Experimental Setup for Testing and Evaluation of Kalman Filter Configurations

Versuchsaufbau zum Test und Evaluierung von Kalman Filter Konfigurationen

Stefan Hensel*, Marin B. Marinov**, Christopher Kupitz*, Dimitre Trendafilov **

* University of Applied Sciences Offenburg, Department for Electrical Engineering,
Badstraße 24, D-77652 Offenburg, Germany, stefan.hensel@hs-offenburg.de
** Technical University of Sofia, Faculty of Electronic Engineering and Technologies, Department of Electronics,
8, Kliment Ohridski Blvd., BG-1756 Sofia, Bulgaria, mbm@tu-sofia.bg

***Abstract* —** Positioning mobile systems with high accuracy is a prerequisite for intelligent autonomous behavior, both in field robotics and in industrial environments. This paper describes the setup of a robotic platform and its use for testing and evaluating Kalman filter configurations. The setup was implemented using a Husky A200 mobile robot and a light detection and ranging (LiDAR) sensor. Five different scenarios were devised to verify the proposed setup. With these, the filters were tested for their performance in terms of position determination accuracy.

***Zusammenfassung* —** Die Positionierung mobiler Systeme mit hoher Genauigkeit ist eine Voraussetzung für intelligentes autonomes Verhalten, sowohl in der Feldrobotik als auch in industriellen Umgebungen. Dieser Beitrag beschreibt den Aufbau einer Roboterplattform und ihre Verwendung für den Test und die Bewertung von Kalman-Filter-Konfigurationen. Der Aufbau wurde mit einem mobilen Roboter Husky A200 und einem LiDAR-Sensor (Light Detection and Ranging) realisiert. Zur Verifizierung des vorgeschlagenen Aufbaus wurden fünf verschiedene Szenarien ausgearbeitet. Mit denen wurden die Filter auf ihre Leistungsfähigkeit hinsichtlich der Genauigkeit der Positionsbestimmung getestet.

I. Einführung und Motivation

Dieses Kapitel gibt einen Überblick über die in Hard- und Software aufgebaute Plattform. Der als Versuchsträger eingesetzte Roboter, die Sensorik und die notwendigen Anpassungen werden ebenso erläutert wie die verteilten Rechnerkomponenten und die zur Kommunikation und Steuerung eingesetzte Software [1].

In diesem Kapitel wird der strukturelle Aufbau des Robotersystems erklärt. Aufgeteilt in den Hardwareaufbau, der die Vernetzung der verschiedenen Rechner untereinander und der Sensoren des Huskys beschreibt. Und in den Softwareaufbau der den Nachrichten Austausch innerhalb des Systems, zwischen den Gerätetreibern und Softwarepakete beschreibt.

Eine der wichtigen Anforderungen, die ein mobiles Robotersystem erfüllen muss, ist die Frage „Wo befinde ich mich gerade?“ beantworten zu können. Insbesondere, wenn man sich den Gedanken der selbst fahrenden Autos vor Augen hält. Die Antwort auf diese Frage kann mithilfe von exterozeptiven Sensordaten ermittelt werden. Jedoch befinden wir uns in einer sehr dynamischen und hochkomplexen Welt, in der Sensoren nicht perfekt und ihre Messungen fehleranfällig sind. Durch Fusionierung der Daten von mehreren Sensoren ist es möglich eine Gesamtpositionsschätzung zu erhalten, deren Fehler geringer ist, als dies mit einem einzelnen Sensor isoliert möglich wäre [2]. Es ist häufig der Fall, dass eine größere Menge von Sensordaten genauere Positionsschätzungen erzeugt [3].

