1. **Vertiefungsaufgabe: Schildererkennung**

Als Vertiefungsaufgabe haben wir uns entschieden, Schildererkennung zu implementieren. Unser Ziel war es drei verschiedene Schilder zu erkennen, auf die das Auto dann reagiert. Im Folgenden werden die zwei Ansätze erklärt, die wir für die Realisierung verfolgt haben.



Geschwindigkeit verringern auf 1km/h für 10s

Auf die andere Spur wechseln

Auf Höhe des Schildes 2s anhalten

* 1. **Erster Ansatz: Find-Object**

Unser erster Ansatz für die Schildererkennung war es, das ROS-Packet Find-Object von IntRoLab[[1]](#footnote-2) zu benutzten. Dieser ermöglicht es, einfach, mit einer grafischen Benutzeroberfläche, Referenzbilder zu laden, welche im Live-Bild der Kamera gesucht werden. Man kann dafür einfach verschiedenen Suchalgorithmen (SIFT, SURF, FAST, BRIEF u.v.m) und Parameter ausprobieren.



**Abbildung 5.?**: Grafische Benutzeroberfläche von Find-Object, wo sich Referenzbilder links, das Kamerabild in der Mitte und die Parametern rechts, befinden.

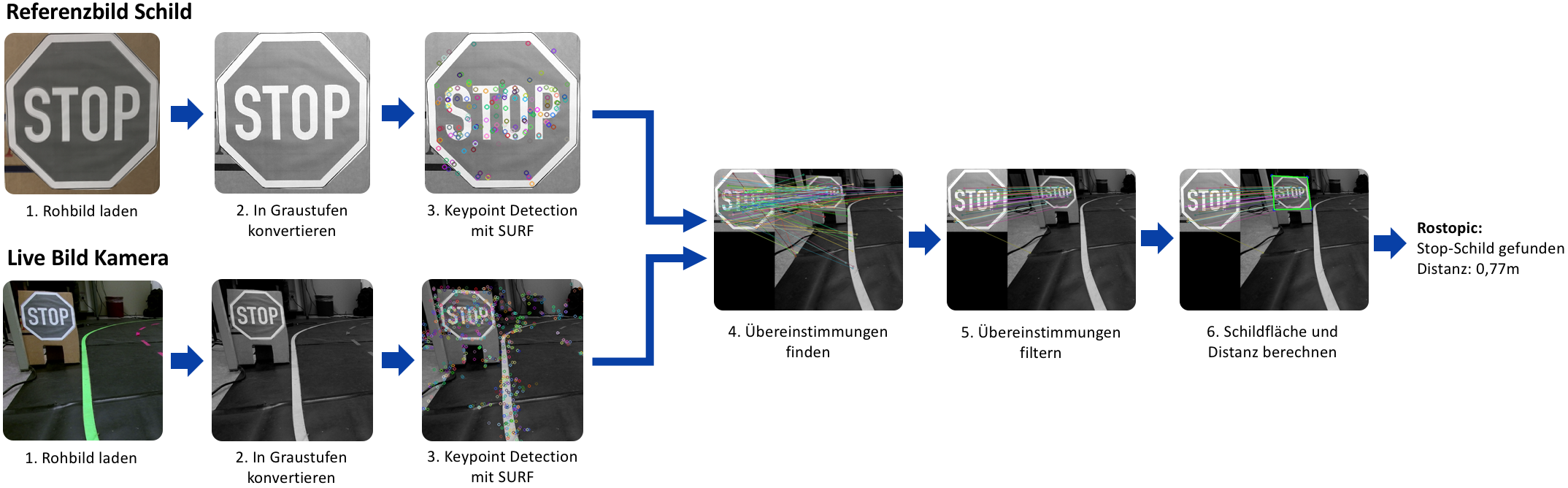
**Probleme mit diesem Ansatz**

Als wir die Schildererkennung und den Regler für die Steuerung gleichzeitig laufen ließen, begann das Auto stark zu schwingen und es war diesem nicht mehr möglich die Spur korrekt zu halten. Die Performanz-Probleme konnten wir auf die CPU zurückführen, welche alleine durch die Verwendung der Kinect-Kamera zu ca. 50 % ausgelastet war. Außerdem ist das Paket Find-Object mit seinen umfangreichen Funktionen (von denen wir nicht alle benötigten), sehr Ressourcen verbrauchend. So musste ein anderer Ansatz her, mit einem simplen, überschaubaren Codeumfang her.

* 1. **Finaler Ansatz: Schildererkennung mit SURF**

Mit dem ersten Ansatz hatten wir experimentell ermittelt, dass der Suchalgorithmus SURF (Speeded Up Robust Features[[2]](#footnote-3)) für unser Anwendungsszenario am schnellsten und robustesten funktioniert. Schilder werden nämlich erkennt, unabhängig davon welche Größe und Orientation sie haben.

