HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

Bericht zum Praxismodul

Praktikumstelle:
Blockchain Competence Center Mittweida (BCCM)

Verfasser:
Johannes Krause

Seminargruppe:
MI15w3-b

Matrikelnummer:
40864

Einreichung:
Mittweida, 12. Juni 2018

Inhaltsverzeichnis

1. Einleitung **1**

2. Blockchain Spring School **2**

3. Blockchain-basierte Kommunikationsschichten für autonome Organisationen **3**
- 3.1. Umriss . . . 3
- 3.2. Aufgabenstellung . . . 5
- 3.3. Rahmenbedingungen . . . 6
- 3.4. Umsetzung . . . 7
  - 3.4.1. Demonstrator . . . 7
  - 3.4.2. Geth . . . 7
  - 3.4.3. OPCUA . . . 8
  - 3.4.4. web3 . . . 8
  - 3.4.5. Joy-It Roboterarm . . . 8
  - 3.4.6. Python-Shell . . . 9
  - 3.4.7. Architektur . . . 10
- 3.5. Entwicklung . . . 10
- 3.6. Ergebnis . . . 12

4. Ausblick **13**

5. Tools **14**
- 5.1. Generell . . . 14
- 5.2. Weitere verwendete Software . . . 15
- 5.3. NodeJS-Bibliotheken . . . 15

6. Wichtige Hinweise! **16**

Glossar **17**

Akronyme **18**

Quellen **19**

7. Selbstständigkeitserklärung **20**

Anhang A. Tätigkeitsberichte **22**

Anhang B. Installation **23**
B.1. Roboterarm 23
B.2. Geth 24
B.3. NodeJS 26
B.4. BCCM Demonstrator 26

Anhang C. Benutzerleitfaden **27**
C.1. Übersicht 27
C.2. startGeth 27
C.3. startServer 28
C.4. robotOPCUAServer.js 28
C.5. Servos.py 29
C.6. robotCode.py 30
C.7. robotHistory 30
C.8. robotOPCUAHistory 31
C.9. lastServoValues 31

Abbildungsverzeichnis

1.1. BCCM-Logo 1

3.1. Produktionszyklus 5
3.2. Aufbau Demonstrator 7
3.3. Joy-It Roboterarm 9
3.4. Architektur 10

B.1. Aufbau 1 23
B.2. Aufbau 2 23
B.3. Aufbau 3 23

Tabellenverzeichnis

5.1. Verwendete Software 15
5.2. Verwendete NodeJS-Bibliotheken 15

A.1. Tätigkeitsbericht 22

C.1. Startparameter für robotOPCUAServer.js 28
C.2. Startparameter für Servos.py 29
C.3. Tastenbelegung robotCode.py 30

1. Einleitung

Dieser Bericht behandelt das Praktikum, das vom 05.03.2018 bis zum 08.06.2018 vom Verfasser dieses Berichts am BCCM abgeleistet wurde. Die Arbeitszeit des Praktikums betrug arbeitstäglich 8 Stunden. Der genaue Beginn und Ende war dabei dem Praktikanten freigestellt. Gleitzeit und HomeOffice war möglich.

Das BCCM forscht und gibt Veranstaltungen rund um das Thema Blockchain. In regelmäßigen Abständen werden die sogenannten Spring- und Autumn-Schools veranstaltet, in der Teilnehmer sich zum Thema Blockchain schulen, in Praxis-Sessions die Technologie selber anwenden, und sich in Meet-Ups vernetzen können.

Abbildung 1.1.: BCCM-Logo

Das Praktikum beschäftige sich primär mit der Recherche und Realisierung von Methoden zur dezentralen Kommunikation von Organisationen und Produktionsketten. Zu diesem Zweck wurden vom Praktikanten gängige Industriestandards recherchiert, die das Ergebnis seiner Arbeit näher an Gegebenheiten der Industrie bringen sollen. Im Anschluss wurde unter Berücksichtigung dieser Standards in Zusammenarbeit mit Kollegen des BCCM ein Demonstrator erarbeitet, der auf der Basis einer Ethereum-Blockchain eine einfache Produktionskette abbildet. Dabei galt es verschiedene Probleme zu lösen, die vom Aufbau der entsprechenden Produktions- und Transportprozesse, zur Entwicklung eines Softwaresystems und des anschließenden Trainings der Gerätschaften reichen, um eine Automatisierung zu ermöglichen.

Der folgende Bericht geht hierbei nun auf die verwendeten Technologien, das Recherchematerial und das Ergebnis des dreimonatigen Praktikums ein, und gibt anschließend einen Ausblick auf die mögliche Weiterentwicklung des Forschungsprojektes.

2. Blockchain Spring School

Der Beginn des Praktikums fiel auf den Start der Blockchain-Spring-School des BCCM am 05.03.2018 , und dauerte eine Woche an. In dieser Zeit ging der Praktikant primär unterstützend zur Hand und half insbesondere als sogenannter Programmierpate in den Praxis-Sessions der Veranstaltung aus. Die verschiedenen schulischen Veranstaltungen wurden ebenfalls genutzt, um den Praktikanten die Möglichkeit zu geben, sich zum Thema Blockchain zu informieren bzw. altes Wissen wieder neu aufzufrischen.

3. Blockchain-basierte Kommunikationsschichten für autonome Organisationen

3.1. Umriss

Seit einigen Jahren nimmt der Trend autonomer, softwaregestützter industrieller Produktion in der Wirtschaft stetig zu. Diese Entwicklung ist unter dem Namen „Industrie 4.0“, oder „die vierte industrielle Revolution“ bekannt. Im Rahmen dieses Trends werden stetig mehr sogenannter Cyber-Physischer Systeme entwickelt, die sich durch den Verbund mehrerer, durch Software gesteuerter physischer oder elektrotechnischer Komponenten auszeichnen. Als Kommunikationsmedium dienen hierbei meist dem Internet ähnliche Vernetzungen der einzelnen Komponenten und als „Sprache“ Protokollstandards wie Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP) oder die Open Plattform Communication Unified Architecture (OPCUA).

