Automation eines BlueROV2 - Rapid Prototyping Plattform für regelungstechnische Anwendungen

Sven Lack, Hochschule Wismar
s.lack@stud.hs-wismar.de
Martin Kurowski, Universität Rostock
martin.kurowski@uni-rostock.de
Peter Dünow, Hochschule Wismar
peter.duenow@hs-wismar.de

Zusammenfassung

Durch die in den letzten Jahrzehnten kontinuierlich steigende Anzahl an Bauprojekten für maritime Infrastrukturen in den europäischen Meeren und Ozeanen, steigt auch gleichermaßen der Bedarf an Inspektions-, Vermessungs- und Wartungsmöglichkeiten dieser Bauwerke. Um die Kosten niedrig zu halten und die Gefährdung von Tauchern zu vermeiden, werden für diese Aufgaben kabelgebundene ROVs (Remotely Operated Vehicles) eingesetzt. Die situationsabhängige Steuerung dieser Unterwasserfahrzeuge, die in allen sechs Freiheitsgraden beweglich sind, stellt eine große Herausforderung für den oder die Piloten da. Daher finden gegenwärtig umfangreiche Entwicklungen im Bereich der Automatisierung und Regelung dieser Fahrzeuge statt. Dafür werden meist von den verschiedenen Forschungsgruppen eigene spezielle Versuchsträger gebaut und diese über systemspezifische Softwarearchitekturen angesteuert. Diese Tatsache führt dazu, dass der Einstieg in den Bereich der Automatisierung von ROVs langwierig, kostenintensiv und unnötig kompliziert wird. Des Weiteren ist der Austausch untereinander nur schwierig möglich, da systembedingt aufwendige Reimplementierungen nötig werden. In diesem Beitrag soll ein Konzept vorgestellt werden, mit dem ein serienmäßig kommerziell erwerbbares ROV durch Softwareanpassungen so verändert wird, dass dieses als Basisplattform für automatisierungs- und regelungstechnische Versuche unter MATLAB®/ SIMULINK® verwendet werden kann. Dazu wurde als Softwarebasis ein ROS-Netzwerk (Robot Operating System) und das „BlueROV2“ der Firma Blue Robotics Inc.[1] gewählt. Die Tauglichkeit des entwickelten Konzeptes konnte mit durchgeführten Laufzeitmessungen bestätigt werden.

