# Raspberry Pi Sailboat Dokumentation

[Parallele Prozesse 2](#_Toc522961034)

[Sensoren 2](#_Toc522961035)

[Abstandsensor 2](#_Toc522961036)

[Steuerung 2](#_Toc522961037)

[Ansteuerung Segel-Servo 2](#_Toc522961038)

[Einstellung des Segel-Winkels 3](#_Toc522961039)

[Ruder-Servo 3](#_Toc522961040)

[Navigation 3](#_Toc522961041)

[Seekarten 3](#_Toc522961042)

[Eingelesene Seekarten als Instanzen 4](#_Toc522961043)

## Parallele Prozesse

Wenn das Skript «startboat.py» zum Starten des gesamten Programms aufgerufen wird, werden sofort die 4 Hauptprozesse parallel gestartet: Co-Pilotin, Navigator, Ausgucker, Segelsetzerin und Rudersetzer.

Die Co-Pilotin kümmert sich um Hindernisse/Unerwartetes, z. B. Ausweichmanöver. Der Navigator berechnet eine sinnvolle Route. Der Ausgucker erfasst Wind und Abstand zu Objekt vor Schiff. Segel- und Rudersetzer\*in steuern die Servos.

Zwecks Übersicht werden alle diese Objekte in «startboat.py» definiert.

## Sensoren

### Abstandsensor

Zur Kollisionsvermeidung wird ein IR-Sensor am Schiffsbug installiert. Der Sensor (Sharp GP2Y0A710K0F) erkennt Distanzen zwischen 100 und 550 cm.

Anschluss:

! Die roten Kabel werden verdrillt (Versorgungsspannung).

! Die schwarzen Kabel ebenfalls (Ground).

Die roten Kabel werden mit 5V Spannung versorgt.

Das weisse Kabel gibt den gemessenen Abstand in analoger Spannung zurück (0-5V) und wird deshalb an den AD-Wandler MCP3008 angeschlossen, auf Kanal 1.

Da der Sensor kurzzeitig hohe Ströme benötigt, muss ein Kondensator (mind. 10 µF) mitangeschlossen werden. Dieser wird parallel zum Sensor geschalten und mit einem Vorwiderstand versehen (in unserem Fall 220Ω). Dies ist nötig, weil der Kondensator ganz entladen widerstandsfrei ist.

Im Skript wird dann der Wert des AD-Wandlers eingelesen und so umgewandelt, dass ein Wert zwischen 0 und 100 entsteht. Je näher Objekte sind, desto grösser ist dieser Wert. Die Werte sind nicht linear verteilt. Da das Boot sich aber nur an einem willkürlich gesetzten Grenzwert (Cutoff) orientiert, wird dies nicht weiter berücksichtigt oder berechnet.

## Steuerung

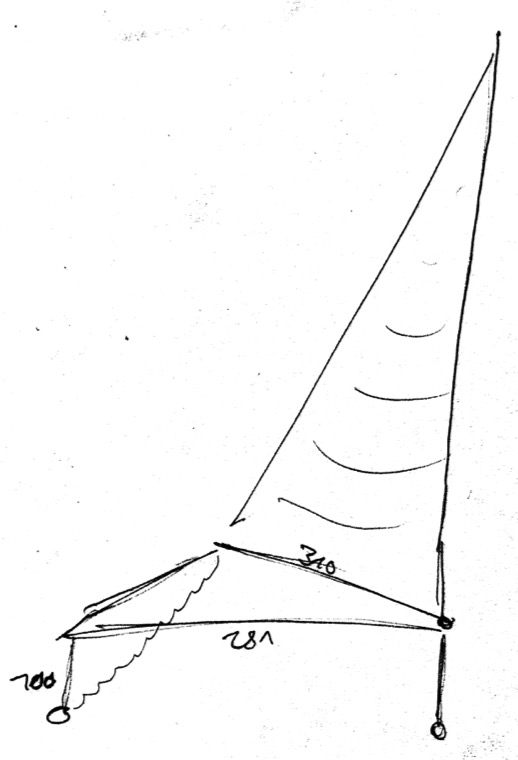
### Ansteuerung Segel-Servo

Der Segelservo dreht sich mehrfach und wird mittels Impulsweitenregulation gesteuert. Die Pulsweiten liegen zwischen 5 und 8,4 Prozent, von 20ms (50Hz). Dies konnte mit Tests herausgefunden werden. Dies reicht, um das Segel ein bisschen weiter als 90 Grad auszufahren, was jedoch nicht nötig ist.

