



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-19-01

Intelligente Gebäudeklimatisierung auf Basis eines Sensornetzwerks und künstlicher Intelligenz

Johannes Dörfelt · Wolfram Hardt · Christian Rosjat

Februar 2019

Chemnitzer Informatik-Berichte

Abstract

Die vorliegende Masterarbeit erforscht die Erweiterung der Steuerung einer bestehenden Klimatisierungsanlage durch ein Sensornetzwerk und künstliche Intelligenz. Ziel ist die Optimierung des Energiebedarfs und die Verbesserung des Raumklimas durch die Überwachung umfangreicher Sensordaten. Ein *Mesh*-Netzwerk mit Aktualisierungsmöglichkeit für die Sensorknoten und die *TensorForce*-Bibliothek bilden dafür die Grundlage. Unter Nutzung von *Deep Q-Learning from Demonstrations*, einer *Reinforcement Learning*-Strategie, wird die praktische Umsetzung eines solchen Systems getestet. Durch die Erprobung verschiedener Konfigurationen gelingt es, die Aktivitätszeit der Klimageräte durch automatische Abschaltung im Vergleich zur regulären Nutzung signifikant zu senken. Zusätzlich wechselt das entwickelte System bei Bedarf selbstständig in den Lüftungsmodus, anstatt dauerhaft im Kühlbetrieb mit hohem Energiebedarf zu arbeiten. Diese Masterarbeit richtet sich an alle, die am praktischen Einsatz von Reinforcement Learning interessiert sind.

Keywords: Reinforcement Learning, TensorForce, HVAC, Sensornetz

Inhaltsverzeichnis

Inhaltsverzeichnis	4
Abbildungsverzeichnis	6
Abkürzungsverzeichnis	7
1. Einleitung	10
2. Grundlagen	12
2.1. Sensornetz	12
2.2. Klimaanlage	14
2.3. Künstliche Intelligenz	15
2.3.1. Neuronale Netze	16
2.3.2. Lernverfahren	17
3. Stand der Technik	22
3.1. Verwandte Arbeiten	22
3.2. Lösungsansätze	24
3.2.1. Betriebssystem	24
3.2.2. Netzwerkprotokolle	26
3.2.3. Nachrichtenübermittlung	28
3.2.4. Update-Systeme	30
3.2.5. Bedienoberflächen	33
3.2.6. Künstliche Intelligenz	34
3.3. Zusammenfassung	36
4. Konzept	37
4.1. Architekturentwurf	37
4.2. Anbindung der Klimaanlage	38
4.3. Betriebssystem	39
4.4. Sensornetzwerk	39
4.5. Systemaktualisierung der Sensorknoten	41
4.6. Nutzerschnittstelle	41
4.7. Datenaufzeichnung	42
4.8. KI-Agent	42
4.9. Bewertungskriterien	44
4.10. Zusammenfassung	45

INHALTSVERZEICHNIS

5. Umsetzung	47
5.1. Inbetriebnahme der Komponenten	49
5.2. Anbindung der Klimaanlage	50
5.3. Betriebssystem	53
5.4. Sensornetzwerk	54
5.5. Systemaktualisierung der Sensorknoten	61
5.6. Nutzerschnittstelle	63
5.7. Datenaufzeichnung	65
5.8. KI-Agent	68
5.9. Zusammenfassung	77
6. Ergebnis	79
7. Fazit und Ausblick	92
Literaturverzeichnis	95
A. Anhang	105
A.1. Inhaltsverzeichnis der beigelegten CD	105

Abbildungsverzeichnis

1.1. Prinzip einer Klimaanlage [72]	10
2.1. Zwischenstationen in Netzwerkverbindungen	13
2.2. Unterscheidung von Split-HVAC-Typen	14
2.3. Feed Forward Network mit zwei Dense Layers	17
2.4. Prinzip Reinforcement Learning	19
3.1. Netzwerktopologien	27
4.1. Architekturüberblick	37
4.2. Architekturkonzept	45
5.1. Übersicht der Netzwerkstruktur mit Sensornetz und LAN	47
5.2. Überblick der Sensorschaltung [4]	50
5.3. Übersicht der MQTT-Topics	53
5.4. MQTT-Client-Architektur mit Sendeschleife	58
5.5. MQTT-Client-Architektur mit Timer	59
5.6. Anzeige-Tab der Node-RED-Oberfläche	64
5.7. Steuerungs-Tab der Node-RED-Oberfläche	65
5.8. Lernschleife des KI-Agenten	73
5.9. Datenflussdiagramm	77
6.1. Agent mit Kenntnis der Luftfeuchtigkeit im kleinen Raum	84
6.2. Agent mit Kenntnis der Luftqualität im kleinen Raum (1/2)	85
6.3. Agent mit Kenntnis der Luftqualität im kleinen Raum (2/2)	86
6.4. Agent mit Kenntnis der Luftfeuchtigkeit im großen Raum	87
6.5. Agent mit Kenntnis der Luftqualität im großen Raum	88
6.6. Referenzbetrieb ohne KI im kleinen Raum	89
6.7. Referenzbetrieb ohne KI im großen Raum	90

Abkürzungsverzeichnis

AMQP	Advanced Message Queuing Protocol
AP	Access Point
API	Application Programming Interface
Bash	Bourne-again shell
BSP	Board Support Package
CNTK	Microsoft Cognitive Toolkit
CoAP	Constrained Application Protocol
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
DHCP	Dynamic Host Configuration Protocol
DQfD	Deep Q-learning from Demonstrations
DTLS	Datagram Transport Layer Security
FFN	Feed Forward Network
GPIO	General Purpose Input/Output
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HVAC	Heating, Ventilation, Air Conditioning
I2C	Inter-Integrated Circuit
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers

Abkürzungsverzeichnis

IoT	Internet of Things
IPv4	Internet Protocol Version 4
iTC	intelligent Touch Controller
IVI	In-Vehicle Infotainment
JMS	Java Message Service
JSON	JavaScript Object Notation
KI	künstliche Intelligenz
LAN	Local Area Network
LED	Light-Emitting Diode
LoRaWAN	Long Range Wide Area Network
LSTM	Long Short-Term Memory
M2M	Machine-to-Machine
MANet	Mobile Ad Hoc Network
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
NTP	Network Time Protocol
OLPC	One Laptop Per Child
OPC UA	Open Platform Communications Unified Architecture
OSI	Open Systems Interconnection
OTA	over-the-air
PUE	Power Usage Effectiveness
QoS	Quality of Service
RAUC	Robust Auto-Update Controller
REST	Representational State Transfer
RL	Reinforcement Learning
RNN	Recurrent Neuronal Network