Von der einfachen, automatisierten Produktion sind solche Cyber-Physischen Systeme abzugrenzen. Der bedeutendste Unterschied besteht darin, dass Komponenten Ersterer nicht oder nur sehr eingeschränkt untereinander kommunizieren. Es handelt sich meist um Automaten, die mit dem nächsten Produktionsschritt einer Ware beginnen, wenn sie einfache, binäre Signale der vorhergehenden Produktionsschritte empfangen, oder über Sensoren die Position bestimmter Formen an ihrem Produktionsstandort erfassen. Zumeist fehlt es ihnen auch an einer informationstechnischen Schnittstelle zur Abfrage von Diagnose- oder Produktionsdaten. Die durch die Digitalisierung gewachsene Anzahl an Daten durch Sensoren in der Produktion, und die durch moderne Technologie gewonnene Möglichkeit, Maschinen diese Informationen in komplexen Zusammenhängen untereinander kommunizieren zu lassen, erlaubt eine größere Autonomie der Produktionskette. Diese sind in der Lage, Verzögerungen und Produktionsfehler automatisch zu erkennen und Mitarbeiter darüber zu informieren,

Logdateien zur einfacheren Fehlerdiagnose anzufertigen, und Informationen bereitzustellen, die eine Optimierung der Produktionskette ermöglichen.

Cyber-Physische Systeme sind in Deutschland besonders in der Automobilindustrie stark vertreten und haben dort einen Siegeszug angetreten. Auch wenn diese Systeme die Produktion *innerhalb* einzelner Betriebe erfolgreich optimiert und automatisiert haben, so besteht nach wie vor kaum Möglichkeit zur automatisierten Abwicklung extra-betrieblicher Geschäftsprozesse. Bestellungen von Rohmaterialien, logistische Dienstleistungen und Warenabnahme benötigen nach wie vor schriftliche Verträge zwischen den beteiligten Parteien, die auf Vertrauen basieren, und große Kosten bei Nichteinhaltung verursachen. Selbst wenn eine juristische Erzwingung der Erfüllung der Vertragsbestandteile erreicht werden kann, sind die damit verbundenen Verzögerungen meist mit zu großen Kosten für alle Parteien verbunden. Dies führt dazu, dass beteiligte Parteien Strategien entwickeln, um dieses Risiko und damit verbundene Kosten zu minimieren. [1]

Das BCCM forscht seit dem 01.07.2017 zum Thema „Blockchain-basierte Kommunikationsschichten für autonome Organisationen“ und erforscht Möglichkeiten, wie solche Probleme mithilfe der von der Blockchain bereitgestellten Technologien gelöst werden können. Ziel ist es dabei, einen vollständigen Produktionszyklus, von der Bestellung bis zur Lieferung, mit Blockchain basierten Technologien umzusetzen.[2] Diese unterteilen sich in folgende vier Bestandteile:

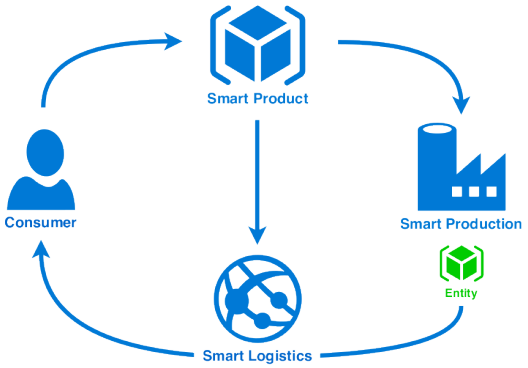

Abbildung 3.1.: Produktionszyklus

- Customer - Bestellt ein Produkt oder Dienstleistung durch einen sogenannten Smart Contract und bezahlt das Smart Product in der Währung der Blockchain selbst. Sollte die Bestellung nicht erfüllbar sein, wird eine Rückerstattung automatisch eingeleitet.
- Smart Product - Der Smart Contract des Smart Products ermittelt autonom Hersteller oder Dienstleister, die die Kriterien der bestellten Ware erfüllen können und ermittelt unter ihnen den Günstigsten. Da das Geld beim Smart Product hinterlegt ist, kann der Hersteller oder Dienstleister sicher sein, dass die angeforderte Ware bezahlt werden kann.
- Smart Production - Die Smart Production fertigt die angeforderte Ware mithilfe Cyber-Physischer Systeme an, die kompatibel zu Blockchain-Technologien sind.
- Smart Logistics - Die Logistik zum Transport der Ware wird ebenfalls vom Smart Product selbst ermittelt und bezahlt. Auch hier werden Blockchain-kompatible Technologien vorausgesetzt.

3.2. Aufgabenstellung

Dem Praktikanten fiel im Rahmen dieses Projektes die Aufgabe zu, mithilfe von Kollegen einen Demonstrator zu entwickeln, welche den Aspekt der „Smart Factory“ in diesem automatisierten Produktionszyklus darstellt. Dazu wurden folgende Arbeiten vorgenommen:

- Recherche über Kommunikationsmöglichkeiten einzelner Fertigungsanlagen über die Blockchain
- Erarbeiten einer Architektur für Blockain-basierte Cyber-Physische Systeme
- Recherchieren bestehender und in der Industrie angewandter Maschine-zu-Maschine (M2M)-Protokolle
- Aufbau eines Roboterarmes als Bestandteil des Demonstrators
- Entwicklung einer Steuerungssoftware des Roboterarmes
- Entwicklung eines OPCUA-Servers für den Roboterarm
- Anbindung des Roboterarmes an die Blockchain
- Entwicklung einer Whisper-Schnittstelle zur Kommunikation des Roboterarmes mit anderen Bestandteilen der Fertigungsanlage
- Entwicklung einer Schnittstelle zwischen der Blockchain als Kommunikationsmedium, dem OPCUA-Server als gängige M2M Lösung in der Industrie, und der Steuerungssoftware des Roboterarmes
- Trainieren des Armes auf korrekte Bewegungen
- Testen und Dokumentation der Ergebnisse

3.3. Rahmenbedingungen

Es galten folgende Einschränkungen:

- Als zu verwendende Blockchain ist die Ethereum-Blockchain vorgeschrieben, da diese die momentan die fortschrittlichste Implementierung von Smart Contracts anbietet
- Die zu entwickelnde Software muss auf einem RaspberryPi 3 Model B lauffähig sein (1,2 GHz, 1GB RAM)
- Die Software soll zur besseren Nachverfolgung von Fehlern Einträge in eine Logdatei schreiben
- Die erarbeitete Lösung soll kompatibel zu bereits bestehenden M2M-Lösungen sein, die in der Industrie verwendet werden

