Gruppe 2: Lageabschätzung

Dokumentation

Verzeichnis:

Aufgaben: (von Pattanachart)

Im Rahmen des Projektes „Autonome Modellfahrzeuge“ beschäftigen wir uns mit der Lageabschätzung des Modelfahrzeugs.

Die Hauptziele sind:

- Erkennung der Orientierung des Fahrzeugs mittels Integration von Gyroskopdaten.
- Kameradaten zur Verbesserung der Genauigkeit und zur Beseitigung des Drifts (Komplementärfilters).
- * Verwendung eines Komplementärfilters: Zur Kombination der Daten von Gyroskop und Kamera, um eine präzisere und stabilere Schätzung der Fahrzeuglage zu erreichen. Der Komplementärfilter nutzt die Vorteile beider Sensoren und kompensiert gleichzeitig deren Schwäche.

Stand: (von Pattanachart)

Der aktuelle Stand des Projekts ist wie folgt.

- Lageabschätzung: ‘Lageabschaetzung.cpp‘ und ‚Lageabschaetzung.h‘, diese Dateien enthalten die Implementierung der Lageabschätzung anhand von Gyroskop- und Kameradaten. Es verarbeitet die Sensordaten und berechnen die aktuelle Orientierung des Fahrzeugs ziemlich gut.
- Gyroskopintegration: Ein MPU6050-Gyroskop erfasst die Winkelgeschwindigkeit des Fahrzeugs. Die Integration funktioniert gut und die Daten werden ziemlich korrekt erfasst.
- Datenfusion: Die Kombination von Kameradaten und Gyroskopwerten erfolgt mittels eines Komplementärfilters, um den Drift zu beseitigen. Der Komplementärfilter ist

implementiert und korrigiert den Drift effektiv, was zu einer präziseren Schätzung der Fahrzeuglage führt.

- Kameradatenverarbeitung: Eine webcam wird verwendet, um Aruco-Marker zu erkennen und deren Orientierung zu bestimmen. Dies funktioniert zuverlässig und liefert stabile Ergebnisse.
- Kommunikation: Die ermittelten Winkel werden über eine serielle Schnittstelle und LoRa- Funkmodule übertragen. Die Kommunikation funktioniert stabil und zuverlässig.
- Protokollierung: Die Daten werden in Log-Dateien gespeichert und zur späteren Analyse verwendet. Die Protokollierung funktioniert wie erwartet und die Daten sind für die Analyse verfügbar.

Anleitung zur Nutzung: (von Pattanachart)

Nur einmal:

1. Muss sicher sein, dass Lageabschätzung.cpp und Lageabschätzung.h im gleichen Ordner sind.
2. Inkludiert lageabschaetzung.h (#include “Lageabschaetzung.h”)
3. Erstellen ein Objekt der Klasse Lageabschaetzung (Lageabschätzung l;)
4. Wert von Filterkonstante festlegen. (l.set_comp_filter_factor(factor);)

In der endlosen Schleife:

5. MPU update (mpu.fetchData();)
6. Falls vorhanden neuen Kamerawert festlegen(l.set_angle_cam_deg(angle_cam_deg);)
7. Mit Gyroskopwert neues Heading berechnen(l.get_heading_deg(mpu.getGyroZ());)
8. Heading aus klasseninterner Variable auslesen (l.heading_deg)

Theorie Komplementärfilter (von Lukas)

Der Komplementärfilter ist dazu da um die gemessenen Winkel der Kamera und des Gyroskops auf einen brauchbaren Wert zu regeln. Man sollte nicht nur nach der Kamera fahren da diese zu langsam Werte liefert und man sollte auch nicht nur nach dem Gyroskop fahren da dieses mit der Zeit wegdriftet. Die Filterkonstante (COMP_FILTER_FACTOR) legt fest wie stark Werte gewichtet werden.
Filterkonstante = 1 bedeutet das man nur nach der Kamera werten fährt
Filterkonstante = 0 bedeutet das man nur nach Gyroskop fährt
Ziel ist es soviel wie möglich nach dem Gyroskop zu fahren, da die Kamerawerte sehr langsam kommen können, wenn mehrere Roboter gleichzeitig fahren. Man sucht also nach einer möglichst kleinen Filterkonstante. Wenn diese aber zu klein wird gleicht sie den Drift des Gyroskops nicht mehr ausreichend aus.
Im Code ist der Komplementärfilter mit folgenden Zeilen umgesetzt:
heading_deg += (1 - COMP_FILTER_FACTOR) * (gyro_z * dt);
heading_deg = heading_deg + (COMP_FILTER_FACTOR) * angle_diff;