Abkürzungsverzeichnis

RTC	Real-Time Clock
RVI	Remote Vehicle Interaction
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SOTA	Software over-the-air
SPI	Serial Peripheral Interface
SSH	Secure Shell
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol-Familie
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
VANet	Vehicular Ad Hoc Network
VOC	Volatile Organic Compound
VRV	Variable Refridgerant Volume
WANet	Wireless Ad Hoc Network
WCF	Windows Communication Foundation
WLAN	Wireless Local Area Network
WSAN	Wireless Sensor and Actor Network
WSN	Wireless Sensor Network
XML	Extensible Markup Language

1. Einleitung

Deutschland liegt in der gemäßigten Klimazone, d. h. die Jahresmitteltemperatur liegt unter 20 °C. Im Gegensatz zum gewerblichen Sektor sind Klimaanlage in Wohnhäusern deshalb eher unüblich. Bürogebäude und Fabriken haben aufgrund der Gestaltung ihrer Fassaden, oft mit großen Glasflächen, und der bspw. durch Produktionsprozesse entstehenden Wärme einen wesentlich höheren Klimatisierungsbedarf [42]. Außerdem erfordern einige Fertigungsverfahren, z. B. in der Elektronikbranche, die Einhaltung bestimmter Grenzwerte bzgl. der Luftfeuchtigkeit, die eine Klimatisierungsanlage voraussetzen. Auch in Supermärkten und der Lebensmittelindustrie wird Kühltechnik benötigt, um verderbliche Waren länger lagern zu können. Abbildung 1.1 zeigt die grundlegende Funktionsweise einer Klimaanlage. All diese Systeme sind durch einen hohen Energiebedarf gekennzeichnet, der hauptsächlich zum Komprimieren und Kühlen eines Kältemittels in einem sogenannten Kondensator aufgewendet wird. Das Gegenstück dazu, die Verdampfer, nutzen das nun flüssige Mittel

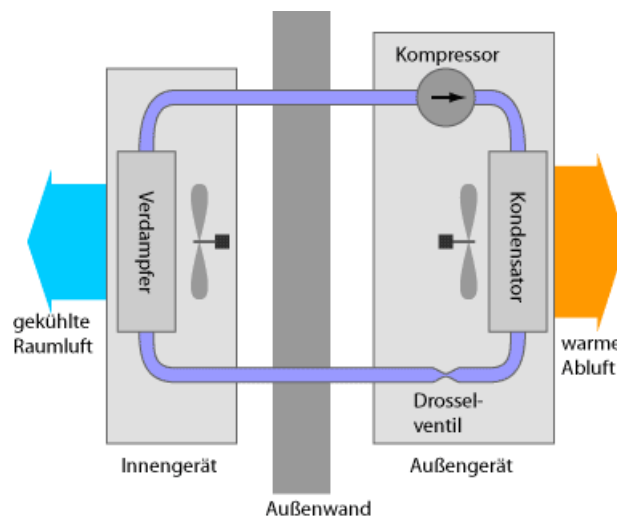


Abbildung 1.1.: Prinzip einer Klimaanlage [72]

zur Kühlung der Luft und leiten es gasförmig zurück. Kompressor und Drosselventil ermöglichen dabei den notwendigen Druckunterschied zwischen beiden Seiten. Allein in Deutschland entfielen 2009 mit 71 Terrawattstunden 14 Prozent des gesamten Stromverbrauchs auf Kältetechnik [39]. Schon bei der Temperatureinstellung ist oftmals Sparpotenzial vorhanden. Zum Beispiel „erhöht sich der Energieverbrauch der Kühlung pro 1 Grad tiefere Raumtemperatur um drei Prozent“ [43]. Einige Hersteller versuchen zusätzlich durch neue Verfahren, wie Wärmerückgewinnung oder dyna-

1. Einleitung

misch anpassbarem Kältemittelvolumen, Abhilfe zu schaffen. In Deutschland ist die vorhandene, gewerblich genutzte Klimatechnik jedoch im Durchschnitt 25 Jahre alt und dementsprechend veraltet. Oft fehlt es „an einer intelligenten Mess-, Steuerungs- und Regelungstechnik“ [41], welche die Effizienz steigern könnte. *DeepMind* bspw. gelang es in einem Versuch, die benötigte Kühlenergie eines Rechenzentrums durch Nutzung von künstlicher Intelligenz bei der Steuerung um bis zu 40 % zu senken [26]. Dabei erlernte das System bis dato unbekannte Steuerungskonzepte.

In Kombination mit den Technologien des *Internet of Things (IoT)*, welche die Vernetzung vieler, räumlich verteilter Geräte ermöglichen, sind die Grundlagen für die Entwicklung intelligenter Steuerungssysteme vorhanden. Darauf aufbauende Sensornetze bieten Möglichkeiten zur umfangreichen Erfassung und Sammlung von Umweltdaten, welche anschließend in die Steuerung einfließen können. Dabei dürfen Sicherheitsaspekte und die Möglichkeit, Systemaktualisierungen durchzuführen, jedoch nicht ins Hintertreffen geraten. Während der gesamten Entwicklungsphase und Lebensdauer der Komponenten müssen diese Gesichtspunkte berücksichtigt werden. Wird das nicht getan, so ist die Infrastruktur Angriffen möglicherweise schutzlos ausgesetzt [33]. Letztendlich kann solch ein Szenario in gigantischen *Botnetzen* aus ebendiesen Geräten resultieren, wie es bei *Mirai* in den Schlagzeilen ersichtlich war [50].

Struktur der Arbeit

Dieses Dokument ist neben dieser Einleitung in die Kapitel Grundlagen, Stand der Technik, Konzept, Umsetzung, Ergebnis und Fazit unterteilt. In Grundlagen wird das Wissen derjenigen Fachgebiete vermittelt, die zum Verständnis der weiterführenden Kapitel dieser Arbeit vonnöten sind. Stand der Technik gibt einen Einblick in thematisch verwandte Arbeiten und begründet die Notwendigkeit, diese Thematik weiter zu verfolgen. Zudem werden bestehende Lösungsansätze verschiedener Aufgabenbereiche dieser Arbeit vorgestellt. Das Kapitel Konzept umfasst die Evaluation der verschiedenen Ansätze. Daraus wird anschließend eine Strategie zur Lösung der gestellten Aufgabe abgeleitet. Im Kapitel Umsetzung wird die konkrete Realisierung des zuvor erarbeiteten Lösungskonzepts beschrieben. Weiterführend erfolgt die Auswertung der erprobten Testszenarien im Kapitel Ergebnis. Es wird bewertet, wie gut verschiedene Ausführungen der Lösung die Aufgabenstellung erfüllt. Abschließend werden im Kapitel Fazit ein Resümee über die gesamte Arbeit gezogen und mögliche Ergänzungen aufgeführt. Der darin enthaltene Ausblick nennt einige Anregungen für weiterführende Arbeiten.

2. Grundlagen

Dieses Kapitel soll das zum Verständnis dieser Arbeit nötige Grundlagenwissen vermitteln. Hierzu wird in den nachfolgenden Abschnitten *Sensornetz*, *Klimaanlage* sowie *Künstliche Intelligenz* auf die verschiedenen Themenschwerpunkte der Aufgabenstellung eingegangen.