II. Systemaufbau

A. Hardwareaufbau des Robotersystems

Dieser Kapitel behandelt den Hardwareaufbau des Robotersystems, der die Vernetzung der verschiedenen Rechner untereinander und der Sensoren des Huskys beinhaltet. Das verwendete Robotersystem besteht aus einem verteilen Computersystem. Das bedeutet, dass die verschiedenen Sensoren nicht an einem Rechner angeschlossen sind, sondern auf verschiedene Rechner aufgeteilt sind. Diese Rechner sind wiederum über ein Computernetzwerk miteinander verbunden. Grafisch abgebildet in einem Strukturdiagramm in Abb. 1.

Das Computernetzwerk beinhaltet einen TP-Link Router, einen mobilen Laptop zur Datenaufzeichnung und späteren Datenauswertung der über das Roboter eigene WLAN angebunden ist und zwei auf/in dem Husky A200 Chassis verbaute Rechner (in Abb. 1. als Husky PC und Pokini bezeichnet). An diesen Computern sind in diesem Projekt vier Sensoren angeschlossen. Am Pokini ist die zur inertialen Messwert Aufnahme nötige XSens IMU, sowie ein RTK fähiger GNSS-Empfänger der Marke u-blox (in Abb. 1 u-blox RTK GPS genannt) angeschlossen, dessen Daten durch ihre ausreichende hohe Genauigkeit als Referent der wahren Position (engl. ground truth) des Roboters verwendet werden. Am Husky PC, der innerhalb des Husky Chassis verbaut ist, ist die MCU des Huskys mittels serieller Schnittstelle angeschlossen. Über diese werden die Motoren gesteuert und die Daten der Odometrie empfangen. Zusätzlich ist am Husky PC ein einfacher GNSS-Empfänger (in Abb. 1 GPS-Modul Arduino genannt) angebunden.

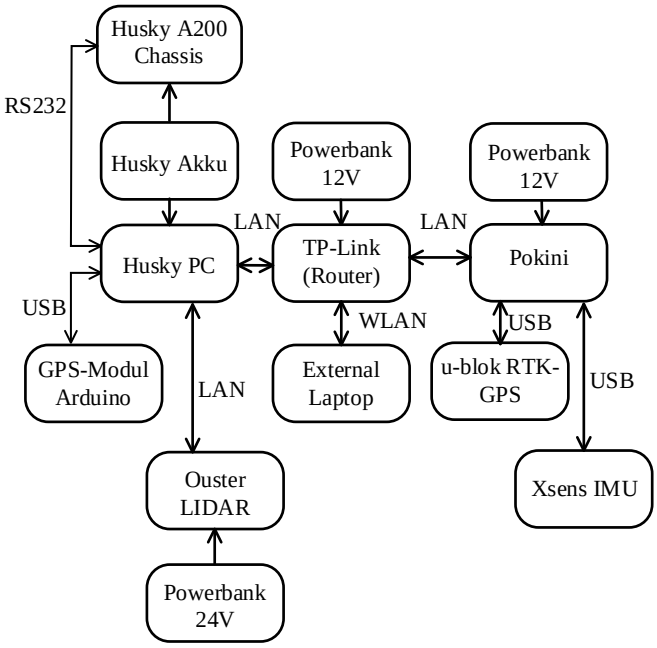

Abb. 1. Hardwareaufbau des Verteilten Robotersystems

Der in diesem Projekt verwendete Roboter ist der **Husky A200** des Unternehmens Clearpath Robotic Inc. Der Husky A200 ist ein robustes, unbemanntes Bodenfahrzeug (engl. unmanned ground vehicle, UGV) für den Innen- und Außenbereich. Durch seine Modifizierbarkeit des Aufbaues ist er sehr für Forschungs- und Rapid-Prototyping-Anwendungen geeignet ist. Alle Funktionen des Huskys werden über eine serielle RS232-Schnittstelle von einem, im Inneren des Chassis verbauten Rechners gesteuert. Auf diesem läuft der Clearpath Robotics offiziell unterstützte ROS-Knoten im Paket clearpath_base.