C++ Implementation: Lageabschaetzung (von Moritz)

Klasse zur Lageabschätzung des Picos um die Z-Achse.
Dazu gehören: Lageabschaetzung.h, Lageabschätzung.cpp

Public Member:

- Lageabschaetzung()

Standard Konstruktor.

- ~Lageabschaetzung()

Standard Destruktor.

- double get_heading_deg(double gyro_z)

Heading aus Gyro- & Kamerawinkel berechnen.

- void parse_angle_cam_packet(uint8_t *PL)

Liest Kamerawinkel aus Lora-Paket aus.

- void set_angle_cam_deg(double angle_cam_deg)

Legt neuen Wert für Kamerawinkel fest.

- double get_angle_cam_deg()

Gibt Kamerawinkel zurück.

- void set_comp_filter_factort(double factor)

Legt neuen Wert für Komplementärfilterfaktor fest.

- double heading_deg

Lage des Picos.

Detaillierte Beschreibung

Klasse zur Lageabschätzung des Picos um die Z-Achse. Dazu werden lokal über ein Gyroskop gemessene Werte mit Werten einer Kamera kombiniert um möglichst gute Ergebnisse zu erzielen. Die Kombination erfolgt über einen Komplementärfilter.

Lageabschaetzung()

Standard Konstruktor.
Setzt Filterfaktor auf .0001.

~Lageabschaetzung()

Standard Destruktor.

double get_heading_deg(double gyro_z)

Berechnet das Heading [°] und gibt dieses zurück.
Integriert vom Gyroskop erhaltene Winkelgeschwindigkeiten und kombiniert diese durch einen Komplementärfilter mit von einer Kamera bestimmten Winkelwerten um bestmögliche Ergebnisse zu erhalten.
Je nach Einstellung der Filterkonstante dominieren entweder die Gyro- oder die Kamerawerte.

Argumente:
double gyro_z: vom Gyroskop ausgelesene Winkelgeschwindigkeit [°/s]

Gibt zurück:
double heading_deg: neu berechneter Wert des Headings [°]

*void parse_angle_cam_packet(uint8_t *PL)*
Liest Kamerawinkel aus der Payload eines Lora-Pakets aus und übernimmt diesen als neuen Kamerawert.
Als Payload werden zwei Byte erwartet. PL[0] ist dabei das höherwertigere und PL[1] das niedrigere Byte eines uint16_t sein, der den Winkel in Minuten beinhaltet.
Beim ersten Aufruf der Funktion wird der Wert des Kamerawinkels auch für das Heading übernommen. Dies vermeidet eine lange Anfangszeit mit sehr falschem Winkel, wenn sich Pico und Kamera nicht über den Nullpunkt einig sind.

Argumente:
uint8_t *PL: Pointer zum ersten Element der Payload des Pakets

void set_angle_cam_deg(double angle_cam_deg)
Legt einen neuen Wert für den Kamerawinkel fest.
Sollte immer aufgerufen werde, sobald ein neuer Wert von der Kamera erhalten wird.

Beim ersten Aufruf der Funktion wird der Wert des Kamerawinkels auch für das Heading übernommen. Dies vermeidet eine lange Anfangszeit mit sehr falschem Winkel, wenn sich Pico und Kamera nicht über den Nullpunkt einig sind.

Argumente:
    double angle_cam_deg: Kamerawinkel [°]

double get_angle_cam_deg()
Gibt den aktuellen Wert des Kamerawinkels [°] zurück.

Gibt zurück:

    double angle_cam_deg: Kamerawinkel [°]

void set_comp_filter_factor(double factor)
Legt einen neuen Wert für den Komplementärfilterfaktor fest.
Ist dieser sehr klein, dominieren die Werte des Gyroskops, ist er groß die der Kamera.