2.1. Sensornetz

Eine der Hauptkomponenten dieser Arbeit ist ein Sensornetzwerk. Sensornetzwerke bestehen aus mehreren Sensorknoten, die durch ein Netzwerk miteinander in Verbindung stehen. Diese Knoten erfassen Messwerte und leiten sie über eine Kommunikationsschnittstelle, meist an eine zentrale Instanz, weiter [116]. Die Kommunikation erfolgt in der Regel drahtlos, da die Teilnehmer möglicherweise mobil und räumlich weit voneinander entfernt sind. Für die Vernetzung kommen prinzipiell zwei verschiedene Architekturen infrage: Einerseits die Einbindung in eine zusätzliche Kommunikationsinfrastruktur und andererseits die selbstständige Vernetzung durch ein sogenanntes Mesh-Netzwerk. Im erstgenannten Fall sind die Teilnehmer direkt an einen zentralen Zugangspunkt angebunden und tauschen sämtliche Daten mit diesem aus. Bei Betrachtung dieser Verbindung, dargestellt in Abbildung 2.1a, müssen die Pakete genau einen sogenannten *Hop*, d. h. einen Sprung von der Quelle zum Ziel zurücklegen. Deshalb wird für diese Art der Kommunikation die Bezeichnung *Single-Hop* verwendet. Mesh-Netzwerke sind auch als *Mobile Ad Hoc Networks (MANets)* bzw. *Wireless Ad Hoc Networks (WANets)* bekannt und können weiter untergliedert werden [112]. Unterkategorien sind bspw. *Vehicular Ad Hoc Networks (VANets)* aus dem Automobilbereich und allgemeinere *Wireless Sensor Networks (WSNs)* oder *Wireless Sensor and Actor Networks (WSANs)*, bestehend aus Sensoren und ggf. Aktoren [111]. Die Teilnehmer eines solchen Sensornetzes werden auch als Knoten bezeichnet. Wie in Abbildung 2.1b zu sehen ist, leiten sie Datenpakete untereinander kettenartig weiter bis sie letzten Endes beim Empfänger ankommen. Das setzt sich oft über mehrere Stationen fort, weshalb man auch von *Multi-Hop*-Netzwerken spricht [114].

Des Weiteren wird die Art und Weise der Datenauslieferung unterschieden. Einerseits können die Messwerte von den Sensorknoten auf Abruf bereitgestellt oder andererseits in festgelegten Abständen von ihnen selbst versendet werden. Man spricht hierbei auch von *Push*- bzw. *Pull-Kommunikation*. Als Hardwareplattform finden in Sensornetzen sämtliche Konfigurationen von Mikrocontrollern bis Einplatinenrechner Anwendung. Meist sind sie besonders energiesparend getrimmt und mit entspre-

ments, welcher vom Sensor erfasst wird. Je niedriger der Widerstandswert ist, desto höher ist die Konzentration solcher Verbindungen und dementsprechend niedriger ist die Luftqualität. Es lässt sich jedoch anhand dieses Werts nicht unterscheiden, welche Gase in welcher Konzentration enthalten sind, sondern nur die Gesamtkonzentration [2].

Das resultierende Sensornetz wird somit eine homogene Struktur haben, da die zugrundeliegende Plattform dieselbe ist. Zudem handelt es sich um ein WSN, je nach Auslegung auch um ein WSAN, da sowohl Sensorik, als auch Aktorik in Form der Klimaanlage zumindest indirekt Teil des Netzwerks ist [89]. Weitere Klassifikationen dieses Sensornetzes werden sich im Lauf der Entwicklung herausbilden.

2.2. Klimaanlage

Es handelt sich bei dieser Arbeit um eine Praxisstudie. Demzufolge bildet eine Klimaanlage einen weiteren Grundbaustein. In diesem Fall handelt es sich um ein System des Herstellers *Daikin* zur Heizung, Lüftung und Kühlung, auch unter der englischen Bezeichnung *Heating, Ventilation, Air Conditioning (HVAC)* bekannt. Klimaanlagen können prinzipiell in zwei Bauarten eingeteilt werden. *Split*-Systeme, wie in Abbildung 2.2 dargestellt, bestehen aus Außengeräten, meist auf Gebäudedächern montiert, und Innengeräten, die sich im Gebäudeinneren befinden. Zwischen ihnen zirkuliert ein Kältemittel, welches außen komprimiert, gekühlt und schließlich innen zur Kühlung der Luft verdampft wird. Viele dieser Systeme können den Prozess zudem umkehren, sodass das System die Raumluft erwärmt, statt sie zu kühlen. Meist sind Bedienteile im Raum montiert, über die die gewünschte Solltemperatur eingestellt werden kann. Die zweite Bauart sind sogenannte *Monoblock*-Geräte, welche alle Komponenten des Kühlkreislaufs in einem kompakten Gehäuse vereinen. Sie können jedoch nur einen Raum kühlen, während *Split*-Systeme mit einem Außengerät mehrere Innengeräte bedienen können. In diesem Fall spricht man auch von einer *Multi-Split*-Anlage, dargestellt in Abbildung 2.2b. Abbildung 2.2a hingegen zeigt ein sogenanntes *Mono- oder Single-Split*-System, bei dem an jedes Außengerät jeweils nur ein Innengerät angeschlossen ist [110].

Die im Rahmen dieser Arbeit genutzte HVAC-Anlage besteht aus mehreren Außen-



(a) Single-Split [24]

(b) Multi-Split [23]

Abbildung 2.2.: Unterscheidung von Split-HVAC-Typen

und Innengeräten, deshalb ist hier der erstgenannte Begriff zutreffend. Das System nutzt zudem eine Technik namens *Variable Refrigerant Volume (VRV)* [81], um eine effizientere Klimatisierung zu ermöglichen. Damit ist das System in der Lage, den Kältemittel-Volumenstrom zu jedem Innengerät, je nach Anforderung, zu variieren. Bei dieser Installation überwacht und beeinflusst eine Zentraleinheit, der *intelligent Touch Controller (iTC)*, insgesamt 23 Innengeräte im ganzen Gebäude, welche zusätzlich über Bedienteile in den jeweiligen Räumen eingestellt werden können. Die Innengeräte sind über eine proprietäre Schnittstelle, den *D3-Bus*, an die Zentraleinheit angebunden. Am iTC selbst und per Web-Interface über eine Ethernet-Verbindung kann jedes Innengerät konfiguriert werden. Zusätzlich wird durch Komitec ein Schnittstellen-Modul zur Verfügung gestellt. Dieses ermöglicht die Steuerung und Abfrage der einzelnen Innengeräte mit speziellen Kommandos über das *Hypertext Transfer Protocol (HTTP)*. Die Befehle werden in der zugehörigen Inbetriebnahmeanleitung [22] detailliert beschrieben.