Abb. 2. Verwendeter Roboter mit Sensoraufbau und verwendetem Laptop

Real Time Kinematics (RTK) ist eine Art des DGNSS und liefert eine hochgenaue Positionsbestimmung unter Verwendung einer genau eingemessenen Basisstation. Das Verfahren nutzt nicht wie das Standard GNSS Verfahren die gesendeten Informationen der Satelliten, sondern die Trägerwelle selbst, mit der die Information übertragen wird (vereinfacht dargestellt in Abb. 3). Diese Trägerphasen werden anhand ihrer Wellenlänge korreliert. Sobald eine Korrelation gefunden ist, tritt ein sog. RTK-FIXZustand ein. Ab diesem Moment wird können hochpräzise Positionsdaten berechnet werden. Unter der Voraussetzung von einer guten Signalstärke von mindestens fünf Satelliten ist eine Positionsbestimmung auf 1 cm bis 2 cm für die Lage (x, y) und bis auf 2 cm bis 3 cm für die Höhe möglich [4, 5].

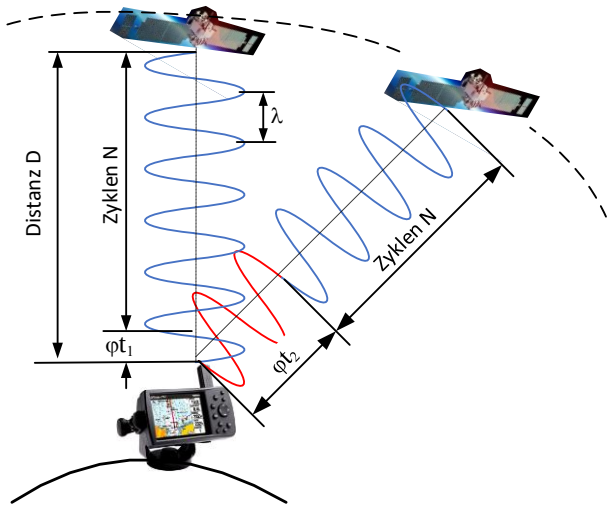

Abb. 3. Vereinfacht dargestellte Funktionsweise eines Real Time Kinematic Systems (Adaptiert von [4])

Eine **Inertiale Messeinheit** (engl. „inertial messurement unit“, IMU) ist eine Kombination mehrerer Sensoren. Meist beinhaltet eine IMU-Beschleunigungssensoren, Drehraten-sensoren und Magnetfeldsensoren. Zur Abdeckung aller Freiheitsgrade im dreidimensionalen Raum, werden drei Sensorpakete orthogonal, auf drei Achsen, kombiniert. In Verbindung mit einer Signalverarbeitung lassen sich dadurch die Beschleunigung, die Geschwindigkeit und die Position im dreidimensionalen Raum bestimmen. Außerdem ist es damit möglich die Winkelgeschwindigkeit zu messen und die absolute Lage, des Systems zu berechnen. IMUs werden häufig in Inertialen Navigation Systemen (INS) verwendet. Beispielsweise in Robotersystemen, Flugzeugen, Raketen oder auch in Navigationssystem normaler PKW. Sie ermöglichen eine Positionsbestimmung, auch wenn beispielsweise ein absolutes Lokalisierungssystem, wie GNSS, ausgefallen oder nicht vorhanden ist.

In dieser Arbeit wird die IMU **MTI-20** der Firma XSens verwendet. Der Vorteil der IMUs der Firma XSens ist, dass sie leicht mit einem USB-Kabel mit dem Computer verbunden werden können. Außerdem beinhalten die IMUs Signalprozessoren die je nach Konfiguration, die Rohdaten, gefilterte Daten oder Sensorfusionierte Daten an den Rechner senden. Die Konfiguration der Hardware wird mittels der vom Unternehmen XSens bereitgestellten Software, MT Software Suite vorgenommen [6].

