

# Mobile Roboterplattform

Benjamin Ranft, Jacques Mayländer und Markus Schreiber

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



## Lernziele

1. Regler-Einstellung nach Ziegler-Nichols
2. Erkennung einer roten Linie in Kamerabildern
3. Least-Squares-Schätzung von Modell-Parametern
4. Flugregelung eines Quadropters
5. Programmierung in C++ mit *ROS* und *OpenCV*

Unter [www.mrt.kit.edu/software/](http://www.mrt.kit.edu/software/) sind sowohl dieses Skript, lizenziert unter *Creative Commons Attribution-ShareAlike 4.0 International*<sup>1</sup>, als auch unsere verwendete Software, lizenziert unter *GNU General Public License*<sup>2</sup>, zum Download verfügbar.

<sup>1</sup><http://creativecommons.org/licenses/by-sa/4.0/>

<sup>2</sup><http://www.gnu.org/licenses/>

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Versuchsaufbau . . . . .	3
1.2	Softwarestruktur . . . . .	4
<b>2</b>	<b>Zielsetzung</b>	<b>4</b>
<b>3</b>	<b>Grundlagen</b>	<b>5</b>
3.1	Regelung des Quadropters . . . . .	5
3.1.1	Reglerparametrierung nach Ziegler-Nichols . . . . .	5
3.2	Erkennung der Linie . . . . .	6
3.2.1	Bildvorverarbeitung . . . . .	7
3.2.2	Schätzung der Linie . . . . .	8
3.3	Bestimmung der Sollwerte des Reglers . . . . .	9
<b>4</b>	<b>Durchführung des Versuchs</b>	<b>12</b>
4.1	Inbetriebnahme . . . . .	12
4.2	Regler-Parametrierung nach Ziegler-Nichols . . . . .	12
4.2.1	Regelung der Flughöhe . . . . .	13
4.2.2	Regelung des Gierwinkels . . . . .	14
4.3	Bildvorverarbeitung und Schätzung der Linie . . . . .	14
4.4	Nachfliegen der Linie . . . . .	15
<b>5</b>	<b>Vorbereitende Aufgaben</b>	<b>17</b>
<b>6</b>	<b>Anhang</b>	<b>19</b>
6.1	Hilfreiche C++ Kenntnisse . . . . .	19
6.1.1	Standardbibliothek . . . . .	19
6.1.2	OpenCV . . . . .	20
6.1.3	Versuchsspezifische Erweiterungen . . . . .	21
6.2	Robot Operating System (ROS) . . . . .	21
6.3	Gamepadbelegung . . . . .	22

# 1 Einleitung

In den letzten Jahren wurden Quadrocopter 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.

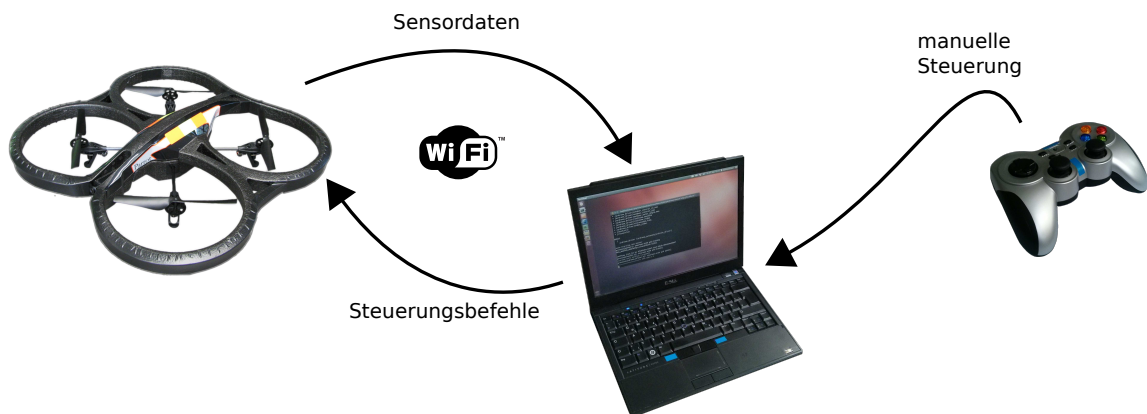
Leider gehen in diesen gekapselten Systemen die technischen Funktionalitäten unter. Man sieht als "Anwender" nicht, wie und warum etwas funktioniert. In diesem Praktikumsversuch soll zunächst die Funktionsweise eines Quadrocopters veranschaulicht werden. Beim nachfolgenden Parametrieren einer Höhen- 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 Quadrocopter-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

