Programmentwurf
Künstliche Intelligenz

Sensorfusion
Autonomes Fahren

Julien Steininger, Maximilian Wilhelm

19.12.2021

DHBW Stuttgart TINF19E

Künstliche Intelligenz

Prof. Dr. Dirk. Reichardt

Inhaltsverzeichnis

Einleitung

Ziel der Aufgabe ist es, auf Basis von Messungen ein vorausfahrendes Fahrzeug zu kategorisieren. Dazu gibt es die vier vordefinierten Fahrzeugklassen PKW, LKW, Motorrad und Fahrrad.
Die Messungen beinhalten lediglich Informationen zur Eigengeschwindigkeit, Angaben zur gemessenen Höhe und Breite und dem Abstand zum vorausfahrenden Fahrzeug.
Die Messungen sind größtenteils ungenau, hierzu zählt die größte Problematik, der Abstandsmesser.
Durch diese Fehleranfälligkeit sind die späteren Berechnungen von Geschwindigkeit und Beschleunigung stark betroffen, dementsprechend wird das Ausschlussverfahren teilweise verfälscht.

Grundsätzlicher Aufbau

Das Skript geht die CSV Datei Zeile für Zeile durch. In einer späteren Echtzeitanwendung würde jede Zeile dem Empfang eines Datensets zu einem Zeitpunkt entsprechen.
Für jedes Datenset werden dann das Seitenverhältnis der Bounding Box, die Geschwindigkeit und die Beschleunigung gemessen. Um Messfehler zu korrigieren, wurde hier mit Durchschnitten gearbeitet (mehr um Kapitel Messungenauigkeiten und Methoden).
Mit Hilfe der berechneten Werte findet dann für jedes Datenset eine Gewichtung nach in der Aufgabenstellung geforderten Parametern statt (mehr im Kapitel Methoden bei zugehörigen Funktionen probabilityFrom[…]). Diese Gewichtung wird dann mit vorher erkannten Wahrscheinlichkeiten kombiniert.
So ist mit zunehmender Zeit meist eine Klassifizierung mit hoher Wahrscheinlichkeit möglich.

Begründung von Entwurf und Umsetzung

In den folgenden Kapiteln werden die Struktur der Lösung, sowie die Gründe, die zu den getroffenen Entscheidungen geführt haben, erläutert.

Wahl der Tools

Zur Lösung der Aufgabe wurde die Programmiersprache Python und ein Jupyter Notebook verwendet.
Hierzu gab es Vorkenntnisse bei beiden Studenten durch verschiedene Studienfächer.
Zusätzlich gab es während der betroffenen Vorlesung, Künstliche Intelligenz, bereits ein Beispiel zur Evidenztheorie mit Python. Daher fiel die Suche nach passenden Bibliotheken weitgehend weg.
Zuletzt ist Python relativ performant. Dies wäre für eine spätere Nutzung bei Echtzeitanwendungen relevant.

Gewichten der Berechneten Werte

Zur Gewichtung haben wir auf die Bibliothek pyds zurückgegriffen, die bereits in der Vorlesung vorgestellt wurde.
Hier sind alle wichtigen Funktionen für die Evidenztheorie vorhanden, darunter das Basismaß als MassFunction und Demptsters Regel als combine_conjunctive.
Die Gewichtung findet in den drei Funktionen probabilityFrom[…] statt (Beschreibung im Kapitel Methoden).

Kombinieren der Gewichtungen

Die Gewichtungen, die in den Funktionen probabilityFrom[…] errechnet werden, stellen lediglich Wahrscheinlichkeiten einzelner Aspekte dar. Deswegen werden sie anschließend zu einer einheitlichen Wahrscheinlichkeit kombiniert.
Die Kombination findet direkt in der Funktion iterateMeasurement statt. Die Kombination ist dabei zweigeteilt.
Zuerst wird eine Kombination der Berechnungen für Seitenverhältnis, Geschwindigkeit und Beschleunigung der aktuellen Iteration durchgeführt.
Im zweiten Schritt wird die Kombination der Werte dann mit dem in der Variable massResult gespeicherten Wert kombiniert und das Ergebnis erneut hinterlegt.