3.4. Umsetzung

3.4.1. Demonstrator

Für den Demonstrator wurde folgender Aufbau gewählt:

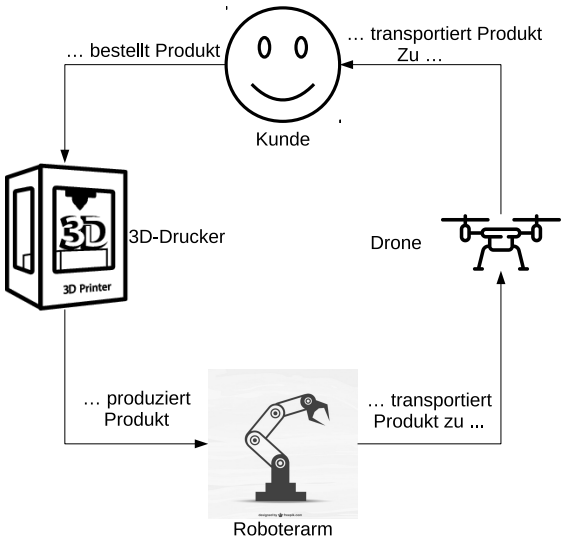

Abbildung 3.2.: Aufbau Demonstrator (Enthält Abbildungen von Ema Hoffman[3], freepik[4] und flaticon[5])

Für den Aufbau und die Eingliederung des 3D-Druckers zeichnete sich Adrian Ortenburger aus, ein weiterer Praktikant am BCCM. Der Aufbau und die Eingliederung des Roboterarms wurde vom Praktikanten übernommen. Zum Zeitpunkt dieses Berichts wurden noch keine Arbeiten an der Drohne vorgenommen.

3.4.2. Geth

Die erarbeitete Lösung muss auf der Ethereum-Blockchain ausführbar sein. Zu diesem Zweck war es als erstes nötig, eine Node-Software zu finden, die auf einem Raspberry Pi mit zufriedenstellender Performance und vollständigem Funktionsumfang lauffähig ist. Die erste Wahl fiel hierbei auf den Geth-Clienten, der sowohl gestattet, auf den offiziellen wie auch privaten Blockchains zu minen, und eine Unterstützung des Whisper-Protokolls anbietet.

Für den Geth-Clienten existieren keine bereits lauffähigen Binarys für ARM-Prozessoren, weswegen er auf dem Raspberry manuell compiliert werden musste. Eine Anleitung dazu findet sich im Anhang B.

Folgende Geth-Version wurde verwendet: 1.8.4-stable

3.4.3. OPCUA

OPCUA ist ein industrieweit verwendeter Standard zur erleichterten Kommunikation zwischen Fertigungsmaschinen verschiedener Hersteller. Um zu demonstrieren, dass eine Blockchain-Lösung gut in existierende Umgebungen integrierbar wäre, wurde entschieden, die einzelnen Bestandteile der Fertigungsanlage mit einer Implementation dieses Standards auszustatten.

Als Implementation wurde NodeOPCUA gewählt. NodeOPCUA bietet eine Implementation auf Basis der Plattform NodeJS, in der die größte Entwicklungsarbeit bezüglich Ethereum geleistet wird. Auf diese Weise wird eine zusätzliche Schnittstelle zwischen zwei Programmiersprachen vermieden.

Verwendete Version: 0.3.0

3.4.4. web3

Um Blockchain-Technologie programmatische einbinden zu können ist die Einbindung des `web3`-Paketes notwendig. Web3 ist ein NodeJS-Packet, dass Methoden zur Arbeit mit Ethereum-Blockchains bereitstellt.

Verwendete Version: 1.0.0-beta.34

3.4.5. Joy-It Roboterarm

Für den Demonstrator wurde als Roboterarm der Robot02-Roboterarm von Joy-It gewählt. Er wird über einen Raspberry Pi 3 Model B angesteuert, und verfügt über sechs Achsen, die über Servos individuell angesteuert werden können. Die Leistung der Servos beträgt $2\ Ampere * 5\ Volt = 10\ Watt$. Für die Stromversorgung der Servos wurde ein individuelles Netzteil benötigt, da die Stromversorgung des Raspberry Pi nicht ausreicht.

Da der Roboterarm über sechs Servos verfügt, die jeweils 2A Stromstärke benötigen, wird ein Netzteil vorausgesetzt, dass 12A bei 5V liefern kann. Ein solches Netzteil war zur Zeit des Praktikums nicht aufzutreiben, weswegen ein Netzteil gewählt werden musste, dass nur in der Lage ist, 10A zu liefern. Trotz dieser Einschränkung wurde keine Leistungseinbuße des Roboterarms festgestellt, selbst wenn alle Servos gleichzeitig in voller Stärke bewegt wurde.

Abbildung 3.3.: Joy-It Roboterarm [6]

Ein Nachteil des Roboterarmes ist, dass die Servos nicht über Javascript angesteuert werden können. Es existiert nur eine in Python geschriebene Bibliothek, weswegen eine Schnittstelle zwischen NodeJS und Python entwickelt werden musste.

3.4.6. Python-Shell

Zur Überbrückung des Kommunikationsproblems zwischen der in NodeJS geschriebenen Serversoftware und der in Python geschriebenen Steuerungssoftware wurde das NodeJS-Packet `Python-Shell` verwendet. `Python-Shell` erlaubt es, Python-Scripts aus einer NodeJS-Anwendung heraus mit Startparametern aufzurufen, Ausgabe und Fehler auszulesen, Eingabe zu simulieren sowie das Script zu beenden und Informationen über Zeitpunkt und Exit-Code des Python-Scriptes zu erhalten.