[1] https://www.bluerobotics.com/

1 Einleitung

1.1 Motivation

Immer mehr Förderplattformen für Öl und Gas, Windanlagen, Pipelines und Kabel für den Kommunikations- und Energieaustausch der Länder werden Offshore installiert. Diese so errichteten maritimen Infrastrukturen sind permanent den rauen Umweltbedingungen der Meere ausgesetzt und müssen daher in regelmäßigen Abständen inspiziert und gegebenenfalls instand gesetzt werden. Für die Sichtkontrolle, Vermessung und Wartung dieser Bauwerke werden vermehrt kabelgebundene Unterwasserfahrzeuge sogenannte ROVs (Remotely Operated Vehicle) eingesetzt. Im Gegensatz zum Einsatz von Tauchern oder bemannten Tauchfahrzeugen bieten sie eine Reihe von Vorteilen. ROVs besitzen einen großen Einsatzbereich der von mehreren hundert bis zu einigen tausend Metern Tiefe reichen kann. Außerdem sind die Einsatzkosten in der Regel geringer und die Gefährdung von Personen ist auf ein Minimum beschränkt. Aus der Tatsache, dass diese Fahrzeuge in allen sechs Freiheitsgraden beweglich und den Umwelteinflüssen wie Strömungen Unterwasser ausgesetzt sind, ergeben sich besondere Herausforderungen bei der Steuerung dieser Geräte. Um die Arbeit der Piloten zu erleichtern und die Wahrscheinlichkeit eines Fahrzeugverlustes zu minimieren, werden aktuell umfangreiche Entwicklungen im Bereich der Automation und Regelung von ROVs durchgeführt. Diese reichen von einfachen und robusten Tiefen- und Kursregelungen bis hin zu der Automation von kompletten Missionen. Für diese Forschungsaktivitäten werden von den verschiedene Arbeitsgruppen meist selbst entwickelte Fahrzeuge verwendet, die mit speziell für das ROV und den Einsatzzweck programmierter Software ausgerüstet sind [1, 2, 3, 4]. Dieses Vorgehen bietet den Vorteil, dass so alle Schnittstellen selbst definiert werden können und die Anpassung des Systems an die jeweilige Aufgabe einfach möglich ist. Nachteilig ist aber das immer wieder die gleichen Problemstellungen, die bei der Entwicklung eines ROVs auftreten, von unterschiedlichen Forschungsgruppen mit anderen Systemen gelöst werden müssen. Daraus resultieren lange und kostenintensive Einarbeitungsphasen, in denen umfangreiche Kenntnisse zur Hard- und Software des zu entwickelnden Fahrzeuges erarbeitet werden müssen. Des Weiteren gestaltet sich der Erfahrungsaustausch unter den einzelnen Arbeitsgruppen schwierig, da entweder die verwendete Hardware nicht frei verfügbar ist oder erhebliche Anpassungen an der Software nötig sind. Aus den genannten Aspekten lässt sich leicht der Wunsch nach einer frei verfügbaren, kostengünstigen, erprobten und erweiterbaren Versuchsplattform mit einfach integrierbarer Schnittstelle für neue Forschungsvorhaben ableiten.

1.2 Das BlueROV2 als Versuchsplattform

Das BlueROV2 ist ein kabelgebundenes Unterwasserfahrzeug, welches von der Firma Blue Robotics Inc. als Bausatz vertrieben wird. Es besitzt vier horizontale und zwei vertikale Antriebe. Trotz des vorhandenen Kabels, welches nur für den Datenaustausch genutzt wird, erfolgt die Spannungsversorgung über einen Lithium-Polymer-Akku. Neben den

Antrieben sind am Fahrzeug noch vier LED-Scheinwerfer, eine Kamera und verschiedene Sensoren verbaut, sodass eine Grundausstattung gegeben ist. Abbildung 1 zeigt die Einzelteile des Bausatzes und das einsatzbereite ROV.

Abbildung 1: Links: Lieferumfang des BlueROV2, Rechts: BlueROV2 fertig montiert[2]

Die zentrale Steuerungs- und Recheneinheit im ROV wird durch einen Einplatinencomputer vom Typ Raspberry Pi Model 3 und einem Mikrocontroller basierten Pixhawk PX4 Flugcontroller gebildet. Dabei wird der Pixhawk vor allem für die Auswertung der Sensorik und für die Ansteuerung der Aktorik verwendet und kommuniziert über eine USB Schnittstelle über das sogenannte MAVLink Protokoll (Micro Air Vehicle Communication Protocol) mit dem Raspberry Pi. Der Raspberry Pi ist hierarchisch gesehen darüber angeordnet und übernimmt die Auswertung der Kamera, sowie die Kommunikation mit der Oberfläche über eine Ethernet Verbindung. Abbildung 2 zeigt das Blockschaltbild der verbauten Komponenten und die jeweiligen Kommunikationswege, wobei sich die hellblau hinterlegten Komponenten im ROV selbst befinden.