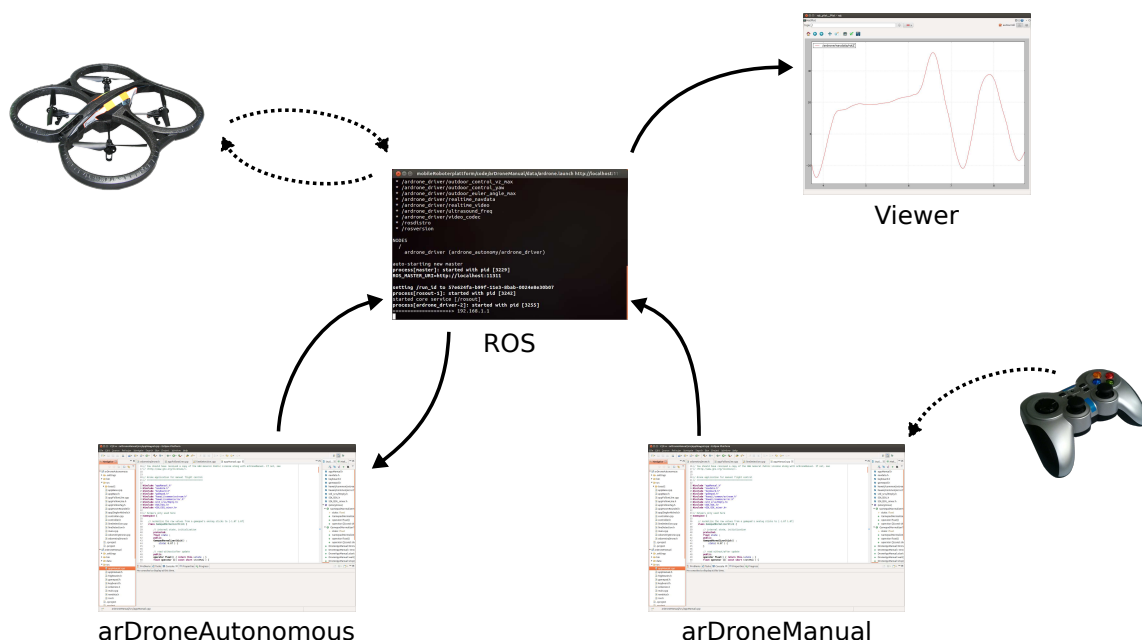
Der Quadrocopter selbst ist ein geschlossenes System, auf dem nicht ohne Weiteres eigener Programmcode ausgeführt werden kann. Stattdessen überträgt er Messwerte und Bilddaten über WLAN an einen Rechner. Dieser verarbeitet die Daten und sendet Steuerungsbefehle an den Quadrocopter zurück. Befehle der 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](http://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 Quadrocopter 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 Quadrocopter 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.

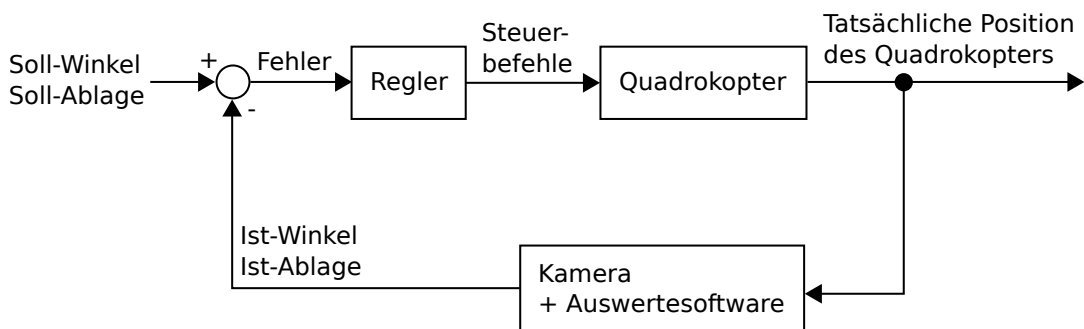
## 2 Zielsetzung

Ziel des Versuchs ist das autonome Nachfliegen einer roten Linie auf dem Boden, welche durch ein flexibles Seil dargestellt wird. Als notwendige Vorarbeit müssen zunächst die PID-Regler für Höhe und Gierwinkel des Systems mit der Methode von Ziegler-Nichols eingestellt werden. Ebenso ist die Erkennung der Linie auf Bildern der nach unten gerichteten Kamera zu programmieren.

## 3 Grundlagen

### 3.1 Regelung des Quadropters

Der Quadroptter 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.



