedeutet etwa "Übereinstimmung mit einer zufälligen Stichprobe". Voraussetzung für eine gute Schätzung mit RANSAC ist, dass ausreichend viele Datenpunkte vorliegen. Der iterative Algorithmus setzt sich aus den folgenden drei Schritten zusammen:

1. Wähle zufällig so viele Punkte aus allen Datenpunkten $\mathbb{D}$, wie nötig sind, um eine Schätzung des angenommenen Modells (hier: das Linienpolynom) berechnen zu können. Es wird dabei erwartet, dass diese Menge frei von Ausreißern ist.

2. Schätze das Modell durch den Least-Squares-Ansatz mit den gewählten Datenpunkten.

3. Bestimme die Teilmenge $\mathbb{S} \subseteq \mathbb{D}$ (das sog. Consensus set) aller Datenpunkte, deren Abstand zur Modellschätzung kleiner als ein definierter Grenzwert ϵ ist. Enthält diese Teilmenge mehr Punkte als eine vorher definierte Anzahl N, wurde vermutlich ein gutes Modell gefunden. Die Teilmenge wird dann gespeichert.

Diese drei Schritte werden dann k-mal durchgeführt, wobei in j, $(0 \leq j \leq k)$ Schritten ein Consensus set gefunden wurde. Schließlich wird diejenige Teilmenge

$$\mathbb{S}_{max}, \;\; max = ind \max \{|\mathbb{S}_1|, |\mathbb{S}_2|, \ldots, |\mathbb{S}_j|\} \tag{9}$$

gewählt (sofern eine existiert), welche die meisten Datenpunkte enthält. Dann wird mit allen Datenpunkten dieser Teilmenge die finale Modellschätzung bestimmt.
RANSAC hängt damit im Wesentlichen von drei Parametern ab:

1. Dem maximalen Abstand ϵ eines Datenpunktes vom gewählten Modell. Im Rahmen des Praktikumsversuchs entspricht dieser Parameter also die Breite des Schlauchs um das zu zu testende Polynom. Alle Punkte innerhalb dieses Schlauchs werden dann als Unterstützer der Hypothese gewertet und dem consensus set hinzugefügt.

2. Die Mindestgröße N des consensus sets, welche angibt wie viele Unterstützer eine Hypothese mindestens benötigt um in die Auswahl möglicher Lösungskandidaten aufgenommen zu werden.

3. Die Anzahl der Iterationen k. Dieser Parameter sollte so gewählt werden, dass ein guter Ausgleich zwischen Laufzeit und Zuverlässigkeit des RANSAC-Algorithmus erreicht wird. Eine kleinere Anzahl von Iterationen führt bspw. zu einer schnelleren Terminierung, erhöht jedoch auch die Wahrscheinlichkeit kein Samplemenge getestet zu haben, welche nur aus Inliern besteht. Für eine geeignete Anzahl von Iterationen kann die Faustformel

$$k = \frac{\log(1-p)}{\log(1-(1-r)^s)} \tag{10}$$

verwendet werden. Hierbei ist p die Wahrscheinlichkeit mindestens einmal eine Samplemenge zu ziehen, die nur aus Inliern besteht, r die mittlere Inlierrate in allen Datennpunkten und s die Anzahl von Datennpunkten die mindestens notwendig ist um das Modell endeutig bestimmen zu können.

3.3 Bestimmung der Sollwerte des Reglers

Für die Regelung zum Folgen der Linie bestehen durch die Freiheitsgrade des Quadrokopters mehrere Möglichkeiten. Zwecks Einfachheit werden hier die Gier- und Seitwärtsbewegung unabhängig voneinander geregelt. Wegen der für diesen Versuch vorzubereitenden Aufgaben verzichtet die folgende Beschreibung auf explizite Formeln.

Gemeinsamer Ausgangspunkt ist die Spalten-Koordinate u als Funktionswert des Linienpolynoms in einer festgelegten Bildzeile v. Letztere wird, wie in Abbildung 8 dargestellt, um die Vorschaulänge l oberhalb der Bildmitte C bzw. vor dem Quadrokopter platziert. Der Abstand zwischen u und der mittleren Bildspalte ergibt die gesehene Ablage Δu. Da der Quadrokopter nicht immer exakt waagrecht fliegt, kann sich der Fußpunkt F senkrecht unter ihm durchaus außerhalb der Bildmitte C befinden. Daher kommt zur gesehenen Ablage Δu auch eine rollwinkel-induzierten Ablage Δu_F hinzu.