Als Code-Basis haben wir den OpenCV-Beispielcode *SURF\_FLANN\_matching\_homography\_Demo.cpp[[3]](#footnote-4)* verwendet. Diesen haben wir für unsere Zwecke angepasst, damit es mit ROS und der Kinect-Kamera kompatibel ist, und haben Funktionen wie Multi-Objekt-Erkennung hinzugefügt.



**Abbildung 5.?**: Schildererkennung-Pipeline

Um die Funktionsweise unseres Ansatzes zu erklären, gehen wir die Schilderkennung-Pipeline Schritt für Schritt durch:

**Schritt 1:** Das Referenzbild des Schildes und das aktuellste Bild der Kamera werden geladen.

**Schritt 2:** Beide Bilder werden in Graustufen konvertiert. Denn Farbinformationen sind für diese Art der Objekterkennung irrelevant, es interessieren uns nur Helligkeitsverläufe im Graustufenbild.

**Schritt 3:** Mit dem Detektor SURF wird in beiden Bildern separat nach markanten Punkten, (auch Keypoint oder Point-of-Interest genannt), gesucht. Jeder Keypoint wird als ein Vektor von Eigenschaften (Koordinaten, Durchmesser, Winkel, Octave, id …) beschrieben.

**Schritt 4:** Nun wird nach Übereinstimmungen zwischen den Keypoints des Referenzbildes und des Kamerabildes gesucht. In diesem Schritt wird für jeden Keypoint des Referenzbildes, zwei Keypoints aus dem Kamera-Bild gesucht, welche die größte Gemeinsamkeit haben. Dafür wird der Nearest-Neighbor Algorithmus von FLANN[[4]](#footnote-5) (Fast Library for Approximate Nearest Neighbors), genutzt.

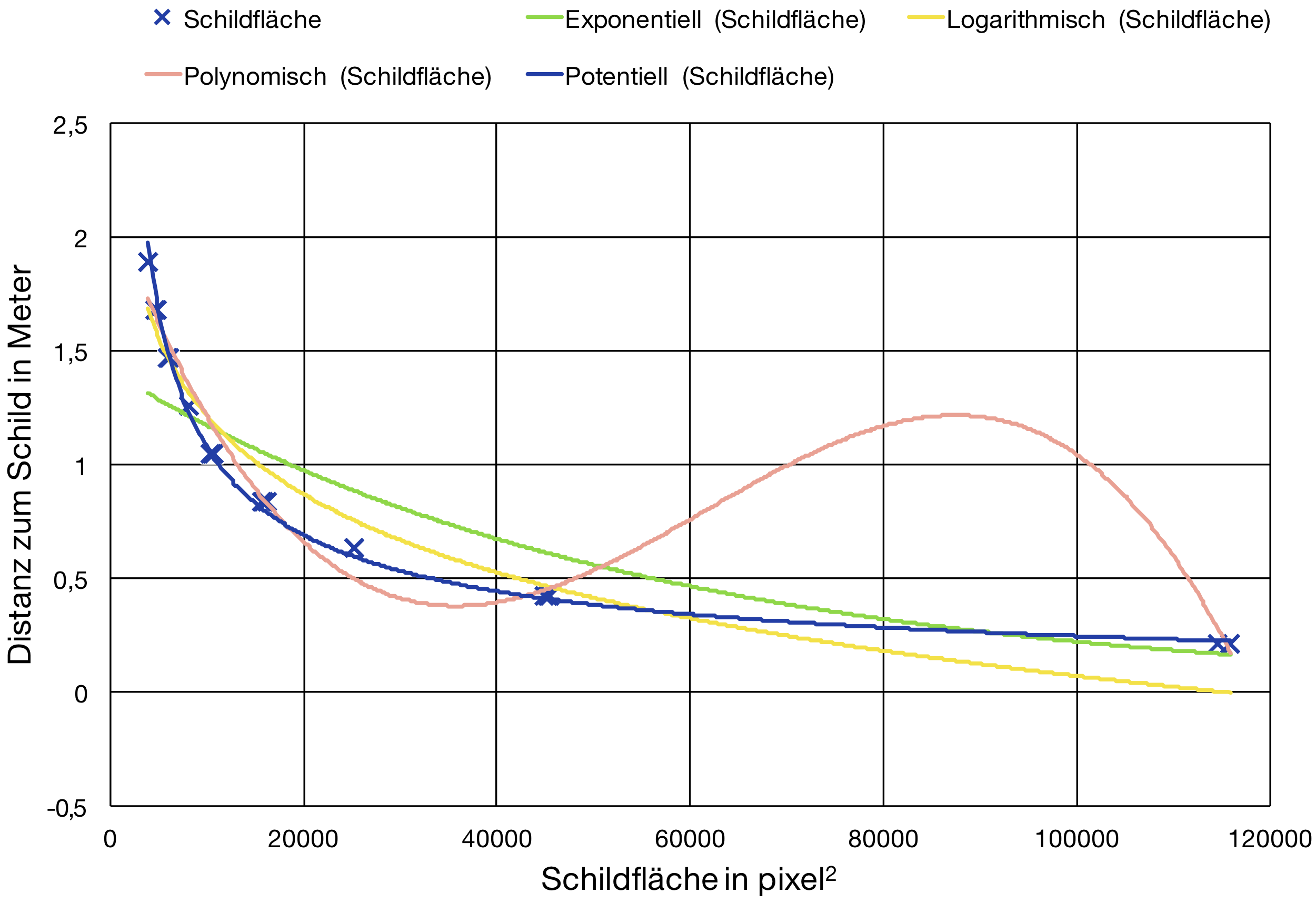
**Schritt 5:** Anschließend werden die Übereinstimmungen gefiltert, so dass wir nur noch die besten Übereinstimmungen haben. Dazu verwenden wir den Lowe’s-Ratio-Test[[5]](#footnote-6), dieser schaut sich die Distanz, zwischen dem Keypoint des Referenzbild und dem des Kamerabild an, und filtert Übereinstimmungen aus, welche den Vorgegebenen Threshold nicht erfüllen. Mit Distanz ist keine physische Distanz gemeint, sondern die Euklidische Distanz von zwei Keypoint Vektoren. Dieser gibt den Wert der Ähnlichkeit an, um so kleiner die Distanz, um so ähnlicher sind sich die Keypoints.

