# Erkennung möglicher Notlandefelder aus Höhendaten

## Skizze eines Algorithmus zur Erkennung

Eingabedaten:

* Es ist eine (rechteckige) Kachel mit Höhenangaben eines zu untersuchenden Gebiets gegeben. Die Kachel ist gerastert, enthält also Quadrate (Rechtecke) mit einer bestimmten Kantenlänge und jedem dieser Quadrate ist eine bestimmte absolute Höhe zugeordnet.
* Eine potentielle Notlandebahn wird charakterisiert durch folgende Angaben:
  + Minimale Länge
  + Minimale Breite
  + Einen Satz an Steigungsdaten, welche in dem potentiellen Notlandefeld nicht überschritten werden dürfen
  + Die Steigungsdaten sind in Längsrichtung und in Querrichtung der Notlandebahn unterschiedlich

Charakterisierung durch Steigungsdaten

Eine Landebahn kann durch einen Satz von Steigungsdaten Charakterisiert werden:

* „global“: Eine Landebahn darf auf Ihrer gesamten Länge nicht über ein gewisses Maß hinaus ansteigen. Ein Flugzeug kann nicht bergauf und auch nicht bergab landen. Es muss also für die gesamte minimale Bahnlänge ein maximaler Anstieg und ein maximaler Abfall gegeben werden.
* „lokal“: Für (deutlich) kürzere Strecken als die Gesamtbahnlänge ist ein anderer Satz von maximalen/minimalen Steigungslimits notwendig, welche die maximale „Holperigkeit“ der potentiellen Bahn beschreiben.
* „Querneigung“: Eine optimale Landebahn ist senkrecht zur langen Achse perfekt eben. Eine reale Bahn darf geneigt bzw. in sich „verwunden“ sein, solange kein kritischer Grenzwert für diese Querneigung überschritten wird. Diese maximale Querneigung wird über die Mindestbreite der Bahn angegeben.  
  (*zu klären*: Benötigen wir auch noch einen Parameter zur maximalen Neigungsänderung auf den Außenrändern der Bahn, oder wird das über die maximale Holprigkeit schon abgedeckt.)

Das globale Limit der Steigung ist sicherlich geringer als das lokale Limit für die Holprigkeit. (Extremfall: Landen auf Kopfsteinpflaster kann lokal ziemlich große Steigungen verursachen, wenn das Raster klein genug ist, stellt aber eigentlich kein Problem dar, da die Strecken zwischen den Rasterpunkten kurz genug sind.)

*Zu klären:*

* Wie viele verschiedene Sätze an Steigungslimits werden benötigt? Sicherlich notwendig:
  + Gesamtsteigung auf der Mindestbahnlänge
  + Kurzes lokales Steigungslimit im 1-Meter Bereich
  + Mittleres lokales Steigungslimit im 20 Meter Bereich (passt zu den Initialen Daten)

(Die Länge, über die eine Steigung angegeben ist soll als „**Vorausschaulänge**“ bezeichnet werden).

* Sind auch Steigungslimits für mittlere Längen notwendig?
* Ist es notwendig „Zwischenlängen“ passend zu ergänzen? Z. B. indem eine kürzere Länge so lange verdoppelt wird, bis sich die nächst längere Länge ergibt. Dann müsste ein „gewichteter“ Mittelwert der erlaubten maximalen Steigungen angegeben werden, damit eine Gesamtsteigung mit einem überlagerten „Holpern“ noch OK ist.

Geeignete Landebahn

Eine geeignete Landebahn ist gefunden, wenn ein rechteckiger Bereich gefunden ist, bei dem für **jeden** Rasterpunkt innerhalb der minimalen geforderten Länge und Breite die gegebenen Steigungslimits nicht überschritten werden.

Auch wenn die kürzeste Vorausschaulänge länger ist als der Abstand der Rasterpunkte, so muss die Überprüfung der Steigungslimits doch für **jeden** Rasterpunkt innerhalb der Bahn erfolgen und entsprechend positiv ausfallen.

Durchmusterung eines Bereichs

Eine Kachel wird so groß gewählt, dass sie vollständig in den Speicher passt und zusätzlich Platz ist für die einzelnen Punkte alle notwendigen Verwaltungs- und Ergebnisinformationen zu speichern.

Eine Kachel wird in jedem Durchgang auf Bahnen in einer vorgegebenen Richtung durchmustert. Diese Richtung gibt die Längsachse der potentiellen Bahn an.