#### 3.1.1 Reglerparametrierung nach Ziegler-Nichols

Ein PID-Regler besitzt die Übertragungsfunktion

$$G_R(s) = k_P + \frac{k_I}{s} + k_D s \quad (1)$$

bzw. im Zeitbereich die Differentialgleichung

$$u(t) = k_P e(t) + k_I \int_0^\infty e(\tau) d\tau + k_D \dot{e}(t) \quad (2)$$

Die Ziegler-Nichols-Methode ist ein heuristisches Verfahren zur Bestimmung von dessen Parametern  $k_P$ ,  $k_I$  und  $k_D$ . Hierbei ist wie folgt vorzugehen:

1. Der Regler wird durch Setzen von  $k_I = k_D = 0$  auf einen reinen P-Regler reduziert.  $k_P$  wird zu Beginn sehr klein gewählt.
2. Die Verstärkung  $k_P$  wird in kleinen Schritten erhöht und jeweils das Verhalten beim Einregeln beobachtet. Zu Beginn wird der Sollwert nur langsam erreicht. Bei höheren Werten tritt ein Überschwingen auf, bei einer weiteren Erhöhung schließlich eine abklingende Schwingung. Sobald diese in eine Dauer-Schwingung übergeht ist die kritische Verstärkung  $k_{Pk}$  erreicht. Bei noch höheren Werten von  $k_P$  würde das System instabil.

3. Aus dieser kritischen Verstärkung  $k_{Pk}$  und der Periodendauer der zugehörigen Dauerschwingung  $T_k$  können die Reglerparameter bestimmt werden. Hierfür stehen mehrere heuristische Formeln zur Auswahl, welche Tabelle 1 zusammenfasst. Ein reiner PI-Regler ist etwa für sehr rauschbehaftete Regelgrößen sinnvoll; ein PD-Regler reicht dagegen aus, wenn bereits die Strecke integrierendes Verhalten aufweist. Die Heuristiken für PID-Regler sind jeweils für ein schnelles Ausregeln von Störungen oder ein geringes Überschwingen angepasst.

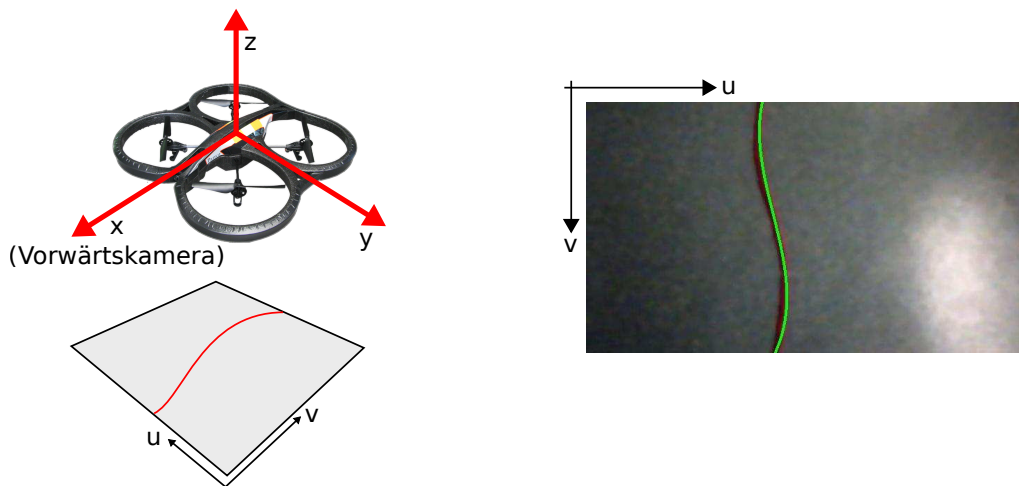
Eine ausführlichere Beschreibung des Verfahrens nach Ziegler-Nichols finden Sie im Skript zur Vorlesung *Grundlagen der Mess- und Regelungstechnik*.

	$k_P$	$k_I$	$k_D$
P-Regler	$0,50 \cdot k_{Pk}$		
PI-Regler	$0,45 \cdot k_{Pk}$	$0,54 \cdot k_{Pk}/T_k$	
PD-Regler	$0,80 \cdot k_{Pk}$		$0,100 \cdot k_{Pk} \cdot T_k$
PID-Regler	$0,60 \cdot k_{Pk}$	$1,20 \cdot k_{Pk}/T_k$	$0,072 \cdot k_{Pk} \cdot T_k$
wenig Überschwingen	$0,33 \cdot k_{Pk}$	$0,67 \cdot k_{Pk}/T_k$	$0,111 \cdot k_{Pk} \cdot T_k$
kein Überschwingen	$0,20 \cdot k_{Pk}$	$0,40 \cdot k_{Pk}/T_k$	$0,067 \cdot k_{Pk} \cdot T_k$
Pessen Integral Rule	$0,70 \cdot k_{Pk}$	$0,28 \cdot k_{Pk}/T_k$	$0,105 \cdot k_{Pk} \cdot T_k$