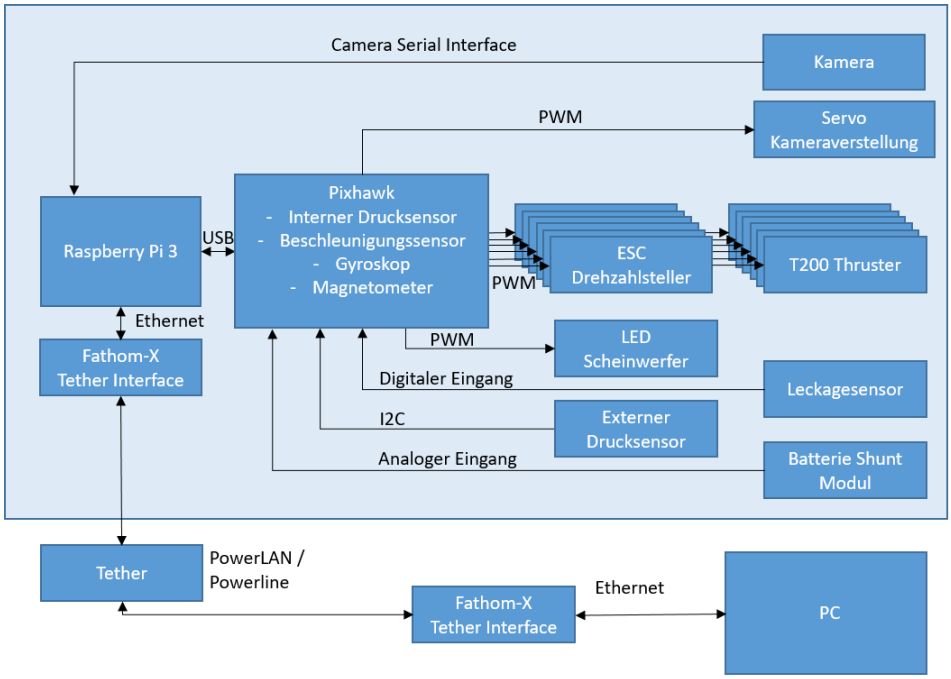

Abbildung 2: Blockschaltbild der elektronischen Komponenten des BlueROV2

[2] http://www.bluerobotics.com/store/rov/bluerov2/

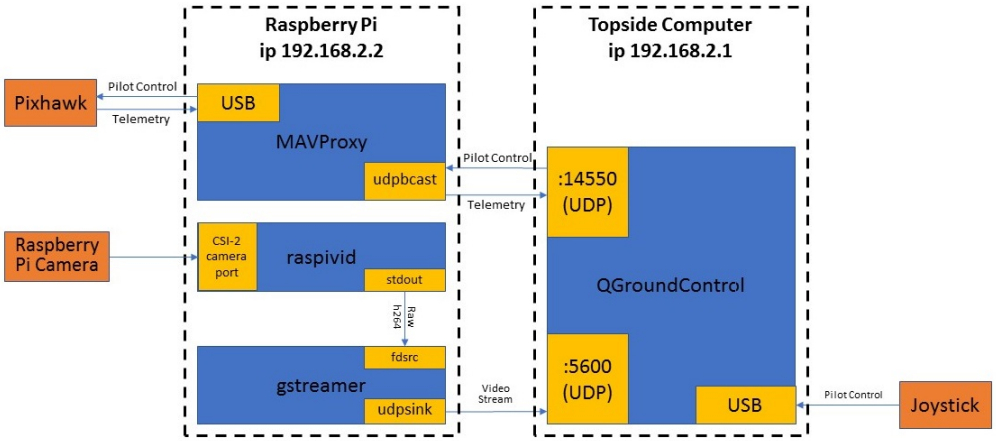

Abbildung 3: Prinzipskizze Softwarekonzept Blue Robotics Inc.[3]

Standardmäßig ist nur die Steuerung des Fahrzeuges und die Visualisierung der Messdaten über die Software QGroundControl vom Hersteller vorgesehen. Das dafür gewählte Softwarekonzept ist in Abbildung 3 dargestellt. Ein Abgriff der Messwerte bzw. die Vorgabe von Sollwerten außerhalb von QGroundControl ist mit diesem System nicht möglich. Im Gegensatz zu vielen anderen käuflich erwerbbaren ROVs, ist beim BlueROV2 die komplette Software Open Source und kann beliebig verändert werden. Dies betrifft nicht nur den Raspberry Pi und das auf ihm installierte Linux Betriebssystem, sondern auch den Pixhawk mit der Firmware ArduSub, sowie die Drehzahlsteller. Diese Tatsache mach das BlueROV2 zu einer soliden Basis für das im folgenden Abschnitt vorgestellte entwickelte Ansteuerungskonzept.