Für die XSens Produktreihen MT, MTI und MTi-G bestehen außerdem offiziell entwickelte ROS-Treiber [7] Die mit ihren Konfigurationsdateien auf das geforderte Verhalten angepasst werden können. Die Art der Daten und die Frequenz der gesendeten Daten und lässt sie hier einstellen. Zudem ist es möglich eine Kalibration einzustellen. Mit ihrer Hilfe lässt sich der Offset bestimmen, der die Genauigkeit der Daten deutlich erhöht.

B. Softwareaufbau

Das **Robot Operating System** (ROS) ist ein Softwareframework das auf den Betriebssystemen Linux, Windows oder MacOS aufbaut. Es wird seit 2013 von der Open Source Robotics Foundation (OSFR) und von nonprofit Vereinigung Open Robotics gepflegt und weiterentwickelt. Die Hauptbestandteile dieses Softwareframework ist die Bereitstellung einer Vielzahl von Gerätetreibern, Softwarealgorithmen, der Nachrichtenaustausch zwischen Softwarediensten auf einem Gerät oder netzwerkübergreifend und die Bereitstellung einer Programmierschnittstelle zur beliebigen Erweiterung durch eignen Software. Durch die häufige Verwendung in Robotersystemen in der Forschung oder im privaten Raum steht eine Vielzahl an Software-Tools und Bibliotheken online zur Verfügung. Zusätzlich bietet die Plattform die Möglichkeit eigene Softwarepackete in den Programmiersprachen C++ und Python, unter der Verwendung der ROS-Softwareschnittstelle (ROS API), hinzuzufügen.

Um beispielsweise Daten eines Sensortreibers in einem Algorithmus nutzen zu können ist ein Nachrichtenaustausch notwendig. Hierzu nutzt das System das Publisher-Subscriber-Kommunikationsprinzip. Der Datenaustausch zwischen oftwareknoten (engl. „Nodes“) findet hierbei über Themen (engl. „Topics“) statt. Nachrichten (engl. „Messages“) werden von einem Knoten in einem Thema publiziert und können von anderen Knoten abonniert werden.

Durch die wichtigsten verwendeten **Softwarekomponenten** wird das strukturelle Zusammenspiel der einzelnen Softwareapplikationen und Hardwaretreiber und deren Nachrichtenaustausch aufgezeigt. Da ROS einen netzwerkübergreifenden Datenaustausch ermöglicht, stellt es kein Problem dar, den bereits beschriebenen Aufbau zu verwenden. Es muss lediglich auf jedem der verwendeten Rechner das ROS-Framework installiert sein und auf einem dieser Rechner muss der ROS-core laufen. Der ROS-core ist die zentrale Instanz des Systems und verwaltet alle gestartet Nodes und vermittelt den Topic Empfängern auf welchem Rechner das gewünschte Topic versendet wird. Das ermöglicht das Senden und Empfangen aller ROS-Topic, von jedem Rechner des Netzwerks, ohne Einschränkungen. Zur Bereitstellung der benötigten Sensordaten wurden nachfolgende Treiber installiert.

nmea_navsat_driver: Dieser Softwarepaket bildet das Interface zu einem GNSS-Empfänger der seriell oder über eine Socket Verbindung NMEA-Sätze versendet. Hierbei werden die vom Empfänger versendeten NMEA-Sätze entschlüsselt und als *sensor_msgs/NavSatFix* unter dem Topic */gps/fix* versendet.

ublox_gps: Dieses Softwarepaket fungiert als einfacher Treiber für verschiedene u-blox Geräte. Es beinhaltet ROS Nodes für u-blox Empfänger und für die Entschlüsselung des binären UBX-Protokolls. Diese Nodes versenden die empfangene Positionsdaten als *sensor_msgs/NavSatFix* unter dem Topic */ublox/fix* und Geschwindigkeits- und Drehratendaten mittels *geometry_msgs/Twist* im Topic */ublox/fix_velocity*.