Verwendete Version: 0.5.0

3.4.7. Architektur

Mithilfe der bisher gesammelten Daten lässt sich folgende Architektur für die Entwicklung und Eingliederung des Roboterarms bestimmen:

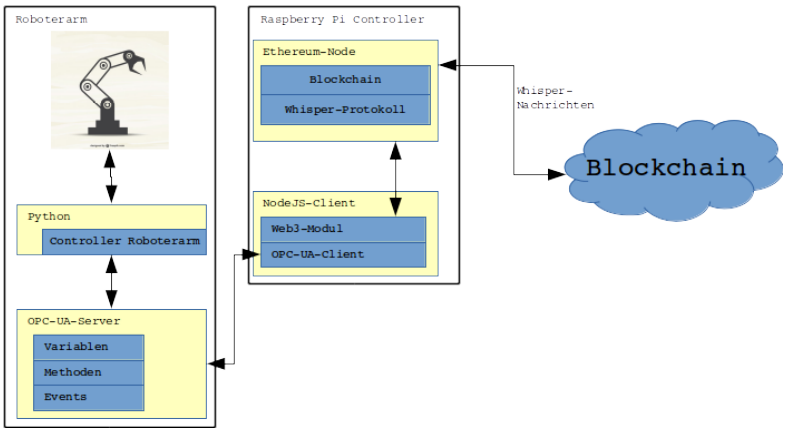

Abbildung 3.4.: Architektur

3.5. Entwicklung

Für die erfolgreiche Umsetzung und Anbindung des Roboterarmes an den Demonstrator war es zuerst vonnöten, eine private Ethereum-Blockchain aufzusetzen, und den Geth-Clienten mit dieser Blockchain zu verbinden, um während des Entwicklungsprozesses die umgesetzten Funktionalitäten testen zu können. Nach erfolgreichem Test des Sendens und Empfangen von Whisper-Nachrichten wurde die Entwicklung mit dem Schreiben der Steuerungssoftware des Roboterarmes fortgesetzt. Diese Software musste in Python geschrieben werden. Als erstes wurde eine einfache Bedienungssoftware verwirklicht, die eine Bedienung des Roboterarmes mit der Tastatur ermöglichte, und das Aufnehmen und Abspielen von Roboterbewegungen ermöglicht.

Nach grundlegenden Funktionstests des Roboterarmes wurde ein einfaches Kommunikationsprotokoll zwischen dem noch zu entwickelndem OPCUA-Server und der ebenfalls noch darauf zu spezialisierenden Steuerungssoftware erdacht. Es wurde beschlossen, eine Steuerungssoftware zu entwickeln, die mit einem Startparameter aufgerufen wird, und sich nach Abarbeitung der damit verbundenen Funktionalität beendet. Diese Option verhindert, dass die Steuerungssoftware in einen

Fehlerzustand verharren kann. Die Steuerungssoftware legt eine Logdatei an, in der die Zeitpunkte eingegangener Aufrufe festgehalten und eventuell aufgetretene Fehler geloggt werden. Weiterhin werden die Werte der Servomotoren in eine externe Datei geschrieben. Dies ist nötig, da die Werte der Servomotoren nicht programmatisch abgefragt werden können.

Die Servos des Roboterarmes werden von der Steuerungssoftware nicht direkt auf den angegebenen Wert gesetzt, da die daraus resultierende Bewegung zu schnell und kraftvoll erfolgt, und den Roboterarm auf Dauer beschädigen kann. Stattdessen wird die Differenz der momentanen Position des Servos und seiner neuen Position berechnet, und diese mithilfe einer gestauchten Kosinus-Funktion schrittweise herangeführt. Dies führt zu einer sich bis zur Mitte der Positionsveränderung beschleunigten, und danach sich langsam verzögernden Bewegung. Die Formel für dieses Verfahren lautet:

$$currentPos = oldPos + (newPos - oldPos) * (-cos(x * Pi) + 1)/2$$

wobei

$currentPos$ die Zwischenposition des Servos,
$newPos$ die neue Position des Servos,
$oldPos$ die Anfangsposition des Servos, und
$\{x \in \mathbb{R} \mid 0 <= x <= 1\}$ darstellen.

x wird schrittweise um einen festen Wert 0.006 inkrementiert. Überschreitet x den Wert 1, wird die Position des Servos fest auf die neue Position gesetzt.

Als letztes musste der OPCUA-Server programmiert, und mit dem Whisper-Protokoll und der Steuerungssoftware verwickelt werden. Der OPCUA-Server öffnet einen Websocket zu einem laufendem Geth-Node und empfängt über diesen eingehende Whisper-Nachrichten. Dieser Geth-Node kann auch auf einem entferntem Rechner ausgeführt werden, und muss nicht zwangsläufig auf derselben Hardware starten.
Der Server wurde weiterhin mit dem Hintergedanken entwickelt, dass die entwickelte Steuerungssoftware nicht mit zukünftigen Bedürfnissen vereinbar ist. Der Server fragt die hinterlegte Steuerungssoftware nach den Werten der Servomotoren in einem festgelegtem Protokoll und bietet diese für sich verbindende Clienten an. Eine genaue Übersicht dieses Protokolls ist dem Benutzerleitfaden im Anhang in der Sektion C.5 für die Steuerungssoftware zu entnehmen.
Weiterhin bietet der Server Clienten eine Funktion zum Abspielen zuvor aufgenommener Armbewegungen. Als Argument für diese Methode muss der Name der abzuspielenden Datei angegeben werden. Diese aufgezeichneten Dateien müssen

sich zudem im selben Verzeichnis wie die Steuerungssoftware des Roboterarmes befinden.
Der Server schreibt, wie die Steuerungssoftware, eine Logdatei über geschehene Ereignisse.

3.6. Ergebnis

Das Ergebnis der Bemühungen ist ein der Aufgabenstellung vollständig entsprechender Roboterarm, der sich problemlos in den BCCM-Demonstrator eingliedern ließ. Eine Demonstration seiner Fähigkeiten wurde bereits zur Zufriedenheit aller Beteiligten durchgeführt. Dem Roboterarm können einfach neue Bewegungen antrainiert werden, und die bereitgestellten Programme sind modular genug gehalten, um eine Ansteuerung anderer Roboterarme zu ermöglichen, falls nötig. Zur Ansteuerung des Armes ist lediglich das Abschicken einer Whisper-Nachricht an die selbe Blockchain nötig, mit der auch der Roboterarm verbunden ist, um eine zuvor aufgezeichnete Bewegung abzuspielen. Die Art der Blockchain spielt hierbei keine Rolle, solange es sich Ethereum-Blockchains handelt. Es können sowohl private, selbst aufgesetzte Blockchains verwendet werden, wie auch das Mainnetz oder die offiziellen Testnetze, solange diese über eine ausreichende Anzahl an Nodes verfügen, welche das Whisper-Protokoll unterstützen.