**Schritt 6:** Wenn ein Schild gefunden wird, wird die physische Distanz vom Schild zum Auto ermittelt und diese wird auf einer ROS-Topic ausgegeben:  
Wenn der Stopp-Schild erkannt wird → /sign\_detection\_node/StopSign  
Wenn der Geschwindigkeitsbegrenzung-Schild erkannt wird→ /sign\_detection\_node/LaneSign  
Wenn der Spurwechsel-Schild erkannt wird → /sign\_detection\_node/SpeedSign

**Distanz vom Auto zum Schild ermitteln**

Um die Distanz vom Auto zum Schild zu ermitteln, war unsere erste Idee, das Tiefenbild der Kinect zu nutzen, das hat aber zu viel CPU-Leistung verbraucht, so das die Steuerung nicht mehr richtig funktioniert hat, und das Auto Schlangenlinien gefahren ist.

Also sind wir auf die Lösung gekommen, die Distanz anhand der Fläche des gefundenen Bildes zu errechnen. Dafür haben wir für verschiedene Distanzen geschaut, wie groß die Fläche ist und haben mittels Regressionsanalyse nach einer Funktion gesucht, welche die Messpunkte möglichst gut beschreibt. Wie man es in Abbildung 5.? entnehmen kann, werden die Messpunkte am Besten von der Potenzfunktion beschrieben:



**Abbildung 5.?**: Messpunkte für Verhältnis Fläche – Distanz und verschiedene Regressionsfunktionen

**5.3 Ausblick – Verbesserungsmöglichkeiten**

Es gibt verschiedene Möglichkeiten die Schildererkennung zu verbessern:

Man könnte überlegen ein anderen Deskriptor als SURF zu benutzen, dieser ist zwar für Forschungszwecke lizenzfrei aber nicht für kommerzielle Zwecke. Womöglich ist der lizenzfreie (BSD-Lizenz) Deskriptor ORB[[6]](#footnote-7) die bessere Lösung.

Bei der aktuellen Implementierung wird jedes Mal wenn ein neues Kamera-Bild reinkommt, beim Referenzbild des Schildes von neuem die Keypoints gesucht. Da diese sich aber nicht verändern, könnte man diesen Schritt auslagern, sodass er nur ein Mal ausgeführt wird.

Außerdem könnte Schildererkennung erweitern werden, also, dass mehr als nur die drei Schilder erkannt werden.

1. Labbé, Mathieu: Find-Object. https://introlab.github.io/find-object/ [↑](#footnote-ref-2)
2. Herbert Bay, Tinne Tuytelaars, and Luc Van Gool: SURF: Speeded Up Robust Features.

   http://www.vision.ee.ethz.ch/~surf/eccv06.pdf [↑](#footnote-ref-3)
3. Feature Matching with FLANN https://docs.opencv.org/3.4.4/d5/d6f/tutorial\_feature\_flann\_matcher.html [↑](#footnote-ref-4)
4. FLANN - Fast Library for Approximate Nearest Neighbors https://www.cs.ubc.ca/research/flann/ [↑](#footnote-ref-5)
5. Lowe, D.: Distinctive image features from scale-invariant keypoints, cascade filtering approach.

   <https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf> [↑](#footnote-ref-6)
6. E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF" [↑](#footnote-ref-7)