**Tabelle 1:** Verschiedene Heuristiken zur Regler-Parametrierung nach Ziegler-Nichols.

### 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).



**Abbildung 1:** Koordinatensystem des Quadrokopters und des Kamerabildes

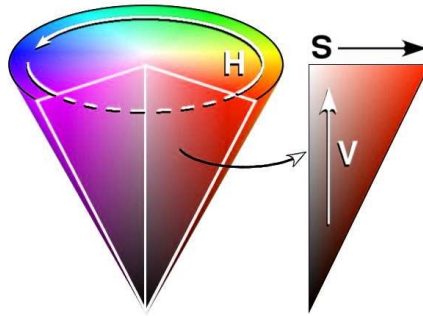


Abbildung 2: Veranschaulichung des HSV-Farbraums<sup>3</sup>

### 3.2.1 Bildvorverarbeitung

Nach der Bildvorverarbeitung soll die Linie deutlich hervortreten und Störungen im Bild möglichst verschwinden. Hierzu wird für jedes Pixel unabhängig 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  $H \in [0^\circ, 360^\circ]$  (*hue*), der Sättigung  $S \in [0, 1]$  (*saturation*) und dem Hellwert  $V \in [0, 1]$  (*value*). Jede Farbangabe mit  $V = 0$  entspricht dabei Schwarz.

**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:

$$(H_{min} < H_{uv} < H_{max}) \quad \wedge \quad (S_{min} < S_{uv}) \quad \wedge \quad (V_{min} < V_{uv}) \quad (3)$$

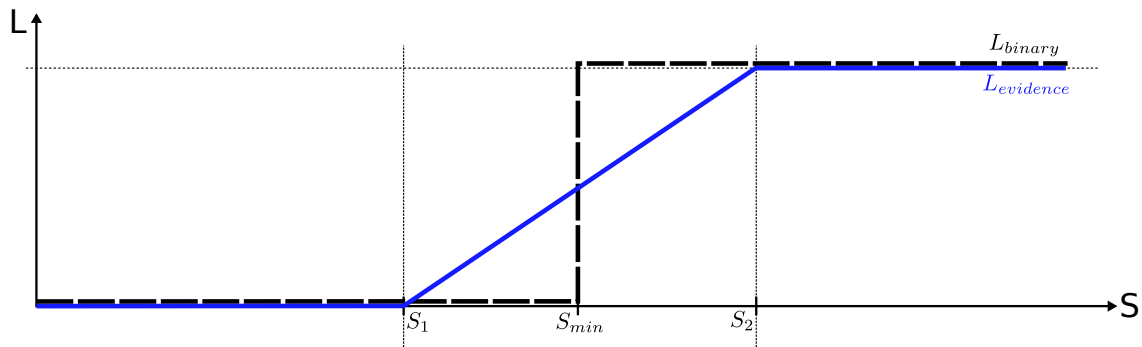
Für jedes Pixel  $(u, v)$  kann entsprechend als Ergebniswert definiert werden:

$$L_{uv} = \sigma(H_{uv} - H_{min}) \cdot \sigma(H_{max} - H_{uv}) \cdot \sigma(S_{uv} - S_{min}) \cdot \sigma(V_{uv} - V_{min}) \quad (4)$$

**Linienvidenz** 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:

$$L_{ges} = L_H \cdot L_S \cdot L_V \quad (5)$$

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



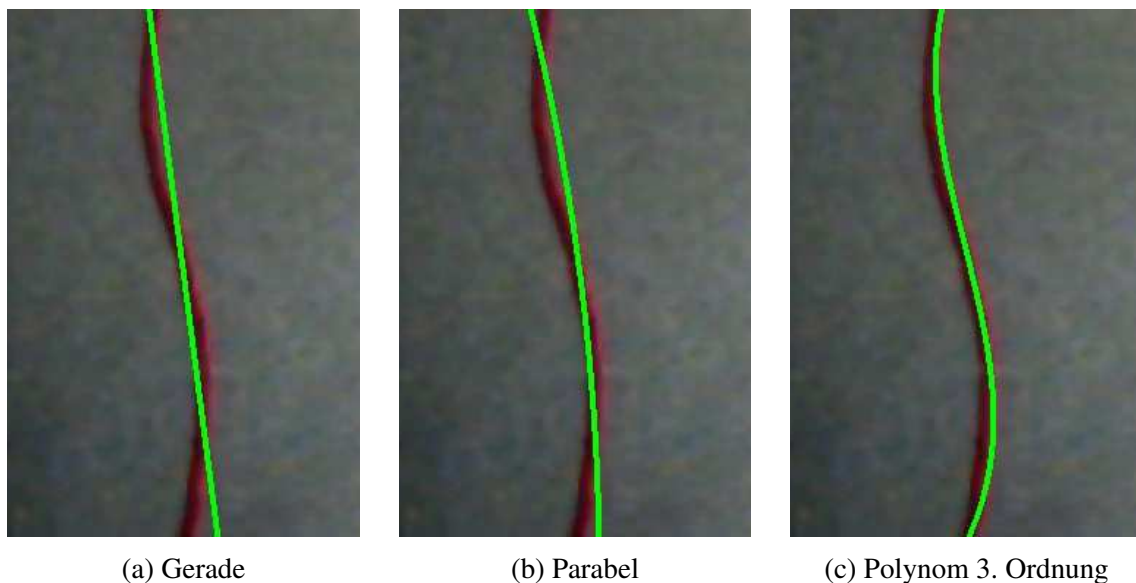
**Abbildung 3:** Vergleich der Bewertungsfunktion für die Sättigung  $S$  eines Pixels bei Binarisierung bzw. mit einem Übergangsbereich.