xsens_driver: Dieses Softwarepaket übermittelt die von einer XSens IMU empfangenen Daten an das ROS-System. Daten wie die globale Orientierung, die Winkelgeschwindigkeit und die lineare Beschleunigung werden in einer *sensor_msgs/Imu*
Nachricht an das Topic */imu/data* verwendet. Zudem wird mit einer *sensor_msgs/MagneticField* Nachricht Informationen über das vorherrschende Erdmagnetfeld im Topic */imu/mag* veröffentlicht. Die Einstellung welche Daten zur Ermittlung der globalen Orientierung verwendet werden, müssen in der von dem Unternehmen XSens bereitgestellten Software, MT Software Suite vorgenommen werden.
Zur Berechnung der Kalman-Filter wird ein bestehendes Softwarepaket zurückgegriffen. Das von Tom Moore entwickelte Softwarepaket *robot_localisation* ist ein offiziell von ROS unterstütztes Paket und ist bereits bei vielen ROS-Distributionen ein fester Bestandteil. Es ist eine Sammlung von Zustandsschätzungsnodes, von denen jeder eine Implementierung eines nichtlinearen Kalman-Filters, für Roboter, die sich im dreidimensionalen Raum bewegen, ist. Das Paket enthält zwei 15-dimensionale Implementierungen der Kalman-Filter. Einmal die *ekf_localization_node* und die *ukf_localization_node*. Des Weiteren enthält das Softwarepaket die *navsat_trtansvorm_node* die, die Integration von GNSS-Daten in einen der erwähnten Kalman-Filter Implementationen ermöglicht. Zusätzlich ist ein Softwareknoten programmiert worden, der die globale Orientierung des mobilen Robotersystems anhand der Empfangenen GNSS-Signale und die dazugehörige Varianz, publiziert. Hierzu wird die errechnete Orientierung in einer *sensor_msgs/Imu* Nachricht in dem Topic */gps/orientation* versendet.

Koordinatensysteme

In diesem Kapitel werden die nötigen Koordinatensysteme und ihre Zusammenhänge eingeführt. Zudem wird beschrieben, warum die einzelnen Koordinatensysteme vonnöten sind, und erklärt, was diese darstellen. Alle Sensoren eines Systems erfassen ihre Messungen bezogen auf ihre eigene Position und Orientierung zu dem gesamten System. Das bedeutet sie publizieren ihre Messergebnisse in ihren eigenen Koordinatensystemen, das zu dem Koordinatensystem der Roboterbasis translatiert oder rotiert sein kann. Um jetzt Daten eines Sensors auf das Basissystem herunterrechnen zu könne, muss die Translation und Rotation des Koordinatensystems des Sensors bezogen auf das Basiskoordinatensystem bekannt sein. Beispielsweise muss die Position und Rotation eines GNSS-Empfängers auf einem Roboter bekannt sein damit auf die Position des Basissystems geschlossen werden kann. Diesbezüglich liefert Open Robotics systeminterne Standards. Die für das Projekt wichtigen Standards sind die REP-103 und die REP-105. In der REP-103 werden die Standards für Einheiten und Orientierung der einfachen Koordinatensysteme behandelt. Die REP-105 spezifiziert die Namenskonventionen und die semantische Bedeutung von Koordinatensystemen in mobilen Roboterplattformen die ROS verwenden [8, 9]. Wichtige Koordinatensysteme für das Projekt sind das base_link-, odom- und map- Koordinatensystem. Diese sind wie folgt in REP-105 definiert:

base_link: Das Koordinatensystem mit der Bezeichnung *base_link* ist fest mit der Basis des mobilen Roboters verbunden. Diese beschreibt Daten aus der Sicht der mobilen Roboterplattform. Zudem werden Translation und Rotation starr verbauter Sensoren in diesem Koordinatensystem beschrieben. In REP-103 ist die bevorzugte Sensorkoordinatensystemausrichtung definiert.