Einfluss der Distanz auf die Unsicherheit

In der Aufgabenstellung wurde definiert, dass die Sensoren bei größeren Entfernungen unzuverlässiger Arbeiten. Um diese Angabe umzusetzen, haben wir die Unsicherheit dynamisch durch den Abstand errechnet.
Für eine genaue Reflektion der tatsächlichen Verhältnisse fehlen Angaben über die Genauigkeit der Sensoren. Daher entstand die Annahme, dass die Sensoren bis zu einem Abstand von 20 Metern mit maximaler Zuverlässigkeit funktionieren. Erst danach wird die Unsicherheit größer.
Auf der anderen Seite wurde eine Obergrenze der Unsicherheit bei 60 Metern defniert, ab der die Sensoren gleichmäßig mit einer Basisunsicherheit von 0.5 bewertet werden.
Zusätzlich wird für die Beschleunigung eine Erhöhung der Unsicherheit von 0.3 vorgenommen. Grund hierfür ist die weiterhin zu große Ungenauigkeit der erkannten Beschleunigung.

Handling von fehlenden Messwerten

Das Skript enthält keine vollständige Fehlerbehandlung. Jedoch werden bestimmte Fehler abgefangen oder umgangen.
Hierzu gehören zum Beispiel das Fehlen einzelner Werte der gemessenen Distanz, die lediglich zu einer Berechnung des Durchschnitts durch weniger Werte und Aussetzen der Gewichtung führen.

Messungenauigkeiten

Die CSV Dateien enthalten starke Messungenauigkeiten. Gründe hierfür sind der Abstand, der nur auf halbe Meter gerundet angegeben ist. Weiter ist in der Aufgabenstellung gegeben, dass die Sensoren mit zunehmendem Abstand ungenauer werden.
Die Messungenauigkeiten betreffen hauptsächlich die Geschwindigkeit und Beschleunigung, das Seitenverhältnis liefert recht konstante Ergebnisse. Eine Korrektur wird also nur für Geschwindigkeit und Beschleunigung vorgenommen.
Um sehr starke, durch Messungen verursachte, Schwankungen für die Geschwindigkeit zu verhindern, werden statt der letzten beiden Abstände die Abstände der letzten halben Sekunde (fünf Messungen) genutzt. Dies wird durch eine Liste mit rotierenden Werten umgesetzt.
Durch die Nutzung der Durchschnittswerte der letzten halben Sekunde sind die Geschwindigkeiten deutlich konstanter. Nachteil der Korrektur ist eine leicht verzögerte

Erkenntnis der Messwerte. Für Systeme, die Sicherheitsrelevant sind, sollten daher zusätzliche Fehlermechanismen eingeführt werden.
Für die Berechnung der Beschleunigung werden direkt die Werte der Durchschnittsgeschwindigkeit genutzt. Damit sind grundsätzlich keine weiteren Korrekturen nötig.
Da die Ungenauigkeit der Beschleunigung weiterhin sehr groß ist, wurde beschlossen, eine weitere Korrektur durchzuführen. Hierfür wird die Beschleunigung erneut in einem Array zwischengespeichert. Dadurch wird die Beschleunigung als Durchschnittswert über die letzte halbe Sekunde angegeben.

Erläuterung der Ausgabe

Das Skript ermöglicht die Ausgabe einiger hilfreicher Werte. Standardmäßig ist hier nur die Ausgabe der errechneter Wahrscheinlichkeiten für die Kategorisierung des Fahrzeuges aktiviert.
Durch Angabe des Parameters debug=True für die Methode iterateMeasurement ist es möglich, für jede Iteration Ausgaben zu erhalten.

Bewertung der Umsetzung

Positiv zu bewerten ist, dass das Skript für alle Testdatensätze mit hoher Wahrscheinlichkeit eine Klassifizierung vornehmen kann. Meist ist dies bereits nach wenigen Sekunden möglich.
Ebenso gut funktioniert die Erkennung des Fahrzeugs aufgrund der Bounding Box. Hier sind in den Datensätzen bereits recht zuverlässige, konstante Werte vorhanden.
Zu ungenau funktionieren dagegen die Berechnung der Geschwindigkeit und der Beschleunigung. Gründe hierfür liegen in der beschriebenen Ungenauigkeit der Messwerte für den Abstand.
Korrekturversuche haben grundsätzlich starke Verbesserungen gebracht. Jedoch führt die Abstraktion zu einer Verzögerung der Erkennung von tatsächlichen Änderungen. Da das Skript für eine Kategorisierung gedacht ist, wurde dies als annehmbar bewertet.
Ebenso führt die wiederholte Abstraktion zur Verfälschung der tatsächlichen Messwerte. Vermutlich wäre hier eine zuverlässige Lösung mit weniger Abstraktion möglich gewesen.
Die Berechnung der Unsicherheit findet derzeit lediglich nach Distanz statt. Die Einordnung einer Bounding Box zu einem Gefährt ist dagegen streng kategorisiert.
Für eine Optimierung wäre es hier möglich, die Unsicherheit gegen Werte, die klar einer Kategorie zuzuordnen sind, zu verringern. Werte, die dagegen am Rande einer Kategorie

stehen, könnten mit höherer Unsicherheit gewertet werden. Dies wurde aus Zeitgründen nicht implementiert.