2 Das entwickelte Konzept

Für das entwickelte Softwaresystem, welches in Abbildung 4 gezeigt ist, wurden mehrere Forderungen im Vorfeld aufgestellt. Diese sind in der folgenden Auflistung dargestellt.

- Möglichkeit der unabhängigen Ansteuerung der einzelnen Aktorikkomponenten
- Möglichkeit der Auswertung aller relevanten Sensordaten
- Schneller Datenaustausch im Verhältnis zum Systemverhalten
- Kein Einsatz von Spezialsoftware auf dem Oberflächencomputer
- Einfache Kommunikationsschnittstelle

Als Grundlage für die Erfüllung dieser Forderungen wurde ein ROS-System gewählt, welches auf dem Raspberry Pi unter dem Linux Betriebssystem Ubuntu Mate in der Version 16.04 installiert wurde. ROS ist ein Quellen offenes erprobtes Framework mit klarer Struktur und definierten Schnittstellen, welches für die verteilte Verarbeitung von Teilaufgaben in Robotersystemen entwickelt wurde. Auf dem Pixhawk kommt eine angepasste Variante

[3] `https://www.ardusub.com/software/components.html`

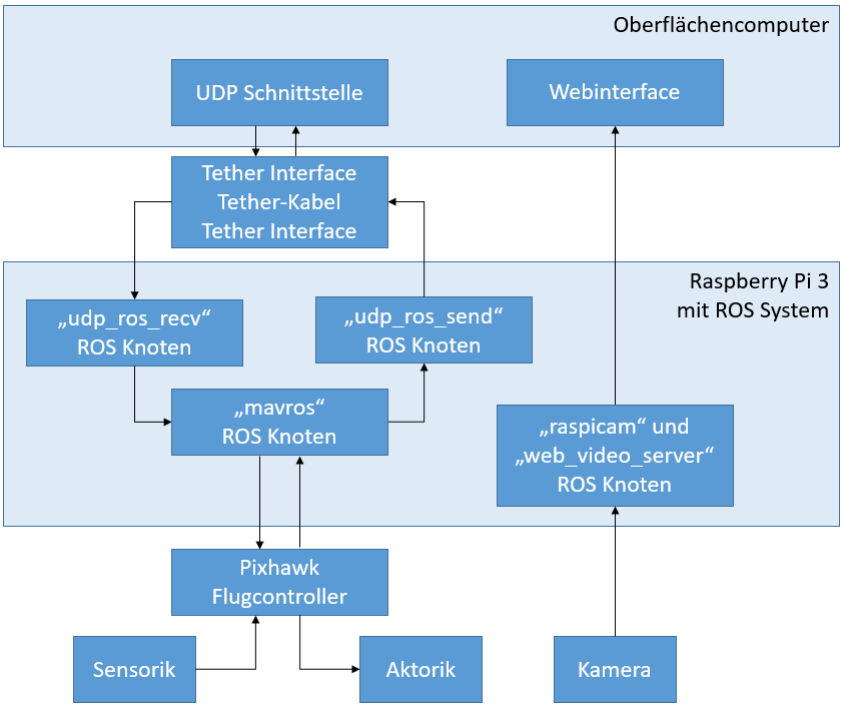

Abbildung 4: Blockschaltbild des entwickelten Softwarekonzeptes

der ArduSub Software der Version 3.5 zum Einsatz. Diese Firmware wurde im Bezug zur Standardversion dahin gehend angepasst, dass nicht die kombinierte Ansteuerung mehrerer Antriebe in die verschiedenen Bewegungsrichtungen, sondern der einzelnen Antriebe nun möglich ist. Des Weiteren wurden alle intern im Flugcontroller ablaufenden Regelungen deaktiviert und die Updaterate, sowie die Übertragung der Sensordaten auf 50 Hz erhöht. Außerdem wurde eine Laufzeitmessstrecke in die Software implementiert, die den Weg der Daten vom Eingang bis zum Controller seitigen Ausgang und wieder zurück umfasst. Die Kommunikation zwischen dem ROS-System und dem Pixhawk erfolgt über das MAVLink-Protokoll und über den ROS-Knoten „mavros“[4]. Da eine unkomplizierte Nutzung des ROVs, ohne spezielle Kenntnisse auf dem Gebiet der ROS-Netzwerke oder anderen speziellen Softwarelösungen gegeben sein sollte, wurde für die Verbindung zum Oberflächencomputer eine