### 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 Quadropter 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:

$$f(v) = av^3 + bv^2 + cv + d \quad (6)$$

**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.



**Abbildung 4:** vom Quadropter aus aufgenommene Bilder mit geschätzter Linie



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 Linienvidenz-Bild bei geeigneten Parametern  $K_1$  und  $K_2$  auch ohne Filterung eindeutige Maxima hervor.

Aus Zeile  $v_i$  und geschätzter 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:

$$\begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_n \end{pmatrix} + \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \vdots \\ \hat{e}_n \end{pmatrix} = \begin{bmatrix} v_1^3 & v_1^2 & v_1 & 1 \\ v_2^3 & v_2^2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ v_n^3 & v_n^2 & v_n & 1 \end{bmatrix} \begin{pmatrix} \hat{a} \\ \hat{b} \\ \hat{c} \\ \hat{d} \end{pmatrix} \iff \hat{\mathbf{u}} + \hat{\mathbf{e}} = \mathbf{H}\hat{\mathbf{p}} \quad (7)$$

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:

$$\hat{\mathbf{p}} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \hat{\mathbf{u}} \quad (8)$$

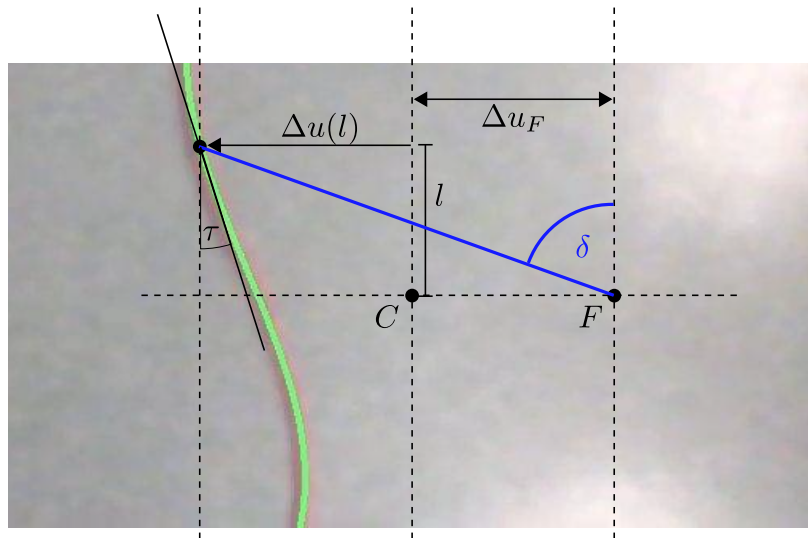
Eine Herleitung dieser Formel finden Sie ebenfalls im Skript zur Vorlesung *Grundlagen der Mess- und Regelungstechnik*.

### 3.3 Bestimmung der Sollwerte des Reglers

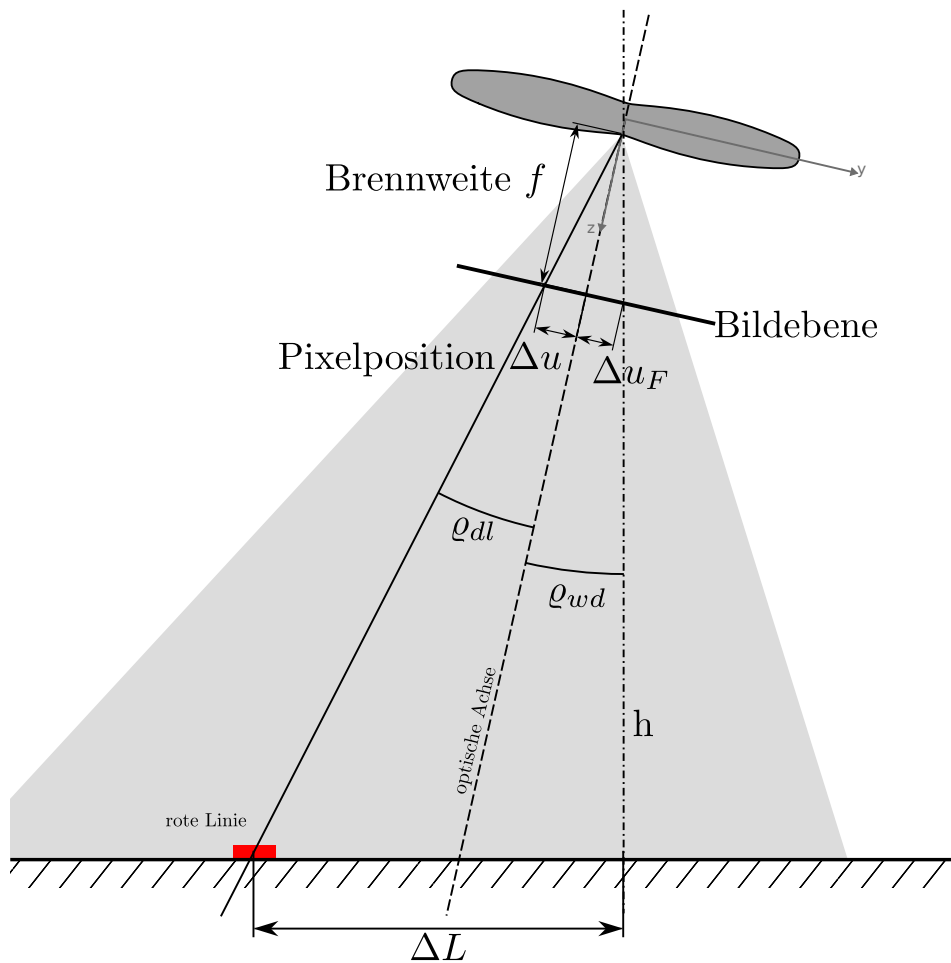
Für die Regelung zum Folgen der Linie bestehen durch die Freiheitsgrade des Quadropters mehrere Möglichkeiten. Zwecks Einfachheit werden hier die Gier- und Seitwärtsbewegung unabhängig voneinander geregelt. Wegen der für diesen Versuch vorzubereitenden Aufgaben verzichtet die folgende Beschreibung auf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