Argumente:
    double factor: neuer Wert für Komplementärfilterfaktor,
        sollte zwischen 0 und 1 liegen

double heading_deg
Zuletzt berechneter Wert für das Heading [°].

Winkelbestimmung mit Kamera: Camera.py (von Islam)

Methode:

- Kamera öffnen:

```
cam_path = "/dev/v4l/by-id/usb-046d_C922_Pro_Stream_Webcam_CD90D4BF-video-index0"
cam = cv.VideoCapture(cam_path)
if not cam.isOpened:
    print("Kamera konnte nicht geöffnet werden")
    exit(1)
```

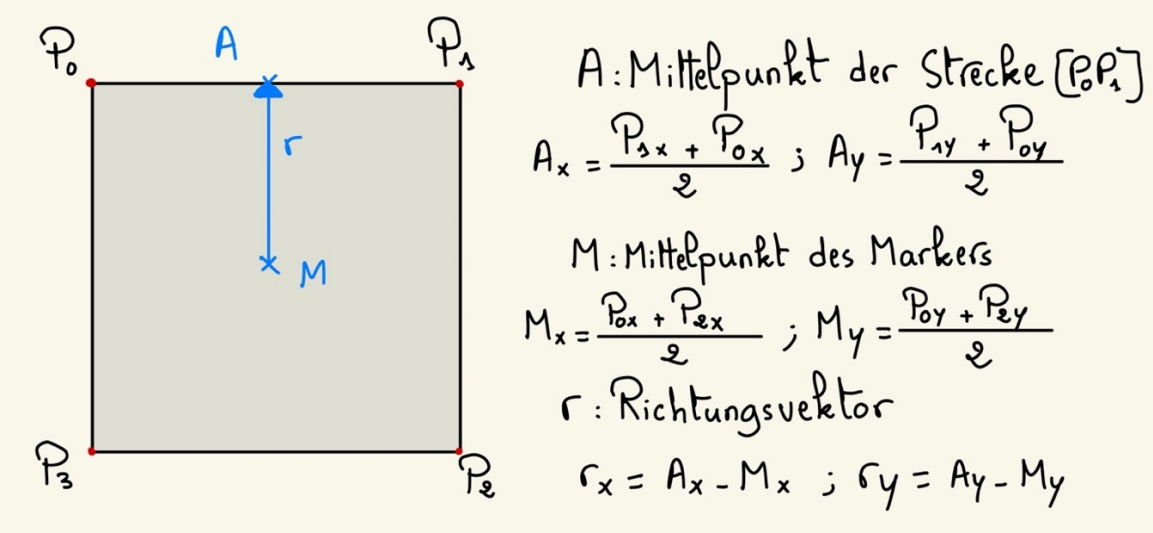
- Aruco-Detector aus der openCV-Bibliothek mit den entsprechenden „Dictionary“ und Parametern aufstellen (siehe opencv Dokumentation im Internet)

```
aruco_dict = cv.aruco.getPredefinedDictionary(cv.aruco.DICT_4X4_50)     # zum
detector = cv.aruco.ArucoDetector(aruco_dict, cv.aruco.DetectorParameters())
```

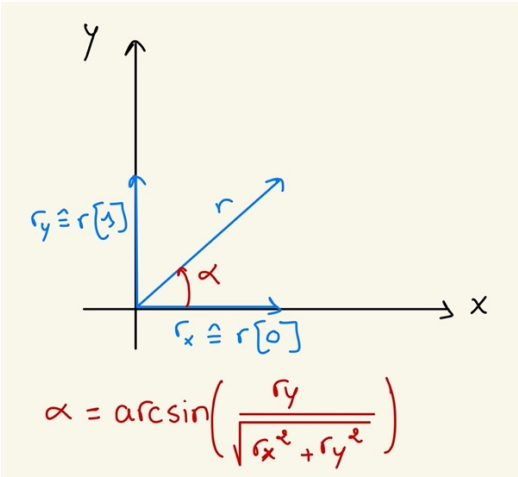
- In der endlosen Schleife werden Bilder ausgelesen und in Graustufen umgewandelt. Der Detektor „detectMarkers“ speichert die Ergebnisse in 3 Variablen: in der ersten (corners) die Koordinaten von den Eckpunkten aller detektierten Marker, in der zweiten (ids) die ID-Nummer aller detektierten Marker z.B 4, 27 als Liste und in der letzten (rejected) eine Liste von möglichen Markerkandidaten, die während des Erkennungsprozesses identifiziert, aber letztendlich als ungültig eingestuft wurden. (nicht relevant für uns)

```
while 1:
    result, image = cam.read()

    if result:
        image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

        (corners, ids, rejected) = detector.detectMarkers(image)
```