UDP-Protokoll basierte Schnittstelle ausgewählt. Diese wurde mit zwei selbst entwickelten ROS-Knoten „ros_udp_send“ und „ros_udp_recv“ realisiert. Diese Knoten arbeiten Interrupt basiert und gewährleisten somit den schnellen Datentransport an den Oberflächencomputer bzw. zum ROV. Für die Registrierung von Paketverlusten wird eine fortlaufende Paketnummer verwendet. Im Knoten „udp_ros_recv“ wurde des Weiteren eine Laufzeitüberprüfung eingefügt, die die Antriebe im Falle eines Kommunikationsverlustes deaktiviert. Die realisierte UDP-Schnittstelle ist durch die implementierten Sicherungsmaßnahmen somit robust gegenüber Störungen und kann über benutzerspezifische Software angesprochen werden. Getestet wurde die Schnittstelle erfolgreich unter MATLAB®/ SIMULINK®, denkbar ist aber auch die Verwendung von LabVIEW, Python oder jeglicher anderer Software, die auf eine UDP-Schnittstelle zugreifen kann. Ebenfalls um die vierte Forderung zu erfüllen, wurde für den Austausch der Vi-

[4] http://wiki.ros.org/mavros

deodaten des ROVs die ROS-Knoten „raspicam“[5] und „web_video_server“[6] verwendet. Der erste der beiden Knoten liest die Kamerabilder ein, der zweite erstellt eine auf dem HTTP-Protokoll basierende Webseite. Diese Webseite ist mit jedem gängigen Webbrowser abrufbar und zeigt das Kamerabild des ROVs, welches im Format mjpeg übertragen wird. Dieser so übertragene Videostream kann auch einfach in MATLAB® eingelesen (z. B.: mit dem Paket „HebiCam“[7]) und dort weiterverarbeitet werden.

3 Experimentelle Konzeptbestätigung

3.1 Durchführung

Das auf dem Raspberry Pi eingesetzte Linux Betriebssystem ist nicht echtzeitfähig. Aufgrund dieser Tatsache ist auch das auf diesem aufsetzende ROS-System ebenfalls nicht echtzeitfähig. Um die Eignung des entwickelten Konzeptes für regelungstechnische Anwendungen zu überprüfen, wurden daher Messungen der Paketumlaufzeit im Gesamtsystem durchgeführt. Bei diesen Messungen wurden die Laufzeiten an zwei Stellen im System untersucht. Abbildung 5 zeigt die beiden gewählten Messpunkte.