Gemeinsamer Ausgangspunkt ist die Spalten-Koordinate  $u$  als Funktionswert des Linienpolynoms in einer festgelegten Bildzeile  $v$ . Letztere wird, wie in Abbildung 5 dargestellt, um die Vorschaulänge  $l$  oberhalb der Bildmitte  $C$  bzw. vor dem Quadropter platziert. Der Abstand zwischen  $u$  und der mittleren Bildspalte ergibt die gesehene Ablage  $\Delta u$ . Da der Quadropter nicht immer exakt waagrecht fliegt, kann sich der Fußpunkt  $F$  senkrecht unter ihm durchaus außerhalb der Bildmitte  $C$  befinden. Daher kommt zur gesehenen Ablage  $\Delta u$  auch eine rollwinkel-induzierten Ablage  $\Delta u_F$  hinzu.

Aus der Summe beider Ablagen sowie der Vorschaulänge  $l$  kann der Deichselwinkel  $\delta$  trigonometrisch berechnet und als Fehler an den Gierwinkel-Regler übergeben werden. Der so entstandene Deichselregler wird etwa in der Vorlesung *Verhaltensgenerierung für Fahrzeuge*



**Abbildung 5:** Geometrischer Zusammenhang zwischen erkannter Linie, gesehener und rollwinkel-induzierter Ablage  $\Delta u$  bzw.  $\Delta u_F$ , Vorschaulänge  $l$ , Tangentenwinkel  $\tau$  und Deichselwinkel  $\delta$ .



**Abbildung 6:** Rückansicht des Quadropters mit Bildebene, gesehener Ablage  $\Delta u$ , Rollwinkel  $\rho_{wd}$  und daraus induzierter Ablage  $\Delta u_F$ , sowie der gesamten lateralen Ablage  $\Delta L$ .

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  $\tau$  zwischen einer Tangente an die Linie und der Längsachse des Quadropters bestimmt und auf den Sollwert null eingeregelt werden. Da der Quadropter so jedoch nur parallel zur Linie, nicht aber genau darüber fliegen würde, wird der letztendlich einzuregelnde Gierwinkelfehler  $\Delta\psi$  aus einer gewichteten Summe von  $\delta$  und  $\tau$  gebildet:

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

Die Regelgröße des Seitwärts-Freiheitsgrades ist die in Abbildung 6 dargestellte Ablage  $\Delta L$  zwischen roter Linie und dem Fußpunkt senkrecht unter dem Quadropter. Diese kann trigonometrisch mit Hilfe der per Ultraschall gemessenen Höhe  $h$  berechnet werden. Der ebenfalls benötigte Winkel  $\varrho_{wl}$  zwischen dem Sichtstrahl zur Linie (“ $l$ ”) und einer Senkrechten in Welt-Koordinaten (“ $w$ ”) entspricht der Summe von  $\varrho_{dl}$  zwischen Sichtstrahl und Senkrechter in Quadropter-Koordinaten (“ $d$ ”) und  $\varrho_{wd}$  zwischen den jeweiligen Senkrechten von Quadropter- 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 Quadropters 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 Quadropters 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 Quadropters an der Rückseite an, setzen Sie einen Akku ein und schließen Sie diesen an. Der Quadropters 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 Quadropters. 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 Regler-Parametrierung nach Ziegler-Nichols

Das Verfahren nach Ziegler-Nichols erfordert eine Dauerschwingung des Quadropters. Der beschränkte Platz im Versuchsraum lässt daher nur die Einstellung der Höhen- und Gierwinkel-Regelung zu. Die Parameter der Quer- und Längs-Regelung werden aus früheren Experimenten übernommen.