odom: Das Koordinatensystem namens *odom* ist ein Weltfestes Koordinatensystem. Die Pose (Position und Orientierung des *base_link*-Koordinatensystems) einer mobilen Plattform im *odom*-Koordinatensystem kann im Laufe der Zeit unbegrenzt driften. Dieser Drift macht das *odom*-Koordinatensystem als langfristige globale Referenz unbrauchbar. Die Pose eines Roboters im *odom*-Koordinatensystem ist jedoch garantiert kontinuierlich, was bedeutet, dass sich die Pose einer mobilen Plattform im *odom*-Koordinatensystem immer reibungslos und ohne diskrete Sprünge entwickelt. Das *odom*-Koordinatensystem ist als genaue, kurzfristige lokale `Referenz nützlich, aber die Drift

macht ihn zu einer schlechten Referenz für langfristiges Planen und Handeln.

map: Das als *map* oder *map frame* bezeichnete Koordinatensystem ist ein weltfestes Koordinatensystem. Die Position einer mobilen Plattform relativ zum *map*-Koordinatensystem sollte im Laufe der Zeit nicht wesentlich abweichen. Die Translation zwischen *map*-Koordinatensystem und *odom*-Koordinatensystem ist nicht stätig, was bedeutet, dass sich die Position einer mobilen Plattform im *map*-Koordinatensystem jederzeit in diskreten Sprüngen ändern kann. Das *map*-Koordinatensystem ist als langfristige globale Referenz nützlich, aber diskrete Sprünge in Positionsschätzern machen ihn zu einer schlechten Referenz für lokales Erfassen und Handeln.

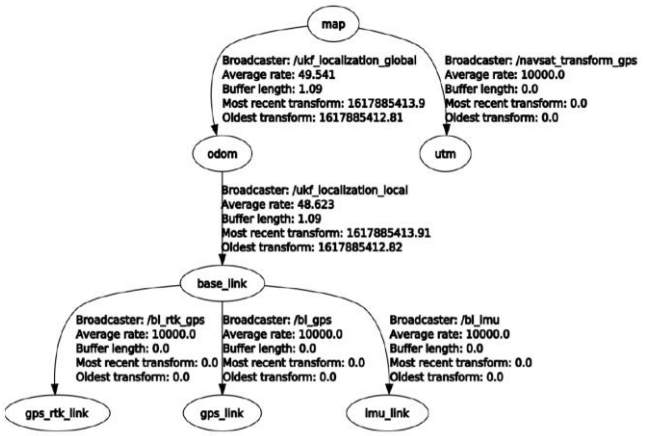

Abb. 4. Vereinfacht dargestellte Funktionsweise eines Real Time Kinematic Systems (ohne Darstellung einer Basisstation)

Abb. 4 beschreibt die Zusammenhänge der verwendeten Koordinatensysteme. Der Aufbau entspricht einem, in REP-105 beschriebenen, typischen Aufbau. Die Koordinatensysteme *imu_link*, *rtk_gps_link* und *gps_link* sind starr mit dem *base_link* verbunden. Die Translation zwischen den Sensorkoordinatensystemen und dem *base_link* entspricht der Position der XSens IMU, des u-blox RTK GNSS Empfängers und der Arduino GNSS Empfängers im *base_link*-Koordinatensystems. Das *utm*-Koordinatensystem ist ein weltfestes Koordinatensystem und beinhaltet die Position des GNSS-Empfängers in Kugelkoordinaten angegeben („GPS-Koordinaten“). Es ist starr mit dem *map*-Koordinatensystem verbunden. Diese starren Translationen werden im ROS-System als *tf2_msgs/TFMessage* unter dem Topic */tf_static* veröffentlicht.

Die Position des *base_link* im *odom*-Koordinatensystem wird in einem typischen Aufbau, basierend auf einer Odometriequelle und einer inertialen Messeinheit berechnet. Die Translation zwischen den genannten Koordinatensystemen wird mittels *tf2_msgs/TFMessage* unter dem Topic */tf* veröffentlicht.