Aus der Summe beider Ablagen sowie der Vorschaulänge l kann der Deichselwinkel δ trigonometrisch berechnet und als Fehler an den Gierwinkel-Regler übergeben werden. Der so entstandene Deichselregler wird etwa in der Vorlesung *Verhaltensgenerierung für Fahrzeuge* ausführlich behandelt. Da die Vorschaulänge l durch den oberen Bildrand begrenzt ist, erzeugt ein reiner Deichselregler beim vorliegenden System ein recht unruhiges Flugverhalten.

Hier bietet es sich aufgrund des geschätzten Linienmodells in Form eines kubischen Polynoms an, auch dessen Ableitung auszuwerten. Aus dieser kann der Winkel τ zwischen einer Tangente an die Linie und der Längsachse des Quadrokopters bestimmt und auf den Sollwert null eingeregelt werden. Da der Quadrokopter so jedoch nur parallel zur Linie, nicht aber genau darüber fliegen würde, wird der letztendlich einzuregelnde Gierwinkelfehler $\Delta\psi$ aus einer gewichteten Summe von δ und τ gebildet:

$$\Delta\psi = w_1 \cdot \delta + w_2 \cdot \tau \qquad \text{mit} \qquad \sum_i w_i = 1$$

Die Regelgröße des Seitwärts-Freiheitsgrades ist die in Abbildung 9 dargestellte Ablage ΔL zwischen roter Linie und dem Fußpunkt senkrecht unter dem Quadrokopter. Diese kann trigonometrisch mit Hilfe der per Ultraschall gemessenen Höhe h berechnet werden. Der ebenfalls benötigte Winkel ϱ_{wl} zwischen dem Sichtstrahl zur Linie ("l") und einer Senkrechten in

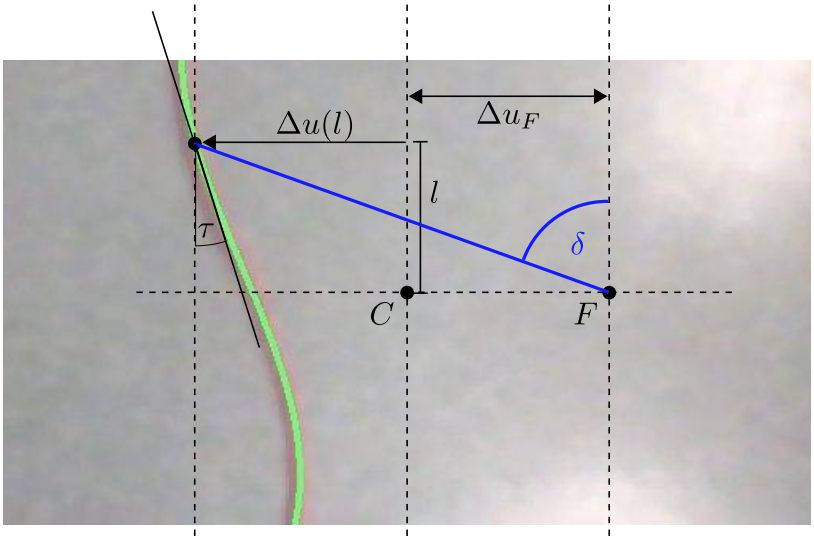


Abbildung 8: Geometrischer Zusammenhang zwischen erkannter Linie, gesehener und rollwinkel-induzierter Ablage Δu bzw. Δu_F, Vorschaulänge l, Tangentenwinkel τ und Deichselwinkel δ.