Der OPCUA-Server kann über einem Startparameter ein anderes, in Python geschriebenes Steuerungsscript zur Anwendung mitgegeben werden, solange das gleiche Kommunikationsprotokoll eingehalten wird. Auf diese Weise sollten auch andere Roboterarme ansteuerbar sein, da auch die Anzahl der Servomotoren über Startparameter bestimmbar sind. Für den Fall, dass das bestehende Protokoll zwischen Server- und Steuerungssoftware den Bedürfnissen nicht mehr entspricht, muss die Serversoftware entsprechend daran angepasst werden.

4. Ausblick

Die Aufgabe der Umsetzung des Smart-Factory-Aspektes eines Cyber-Physischen Systemes, welches Blockchain-kompatibel ist, wurde erfüllt. Es bleiben aber dennoch weitere Arbeiten zur vollständigen Fertigstellung des BCCM-Demonstrators offen. So wurde der Aspekt des Transports mittels einer autonom fliegenden Drohne bisher noch nicht angefangen. Hier ist insbesondere die Frage zu klären, ob die Steuerungssoftware der Drohne, welche die Anbindung zur Blockchain enthält, physisch Teil der Drohne ist, oder die Drohne stattdessen vollständig von einer stationären Hardware ferngesteuert wird. Weiterhin wird der zur Produktion verwendete 3D-Drucker noch nicht über einen OPCUA-Server angesteuert, wie es wünschenswert wäre, um eine möglichst industrienahe Simulation zu ermöglichen.

Um den vollen Funktionsumfang des Demonstrators zu zeigen wäre es ebenfalls wünschenswert, virtuelle Konkurrenzangebote zu erschaffen, aus denen der Smart Contract des bestellten Produktes automatisch den günstigsten Anbieter aussucht, sowie auch Rückerstattungen bei Nichteinhaltung von Vertragsbestandteilen durchführt.

In seiner Weiterentwicklung könnte der Roboterarm um weitere Funktionalitäten erweitert werden. So ist das Aufzeichnen von Bewegungen bisher ausschließlich lokal über ein separates Programm möglich. Dies könnte über Einpflegen weiterer OPCUA-Methoden erweitert werden. Weiterhin könnten eigene aufgezeichnete Bewegungen hochgeladen werden, bei denen zuvor sichergestellt wurde, dass sie mit dem verwendetem Roboterarm kompatibel sind. Dieser Aspekt könnte auch für den 3D-Drucker verwendet werden. Mit dieser Erweiterung würde der Kunde eine gewisse Unabhängigkeit von den festen Angeboten eines Betriebes oder Dienstleisters erlangen, in dem er selbst die zu produzierende Ware spezifiziert, und die Fertigungsanlagen eines Betriebes hierfür nur zur Produktion mietet. Hierfür wäre ebenfalls eine entsprechende Anpassung der Smart Contracts vonnöten, um eine zufriedenstellende Entlohnung des Betriebes zu definieren.

5. Tools

5.1. Generell

Das Projekt wurde auf einem Linux-Laptop mit LinuxMint 18.3 als Distribution, sowie auf einem Raspberry Pi 3 Model B entwickelt. Als Editor zur Entwicklung der Scripte wurde emacs in der Version 24.5.1 verwendet. Zur Versionskontrolle wurde git in der Version 2.7.4 benutzt. Es wurde in den Programmiersprachen Python und JavaScript entwickelt. Für JavaScript wurde die NodeJS-Engine in der Version 8.11.2 verwendet.

Dieser Bericht wurde in LaTeX mithilfe des Online-Dienstes *Overleaf* verfasst. Die LaTeX-Engine war XeLaTeX. Eine vollständige Kopie dieses Berichts im Quelltext ist auf dem github-Repository dieses Projektes und auf *Overleaf* zu finden:
https://github.com/jkrause1/BCCMDemonstrator
https://www.overleaf.com/16588015txzngvnyzbqb

5.2. Weitere verwendete Software

Name	Beschreibung	Version
screen	Screen ist ein Fullscreen-Fenster-Manager das eine Konsole zwischen mehrere unterschiedliche Prozesse multiplext.	4.03.01
python	Python ist ein Interpreter, mit dem in Python geschriebene Scripte ausgeführt werden können.	2.7.12
opcua-commander	Der OPCUA-Commander verbindet sich zu einem OPCUA-Server und liest die von ihm bereitgestellten Werte und Methoden aus. Wurde ausgiebig zum Testen des OPCUA-Servers verwendet.	0.2.3
npm	Packet-Manager für NodeJS	4.6.1
Filezilla	Bietet Implementationen des FTP und SFTP-Protokolls. Wurde verwendet zum Kopieren von Daten zwischen Arbeitslaptop und dem Raspberry Pi	3.15.0.2

Tabelle 5.1.: Verwendete Software

5.3. NodeJS-Bibliotheken

Zur Entwicklung des BCCM-Demonstrators wurden folgende NodeJS-Pakete verwendet:

Name	Beschreibung	Version
web3	Bibliothek zur Programmierung von Anwendungen, die web3- & Blockchain-Technologien verwendet.	1.0.0-beta.34
node-opcua	Bibliothek zur Implementierung eigener OPCUA-Server und -Clients.	0.3.0
python-shell	Stellt eine Schnittstelle zwischen NodeJS und Python bereit	0.5.0
yargs	Bibliothek zum vereinfachten Parsen von Kommandozeilenparameter	11.0.0

Tabelle 5.2.: Verwendete NodeJS-Bibliotheken

6. Wichtige Hinweise!

Beim Betrieb des Roboterarmes ist dringend darauf zu achten, dass der Roboterarm seine Destination auch erreichen kann. Eine Bewegungsblockade führt innerhalb von Sekunden zu einer drastischen Überhitzung der Servomotoren und beschädigt diese irreparabel. Dies ist besonders im Hinblick auf die Zange zu beachten, die Objekte nur wenige Sekunden tragen darf.

Sollte die in Python geschriebene Steuerungssoftware des Roboterarmes ausgetauscht werden, so ist weiterhin darauf zu achten, dass die Funktionalität der neuen Steuerungssoftware mit einer älteren Version des Python-Interpreters als 3.5.0 getestet wurde. Die verwendete Schnittstelle zwischen dem OPCUA-Server und Python unterstützt diese oder neuere Versionen des Python-Interpreters nicht, weswegen es zu Fehlern, unvorhergesehenem Verhalten und im schlimmstem Falle sogar zu Beschädigungen der Servomotoren führen kann.