2.3. Künstliche Intelligenz

Grob umrissen bezeichnet der Begriff *künstliche Intelligenz (KI)* Algorithmen bzw. Programme, die selbstständig Aufgaben erfüllen, zu deren Lösung, aus menschlicher Sicht, ein gewisses Maß an intelligentem Verhalten notwendig ist [27]. Intelligenz wird im Duden auch als „Fähigkeit [des Menschen], abstrakt und vernünftig zu denken und daraus zweckvolles Handeln abzuleiten“ [28] umschrieben. Eine genaue Abgrenzung des Begriffs ist daraus nicht ersichtlich. Prinzipiell bietet sich die Verwendung von KI für Probleme an, die sehr komplex oder gar unmöglich algorithmisch formulierbar sind. Oft findet sie auch Anwendung bei der Verarbeitung sehr großer Datenmengen [16]. Mit menschlicher Intelligenz ist dies aber keinesfalls vergleichbar, da es sich stets um Anwendungen, wie Spracherkennung, handelt, auf die ein KI-System speziell ausgerichtet ist. Diese spezialisierte Art wird auch als schwache KI bzw. *Narrow AI* bezeichnet. Starke KI oder *Strong AI* ist das bisher unerreichte Ziel der Forschung, welches eine allgemein anwendbare, künstliche Intelligenz bezeichnet [82]. Im selben Kontext wird oft der Begriff *Machine Learning (ML)* erwähnt, was ein Teilgebiet der KI ist. ML bezeichnet den eigenständigen, künstlichen Wissenserwerb einer Maschine zur Lösung eines Problems. Grundlage dafür bilden Beispiele, aus denen das Wissen automatisiert extrahiert und abstrahiert wird, um es zur Lösung der gestellten Aufgabe anzuwenden. Zur Repräsentation des Wissens existieren symbolische und nicht-symbolische Arten. Erstgenannte bilden das Wissen, bspw. durch Prädikatenlogik, explizit und nachvollziehbar ab. Bei nicht-symbolischen Verfahren, wie künstlichen neuronalen Netzen, erfolgt diese Abbildung implizit, sodass kein Einblick möglich ist [106].

In den folgenden Abschnitten wird hauptsächlich auf das zuletzt genannte ML-Teilgebiet der künstlichen neuronalen Netze eingegangen, weil es die Basis für das hier geplante System darstellt.

2.3.1. Neuronale Netze

Mit künstlichen neuronalen Netzen ist es möglich, Funktionen zu approximieren. Sie sind dem biologischen Vorbild des Gehirns nachempfunden, jedoch stark vereinfacht und bei Weitem nicht im selben Maße leistungsfähig. Dennoch bieten sie in angepassten Konfigurationen für viele Anwendungsfälle eine gute Lösung. Elementarer Bestandteil eines solchen Netzes sind Neuronen. Sie lassen sich als Verarbeitungseinheiten beschreiben, die gewichtete Eingaben entgegennehmen und diese auf eine Ausgabe abbilden. Das geschieht über drei wesentliche Merkmale:

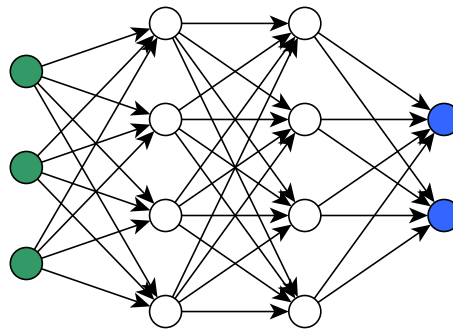
- Propagierungsfunktion, zur Verarbeitung der Eingaben, meist die gewichtete Summe der Eingaben
- Aktivierungs- bzw. Transferfunktion, zur Berechnung der Aktivierung, bspw. *rectified linear unit (ReLU)*
- Ausgabefunktion, zur Umwandlung der Aktivierung in die letztendliche Ausgabe, meist Identitätsfunktion

Sogenannte Bias-Neuronen, welche einen konstanten Wert ausgeben, gewährleisten, dass ein Neuron aktiviert bleibt, wenn sonst alle Eingaben 0 sind und können den Lernvorgang durch eine sonst komplexere Schwellwertfunktion vereinfachen. In Abbildung 2.3 ist der Aufbau eines einfachen neuronalen Netzes dargestellt. Die Neuronen werden in Schichten, sogenannten *Layers*, zusammengefasst. Die erste Schicht wird dabei als Eingabe-, die letzte als Ausgabeschicht und alle dazwischenliegenden als versteckte Schichten bzw. *Hidden Layers* bezeichnet. Die Anzahl von versteckten Schichten und deren Neuronen ist dabei, je nach Anforderung der spezifischen Aufgabenstellung, einstellbar [98]. Bei den Ein- und Ausgabeschichten ist die Neuronenzahl durch die Problemstellung vorgegeben. Je nachdem, wie der Zustandsraum als Eingabe des Netzes modelliert wird, ergibt sich daraus die Größe der Eingabeschicht. Gleichermäßen bestimmt die Definition des Ausgabe- bzw. Aktionsraums die Gestaltung der Ausgabeschicht. Alle Neuronen einer Schicht sind im Regelfall gleich konfiguriert, besitzen also die gleichen Funktionen und Parameter. Einfache neuronale Netze werden auch als *Perzeptron*, bei mehreren versteckten Schichten als *Multilayer-Perzeptron* bezeichnet. Das Verwenden komplexer Netzstrukturen mit einer großen Anzahl an Hidden Layers wird auch mit dem Begriff *Deep Learning* bezeichnet [104].

Die Verbindung der einzelnen Schichten kann auf verschiedene Arten und Weisen erfolgen, wodurch sich letztendlich die Netztopologie ergibt. Als Darstellungsform neuronaler Netze bietet sich ein gewichteter Graph an, dessen Knoten die Neuronen und dessen Kanten die Verbindungen zwischen ihnen sind. Zu den Verbindungsarten zählen bspw. vollständig verbundene Schichten, auch *Dense Layer* genannt, welche jedes Neuron einer Schicht mit allen der folgenden verknüpfen. Des Weiteren gibt es sogenannte *Short-Cut-Layer*, deren ausgehende Verbindungen erst mit der übernächsten oder einer der folgenden Schichten verbunden werden und somit eine Art Abkürzung im Netz bilden. Außerdem besteht die Möglichkeit, Rückkopplungen