Die Steigungsüberprüfung erfolgt immer vorwärts gerichtet und berechnet den Quotienten aus Höhenänderung über Vorausschaulänge. Es wird also überprüft, ob vom aktuellen Punkt aus vorwärts kein Hindernis eine Landung unmöglich macht.

**Brute Force:**

1. Berechne für einen Punkt die Steigungsmaße für die vorgegebenen Vorausschaulängen und speichere sie pro Punkt.
2. Wenn ein Limit gerissen wurde, dann schließe Punkt aus. Dieser darf nicht innerhalb einer Landebahn liegen.
3. Gehe zum nächsten Punkt.

Alle Punkte, die sämtliche Steigungstests bestehen sind Kandidatenpunkte für eine Landebahn.

1. Fasse die Kandidatenpunkte entlang der durchmusterten Bahn in möglichst lange Teilstrecken zusammen.
2. Streiche alle Teilstrecken kürzer als die Mindestbahnlänge
3. Überprüfe entlang der übrig gebliebenen Strecken die Querneigung.

**Idee zur Optimierung:**

Um die Menge der (genau) zu überprüfenden Daten möglichst schnell zu minimieren, ist es wichtig, dass bei einem frühen (grob gerasterten) Durchlauf schon möglichst viele Punkte als Kandidaten herausfallen.

Wenn ein Punkt aus der KandidatenMenge herausfällt, dann wirft er einen „Schatten“ auf das zurückliegende Stück. Eine Bahn kann nicht geeignet sein, wenn sie innerhalb ihrer Mindestlänge diesen ungeeigneten Punkt zwangsläufig enthält.

* Wenn ein Punkt aufgrund eines Kriteriums herausfällt, dann kann die direkt vor ihm liegende Strecke auf keinen Fall Teil einer geeigneten Landebahn sein.   
  Es können also alle Punkte vom Ende der zugehörigen Vorausschaulänge bis zur minimalen Bahnlänge vor diesem Ende der Vorausschaulänge für weitere Berechnungen ausgeschlossen werden. **Die Strecke zwischen Punkt und Vorausschauende darf dabei \_nicht\_ ausgeschlossen werden. Dort könnte ja gerade eine neue Ebene beginnen!**

Mindestbahnlänge

Vorausschaulänge

Aktuelle Prüfung (schlägt am Anstieg fehl)

„Schattenwurf“ des Ausschlusses

Dadurch Ausschluss von weiter vorne liegenden Kandidatenpunkten

* Wenn also ein ungeeigneter Punkt gefunden wird, dann brauchen die Punkte, die innerhalb der Strecke vom aktuellen Punkt bis zur Mindestbahnlänge vor dem Vorausschauende liegen nicht mehr überprüft werden.
* Bei einer kurzen Vorausschau können also besonders viele Punkte aufgrund einer einzelnen Steigungsverletzung ausgeschlossen werden. (Fange mit den kurzen Vorausschaulängen an zu durchmustern)
* Die Schrittweite bei jedem Durchlauf sollte also der Mindestbahnlänge entsprechen. Wenn ein Punkt herausfällt, dann werden die davor liegenden Punkte aus der weiteren Durchmusterung ausgeschlossen.
* In einem nächsten Durchlauf wird der nächste noch zu besuchende Punkt gewählt, aber die Schrittweite bleibt bei der minimalen Bahnlänge. Wenn auf einen Punkt getroffen wird, der in einem vorherigen Durchlauf ausgeschlossen wurde, fahre mit dem Punkt fort, der eine minimale Bahnlänge Abstand vom auszulassenden Bereich hat.
* Wiederhole so lange, bis alle Punkte entlang einer Richtung entweder ausgeschlossen wurden, oder für das überprüfte Steigungskriterium als geeignet erscheinen.

Diese Optimierung bringt nur etwas, wenn die Vorausschaulänge kleiner als die minimale Landebahnlänge ist. Bei einem wirklich „globalen“ Kriterium (Maximalanstieg auf der gesamten Bahnlänge) muss also trotzdem jeder Punkt durchmustert werden.

Bei einer tatsächlich geeigneten Bahn muss jeder Punkt durchmustert und überprüft werden. Selbst wenn das Raster deutlich feiner als die kleinste Vorausschaulänge ist.