Glossar

Constrained Application Protocol

Ein speziell für eingebettete, wenig leistungsfähige Geräte entwickeltes M2M-Protokoll.

Message Queue Telemetry Transport

Ein M2M-Protokoll, dass speziell für eingeschränkte oder fehleranfällige Netzwerke entwickelt wurde.

Node

Nodes bezeichnen in der Graphentheorie die Endpunkte von Linien. In Blockchain-Netzwerke stellen diese Endpunkte Blockchain-Teilnehmer dar.

Open Plattform Communication Unified Architecture

Ein Industriestandard zur Vereinheitlichen digitaler Kommunikationsprotokolle von Industriemaschinen. Gestattet Browsing-Mechanismen zum automatischen Entdecken von Informationen und Methoden, die eine Maschine einer anderen anbietet, ohne dessen spezielle Sprache kennen zu müssen.

Whisper

Whisper ist ein noch nicht sehr weit verbreitetes Protokoll zum Nachrichtenaustausch von Nodes einer Ethereum-Blockchain. Whisper-Nachrichten werden mit einer einstellbaten TTL verschickt, innerhalb der sie an alle Nodes verschickt und von diesen weitergeleitet, aber nicht von allen gelesen werden können.

Akronyme

BCCM	Blockchain Competence Center Mittweida
CoAP	Constrained Application Protocol
M2M	Maschine-zu-Maschine
MQTT	Message Queue Telemetry Transport
OPCUA	Open Plattform Communication Unified Architecture

Quellen

[1] M. Olsen, *How firms overcome weak international contract enforcement: Repeated interaction, collective punishment, and trade finance*, Jan. 2015. Adresse: http://morten-olsen.com (besucht am 08. 06. 2018).

[2] Prof. A. Ittner, „Projektskizze blockchain-basierte kommunikationsschichten für autonome organisationen“, zur Zeit des Praktikums nur Intern einsehbar.

[3] E. Hoffman, *3d printer symbol vector*. Adresse: https://bit.ly/2l1gPLQ (besucht am 08. 06. 2018).

[4] freepik, *Robotic arm*, 2014. Adresse: https://bit.ly/2xUir3k (besucht am 08. 06. 2018).

[5] ——, *Drone free icon*. Adresse: https://bit.ly/2sWgvS2 (besucht am 08. 06. 2018).

[6] joy-it, *Robot02-1*. Adresse: https://bit.ly/2LyXCwc (besucht am 08. 06. 2018).

[7] OPC Foundation, *Opc unified architecture, Wegbereiter der 4. industriellen (r)evolution*. Adresse: https://bit.ly/2pWJGT3 (besucht am 08. 06. 2018).

[8] joy-it, *Roboter-arm-bausatz, Robot02*. Adresse: https://bit.ly/2HxDjgq (besucht am 08. 06. 2018).

7. Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich den vorliegenden Bericht selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort: Mittweida
Datum: 11.06.2018

Unterschrift:

Johannes Krause

Anhang

A. Tätigkeitsberichte

KW	Tätigkeit
10	BCCM-Springschool, Programmierpate und Aushilfe
11	Recherche verwendbar Industriestandards, Protokoll, Bibliotheken und Werkzeuge für Demonstrator
12	Testentwicklung an Roboterarm; Schreiben einer Steuerungssoftware für den Roboterarm
13	Proof-Of-Concepts bisher entwickelter Architektur
14	Fortsetzung Entwicklung Steuerungssoftware für Roboterarm
15	Entwicklung Programm zur Aufnahme und Abspielen von Roboterbewegungen
16	Testen bisherigen Fortschritts; Installation nötiger Software auf Raspberry Pi
17	Geth-Clienten auf Raspberry Pi compilieren; Whisper-Nachrichten empfangen; Roboterbewegung auf Basis der Nachrichten realisieren
18	Whisper-Python-Schnittstelle fertig entwickelt; OPCUA-Steuerung des Roboterarmes recherchiert bzw. begonnen.
19	Armkalibrierung und Neuaufbau. Testen neuer Stromversorgung. OPCUA-Server weiterentwickelt; Roboterbewegungen aufgenommen
20	Armkalibrierung und Neuaufbau. Testen bisherigen Fortschritts.
21	Roboterarm an private Ethereum-Blockchain angebunden; OPCUA-Server fertiggestellt; Weitere Roboterbewegungen aufgenommen
22	Roboterarm auf festem Untergrund fixiert; Demonstrationsaufbau definiert, Abholbewegung des Druckstückes trainiert
23	Verfassen des Abschlussberichtes; Dokumentation; Testen der Software; Feinschliff

Tabelle A.1.: Tätigkeitsbericht

B. Installation

B.1. Roboterarm

Auf der Website des Herstellers, Joy-It, finden sich ausführliche Anleitungen zum Aufbau und Installation nötiger Pakete des Robot02-Roboterarmes. Es sei an dieser Stelle daher nur auf deren Anleitungen hingeweisen, da eine weitere Erwähnung nur Redundanz erzeugen würde.

Link: http://anleitung.joy-it.net/?goods=roboterarm-robot02

Es existiert jedoch keine Referenz in diesen Anleitungen, wie ein fertig aufgebauter Roboterarm auszusehen hat. Zur besseren Reproduzierbarkeit sei hier daher ein Bild des im Praktikum aufgebauten und fertig kalibrierten Roboterarmes veröffentlicht.