Fazit

Das Programm zur Sensorfusion im Bezug auf das autonome Fahren zeigt in seiner ersten implementierten Version gute Erfolge.
Allerdings stellte die Berechnung der Geschwindigkeit und Beschleunigung eine große Schwierigkeit beim Programmieren dar. Da das Skript auch mit fehlerhaften Daten aus dem realen Verkehr umgehen können muss, sind viele Anpassungen vonnöten. Das perfekte Ergebnis zu erzielen scheint hier fast unmöglich.
Positiv dagegen aufgefallen ist der geringe Zeitaufwand der Implementierung der Klassifizierung selbst. Aufgrund der gewählten Bibliotheken war hier nur wenig eigene Programmierarbeit nötig.
Die Berechnung einigt sich bereits nach wenigen Schritten auf einen möglichen Fahrzeugtyp, sodass sich das Programm in nur maximal einer Sekunde zu mindestens 90 Prozent sicher ist, den korrekten Fahrzeugtypen gefunden zu haben.
Diese Wahrscheinlichkeit stieg sogar in drei von vier getesteten Fällen auf eine Wahrscheinlichkeit von 99,9 bis 100 Prozent, trotz implementierten Unsicherheiten, an.
Dementsprechend ist aktuell eine hohe Sicherheit des Programmes versprochen.
Um eine abschließende Bewertung für die Zuverlässigkeit des Skriptes vorzunehmen, sind jedoch mehr Tests mit weiteren Datensätzen nötig.

Dokumentation

Folgend ist eine Dokumentation der verwendetet CSV-Daten, der definierten globalen Variablen und der geschriebenen Methoden.

Die Daten und ihre Nutzung

Zum Kategorisieren der Fahrzeuge stehen uns zum Testen während der Entwicklungszeit CSV Dateien zur Verfügung.

Bezeichnung	**Beschreibung**
t	Entspricht dem Zeitpunkt, der sich in Abhängigkeit zu dem Beginn der Messung anfängt zu erhöhen. Die Einheit des Zeitablaufes entspricht s (Sekunde).
BBox_Hoehe	Die gemessene Höhe der Bounding Box des vorausfahrenden Fahrzeuges.
BBox_Breite	Die gemessene Breite der Bounding Box des vorausfahrenden Fahrzeuges.
Eigengeschwindigkeit	Spiegelt die aktuelle geschwindigkeit des hinteren Fahrzeuges wider. Die Einheit der Eigengeschwindigkeit entspricht m/s (Meter / Sekunde).
Abstand	Stellt den Abstand zwischen den beiden betroffenen Fahrzeugen dar. Die Einheit des Abstandes entspricht m (Meter).

Zusätzliche Nutzung der Bounding Box für LKW Kategorisierung

Über die explizit geforderten Gewichtungen hinaus wurden die Werte mit gleichen Seitenverhältnissen auf einen LKW gewertet. Dieser Zusatz erzielt eine Erhöhung der Zuverlässigkeit und Korrektheit der Kategorisierung.

Globale Variablen

Name	**Beschreibung**
speedArray	Speichert die letzten fünf abgerufenen Geschwindigkeiten in einem Array ab. Dadurch kann eine abstraktion der Daten stattfinden, um realitätsnähere Geschwindigkeitswerte zu berechnen.
avgSpeedArray	Speichert die letzten fünf abgerufenen Durchschnittsgeschwindigkeiten in einem Array ab. Dadurch kann eine abstraktion der Daten stattfinden, um realitätsnähere Beschleunigungswerte zu berechnen.
accArray	Beinhaltet die letzten fünf Beschleunigungswerte. Diese werden mit Durchschnittsgeschwindigkeiten berechnet. Ermöglicht zusätzliche Abstraktion der Beschleunigung durch Nutzung des Durchschnitts.