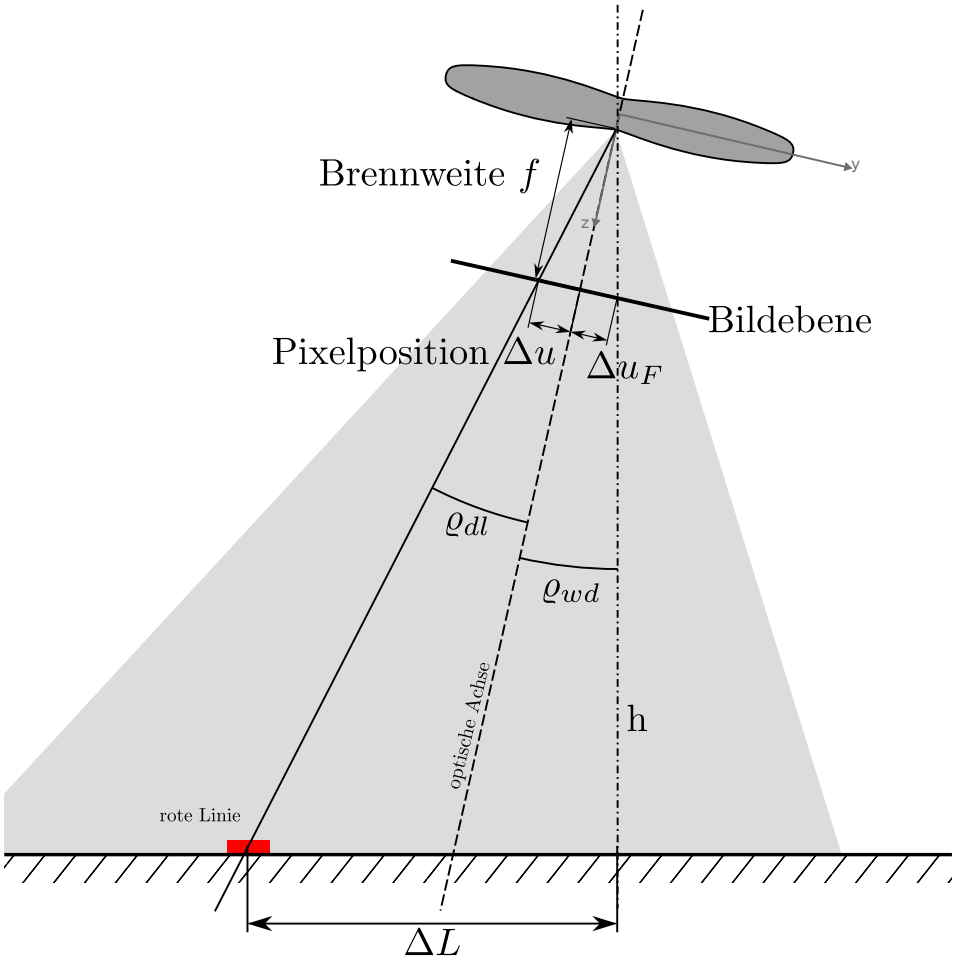


Abbildung 9: Rückansicht des Quadrokopters mit Bildebene, gesehener Ablage Δu, Rollwinkel ϱ_{wd} und daraus induzierter Ablage Δu_F, sowie der gesamten lateralen Ablage ΔL.

Welt-Koordinaten ("w") entspricht der Summe von ϱ_{dl} zwischen Sichtstrahl und Senkrechter in Quadrokopter-Koordinaten ("d") und ϱ_{wd} zwischen den jeweiligen Senkrechten von Quadrokopter- und Weltkoordinaten, also dem Rollwinkel.

Im nun folgenden Praxisteil haben Sie Gelegenheit, die verschiedenen Regelungsansätze zu untersuchen.

4 Durchführung des Versuchs

Hinweis: Während des Flugs muss eine Person zur Sicherheit immer das Gamepad zur Steuerung des Quadrokopters bereithalten. Sollte es zu einem unerwünschten Flugverhalten kommen, kann so jederzeit eine Landung ausgelöst oder die Steuerung übernommen werden. **Bleiben Sie beim Flug in sicherer Entfernung zum Quadrokopter und fassen Sie insbesondere nicht in dessen Rotoren.**

In Ergänzung zu dieser Durchführungsanweisung ist der Quellcode der zu bearbeitenden Programme kommentiert. Es handelt sich hierbei um einen Lückentext, in welchem alle auszufüllenden Stellen mit `// TODO` gekennzeichnet und mit Hinweisen versehen sind. *Eclipse* hebt diese Markierungen am rechten Fensterrand hervor.

4.1 Inbetriebnahme

Heben Sie die Styropor-Abdeckung des Quadrokopters an der Rückseite an, setzen Sie einen Akku ein und schließen Sie diesen an. Der Quadrokopter baut nun ein eigenes WiFi-Netzwerk auf, zu dem sich der Laptop nach kurzer Zeit automatisch verbindet.