2. Grundlagen

einzubauen, sodass von einer Schicht Verbindungen zur vorhergehenden oder noch weiter vorn liegenden Schichten existieren. Damit erhält das neuronale Netz die Fähigkeit, auf Informationen innerhalb von Eingabesequenzen zuzugreifen. Ein Spezialfall davon ist der *Long Short-Term Memory (LSTM)*-Layer, wörtlich übersetzt ein „langanhaltendes Kurzzeitgedächtnis“. Infolgedessen wird es möglich, statt eines einzelnen Zustands einen Verlauf von Zuständen zu betrachten und somit bspw. einen Kontext zu erfassen. Im Gegensatz zu normalen Neuronen besteht eine LSTM-Einheit aus drei *Gates* mit den Bezeichnungen *Input*, *Forget* und *Output* sowie einer inneren Zelle. Dabei bestimmen die Gates entsprechend ihrer Namen jeweils den Einfluss eines neuen Werts auf die Zelle, das Vergessen eines bestehenden Werts und die Berücksichtigung des aktuellen Zellenwerts für die Ausgabe [101]. Das macht sie wesentlich besser kontrollierbar und ermöglicht zudem das Speichern von Informationen über länger Zeiträume, erfordert allerdings auch mehr Rechenaufwand. Diese Art von Layer bilden heutzutage die Grundlage für populäre Spracherkennungs- und Assistenzsysteme sowie das KI-Programm *AlphaGo* [37]. Neuronale Netze ohne Rückkopplungen werden auch als *Feed Forward Networks (FFNs)* bezeichnet. Im Gegensatz dazu lautet die Bezeichnung für Netze, welche Rückkopplungen enthalten, *Recurrent Neuronal Networks (RNNs)* [104].



Eingabeschicht verdeckte Schichten Ausgabeschicht

Abbildung 2.3.: Feed Forward Network mit zwei Dense Layers

2.3.2. Lernverfahren

Künstliche neuronale Netze werden durch Training an das jeweilige Anwendungsszenario angepasst. D. h., dass das Netz erlernt, welche Ausgaben für die erhaltenen Eingaben korrekt sind. Zusätzlich wird das Wissen abstrahiert, um auch für unbekannte Eingabedaten richtige Ergebnisse zu liefern. Dieser Lernvorgang besteht hauptsächlich in dem Anpassen der Kantengewichte zwischen den einzelnen Neuronen durch einen Lernalgorithmus. Prinzipiell existieren dafür drei verschiedene Verfahren: überwachtes Lernen bzw. *Supervised Learning*, unüberwachtes Lernen, auch *Unsupervised Learning* und bestärkendes Lernen, alternativ *Reinforcement Learning (RL)*. Jedes der drei genannten Verfahren eignet sich für bestimmte Anwen-

2. Grundlagen

dungsgebiete. Überwachtes Lernen bietet sich an, wenn Trainingsdaten, im Idealfall in großer Zahl, vorhanden sind. Diese umfassen Datensätze aus Eingaben und den zugehörigen, korrekten Ausgaben, welche das neuronale Netz erlernen und generalisieren soll. Es erfolgt also letztendlich eine Klassifizierung der Eingaben in vorher bekannte Ausgabeklassen. Durch die Generalisierung der Daten soll ein neuronales Netz nach dem Training in der Lage sein, auch für unbekannte, also nicht in den Trainingsdaten vorkommende, Eingaben korrekte Ausgaben zu liefern [104]. Die Kunst ist es hierbei, den Lernvorgang so zu dimensionieren, dass es einerseits nicht zur Überanpassung, dem sogenannten *overfitting* und andererseits nicht zur Unteranpassung, dem *underfitting*, kommt. Letztgenanntes Phänomen tritt meist auf, wenn der Trainingsdatensatz zu klein für das Netz ist oder zu kurz trainiert wird. Somit erzeugt das Netz nicht die gewünschten Ausgaben oder sie unterliegen einer großen Streuung. Overfitting kann hingegen vorkommen, wenn zu viele Trainingsdaten verwendet bzw. zu viele Trainingsiterationen auf den Daten ausgeführt werden und das Netz diese im Endeffekt auswändig lernt. Dadurch ist es zwar in der Lage, die erlernten Beispiele sehr gut wiederzugeben, aber die Abstraktion auf unbekannte Eingaben ist nicht brauchbar. Es gilt deshalb der Grundsatz, bei der Erstellung des Netzes sowohl die Komplexität des Problems als auch die Menge der vorhandenen Trainingsdaten zu berücksichtigen. Zu Beginn sollten entweder scheinbar zu kleine oder zu große Strukturen gewählt und diese schrittweise angepasst werden, solange noch signifikante Verbesserungen zu verzeichnen sind [38].

Bei unüberwachtem Lernen werden ebenfalls Trainingsdaten benötigt. Diese umfassen aber lediglich die Eingaben, auf denen das Netz trainiert wird und die Anzahl der Ausgabeklassen. Das neuronale Netz soll erlernen, selbstständig Muster in den Eingaben zu erkennen und somit eine Klassifizierung derselben durchführen. Im Gegensatz zum überwachten Lernen werden die Klassen aber nicht explizit in den Trainingsdaten vorgegeben. Dementsprechend erfolgt eine Einteilung der Eingaben in eine bekannte Anzahl unbekannter Klassen. Dieses Vorgehen ist unter dem Begriff *Clustering* zusammengefasst [104].

Bestärkendes Lernen enthält Komponenten aus beiden zuvor genannten Lernverfahren. Hierbei werden keine separaten Trainingsdaten verwendet, sondern das neuronale Netz ist Bestandteil eines sogenannten Agenten, der in direkter Interaktion mit seiner Umgebung lernt. Grundlage dafür ist der sogenannte Markow-Entscheidungsprozess. Der Agent befindet sich in einem Zustand und gelangt durch eine Aktion mit einer Wahrscheinlichkeit in einen neuen Zustand. Dieser Übergang ist dabei nur vom Ausgangszustand abhängig und nicht von vorhergehenden Zuständen oder Aktionen. Damit ist die Markow-Eigenschaft erfüllt. Wie in Abbildung 2.4 zu sehen ist, erhält der Agent den aktuellen Zustand als Eingabe. Der Zustandsraum, welcher letztendlich das Format der Eingaben bestimmt, muss vorher modelliert werden. Dabei ist darauf zu achten, dass in vielen Fällen, vor allem bei Realwelt-Einsätzen, nicht die komplette Umwelt erfasst werden kann, da sie zu komplex ist. Deshalb müssen die Zustandsparameter für das jeweilige Problem sinnvoll selektiert und ggf. in ein passendes Format konvertiert werden, damit der Agent nicht durch unnötigen Datenmengen irritiert wird. Diese eingeschränkte Sichtweise der Umge-