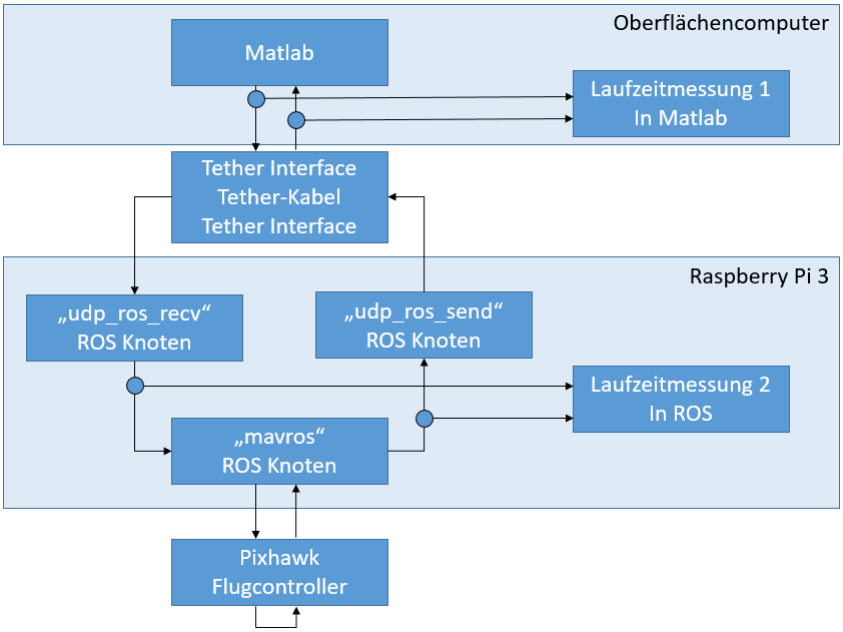

Abbildung 5: Messpunkte der Laufzeitmessungen

Die erste Messung, die in Matlab durchgeführt wurde, repräsentiert dabei die Umlaufzeit des Datenpakets für das gesamte System. Die zweite Messung im ROS-Netzwerk (mittels ROS Bag File) entspricht nur der Kommunikationsdauer zwischen dem ROS-System und dem Pixhawk sowie der Verarbeitungsdauer des Pixhawks selbst. Die beiden Messsysteme wurden nicht synchronisiert, sodass nur relative Aussagen über die Verzugszeiten gemacht werden können. Ein genaue und getrennte Analyse der Laufzeiten auf dem

[5] `https://github.com/fpasteau/raspicam_node`
[6] `http://wiki.ros.org/web_video_server`
[7] `https://github.com/HebiRobotics/HebiCam`

Hin- und Rückweg ist somit nicht möglich. Als Prüfsignal wurde eine Rechteckfolge mit 2 Hz und 50 % Taktgrad gewählt und über einen Zeitraum von 100 Sekunden analysiert. Um zu evaluieren welchen Einfluss die Prozessorauslastung, sowie die Auslastung der Ethernet Verbindung auf die Laufzeiten des Systems haben, wurden vier Messungen bei unterschiedlichen Rahmenbedingungen durchgeführt. Die Auslastung der Ethernet Verbindung wurde durch das Ein- bzw. Ausschalten des Videostreams variiert und durch das Aufnehmen bzw. nicht Aufnehmen des Bag Files im ROS-Netzwerk wurde die Auslastung der CPU beeinflusst. Die letzte genannte Messung hatte zusätzlich den Hintergrund die Beeinflussung der Messergebnisse durch das Messsystem selbst zu untersuchen.

3.2 Ergebnisse und Auswertung

In Tabelle 1 sind die über 400 Flanken gemittelten Ergebnisse der im vorherigen Abschnitt vorgestellten Experimente dargestellt. Die aufgelisteten Zeiten stellen immer die Umlaufzeiten da und ergeben sich somit aus der Summe der Verzugszeit auf den Hin- und Rückweg. Aus den gezeigten Ergebnissen ist ersichtlich, dass es eine Erhöhung der Prozessorauslastung gibt, wenn ein Bag File mit aufgenommen wird. Diese Tatsache hat aber keinen messbaren Einfluss auf die Laufzeiten der Daten (vgl. Messung 1 und 3 sowie 2 und 4). Daher lässt sich ableiten, dass das Messsystem keinen relevanten Einfluss auf die aufgenommenen Messwerte hat. Betrachtet man hingegen die Laufzeit mit und ohne Videostream fällt auf, dass eine geringe nicht wesentliche Erhöhung der Laufzeit von etwa 2,4 ms entsteht (vgl. Messung 1 und 2 sowie 3 und 4). Die Entstehung dieser Abweichung lässt sich eindeutig auf den Raspberry Pi bzw. das ROS-Netzwerk eingrenzen, da die Laufzeiten zwischen dem ROS-Netzwerk und dem Pixhawk konstant bleiben. Dies war zu erwarten, da wie schon erwähnt das eingesetzte System nicht echtzeitfähig ist.