Starten Sie die folgenden Programme in je einem eigenen Terminal-Fenster, welches Sie mit Strg + Alt + T öffnen können:

- Starten Sie die zentrale *ROS*-Datenbank wie im Anhang beschrieben.
- Benutzen Sie die ebenfalls im Anhang genannten *ROS*-Werkzeuge, um sich das Bild der nach unten gerichteten Kamera und verschiedene Messwerte anzeigen zu lassen.
- Machen Sie sich sorgfältig mit der Tastenbelegung des Gamepads im Anhang vertraut. Starten Sie dann das Programm zur manuellen Steuerung des Quadrokopters. Üben Sie das Starten, Landen und Ansteuern einer Position im Raum.
  `cd ~/Desktop/code/arDroneManual && bin/arDroneManual`
  Absichtliche gefährliche Flugmanöver haben einen Ausschluss vom Versuch zur Folge.

4.2 Bildvorverarbeitung und Schätzung der Linie

Die Linien-Erkennung soll zunächst noch nicht im echten Flugbetrieb, sondern off-line mit Einzelbildern fertiggestellt und getestet werden. Tragen Sie hierfür den Quadrokopter von Hand entlang der Linie, ohne dass er dabei aus der Styropor-Abdeckung fällt, und zeichnen Sie währenddessen verschiedene Bilder auf. Nutzen Sie dazu das *ROS*-Tool `image_view` wie in Abschnitt 6.2 beschrieben. Starten Sie es vom Verzeichnis `~/Desktop/code/offLineDetection/data/` aus, um die Bilder per Rechtsklick direkt dort zu speichern.

Öffnen Sie anschließend in *Eclipse* das Projekt `offLineDetection`. In `main.cpp` können Sie festlegen, welches Bild geladen wird. Alles weitere findet in der Datei `lineDetection.cpp` statt:

- Konvertieren Sie das Eingangsbild `image` mit Hilfe einer geeigneten *OpenCV*-Funktion vom BGR- in den HSV-Farbraum.

- Über `linenessMode` kann festgelegt werden, ob die "Linienartigkeit" jedes Pixels durch einfache Schwellwerte oder durch die o. g. Linienevidenz bestimmt wird.

  Wählen Sie zunächst `linenessMode = linenessModeThreshold` und betrachten Sie den entsprechenden Abschnitt `case linenessModeThreshold :`. Tragen Sie hier die fehlenden Schwellwerte ein. Erzeugen Sie anschließend mit Hilfe eines Gauß-Filters eindeutige Maxima im `lineness`-Bild.

- Starten Sie nun bereits das Progamm `offLineDetectionDbg`. In dieser Debug-Version werden mehr Zwischenergebnisse angezeigt und die Verarbeitung jeweils erst nach einem Tastendruck fortgesetzt. Passen Sie die zuvor eingestellten Parameter an, bis Sie mit den Resultaten mehrerer Testbilder zufrieden sind.

- Setzen Sie nun `linenessMode = linenessModeFuzzy` und betrachten Sie den Abschnitt `case linenessModeFuzzy :`. Vervollständigen Sie die Berechnung der Bewertungsfunktionen. Führen Sie das Programm nochmals aus und optimieren Sie erneut die Parameter auf mehreren Bildern.

- Scrollen Sie weiter zum Kommentar `// 2) find most likely line point of each image row`. Dort soll in jeder Bildzeile der Pixel mit der maximalen "Linienartigkeit" gefunden werden. Vervollständigen Sie den Lückentext mit Hilfe des Anhangs. Starten Sie das Programm und überprüfen Sie die dargestellten Ergebnisse.

- Scrollen Sie weiter zum Kommentar `// 3) find coefficients of polynomial line model`. Stellen Sie sicher, dass `estimatorLeastSquares` ausgewählt ist.

- Legen die Beobachtungsmatrix `H` mit der richtigen Größe an. Die folgende `for`-Schleife iteriert über alle zuvor gefundenen Punkte. Nutzen Sie die Koordinaten des aktuellen Punktes `p.u` und `p.v`, um den (bis dahin zufälligen) Inhalt von Beobachtungsvektor `y` und Designmatrix `H` mit den richtigen Werten zu überschreiben.

- Suchen Sie zum Abschluss den Abschnitt `case estimatorLeastSquares :`. Implementieren Sie dort die Formel des Least-Squares-Schätzers und weisen Sie das Ergebnis der Variablen p zu. Überprüfen Sie, ob die nun vollständige Linienerkennung auf allen Bildern funktioniert. Im Falle der korrekten Implementierung werden Sie feststellen dass ein gutartiges Polynom im Falle von ungestörten Bildern und ein falsches Polynom im Falle von Bildern mit Störeffekten geschätzt wird.