Abbildung B.1.: Aufbau 1

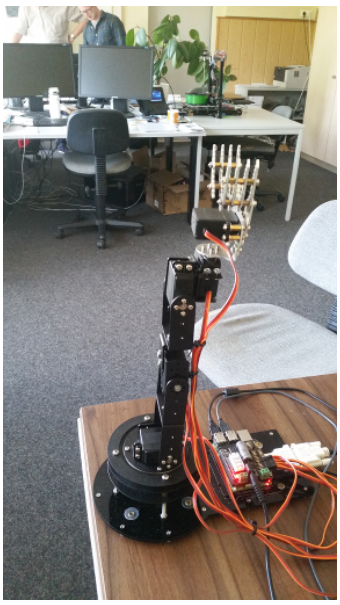

Abbildung B.2.: Aufbau 2

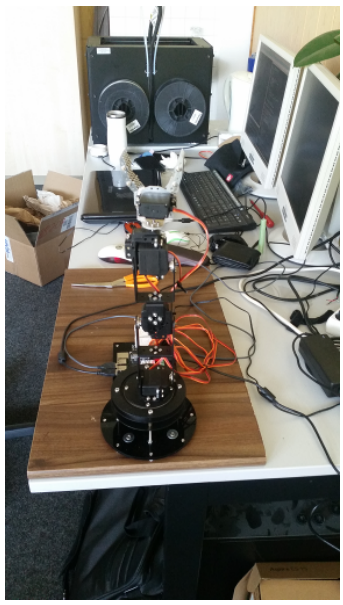

Abbildung B.3.: Aufbau 3

ACHTUNG!

Beim Kalibrieren der Servomotoren ist darauf zu achten, welche Version des Python-Interpreters für die Kalibrierung verwendet wird. In der Version 3.5.0 und neuer konvertiert der Interpreter Divisionen automatisch in Double-Werte, auch wenn die zur Division verwendeten Werte keine Gleitkommazahlen waren. Dies ist solange unproblematisch, solange immer mit der gleichen Version des Interpreters gearbeitet wird, aber führt zu Problemen, wenn nicht.

Der Roboterarm besteht aus 6 Servos, die bei maximaler Auslastung je 2 Ampere bei 5 Volt Spannung aufnehmen. Ein Netzteil, welches diese Leistung aufbringt, muss selbst beschafft werden, und ist nicht im Lieferumfang des Joy-It Robot02-Roboterarmes enthalten.

Der im Lieferumfang enthaltene MotoPi enthält 2 Eingänge zur Stromversorgung:

- 3,5mm Hohlstecker (5V DC)
- Lüsterklemme (4,8V - 6V DC)

Zum Betrieb des Roboterarmes wurde eine Netzteil verwendet, welches nur eine Stromversorgung von 10 Ampere bei 5 Volt sicherstellt. Trotz der fehlenden 2 Ampere konnte nie eine Beeinträchtigung des Roboterarmes bei gleichzeitiger Bewegung aller Servos festgestellt werden.

B.2. Geth

Zum erfolgreichen compilieren von Geth auf einem Raspberry Pi 3 Model B sind folgende Schritte erforderlich:

1. Installation von git, g++, gcc und make mittels

   ```
   sudo apt-get install git g++ gcc make
   ```

2. Download und Installation von GoLang 1.9.3 mit folgenden Kommandozeilen:

   ```
   # Sicherstellung, dass ältere Versionen von GoLang deinstalliert und vom
   System entfernt sind:
   sudo apt remove golang
   sudo apt-get autoremove
   source .profile
   ```

```
# Manualles Herunterladen des Paketes:
wget https://storage.googleapis.com/golang/go1.9.3.linux-armv61.tar.gz

# Entpacken via:
sudo tar -C /usr/local -xzf go1.9.3.linux-armv61.tar.gz
```

3. Setzen der Umgebungsvariable für golang durch Hinzufügen folgender Zeile in ~/.profile:

   ```
   export PATH=$PATH:~/go-ethereum/build/bin
   ```

4. Neustart

5. Prüfen der GoLang-Version

   ```
   go version
   > go version go1.9.3 linux/arm
   ```

6. Geth-Source mittels github herunterladen:

   ```
   sudo git clone https://github.com/ethereum/go-ethereum.git
   ```

7. Compilieren

   ```
   make geth
   ```

8. Umgebungsvariable wie bei GoLang setzen (hier wird angenommen, das von git geklonte go-ethereum Verzeichnis befindet sich im HOME-Verzeichnis):

   ```
   export PATH=$PATH:~/go-ethereum/build/bin
   ```

9. Neustart

10. Geth-Version prüfen:

    ```
    geth version
    > Version: 1.8.3-stable
    ```

ACHTUNG!

Es kann vorkommen, dass der Compilationsprozess des Geth-Quellcodes aufgrund ungenügendem Arbeitsspeichers abgebrochen wird. In diesem Falle sollten eventuell laufende Dienste und Hintergrundprozesse beendet, und der Compilationsprozess erneut gestartet werden.

B.3. NodeJS

Von Haus aus bietet das voreingestellte APT-Repository des Raspberry PI nur eine veraltete, inkompatible Version der benötigten NodeJS-Umgebung. Um eine aktuellere Version zu installieren, sollte folgender Kommandozeilenbefehl auf der Konsole ausgeführt werden:

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Dies installiert NodeJS Version 8.11.2 und npm Version 5.6.0.

B.4. BCCM Demonstrator

Die Software für den BCCM-Demonstrator befindet sich auf einem GitHub-Repository, und kann von dort geklont werden.

```
git clone https://github.com/jkrause1/BCCMDemonstrator.git
```

Nach erfolgreichem klinen des Repositorys sollten alle benötigten NodeJS-Pakete installiert werden.

```
npm install
```

Der Inhalt des Scripts `startGeth` muss um Einträge des Datenverzeichnisses und der Network-Id der verwendeten private Blockchain ergänzt werden.

ACHTUNG!

Der in Sektion B.3 vorgenommene Schritt installiert eine zu neue(!) Version des npm-Packet-Managers. Mit dieser Version war es zur Zeit des Praktikums nicht möglich, das web3- und node-opcua-Packet fehlerfrei herunterzuladen. Aus bisher unbekanntem Grund ist diese Version nicht in der Lage, sogar mit Administratorrechten Rechte bestimmter Verzeichnisse zu setzen. Es wird empfohlen, npm zur Installation dieser Pakete auf Version 4.6.1 downzugraden. Nach der Installation kann npm wieder in seiner aktuellen Version heruntergeladen verwendet werden.

C. Benutzerleitfaden

C.1. Übersicht

Nach erfolgreicher Installation der BCCM-Demonstrator-Software sollten sich im Verzeichnis folgende Dateien vorfinden:

- startGeth
- startServer
- robotCode.py
- Servos.py
- robotOPCUAServer.js

Weiterhin werden im Verzeichnis der Software weitere Dateien während des Betriebs angelegt:

- robotHistory
- robotOPCUAHistory
- lastServoValues

Im folgendem wird die Bedeutung und Benutzung dieser Dateien erklärt.