- Mithilfe der Koordinaten (x und y) der Eckpunkten aller detektierten Marker können wir den entsprechenden Winkel bestimmen:



Im Programm brauchen wir die Koordinaten nicht einzeln zu berechnen. Z.B
r = a – m reicht aus. Beide Koordinaten sind da enthalten.
Wenn man Zugriff auf eine bestimmte Koordinate haben will, dann schreibt man r[0], A[0], … für x-Koordinate und r[1], A[1], … für y-Koordinate.
Die Berechnung des Winkels sieht dann so aus:

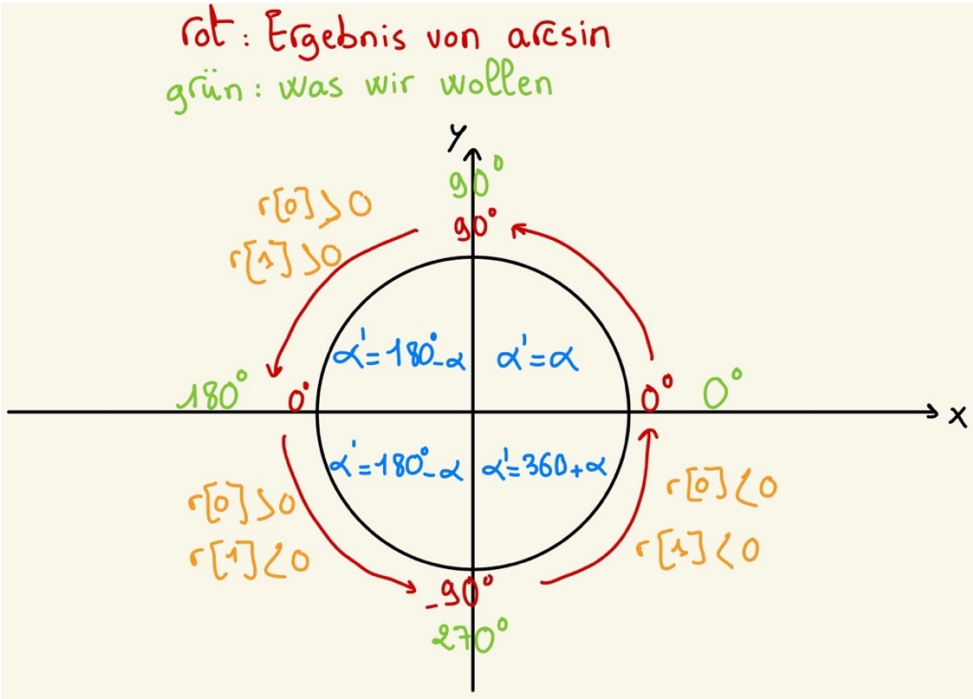


Der mit arcsin berechnete Winkel liegt im Bereich -90°;90°. Wir rechnen deswegen diesen um damit er im Bereich von 0° bis 360° liegt. Die Umrechnung erfolgt so:

- Zuerst entscheiden wir uns, wo wir 0° haben wollen. (beliebig und die Umrechnung hängt dann davon ab)
- Ein Test durchführen: die Koordinaten des Richtungsvektors ausgeben lassen während wir den Marker drehen. Und wir