4.3 Robuste Schätzung der Linie

Im Folgenden sollen die robusten Schätzer in Betrieb genommen und getestet werden. Beginnen Sie mit dem M-Estimator. Scrollen Sie dazu zurück zum Kommentar `// three estimation modes available` und wählen Sie die Option `estimatorM` aus. Scrollen Sie dann wieder vor zum Kommentar

`// mode B: M-estimator`. Hier beginnt die Bearbeitung des Programmcodes, welcher den M-Estimator realisiert.

- Gehen Sie weiter an die Stelle
  `// maximum acceptable error in [px] for "Huber" weight function`
  und legen Sie die Breite des Plateaus der Hubergewichtsfunktion fest. Überlegen Sie sich dazu einen sinnvollen Wert. Die Einheit dieses Wertes ist Pixel.
- Scrollen Sie weiter und geben Sie die korrekte Dimension des Residuenvektors an.
- Geben Sie die danach die korrekte Formel für die Berechnung des Residuenvektors an.
- Vervollständigen Sie an der Stelle `// implement Huber weight function...` die Realisierung der Hubergewichtsfunktion.
- Berechnen Sie die gewichtete Least-Squares-Schätzung an der Stelle `// estimate new coefficients ....` Kompilieren Sie schließlich das Programm neu und überprüfen Sie, ob die robuste Schätzung mittels M-Estimator funktioniert. Analysieren Sie die Schätzergebnisse anhand der angezeigten Bilder.

Als Nächstes soll nun der RANSAC verwendet werden. Scrollen Sie dazu zurück zum Kommentar `// three estimation modes available` und wählen Sie die Option `estimatorRANSAC` aus. Scrollen Sie dann zum Beginn der RANSAC-Implementierung und beginnen Sie dort mit der Vervollständigung.

- An der Stelle `// maximum acceptable error ...` soll ein geeignetes Abstandskriterium zum Modellpolynom für den RANSAC eingesetzt werden.
- Bestimmen Sie als Nächstes die Anzahl der Iterationen für den RANSAC. Verwenden Sie dazu die im Grundlagenkapitel angegebene Gleichung.
- Scrollen Sie weiter zum Kommentar `// find absolute error between ....`
  An dieser Stelle befinden Sie sich in einer Schleife, welche das consensus set für die aktuelle Hypothese bestimmt. Implementieren Sie hier die Berechnung des Abstands des Datenpunktes zur Hypothese.
- Geben Sie danach die Bedingung an, welche das bisher beste gefundene consensus set gegen das aktuelle vergleicht. Die Bedingung muss so definiert werden, dass im Falle einer Verbesserung das aktuelle consensus set durch das Neue ersetzt wird. Anschließend testen Sie ihre RANSAC-Implementierung und analysieren sie die Ergebnisbilder.

4.4 Nachfliegen der Linie

Wechseln Sie nun wieder in das Projekt `arDroneAutonomous` und aktivieren Sie in `main.cpp` nun `DroneAppFollowLine`. Öffnen Sie dazu passend `appFollowLine.cpp`.

Starten Sie das Programm und verifizieren Sie zunächst, dass ihre Linienerkennung und -schätzung auch im Live-Betrieb funktioniert, indem Sie den Quadrokopter entlang der Linie tragen.

- Lassen Sie sich die o. g. Zwischenergebnisse auf der Kommandozeile ausgeben und überprüfen Sie deren Größenordnung und Vorzeichen nochmals bei von Hand getragenem Quadrokopter. Ein positiver Fehler steuert den Quadrokopter nach links.
- Bei der Regelung ergibt sich der Soll-Gierwinkel aus einer gewichteten Summe von Deichsel- und Tangentenwinkel. Versuchen Sie zunächst im Versuchsraum automatisch einer geraden Linie nachzufliegen. Wechseln Sie jeweils am Ende des Raums in den manuellen Steuerungsmodus und wenden Sie den Quadrokopter.
- Zum Ende des Versuchs bringen Sie die Quadrokopter, Laptops und das rote Seil in die Maschinenhalle des Instituts. Dort kann der Linienverlauf freier und als Rundkurs gewählt werden. Knicke von mehr als etwa $60\,^\circ$ sind dennoch zu vermeiden.