Methoden

Name(Parameter), return	**Beschreibung**
calcBoundingRatio(height, width) return: ratio	Berechnet das Verhältnis von Höhe zu Breite der Bounding Box.

calcSpeed(oldOwnSpeed, newOwnSpeed, oldDistance, newDistance) return: avgSpeed	Berechnet und speichert die neue Geschwindigkeit des vorausfahrenden Fahrzeuges anhand von zwei Datensätzen. Hierfür werden zwei Eigengeschwindigkeiten benötigt, um eine approximierte Geschwindigkeit des Zeitintervalls zu berechnen. Zusätzlich werden die Abstände zwischen den Fahrzeugen benötigt, um die Geschwindigkeitszunahme oder -abnahme des vorderen Fahrzeuges zu berechnen. Gibt anschließend die Durchschnittsgeschwindigkeit der letzten 0.5 Sekunden zurück. Die Formel für die Berechnung ist gegeben durch $v = \Delta s/\Delta m$ mit v entspricht Geschwindigkeit in m/s, s entspricht Strecke in m und t entspricht der Zeit in s. Durch die Eigengeschwindigkeit muss diese Formel abgeändert werden zu $v = \Delta s/\Delta m + v_e$ mit v_e entspricht der Eigengeschwindigkeit in m/s.
calcAcceleration() return: avgAcc	Berechnet die Beschleunigung des vorausfahrenden Fahrzeuges. Aufgrund vieler Messfehler bezieht sich diese Methode auf ein globales Array, in dem Durchschnittswerte der Geschwindigkeit abgespeichert werden. Die Beschleunigung wird mit vorherigen Messwerten nochmals abstrahiert, um schließlich eine realitätsnähere Beschleunigung zu bekommen. Berechnung der Beschleunigung durch folgende Formel: In der Grundform ist diese gegeben durch $a = \Delta v/\Delta t$ mit a entspricht Beschleunigung in m/s^2.
calcOmegaFromDistance(distance) return: omeg	Berechnet das Omega in Abhängigkeit von dem Abstand zwischen den Fahrzeugen.

	Da die Messwerte an sich schon Ungenauigkeiten aufweisen, wurde ein fester Omega-Wert definiert. Dieser soll mit einer höheren Entfernung des zu messenden Fahrzeuges steigen. Hierfür wurde eine lineare Zunahme definiert, die bei einer Entfernung von 60 Metern den maximalen Wert erreicht und für weiter entferntere Fahrzeuge stagniert.
probabilityFromRatio(ratio, distance) return: mass	Berechnet die Wahrscheinlichkeiten für die einzelnen Fahrzeugtypen anhand des übergebenen Höhe-Breite-Verhältnis der Bounding Box. Die Verhältniswerte für die einzelnen Typen wurden vorher fest definiert und anhand dessen ein Ausschlussverfahren definiert.
probabilityFromSpeed(speed, distance) return: mass	Berechnet die Wahrscheinlichkeiten für die einzelnen Fahrzeugtypen anhand der übergebenen Geschwindigkeit des vorausfahrenden Fahrzeuges. Die Intervalle der möglichen Geschwindigkeiten wurden innerhalb der Aufgabenstellung übergeben. Zusätzlich wurden Optionen für niedrigere Geschwindigkeiten umgesetzt.
probabilityFromAcc(acc, distance) return: mass	Berechnet die Wahrscheinlichkeiten für die einzelnen Fahrzeugtypen anhand der übergebenen beschleunigung des vorausfahrenden Fahrzeuges. Die Aufgabenstellung übergab nur die Anforderungen für ein Motorrad. Die Implementierung für weitere Fahrzeugtypen ergab keinen Sinn, da alle Fahrzeugtypen niedrige Beschleunigungen umsetzen können.

saveSpeed(speed)	Managed die Verwaltung des Arrays für die Geschwindigkeiten. Fügt die aktuell gemessene Geschwindigkeit hinzu. Entfernt den ersten noch gespeicherten Wert. Genutzt für die Berechnung der Durchschnittsgeschwindigkeit.
saveAvgSpeed(speed)	Managed die Verwaltung des Arrays für die Durchschnittsgeschwindigkeiten. Fügt die zuletzt berechnete Geschwindigkeit hinzu. Entfernt den ersten noch gespeicherten Wert. Genutzt für die Berechnung der Beschleunigung.
saveAcc(acc)	Managed die Verwaltung des Arrays für die Beschleunigung. Fügt die aktuell gemessene Beschleunigung hinzu. Entfernt den ersten noch gespeicherten Wert. Genutzt für die Berechnung der Durchschnittsbeschleunigung, um die Messschwankungen weiter zu reduzieren.
iterateMeasurments(data, debug)	Koordiniert das Skript. Iteriert über alle Elemente eines Dataframes. Ruft anschließend Methoden auf, um Werte und Gewichtungen zu berechnen. Gibt optional pro Iteration Informationen aus. Gibt am Ende Kategorisierung aus.