Tabelle 1: Ergebnisse Paketumlaufzeiten Messung

	Messung 1: ohne Bag File ohne Stream	Messung 2: ohne Bag File mit Stream	Messung 3: mit Bag File ohne Stream	Messung 4: mit Bag File mit Stream
Laufzeit gesamt	27,18 ms	29,55 ms	27,02 ms	29,31 ms
Laufzeit nur ROS	-	-	17,33 ms	17,34 ms
Laufzeit Matlab bis ROS	-	-	9,69 ms	11,97 ms
Prozessorauslastung	39,4 %	52,1 %	45,5 %	66,8 %
Bandbreite Upload	144 KBit/s	256 KBit/s	136 KBit/s	248 KBit/s
Bandbreite Download	80 KBit/S	6,2 MBit/s	112 KBit/s	6,8 MBit/s

Aus den Ergebnissen kann abgeleitet werden, dass der Datenaustausch zwischen Matlab und dem ROS-Netzwerk im Durchschnitt 5 - 6 ms pro Richtung benötigt, je nach Auslastung des Kommunikationskanals. Für die Kommunikation zwischen dem ROS-System und dem Pixhawk werden im Durchschnitt weitere 9 ms pro Richtung benötigt, sodass 15 ms die typische durchschnittliche Laufzeit der Daten pro Richtung unter Verwendung des vorgestellten Softwarekonzepts ist.

Betrachtetet man hingegen nicht die durchschnittliche Laufzeit, sondern die Ergebnisse der einzelnen Flankenauswertungen, so ist eine starke stochastische Varianz in den Messwerten erkennbar. Abbildung 6 zeigt die einzelnen Umlaufzeiten die aus den 400 Flanken der vierten Messung berechnet wurden.

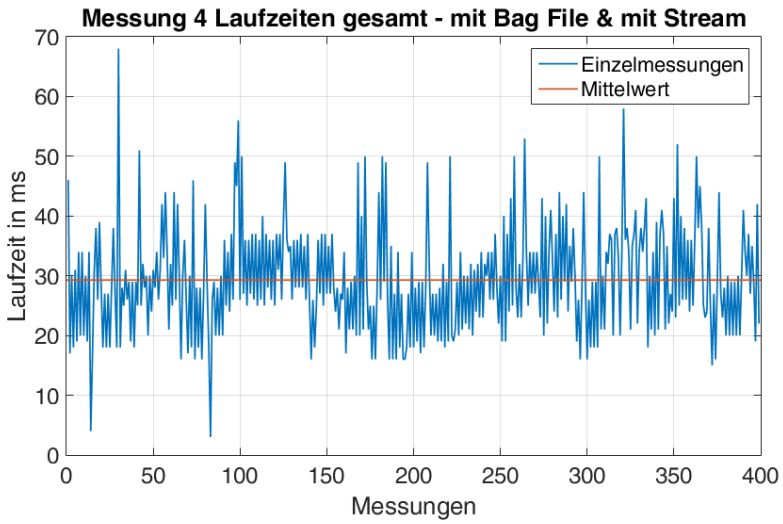

Abbildung 6: Einzelergebnisse Messung 4

Die überwiegende Mehrheit der Ergebnisse liegt unterhalb von 50 ms, ein kleiner Teil aber auch darüber. Daher sind für Regler Abtastrate von kleiner 30 Hz zu wählen. In der Praxis werden für kleine ROVs (Größenordnung ähnlich dem BlueROV2 oder kleiner) oft Abtastraten von 10 Hz gewählt [5, 6], für größere Fahrzeuge sogar noch kleiner Frequenzen [7]. Daher ist die mit dem vorgestellten Konzept erreichbare Abtastrate ausreichend hoch, um regelungstechnische Versuche mit dem BlueROV2 durchzuführen. Die Eignung des Konzeptes konnte so nachgewiesen werden.