5 Vorbereitende Aufgaben

Hinweise:

- Diese Aufgaben müssen bereits im Vorfeld des Versuchs bearbeitet werden.
- Zu Beginn des Kollquiums müssen die bearbeiteten Aufgaben dem Betreuer gezeigt werden.
- Manche Aufgaben sollen mit *Pseudocode* bearbeitet werden. Das muss kein lauffähiger Code einer bestimmten Sprache sein, sondern die Grundprinzipien sollen klar ersichtlich werden. Beschränken Sie sich auf einfache Zuweisungen, Rechenoperatoren und Schleifen. Ein Anhaltspunkt liefert der Anhang.

1. **HSV-Farbraum**
   Welche HSV-Farbwerte hat die Farbe Rot?

   Welche Schwellwerte würden Sie für die Binarisierung einer roten Linie anwenden?

2. **Binarisierung**
   Pseudocode: Die HSV-Farbwerte eines Pixels stehen in den Variablen `H`, `S`, `V`. Die Schwellwerte stehen in den Variablen `H_min`, `H_max`, `S_min` und `V_min`. Setzen Sie die Variable `ist_Linie` auf 1 wenn der Pixel zu der Linie gehört!

3. **Linienevidenz**
   Wie müssen die Gewichtungsfunktionen w_K aussehen, damit ein Maximum möglichst deutlich hervortritt?

   Pseudocode: Der Sättigungswert eines Pixels steht in der Variablen `S`. Die Grenzen des Übergangsbereichs haben Sie in `S_min` und `S_max` gespeichert. Bestimmen Sie die zur Sättigung gehörende Bewertung dieses Pixels! (siehe auch Abbildung 3)

6 Anhang

6.1 Hilfreiche C++ Kenntnisse

6.1.1 Standardbibliothek

Die Standardbibliothek ist bei jedem C++ Compiler enthalten und unabhängig vom Betriebssystem oder sonstigen Randbedingungen verfügbar: `de.cppreference.com`.

- Datentyp für ganze Zahlen, etwa Bildzeilen/-spalten oder Pixel-Koordinaten: `int`
- Datentyp für Fließkomma-Zahlen mit doppelter Genauigkeit (64 Bit), etwa Polynom-Koeffizienten: `double`
- Variablen anlegen bzw. initialisieren: `double x ; double y = 1.23 ;`
- Rechnen mit bestehenden Variablen ohne Angabe von Datentypen: `x = 4.0/y ;`
- logische Operationen: gleich `x == y`, ungleich `x != y`, kleiner `x < y`, größergleich `x >= y`, nicht `!( x < 0 )`, und `( x > 0 ) && ( x < 10 )`, oder `( y < 0 ) || ( x < 0 )`
- Ausgabe von Text und Variablen auf der Kommandozeile:
  `std::cout << "x-Wert = " << x << std::endl ;`
- bedingte Ausführung von Programmcode:
  `if( x > 5 ) { y = x - 5 ; } else { y = 0 ; }`
- Schleifen, etwa um den gleichen Code nacheinander auf jede Bildzeile anzuwenden:
  `for( int row = 0 ; row < image.rows ; ++row ) { ... }`
- den gleichen Code einmal ausführen und bis zu einem Abbruchkriterium wiederholen:
  `do{ x = 2*x ; } while( x < 100 ) ;`
- natürlicher Logarithmus: `double y = std::log( x )`
- Potenzieren, etwa x^3: `std::pow( x, 3 )` oder einfach `x * x * x`, jedoch nicht (!) `x ^ 3`
- Trigonometrie: `std::sin( x )`, `std::cos( y * CV_PI/180.0 )`, `std::tan( CV_PI/4 )`
- Arkustangens ohne Mehrdeutigkeit bzgl. der Vorzeichen von Gegen- und Ankathete:
  `std::atan2( gegenkathete, ankathete )`
- Vektor als Container für eine variable Menge von Werten:
  `std::vector< int > values ;`
- Element an einen Vektor anhängen: `values.push_back( 7 ) ;`
- Anzahl der Elemente in einem Vektor: `values.size()`

- Zugriff auf drittes Vektor-Element (gezählt ab Index 0): `values[ 2 ]`
- Tupel aus Wert und Position eines Vektor-Elements (sog. Iterator):
  `std::vector< int >::Iterator iter ;`
- Anfang/Ende eines Vektors: `iter = values.begin/end() ;`
- größtes Element eines Vektors:
  `iter = std::max_element( values.begin(), values.end() ) ;`
- Abstand zweier Vektor-Elemente:
  `std::distance( values.begin(), iter ) ;`

6.1.2 OpenCV

Open Computer Vision (OpenCV) ist eine freie Softwarebibliothek mit modernen Algorithmen für die Bildverarbeitung und maschinelles Sehen: `docs.opencv.org`. Sie unterstützt neben C++ auch die Programmiersprache *Python*. Ihre Stärke liegt vor allem in der hohen Verarbeitungsgeschwindigkeit und dem benutzerfreundlichen Interface.