## Offene Punkte:

* Mit dem oben skizzierten Algorithmus können potentielle Landebahnen nur in einer einzigen Richtung erkannt werden. Ist es wirklich notwendig in allen (mindestens) 8 Richtungen einen einzelnen Durchlauf zu starten? Oder ist es eventuell möglich in jedem Punkt die Richtung des Gradienten auszuwerten und damit durch eine Projektion auf die betrachtete Richtung Rechenzeit zu sparen.
* Es wird der Zeitpunkt kommen, dass die Betrachtete Karte nicht mehr komplett in den Speicher passt. Sei es durch Ausdehnung des Gebiets oder durch Steigerung der Auflösung. Dann muss die Karte gekachelt werden. Es ist zu klären, wie mit den Randbereichen zwischen zwei Kacheln umgegangen wird und wie es möglich wird an den Rändern zweier Kacheln zwei zu kurze Bahnen zu einer ausreichend langen zusammenzufügen. Es muss ein „Randprotokoll“ entwickelt werden. Ein solches könnte auch zum Übergang von shared memory Systemen auf lose gekoppelte Parallelrechner (MPI) benutzt werden.
* In welchem Format werden mögliche Notlandebahnen später in der Datenbank abgelegt? Wie kann ich größere Gebiete sinnvoll zusammenfassen?

# Einlesen der Daten

Zum Einlesen der Daten kann auf Routinen aus tifffastcrop zurückgegriffen werden. Es wurde eine Einleseroutine erstellt, die einen rechteckigen Bereich aus einem TIFF in den Speicher liest, der in Form von Pixelkoordinaten relativ zur linken oberen Ecke der vorliegenden TIFF-Datei angegeben wird.

Datenformat:

Die Höhendaten des vorliegenden Beispieldatensatzes enthalten für jedes Pixel ein 32bit double mit der Höhe in Metern.

Die zur Umrechnung von Geo-Koordinaten auf Pixelkoordinaten notwendigen Angaben liegen in Form von weiteren Dateien vor.

TODO: Erweitere die Einleseroutine um Funktionen zur Koordinatentransformation Geo-Koordinaten ↔ Pixel.

Bereichsauswahl:

Zur Bereichsauswahl des zu durchmusternden Bereichs ist es, soll die Ausdehnung/Überdeckung eines Datensatzes als Overlay in OSM eingeblendet werden. (TODO: entscheide, ob lediglich die rechteckige Bounding Box der Koordinaten, oder aber die echte Kontur eingeblendet werden soll.)

In OSM kann mittels „boundingBox“ (<http://wiki.openstreetmap.org/wiki/Bounding_Box>) eine Auswahl erfolgen. Es wird ein Rechteck in Geo-Koordinaten zurückgegeben. Wenn diese Geo-Koordinaten in Pixel umgerechnet sind, dann kann der entsprechende Bereich –entweder am Stück, oder gekachelt– eingelesen und Durchmustert werden.

# Ausgabe der Daten

Zur Ausgabe der Daten soll eine Datenbank benutzt werden. In dieser Datenbank sollen potentiell geeignete Landebahnen eingetragen werden können.

TODO: Lege das Datenformat eines Landeflächeneintrags fest. Lediglich 2 Punkte (Start- und Ende), oder auch hier komplexere Formen oder zumindest Rechtecke mit einer Breite. Es muss ein geeigneter Algorithmus gefunden werden mehrere angrenzende Kandidaten zu mergen, damit bei besser aufgelösten Karten nicht eine riesige Anzahl parallel laufender Bahnen entsteht, die teilweise überlappen. Ist es besser die Landebahnen als gefüllten Polygonzug entlang des Randes einer potentiellen Landefläche zu speichern? Dann ist sicherlich noch die angabe einer Richtung notwendig, in der gelandet werden kann.

Die in der Datenbank eingetragenen Daten können von dort in eine separate View Komponente eingelesen/abgefragt werden. Diese Komponente sollte die gefundenen Kandidatenflächen dann in OSM als Overlay einblenden.

TODO: Ist es notwendig die Durchmusterung mit der Darstellung zu synchronisieren (push)? Oder ist es ausreichend, die Abfrage neuer Daten aus der Datenbank manuell zu aktualisieren (pull)?

TODO: Es ist zu klären, ob und wie aus den OSM-Daten bestimmte Bereiche potentieller Landebahnen ausgeschnitten werden können (Autobahnen, Seen, militärische Sperrgebiete, …).

Für die Visualisierung bietet sich eine kleine JavaScript-Anwendung an, die die Darstellung im Browser übernimmt. Damit kann auf Standardtechnologien zurückgegriffen werden und es muss nicht zu viel Hirnschmalz in die Visualisierung und eine Benutzeroberfläche gesteckt werden. Außerdem sind heterogene, kommunizierende Prozesse ganz im Sinne der Veranstaltung „Parallelprogrammierung“