4 Zusammenfassung und Ausblick

In diesem Beitrag wurde ein Softwarekonzept vorgestellt, welches ein kommerziell erwerbbares Serienunterwasserfahrzeug BlueROV2 in eine Versuchsplattform für regelungstechnische Experimente umwandelt. Dabei wurde auf eine einfache und sichere Kommunikationsschnittelle wert gelegt, die vom Nutzer über verschiedene Programme ansprechbar ist. Mit Interrupt gesteuerten Auswertungsroutinen in den ROS-Knoten „ros_udp_send“ und „ros_udp_recv“ konnte eine schnelle Weiterleitung der Daten in und aus dem ROS-Netzwerk realisiert werden. Die Eignung des vorgestellten Konzepts wurde mit Laufzeitmessungen von Datenpaketen überprüft und konnte durch den Vergleich mit anderen Arbeiten bestätigt werden. Somit ist durch die Kombination des BlueROV2 und dem vorgestellten Softwarekonzept ein einfacher, schneller und kostengünstiger Einstieg in die Automation von ROVs nun möglich.

Weitere folgende Arbeiten werden sich mit der Modellierung des BlueROV2 befassen, sodass danach ein komplettes Framework bereitsteht, um auf dem Gebiet der maritimen Regelungstechnik von Unterwasserfahrzeugen aktiv zu werden.

Literatur

[1] N. H. Tehrani, M. Heidari, Y. Zakeri: *Development, depth control and stability analysis of an underwater Remotely Operated Vehicle (ROV)*, In 8th IEEE International Conference on Control and Automation (ICCA), IEEE, 2010.

[2] J. C. Molina-Molin, A. Guerrero-Gonzá, J. Gilabert-Cervera: *A New Electronic Control System for Unmanned Underwater Vehicles*, In SIXTH INTERNATIONAL WORKSHOP ON MARINE TECHNOLOGY, SARTI, 2015.

[3] J. C. Molina-Molin, A. Guerrero-Gonzá, J.Gilabert-Cervera: *DEVELOPMENT AND MODELING OF UNMANNED UNDERWATER REMOTELY OPERATED VEHICLE USING SYSTEM IDENTIFICATION FOR DEPTH CONTROL*, In Journal of Theoretical and Applied Information Technology, Vol 56 (1). pp. 136-145. ISSN 1992-8645, 2005- 2013 JATIT & LLS, 2013.

[4] R. M. F. Gomes, J. B. Sousa, F. L. Pereira: *Modeling and control of the IES project ROV*, In European Control Conference (ECC), IEEE, 2003.

[5] N. Mičkoviś, Z. Vukić: *Fast In-Field Identification of Unmanned Marine Vehicles*, In Journal of Field Robotics Vol 28 (1). pp. 101-120 (2011), 2010 Wiley Periodicals, Inc., 2011.

[6] S. A. Watson, P. N. Green: *Depth Control for Micro-autonomous Underwater Vehicles (μAUVs): Simulation and Experimentation*, In International Journal of Advanced Robotic Systems, Vol 11 (3). ISSN: 1729-8814, SAGE Publications Ltd., 2014.

[7] D. A. Fernandes, A. J. Sørensen, K. Y. Pettersen, D. C. Donha: *Output feedback motion control system for observation class ROVs based on a high-gain state observer: Theoretical and experimental results*, In Control Engineering Practice, Vol 39. pp. 90-102. ISSN 0967-0661, 2014 Elsevier Ltd., 2014.