C.2. startGeth

Dieses Script startet Geth mit voreingestellten Startparametern, die für die problemlose Funktionalität der BCCM-Demonstrator-Software nötig ist.
Bevor erstmaliger Verwendung ist es nötig, das Datenverzeichnis und die Netzwerk-ID der verwendeten Ethereum-Blockchain innerhalb des Scripts anzupassen.

C.3. startServer

Dies startet die BCCM-Demonstrator-Software mit bevorzugten Startparametern.

C.4. robotOPCUAServer.js

Dieses NodeJS-Script beinhaltet den Quellcode für den OPCUA-Server, des Whisper-Clients und der Schnittstelle zum Python-Interpreter des BCCM-Demonstrators. Nachdem `robotOPCUAServer.js` erfolgreich gestartet wurde kann der Roboterarm angewiesen werden, zuvor mit `robotCode.py` aufgezeichnete Dateien abzuspielen, indem eine Whisper-Nachricht mit dem Inhalt `File=<Dateiname>` unter dem Topic geschickt wird, auf dass der OPCUA-Server lauscht. Das Script verfügt über folgende Startparameter:

Parameter	Beschreibung
`-p`	Dieser Wert gibt den Port des OPCUA-Servers an, über den OPCUA-Clienten mit dem Server kommunizieren können.
`-y`	Gibt das Python-Script an, welches die Schnittstelle zwischen dem BCCM-Demonstrator und den in Python geschriebenen Controller an zur Steuerung des Roboterarmes an.
`-i`	Das Intervall, in welchem der OPCUA-Server den Roboterarm-Controller nach den Servo-Werten fragt.
`-w`	Gibt die vollständige URI zu einem laufendem Geth-Node an. Über diesen Node kommuniziert die Software mit dem Rest des Demonstrators.
`-a`	Interner Name der Anwendung des OPCUA-Servers
`-x`	Gibt das Passwort an, mit dem eingehenden Whisper-Nachrichten entschlüsselt werden. Dieses Password muss für Sender und Empfänger identisch sein.
`-t`	Gibt das Thema an, unter dem auf Whisper-Nachrichten gehört werden soll. Muss beim Sender und Empfänger einer Whisper-Nachricht identisch sein.
`-s`	Teilt dem OPCUA-Server die Anzahl der Servos des verwendeten Roboterarmes mit.

Tabelle C.1.: Startparameter für robotOPCUAServer.js

C.5. Servos.py

Servos.py ist die geschriebene Controller-Software zur Ansteuerung des Roboterarmes, welches auf Verwendung durch den BCCM-Demonstrator hin entwickelt wurde. Es ist zu empfehlen, dieses Script für den `--python`-Startparameter des robotOPCUAServer-Scriptes zu verwenden. Servos.py lässt sich mit folgenden Startparametern benutzen:

Parameter	**Wertebereich**	**Beschreibung**
`--Servo`	1-6	Selektiert einen einzelnen Servo für den `--Method`-Parameter
`--Method`	read; write	Bestimmt, ob der Wert des mit `--Servo` selektierte Servo ausgelesen, oder gesetzt werden soll.
`--Value`	0.4-2.5	Der neue Wert des mit `--Servo` selektierten Servos.
`--Servos`	0.4-2.5 x6	6 hintereinander, kommagetrennte float-Werte. Diese Werte werden ihrer Reihenfolge nach für die Servos des Roboterarmes gesetzt.
`--File`	Ein mit robotCode.py aufgezeichnete Roboterbewegung	Spielt die aufgezeichnete Roboterbewegung ap.
`--Speed`	0, 1, 2, 3	Bestimmt die Geschwindigkeit der Bewegung des Roboterarmes. (Wird nicht mehr verwendet!)

Tabelle C.2.: Startparameter für Servos.py

C.6. robotCode.py

Dieses Script erlaubt die manuelle Steuerung und Aufzeichnung des Roboterarmes. Nach Ausführung erscheint ein Programmfenster dass zur Steuerung des Roboterarmes fokussiert sein muss. Das Programm besitzt folgende Tastenbelegung:

Key	**Beschreibung**
W, S	Bewegt Servo 1 (Links-Rechts-Bewegung)
A, D	Bewegt Servo 2 (Auf-Ab-Bewegung)
R, F	Bewegt Servo 3 (Beuge-Bewegungen)
T, G	Bewegt Servo 4 (Kopf-Auf-Ab-Bewegungen)
Q, R	Bewegt Servo 5 (Kopf-Dreh-Bewegungen)
SHIFT, STRG	Bewegt Servo 6 (Zangen-Bewegung)
+, -	Einstellung Bewegungssensibilität
LAlt+R	Schaltet in/aus den Aufzeichnungsmodus. Erfordert Eingabe eines Namens für die Aufzeichnungsdatei. Diese muss in Anführungszeichen angegeben werden.
O	Abspeichern der momentan Position des Roboterarmes in eine Datei oder, wenn im Aufzeichnungsmodus, als Position in die zuvor erstellte Aufzeichnungsdatei.
ESC	Beendet das Steuerungsprogramm

Tabelle C.3.: Tastenbelegung robotCode.py

C.7. robotHistory

Diese Datei wird vom Steuerungsscript Servos.py angelegt. Es wird für jeden Aufruf ein Eintrag angelegt. Hinterlegt werden Zeitpunkt und Kommandozeilenparameter, mit der das Script aufgerufen wurde, sowie aufgetretene Fehler.

C.8. robotOPCUAHistory

Diese Datei wird vom OPCUA-Server angelegt. Für alle Ereignisse werden Einträge vorgenommen, die aus dem Inhalt des Ereignisses und einem Zeitstempel bestehen. Weiterhin werden aufgetretene Fehler hinterlegt.

C.9. lastServoValues

Diese Datei beinhaltet die letzten Werte für die Servomotoren des Roboterarmes. Es ist nicht möglich, die Werte dieser Servomotoren abzufragen, weswegen sie hinterlegt werden müssen. Diese Werte werden von der Steuerungssoftware geschrieben und bei Bedarf an anfragende Software, hier den OPCUA-Server, mitgeteilt.