An den Servo muss zwingend 6V Betriebsspannung angelegt werden. Die Steuersignale können aber tiefer liegen, sodass die 3.3V vom GPIO-Pin reichen. Wichtig: Die unterschiedlichen Spannungen müssen am selben Ground liegen.

### Einstellung des Segel-Winkels

Das Segel wird mit einer Schnur gelöst und dreht sich je nach Wind nach links oder rechts. Dabei sind jedoch der Winkel des Segels und die Länge der Schnur nicht in linearer Beziehung. Deshalb wurde eine Formel entwickelt, die gedanklich zwei Dreiecke aufspannt (siehe Grafik) und mittels Pythagoras und Kosinussatz die Seillänge berechnet. Die Pulsweite steht nun aber in einem linearen Verhältnis zur Schnurlänge.



### Ruder-Servo

Dieser Servo kann auch mit 5V betrieben werden und kann daher direkt am Raspberry Pi angeschlossen werden. Die Ansteuerung ist einfach: Eine Pulsweitenlänge von 5 bis 10 Prozent von 20ms (50Hz) entspricht dem maximalen Bewegungsradius des Servos. Dabei bewegt sich das Ruder über ca. 90 Grad, entsprechend 45 Grad nach links resp. rechts.

## Navigation

### Seekarten

Die Seekarten werden im Ordner «lakemaps» abgelegt. Sie beinhalten pro Zeile ein Punkt (Latitude, Longitude). Die Seen werden dabei zunächst als ein grosses Polygon beschrieben. Eingeleitet wird dies mit der Zeile «Navigable:», also der fahrbare Bereich. Dann folgen die Punkte, sodass sie ein Polygon ergeben, wenn man sie der Reihe nach verbindet.

Sollten im See Hindernisse stehen, können diese ebenfalls als Polygone definiert werden. Diese werden mit «Exclude Floss:» eingeleitet, wobei «Floss» nur ein Beispiel für eine Bezeichnung ist.

Weiter können noch virtuelle Bojen gesetzt werden. Diese sind überall dort notwendig, wo der Uferverlauf ein direktes Anfahren aller möglichen Zielpunkte verhindert, z. B. weil der See sehr verwinkelt ist. Die Bojen können dann als Zwischenstation angefahren werden, um von dort aus entweder den Zielpunkt oder davor noch eine weitere Boje anzufahren. Die Bojen werden wie die Hindernisse eingeleitet, einfach mit «Buoy»: «Buoy North:».

Bojen und Hindernisse können definiert werden, müssen aber nicht. Dann können sie einfach komplett weggelassen werden. Die Reihenfolge («Exclude», «Navigable», «Buoy») spielt keine Rolle; das Einlese-Skript orientiert sich an den Zwischentiteln.

Zur Übersicht werden die Seekarten auch manuell grafisch dargestellt und in «doc/lakemaps-img» abgelegt.

### Eingelesene Seekarten als Instanzen

Die Seekarten (Textfiles im Ordner «lakemaps») werden beim Aufruf von startboat.py mit dem Befehl «mdrouting.InitializeLakemaps()» eingelesen. Dabei wird jedes Textfile in eine Instanz überführt. Die Instanz wird nach dem Textfile benannt, bzw. mit dem Dateinamen vor «.txt». Die Instanzen sind folgendermassen aufgebaut:

Attribute:

Instanzname.lake\_name (Name des Sees aus Dateiname generiert)

Instanzname.navigable (Koordinatenpunkte als Liste in Liste: [[Lat1, Lon1],[Lat2, Lon2] etc. ]

Instanzname.excludes (Koordinatenpunkte als Liste in Liste in Liste, weil hier mehrere Polygone beschrieben werden können: Polygon > Punkte > Lat/Lon.)

Instanzname.exclude\_names (Bezeichnungen der Hindernisse in derselben Reihenfolge, wie die Polygone in «Instanzname.excludes» definiert sind.)

Instanzname.buoys (Koordinatenpunkt als Liste in Liste: Boje > Lat/Lon.)

Instanzname.buoy\_names (Genau wie bei den Hindernissen werden hier alle Bezeichnungen der Bojen in jener Reihenfolge aufgeführt, wie sie zuvor in «Instanzname.buoys» definiert wurden.)

Da jede Seekarte nun eine Instanz (von «lakemap») ist, ist sie global verfügbar. Alle aktuellen Instanzen sind unter «lake\_instances» als Liste aufgeführt.