Die Bearbeitung des Quellcodes findet in der Entwicklungsumgebung *Eclipse* statt. Starten Sie diese über das Menü am linken Bildschirmrand und öffnen Sie im Projekt `arDroneAutonomous` die Dateien `main.cpp` und `appZieglerNichols.h`.

- Prüfen Sie, ob in `main.cpp` die Applikation `DroneAppZieglerNichols` als einzige aktiv (d. h. nicht auskommentiert) ist.

- Suchen Sie in `appZieglerNichols.h` zunächst die Konstruktor-Methode `DroneAppZieglerNichols()`. Weisen Sie darin dem Verstärkungsfaktor  $k_P$  bzw. `this->kP` einen sinnvoll kleinen Startwert zu.

Im gleichen Abschnitt finden Sie auch Variablen für die kritische Verstärkung  $k_{Pk}$  und Periodendauer  $T_k$  für Höhen- und Gierwinkel-Regler, die Sie erst später initialisieren können.


#### 4.2.1 Regelung der Flughöhe

Vor der praktischen Durchführung der Ziegler-Nichols-Methode muss ein geeigneter P-Regler implementiert werden. Suchen Sie hierzu in der Datei `appZieglerNichols.h` die Methode `processNavdata()`. Diese wird automatisch jedes Mal aufgerufen, wenn ein neuer Messwert der Flughöhe vom Quadrokopter übertragen wird. Nehmen Sie dort folgende Änderungen vor:

- Wählen Sie über `mode = vertical` ; die Höhenregelung aus und scrollen Sie dann weiter zum Kommentar `// height control`.
- Setzen Sie den Sollwert `target` für die Flughöhe auf 1 Meter. Suchen Sie aus `navdata` den aktuellen Ist-Wert für die Höhe und speichern Sie diesen in der Variable `actual`. Führen Sie hierbei eine Umrechnung des Messwertes nach Meter durch.
- Berechnen Sie daraus die Regel-Abweichung und speichern Sie diese in `error`. Weisen Sie `flightCommands.vertical` die hieraus resultierende Stellgröße des P-Reglers zu. Den benötigten Wert  $k_P$  finden Sie in `this->kP`.

Kompilieren und starten Sie das Programm nun mit dem grünen “Play”-Button. Plotten Sie gleichzeitig den Verlauf der Höhe mit dem ROS-Tool `rqt_plot`.

Die Sicherungsperson startet nun den Quadrokopter und aktiviert den autonomen Flugmodus. Danach ist es Ihre Aufgabe, ein eventuelles Abdriften des Quadrokopters mit dem Gamepad auszugleichen. Wiederholen Sie nun die folgenden Schritte:

- Ein leichtes Herunterdrücken des Quadrokopters erzeugt eine Höhenabweichung, welche durch die Regelung rückgängig gemacht wird.
- Zunächst stellt sich hierbei noch keine Dauerschwingung ein. Erhöhen Sie in diesem Fall mit  die Verstärkung  $k_P$  und beginnen Sie von vorn.

Sobald sich eine Dauerschwingung eingestellt hat, notieren Sie das eingestellte und auf der Kommandozeile ausgegebene  $k_P$ . Pausieren Sie `rqt_plot`, sobald darin einige Schwingungen zu sehen sind. Danach können Sie den Quadrokopter wieder landen.

Aus dem Plot können Sie nun die Periodendauer  $T_k$  ablesen. Bilden Sie hierbei zwecks Genauigkeit den Mittelwert über alle beobachteten Schwingungen. Die Einheit der Zeitachse ist Sekunden.

Öffnen Sie nochmals den Konstruktor `DroneAppZieglerNichols()`. Tragen Sie dort die eben gemessenen Werte ein. Vervollständigen Sie die Berechnung der Reglerparameter  $k_P$ ,  $k_I$ ,  $k_D$  anhand von Tabelle 1 in Abschnitt 3.1.1. Beachten Sie beim Auswählen der Heuristik, dass Sie eine Position als Regelgröße über eine Geschwindigkeit als Stellgröße einregeln möchten: Weist die Strecke also bereits selbst integrierendes Verhalten auf?

## 4.2.2 Regelung des Gierwinkels

Für die Gierwinkelregelung setzen Sie die Variable `mode = yaw`; und scrollen Sie weiter zum Kommentar `// heading control`.

Definieren Sie analog zur Höhenregelung einen Sollwert, lesen Sie den aktuellen Ist-Wert aus und bestimmen Sie die Regel-Abweichung. Speichern Sie die daraus resultierende Stellgröße nun in `flightCommands.yaw`.