2. Grundlagen

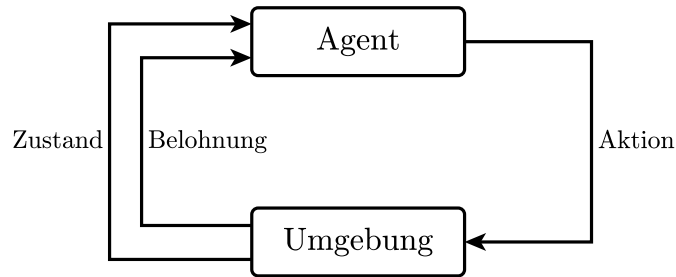


Abbildung 2.4.: Prinzip Reinforcement Learning

bung bezeichnet man auch als *partielle Wahrnehmung*. Entsprechend des Eingabezustands und seines bisherigen Erfahrungsschatzes wählt der Agent eine Aktion aus seinem Aktionsraum, welcher analog zum Zustandsraum modelliert werden muss. Diese wird durchgeführt und resultiert in einer Zustandsänderung der Umgebung. Zusammen mit einer Bewertung bildet der neue Zustand wiederum die Eingabe des Agenten. Durch die Bewertung, auch *Reward* oder *Return* genannt, welche als Belohnung oder Bestrafung formuliert sein kann, erfährt der Agent, wie gut oder schlecht die ausgeführte Aktion im letzten Zustand war. Darauf aufbauend ist er in der Lage, seine Strategie in Bezug auf die Wahl der Aktion, die sogenannte *Policy*, zu optimieren, sodass er den maximalen Reward erhält. Diese Bewertung wird mit Hilfe einer Belohnungsfunktion beschrieben. Die Gestaltung der Funktion ist auch unter dem Begriff *Reward-Shaping* bekannt. So kann bspw. schon die Annäherung an einen Sollwert ansteigend belohnt werden, was zu einem stetigeren Lernerfolg führt, aber viel Zeit zum Finden einer optimalen Lösung beansprucht. Alternativ kann nur für einen Zielzustand eine Belohnung und sonst 0 vergeben werden, was u. U. eine längere Zeit ohne Lernerfolg bedeutet, aber dann schnell zur optimalen Lösung führt. Da Agenten den erwarteten Return für die Wahl einer Aktion intern aufsummieren, würde er bei kontinuierlicher Operation ins Unendliche divergieren. Um dies zu verhindern, können zwei Maßnahmen getroffen werden. Die erste umfasst das Setzen der *Discount-Rate* auf kleiner 1 . Damit wird umgangssprachlich die Weitsicht des Agenten für seine Aktionswahl eingeschränkt, also wie stark in der Zukunft liegende, erwartete Returns bei der Berechnung berücksichtigt werden. In der Lernformel am Ende dieses Abschnitts ist die Verwendung des Parameters zu sehen. Zum anderen werden sogenannte Episoden definiert. Sie stellen jeweils einen Zeitabschnitt dar, bestehend aus einer bestimmten Anzahl von Schritten, in dem der Return betrachtet wird. Nach der festgelegten Anzahl von Schritten wird der aktuelle Abschnitt beendet und eine neue Episode begonnen. Ein Agent muss nach der *Trial-and-Error-Methode* alle möglichen Aktionen in jedem Zustand ausprobieren, um seine Umgebung zu erkunden. Nur so kann er Erfahrungen sammeln und daraus eine möglichst optimale Lösung des Problems finden. Diesen Erkundungsvorgang nennt man auch *Exploration*. Das Gegenstück dazu stellt die sogenannte *Exploitation* dar, welche das Ausnutzen der bisherigen Erfahrungswerte zur Maximierung der Belohnung bezeichnet. Zwischen den Anteilen dieser Komponenten

2. Grundlagen

muss ein gutes Verhältnis eingestellt werden, sodass der Agent einen größtmöglichen Teil seiner Umwelt erkunden und eine optimale Strategie entwickeln kann. Ein häufig gemachter Kompromiss besteht darin, zu Beginn des Lernens einen hohen Anteil an Exploration zuzulassen und diesen über den Zeitverlauf zugunsten einer höheren Exploitation zu reduzieren [104]. Bezüglich der Policy existieren zwei Klassifizierungen: *Off-Policy* und *On-Policy* sowie *Open Loop* und *Closed Loop*. Findet die Auswahl einer Aktion gemäß der aktuell vorliegenden Policy statt, so spricht man von einem On-Policy-Verfahren [77]. Ist die Auswahl unabhängig von der aktuellen Policy, bspw. zufällig oder konstant vorgegeben, so bezeichnet man das Vorgehen als Off-Policy. Diese Art bietet sich vor allem für die Erkundung der Umwelt an. Eine *Open Loop Policy* beschreibt die Ausführung einer zuvor geplanten Sequenz von Aktionen, ohne zwischen den Schritten auf äußere Einflüsse oder Veränderungen zu reagieren. Bei einer *Closed Loop Policy* hingegen handelt es sich um ein reaktives Vorgehen, bei dem der aktuelle Zustand Einfluss auf die Wahl der nächsten Aktion hat. Erfolgt das Lernen des KI-Agenten Schritt für Schritt in der echten Umgebung, so wird es als *Online-Training* bezeichnet. Alternativ kann das Training auch *offline* stattfinden. D. h., es werden bspw. mit einem Simulator der eigentlichen Umgebung viele aufeinanderfolgende Trainingsschritte in kurzer Zeit durchgeführt. Diese werden zusammengefasst und anschließend als sogenannter *Batch* ausgewertet [104]. Wenn der Agent aus den Beobachtungen ein Modell der Zustandsübergänge und Rewards seiner Umwelt erstellt und daraus eine Policy entwickelt, spricht man von *model-based RL*. Im Gegensatz dazu handelt es sich um *model-free RL*, wenn der Agent eine Abbildung der Zustände oder Aktionen auf einen Nutzen verwendet, um eine Strategie zu entwickeln [77]. Viele Algorithmen basieren auf *Temporal Difference Learning*. Dabei bewertet der Agent einen Zustand durch eine *State-Value-Funktion*. Es handelt sich also um *model-free RL*. In diese Funktion fließen nach jedem Schritt die erhaltene Belohnung und die geschätzte zukünftige Erwartung ein. Ältere Algorithmen mussten auf die Bewertung der Strategie am Ende einer Episode warten, bevor sie die Policy anpassen konnten. Mit Hilfe einer Strategie auf Grundlage des geschätzten zukünftigen Rewards der State-Value-Funktion wird eine Aktion ausgewählt. Ein häufig verwendetes Verfahren ist hierbei *ϵ -Greedy*. Entsprechend der Wahrscheinlichkeit ϵ wird entweder die Aktion mit dem höchsten erwarteten Reward oder eine zufällige ausgeführt. Damit arbeitet der Algorithmus nach dem Off-Policy-Prinzip [109]. Die sogenannte *Monte-Carlo-Methode* bezeichnet ein anderes Verfahren, welches die Aktionen nur durch Zufall und somit komplett unabhängig von der aktuellen Policy wählt [104].