- Bild oder Matrix mit variablem Datentyp: `cv::Mat image ;`
- Datentyp dann bei Zugriff auf einzelne Pixel erforderlich:
  `HSV8 pixel = image.at< HSV8 >( v, u )`
- auf Festplatte gespeichertes Bild einlesen:
  `image = cv::imread( "data/frame0000.jpg" ) ;`
- Anzahl von Zeilen bzw. Spalten: `image.rows/cols`
- eine Zeile extrahieren: `std::vector< uint8_t > = image.row( v ) ;`
- Bild vom BGR- in den HSV-Farbraum konvertieren:
  `cv::cvtColor( bgr, hsv, CV_BGR2HSV ) ;`
- Bild mit einem Gauß-Filter (Größe 25x9 Pixel) glätten:
  `cv::GaussianBlur( raw, smooth, cv::size( 25, 9 ), 0 ) ;`
- Bild oder Matrix mit festem Datentyp und Größe 10x4:
  `cv::Mat_< double > matrix( 10, 4 ) ;`
- einfacherer Zugriff auf Pixel z. B. in zweiter Zeile und vierter Spalte (jeweils gezählt ab Index 0): `double pixel = matrix( 1, 3 ) ;`
- Matrix transponieren: `matrix.t()`
- Matrix invertieren: `matrix.inv()`
- Matrix-Inhalt kopieren: `source.copyTo( target ) ;`

6.1.3 Versuchsspezifische Erweiterungen

Zur Vereinfachung des praktischen Programmierteils dieses Versuchs stehen spezielle Datentypen für verschiedene Zwecke zur Verfügung:

- Pixel-Koordinaten: initialisieren `Point p( u, v ) ;` bzw. zugreifen `p.u/v`
- Koeffizienten eines Polynoms dritter Ordnung:
  anlegen und initialisieren `Coeffs coeffs ; coeffs[ 0/1/2/3 ] = 1.0 ;`
  Gleichung $u = c_0v^3+c_1v^2+c_2v+c_3$ auswerten `double u = coeffs.u( v ) ;`
- PID-Regler:
  Parameter setzen `controller.kP/kI/kD = 0.8 * kPCrit ;`
  Stellgröße aus Regelfehler bestimmen (optional mit Ableitung des Fehlers, um rauschbehaftete numerische Ableitung zu vermeiden)
  `commands.vertical = controller( error, errorDeriv ) ;`

6.2 Robot Operating System (ROS)

ROS bietet nicht nur eine zentrale Datenbank, über die verschiedene Programme Nachrichten austauschen können. Es enthält auch einige Tools, die direkt von der Kommandozeile aus genutzt werden können.

- *ROS* starten: `cd ~/Desktop/code && ./ros.sh`
  (Das Skript `ros.sh` ruft den Befehl `roslaunch` mit einer Konfigurationsdatei aus, die für jeden der beiden Quadrokopter spezifische Werte definiert.)
- Video-Stream der aktuell ausgewählten Kamera anzeigen:
  `rosrun image_view image_view image:=/ardrone/image_raw`
  Ein Rechtsklick auf das Bildfenster speichert das Bild im aktuellen Verzeichnis unter einer fortlaufenden Nummer.
- zwischen nach vorne und nach unten gerichteter Kamera umschalten:
  `rosservice call /ardrone/togglecam`
- alle Messwerte des Quadrokopters auf Kommandozeile schreiben:
  `rostopic echo /ardrone/navdata`
- zeitlichen Verlauf eines Messwerts plotten, z. B. des Gierwinkels:
  `rqt_plot /ardrone/navdata/rotZ`
  Der Pause-Button stoppt die laufende Aktualisierung des Plots. Bei gedrückter linker Maustaste kann der betrachtete Ausschnitt verschoben, bei gedrückter rechter Maustaste horizontal und vertikal gezoomt werden.

6.3 Gamepadbelegung