Gehen Sie auch beim praktischen Teil analog zur Höhenregelung vor, aber plotten Sie nun den Gierwinkel. Erzeugen Sie einen Winkelfehler, indem Sie den Quadrocopter leicht “andrehen”.

## 4.3 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 Quadrocopter 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. Linienvidenz 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 Programm `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. Die übrigen Modi sind robustere Schätzverfahren und werden nur im Praktikum *Rechnergestützte Verfahren der Mess- und Regelungstechnik* behandelt.
- 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.

#### 4.4 Nachfliegen der Linie

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

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

Geben Sie im Konstruktor `DroneAppFollowLine()` nochmals die anfangs bestimmten kritischen Verstärkungen und Periodendauern von Höhen- und Gierwinkelregelung ein. Die eigentliche Folge-Regelung findet in der Methode `processImageBottom()` statt:

- Nutzen Sie den Rollwinkel aus `navdataAtImage` sowie die Kamera-Brennweite `this->focalLengthBottomU`, um die rollwinkel-induzierte Ablage `deltaUF` zu berechnen.
- Definieren Sie eine Vorschaulänge  $l$ , welche eine Bildzeile  $v$  oben im Bild bzw. vor dem Quadrokopter ergibt. Berechnen Sie dort den Funktionswert  $u$  des geschätzten Polynoms und daraus die gesehene Ablage `deltaU`.
- Nutzen Sie die zuvor berechneten Werte, um den Deichselwinkel  $\delta$  zu bestimmen, und speichern Sie diesen in der Variable `drawbar`.

- Berechnen Sie zusätzlich die Ableitung des Polynoms in Zeile  $v$  und daraus den Winkel  $\tau$  zwischen dessen Tangente und der Quadrokopter-Längsachse. Speichern Sie diesen Winkel in `tangent`.
- Bestimmen Sie zuletzt den metrischen Abstand zwischen der Linie und dem Fußpunkt des Quadrokopters.
- Lassen Sie sich die o. g. Zwischenergebnisse auf der Kommandozeile ausgeben und überprüfen Sie deren Größenordnung und Vorzeichen nochmals bei von Hand getragem 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, und untersuchen Sie hierbei verschiedene Gewichtungen. 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 Kolloquiums 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

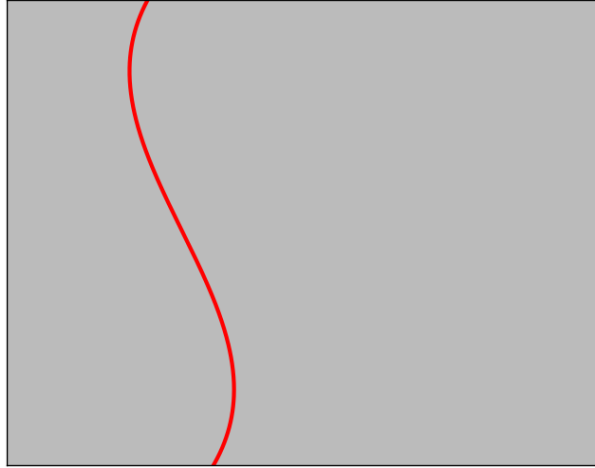
Wie müssen die Gewichtungsfunktionen  $w_K$  aussehen, damit ein Maximum möglichst deutlich hervortritt?

*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)

#### 4. Bestimmung der Ablage

Aus einem Bild schätzen Sie folgendes Polynom:

$$u(v) = -0.000015 \cdot v^3 + 0,008 \cdot v^2 - 0,74 \cdot v + 150 \quad (9)$$



Das Bild ist  $640 \text{ px}$  breit und  $360 \text{ px}$  hoch. Die Brennweite beträgt  $f = 600 \text{ px}$ . Als Vorschaulänge wählen Sie  $l = 100 \text{ px}$ . (siehe Abbildung 5)  
Bestimmen Sie die zur Vorschaulänge gehörende Bildzeile  $v$ .

Bestimmen Sie die Ablage  $\Delta u$  zur Bildmitte in Zeile  $v$ .

Bestimmen Sie die Orientierung  $\tau$  der Linie in Zeile  $v$  relativ zur Längsachse des Quadropters.

Aus den Inertialmessdaten wissen Sie, dass der Quadropter aktuell mit einem Rollwinkel  $\varrho_{wd} = 5^\circ$  fliegt. Bestimmen Sie die daraus entstehende zusätzliche Ablage  $\Delta u_F$  in Pixeln.

Bestimmen Sie den Deichselwinkel  $\delta$ .

## 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`, größer-gleich `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](http://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 Programmiererteils 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 rausch-behaftete 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 Quadrocopter spezifische Werte definiert.)
- Video-Stream der aktuell ausgewählten Kamera anzeigen:  
`roslaunch 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 Quadrocopters 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