Q-Learning ist eine Abwandlung von Temporal Difference Learning, die statt der State-Value-Funktion eine *Action-Value-Funktion* mit der Bezeichnung Q zur Bewertung verwendet. Dementsprechend wird nicht der Zustand allein zur Schätzung des erwarteten Rewards herangezogen, sondern auch die möglichen Aktionen. Folgende Formel beschreibt den Lernvorgang mit Hilfe der Action-Value-Funktion [104]:

2. Grundlagen

$$Q(s_t, a)' = (1 - \alpha) Q(s_t, a) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a))$$

Q' neuer Q-Wert	r_t Return
Q aktueller Q-Wert	a Aktion
s_t aktueller Zustand	α Lernrate
s_{t+1} Folgezustand	γ Discount-Rate

3. Stand der Technik

Nachdem die Grundlagen der einzelnen Themenschwerpunkte dieser Arbeit vermittelt wurden, werden in diesem Kapitel verwandte Arbeiten vorgestellt und diese Arbeit davon abgegrenzt. Außerdem erfolgt die Betrachtung ausgewählter Lösungsansätze für die einzelnen Bereiche der Aufgabenstellung in den Abschnitten *Betriebssystem, Netzwerkprotokolle, Nachrichtenübermittlung, Update-Systeme, Bedienoberflächen* und *Künstliche Intelligenz*.

3.1. Verwandte Arbeiten

Die folgenden Arbeiten stehen inhaltlich im Zusammenhang mit dieser Arbeit und sollen deshalb näher betrachtet und von der hier behandelten Aufgabenstellung abgegrenzt werden.

Inspiration dieser Arbeit ist ein Blog-Eintrag von *DeepMind* mit dem Titel „DeepMind AI Reduces Google Data Centre Cooling Bill by 40%“ [26], der bereits in Kapitel 1 erwähnt wurde. Das Unternehmen, welches 2014 von *Google* übernommen wurde und nun zu *Alphabet* gehört, hat sich auf die Erforschung und Anwendung künstlicher Intelligenz spezialisiert. Im angesprochenen Artikel aus dem Jahr 2016 wird aufgeführt, wie mehrere neuronale Netze mit gesammelten Sensordaten trainiert und in Rechenzentren eingesetzt werden, um die *Power Usage Effectiveness (PUE)* zu optimieren. Die PUE beschreibt das Verhältnis der Gesamtenergieaufnahme des Rechenzentrums zur Energieaufnahme der informationstechnischen Systeme. Das System war, u. a. durch Vorhersagen der neuronalen Netze, in der Lage, die benötigte Kühlenergie des Gebäudes um 40 Prozent zu reduzieren.

Im Paper „Reinforcement learning in feedback control - Challenges and benchmarks from technical process control“ [97] geht es um die Anwendung von RL zur Regelung technischer Prozesse. Es werden viele problemspezifische Herausforderungen behandelt, u. a. veränderliche Sollwerte, nichtlineare dynamische Effekte und der Einfluss externer Größen. Zusätzlich werden vier verschiedene Benchmark-Experimente durchgeführt, die ebendiese Probleme adressieren.

Das Paper „Reinforcement learning for energy conservation and comfort in buildings“ [94] beschreibt die Nutzung von RL, um mit minimalem Energieaufwand entsprechende Level an Komfort in Gebäuden zu erreichen. Dazu wird eine Steuerung entwickelt, die das Temperaturgefühl der Personen im Gebäude, die Luftqualität und die Energieaufnahme überwacht. Diese wird schlussendlich mit einer Fuzzy-Logik-Steuerung und einer einfachen Ein-/Aus-Steuerung verglichen. Das Ergebnis zeigt, dass der RL-Controller nach einigen simulierten Trainingsjahren besser agiert

3. Stand der Technik

als die beiden Vergleichsobjekte.

„Robust Reinforcement Learning Control“ [102] und „Robust Reinforcement Learning Control (with Static and Dynamic Stability)“ [103] sind zwei Paper, die behandeln, wie Bedingungen zur Robustheit formuliert und in ein neuronales Netz eingebaut werden können. Damit ist es möglich, Phasen instablen Verhaltens während des Trainings bei letztendlich gleichem Resultat zu vermeiden. Erreicht wird dies durch das Festlegen einer maximalen Unschärfe der Kantengewichte zwischen Neuronen, die trotzdem noch stabiles Verhalten ermöglicht. Dieser Wert wird anschließend zur Beschränkung der Gewichtsänderung in der Lernphase genutzt. In der Masterarbeit „Neural Networks and PI Control using Steady State Prediction Applied to a Heating Coil“ [95] wird ein Versuch evaluiert, Heizwendeln eines HVAC-Systems mit einem neuronalen Netz zu steuern. Dabei soll das Netz Parameter zum Erreichen des gewünschten Temperaturgleichgewichts voraussagen. Hierbei wird eine Steuerung in einen bestehenden Regelkreis eingebracht, welche den Regler zum richtigen Zeitpunkt zurücksetzt und damit ein wesentlich schnelleres Erreichen des Zielzustands bewirkt. Außerdem wird ein neues Modell für Heizwendeln entwickelt und diskutiert, was die Grundlage für weitere Verbesserungen bildet.

Im Paper mit dem Titel „Optimal control of HVAC and window systems for natural ventilation through reinforcement learning“ [92] wird eine Strategie entwickelt, um natürliche Lüftung und ein HVAC-System durch eine RL-Steuerung zu koordinieren. Dabei kommt *model-free Q-Learning* zum Einsatz, um sowohl Energieaufnahme als auch Temperaturabweichungen zu minimieren. Dazu werden jeweils Innen- und Außenmesswerte der Umgebung ausgewertet, um die beste Entscheidung für Kurz- und Langzeitziele zu treffen. Der Vergleich gegenüber einer regelbasierten Heuristik in der Simulation zeigt, dass der RL-Ansatz mit geringerem Energieverbrauch und gleichzeitig weniger Stunden mit Temperaturabweichungen und weniger Stunden mit hoher Luftfeuchtigkeit besser arbeitet. Zudem ermöglicht RL das selbstständige Anpassen an bestimmte Ziele und die Einbeziehung von Nutzungszeiten und -verhalten, wozu die heuristische Steuerung gar nicht oder nur durch manuelle Anpassung an die speziellen Gegebenheiten in der Lage ist.

Bei „Deep Reinforcement Learning for Building HVAC Control“ [120] wird RL genutzt, um eine effektivere Steuerungsstrategie für das HVAC-System eines Gebäudes zu entwickeln. Intelligenterer Zeitplanung und Vorteile beim Umgang mit der komplexer Thermodynamik sowie vielfältigen Umwelteinflüssen bieten großes Potential für Verbesserungen. Durch Simulation wird gezeigt, dass der Ansatz effektiver als herkömmliche, regelbasierte Steuerungen ist, während die Temperatur im angestrebten Bereich gehalten wird.

In „A Long-Short Term Memory Recurrent Neural Network Based Reinforcement Learning Controller for Office Heating Ventilation and Air Conditioning Systems“ [119] wird ebenfalls eine HVAC-Steuerung entwickelt, hier mit einem *model-free actor-critic* RL-Controller. Es kommt eine Variante von RNNs, ein sogenanntes LSTM-Netz, zum Einsatz. Die Steuerung wurde konzipiert, um den Temperaturkomfort in Kombination mit der Energieaufnahme zu optimieren. Per Simulation wird gezeigt, dass das genutzte Verfahren den Temperaturkomfort um durchschnitt-

lich 15 % und die Energieeffizienz um durchschnittlich 2,5 % verbessert, verglichen zu Kontrollschemas für idealen Temperaturkomfort und der herkömmlichen Steuerung.