In einem weiterem REP-105 typischen Aufbau wird die Translation zwischen *odom*- und *map*-Koordinatensystem, basierend auf GNSS-Empfängerbeobachtungen ständig neu berechnet, wodurch Drifts eliminiert werden, aber diskrete Sprünge verursacht werden, wenn neue Sensorinformationen eintreffen. Diese Translation wird wie bei der *odombase_link* Translation als *tf2_msgs/TFMessage* unter dem Topic */tf* veröffentlicht.

III. Test von Kalman Filter Konfigurationen

Mit dem entwickelten System wurden drei Systemkonfigurationen von Kalman-Filtern getestet: EKF2D, UKF2D und EKF3D. Fünf verschiedene Szenarien, mit denen die Filter auf ihre Leistungsfähigkeit hinsichtlich der Genauigkeit der Positionsbestimmung getestet wurden, ausgearbeitet:

- Szenario 1 - Einfache Fahrt auf dem Hochschulparkplatz,
- Szenario 2 - Einfache Fahrt mit sporadischen GNSS-Ausfällen,
- Szenario 3 - Abschattung des GNSS-Signals durch Bäume,
- Szenario 4 - Reflektions-, Mutlipath-Effekte und Ausfall des GNSS durch Gebäude,
- Szenario 5 - Steigungen und leichte Abschattung durch Bäume.

IV. Zusammenfassung und Ausblick

Eine interessante Möglichkeit die Lokalisierung Ergebnisse verbessern zu können wäre das Verwenden noch weiterer Sensoren. Dahingehend könnte untersucht werden, wie stark eine visuelle Odometrie mittels einer oder mehrere Kameras die Lokalisierungsergebnisse verbessern würde. Oder zu was für eine Verbesserung die Verwendung von Sensoren zu Höhen Ermittlung (bspw. Barometer) oder globalen Orientierung (bspw. Magnetometer), insbesondere in der dreidimensionalen Anwendung führt.

Acknowledgment

The research is carried out within the frames of the contract № 222ПД0014-03, Scientific and Research Sector at the Technical University of Sofia.

Literaturverzeichnis

[1] Clearpath Robotics Inc. , „Husky UGV - Outdoor Field Research Robot by Clearpath“.

[2] M. Ivanova, P. Petkova und P. Petkov, „Machine Learning and Fuzzy Logic in Electronics: Applying Intelligence in Practice,“ *Electronics*, Bd. 10, Nr. (22):2878, 2021.

[3] Hensel, S., Strauss, T., Marinov, M., „Eddy current sensor based velocity and distance estimation in rail vehicles,“ *IET Science, Measurement & Technology*, Bd. 9, Nr. 7, p. 875–888, 2015.

[4] MagicMaps.de, „GNSS-Wissen,“ [Online]. Available: https://www.magicmaps.de/gnss-wissen/?L=0. [Zugriff am 12 May 2021].

[5] C. Prajanu, „RTK Fundamentals - Navipedia,“ [Online]. Available: https://gssc.esa.int/navipedia/index.php/RTK_Fundamentals. [Zugriff am 12 May 2021].

[6] XSens, „MTi User Manual: MTi 10-series and MTi 100-series 5th Generation (Zugriff am: 19. März“.

[7] ROS.org, „xsens_driver - ROS Wiki,“ [Online]. Available: http://wiki.ros.org/xsens_driver. [Zugriff am 19 March 2021].

[8] Tully Foote, Mike Purvis, „REP 103 - Standard Units of Measure and Coordinate Conventions (ROS.org)“.

[9] Wim Meeussen, „REP 105 -- Coordinate Frames for Mobile Platforms (ROS.org)“.

[10] „magicmaps,“ [Online]. Available: https://www.magicmaps.de/gnss-wissen/praezise-gps-messungen-mit-hilfe-von-dgps-und-rtk/. [Zugriff am October 2022].