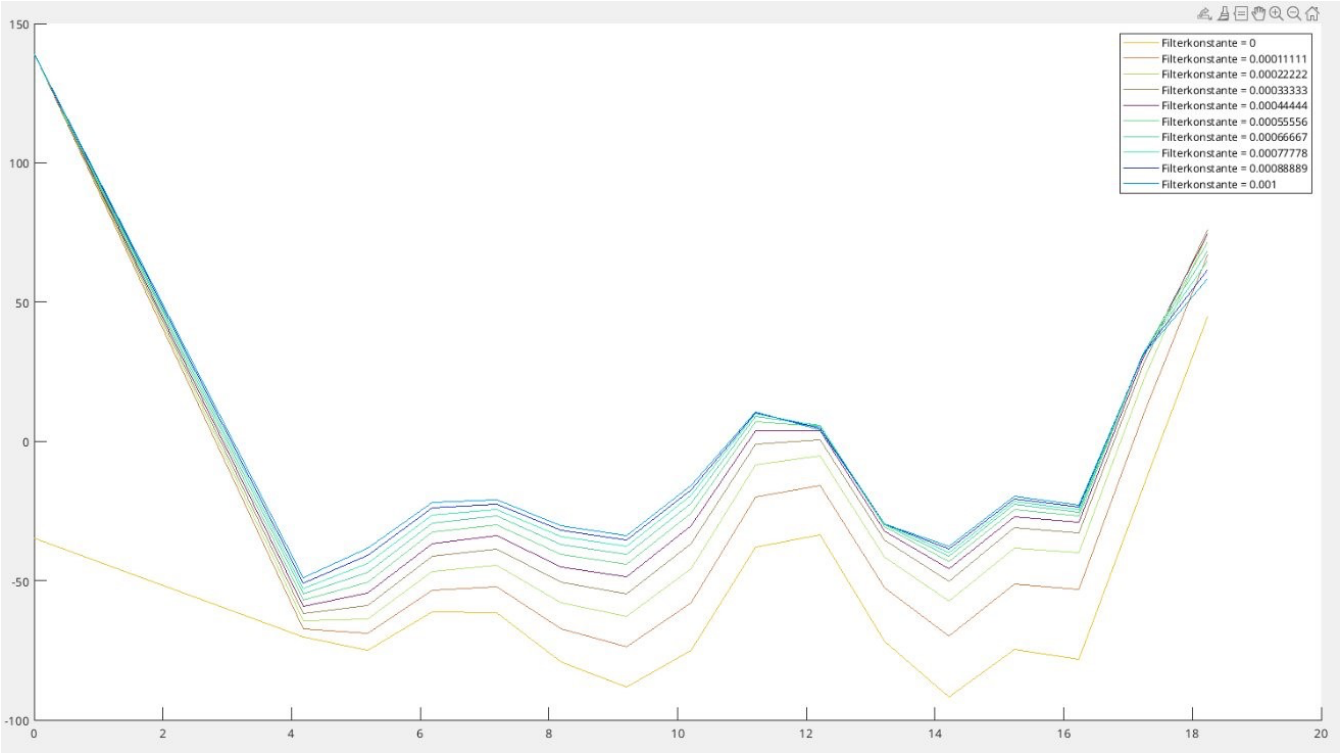
beobachten das Vorzeichen der beiden Koordinaten in jedem Quadranten. Daraus können wir dann umrechnen.

Beispiel:



Simulationen zur Filterkonstante (von Lukas)

Unser Ziel war es, dass falls sich die Filterkonstante als unpassend erweist, nachfolgende Gruppen diese ohne Probleme anpassen können. Deshalb haben wir in Matlab ein paar Simulationen erstellt um das Verhalten der Filterkonstante zu simulieren.
Die Simulation verwendet von uns gemessene Gyro, Kamera und Drift Werte und bestimmt daraus einen neuen Winkel, genauso wie es die
get_heading _deg Funktion tut. Dies geschieht dann für verschiedene Filterkonstanten. Wir haben hier ein paar vorgegeben man kann diese aber gerne ändern.



Die Grafik die bei der Simulation erstellt wird zeigt die Differenz zwischen dem Winkel der Kamera und dem berechneten Winkel. Die ersten 4 Sekunden der Simulation sind zu ignorieren. Wir haben versucht das zu ändern aber haben keine schöne Lösung gefunden.