Die Arbeit „Air Conditioning Control System Learning Sensory Scale Based on Reinforcement Learning“ [121] beschreibt eine Klimaanlagesteuerung, welche auf den Empfindungen der Nutzer basiert. Ziel ist es, die Funktion von bestehenden Klimaanlagen mit schlechter Leistung zu verbessern. Nutzer geben dazu ihre Temperaturempfindung via Computer oder Smartphone an die Steuerung weiter. Diese wählt, entsprechend der mit Q-Learning erlernten Strategie, eine entsprechende Aktion. Für die Integration von Empfindungen mehrerer Nutzer werden verschiedene Möglichkeiten vorgestellt. Außerdem werden, um zusätzlich Energiesparmaßnahmen umzusetzen, verschiedene Arten von Belohnungsfunktionen vorgestellt. Vier der Methoden werden hinsichtlich ihrer Effektivität in einer Simulation evaluiert.

„TUCool“ [80] ist ein Forschungsprojekt der TU Chemnitz, um den Energieverbrauch der bestehenden Umluftklimatisierung von Rechenzentren zu optimieren. Dazu werden flächendeckende Informationen, welche direkt aus den vorhandenen Hardwarekomponenten ausgelesen werden, mit maschinellem Lernen zu einer intelligenten Regelung kombiniert. Diese lernt wiederkehrende Szenarien und ist somit in der Lage, Lastspitzen vorauszuahnen und diese besser auszugleichen.

Das im Rahmen dieser Arbeit geplante System soll im Gegensatz zu vielen der vorgestellten, simulativ erprobten Arbeiten eine Praxisstudie darstellen. Dazu ist es vorgesehen, ein KI-System unter Nutzung von RL, wie es in einigen genannten Ansätzen auch genutzt wird, mit einer bestehenden Klimatisierungsanlage zu kombinieren. Im Gegensatz zu TUCool und DeepMind geht es hierbei jedoch nicht um die Kühlung eines Rechenzentrums, sondern um die Klimatisierung von Räumen in Wohn- bzw. Bürogebäuden.

3.2. Lösungsansätze

In diesem Abschnitt erfolgt die Vorstellung verschiedener Lösungsansätze, welche für die Umsetzung der Aufgabenstellung infrage kommen.

3.2.1. Betriebssystem

Bei eingebetteten Systemen auf Linux-Basis, wie sie im Sensornetzwerk dieser Arbeit vorkommen, existieren verschiedene Möglichkeiten, ein Betriebssystem bereitzustellen. Die wohl unkomplizierteste Methode umfasst die Verwendung einer meist zu den Einplatinenrechnern mitgelieferten und zugeschnittenen Linux-Distribution. Sie enthalten oftmals vorinstallierte Softwaremodule für den schnellen Einstieg, die im laufenden Betrieb ungenutzt bleiben und Sicherheits- oder Stabilitätsrisiken mit sich bringen können. Das erhöht den Speicherplatzbedarf, welcher auf solcher Hardware nicht in Unmengen zur Verfügung steht und sich bei Systemaktualisierungen ggf. negativ auf die benötigte Bandbreite und Zeit niederschlägt. Es sollte deshalb

3. Stand der Technik

in Betracht gezogen werden, ein maßgeschneidertes Linux-System für den jeweiligen Anwendungsfall mittels eines Baukastensystems selbst zu erstellen. Dies verlagert den Aufwand im Projektzeitraum nach vorn und macht ihn vorhersehbarer, da eventuelle Fehler bei der Verwendung fertiger Distributionen durch eine minimale Softwareauswahl von vornherein ausgeschlossen oder deren Ursachen zumindest berechenbarer werden. Im Folgenden sollen dementsprechend insgesamt drei unterschiedliche, freie Baukastensysteme vorgestellt werden [105].

Das *Yocto Project* [85] ist ein weit verbreitetes Projekt der *Linux Foundation*, welches auf das *OpenEmbedded*-Framework als Build-System aufsetzt. Es ermöglicht die Konfiguration eines kompletten Linux-Systems in beliebigem Umfang mit sogenannten *Rezepten*. Sie sind vergleichbar mit Softwaremodulen, ähnlich zu Paketen aus anderen Linux-Distributionen, die in das Zielsystem installiert werden. Rezepte werden entsprechend ihrer Funktionalität zu sogenannten *Layers* zusammengefasst, die in die Build-Umgebung eingebunden werden und enthaltene Rezepte zur Installation bereitstellen. Beispiele dafür sind *meta-network* als Sammlung von Netzwerksoftware und *meta-python* mit vielen Modulen für die Programmiersprache *Python*. Layer-Namen beginnen konventionsgemäß immer mit „meta-“, wobei der Basis-Layer nur mit „meta“ benannt ist. Viele vordefinierte Layer sind im *Layer Index* von OpenEmbedded [61] zu finden. Grundlegend beginnt die Entwicklung eines eigenen Systems mit der Yocto-Referenz-Distribution *Poky*. Diese umfasst bereits die nötigsten Layer und Rezepte für ein lauffähiges Basissystem, bspw. *Bootloader*, *Kernel* und *Init-System*. Durch eigene Layer mit selbsterstellten Rezepten kann man das System den eigenen Wünschen und Vorstellungen entsprechend anpassen. Anschließend wird die Software der ausgewählten Rezepte durch das Erstellungswerkzeug „bitbake“ in mehreren Schritten erstellt und in das Zielsystem installiert. Je nach Konfiguration liegt das Resultat am Ende des Vorgangs als Speicherkartenabbild, Dateiarchiv oder in einem anderen Format vor. Dieses wird auf die Ziel-Hardware gebracht und kann dort gestartet werden. Sogenannte *Board Support Packages (BSPs)* bilden die Abhängigkeiten zur Unterstützung verschiedener Hardware-Plattformen ab.

Mit *PTXdist* [73] steht ein weiteres Baukastensystem zur Verfügung, welches ein abweichendes Konzept verfolgt. Die Konfiguration des Zielsystems erfolgt hier nicht durch das Editieren von textuellen Konfigurationsdateien, sondern grafisch. Dabei kommt das von der Konfiguration des Linux-Kernels bekannte *Kconfig* zum Einsatz. Der Hauptbeitrag zum Projekt kommt von der *Pengutronix GmbH* [67], wobei betont werden muss, dass es sich dennoch um freie Open-Source-Software handelt. Um ein Linux-System mit *PTXdist* zu erstellen, wird zunächst eine Version der Software an sich benötigt, dazu die *OSELAS-Toolchain* und ein BSP für die vorliegende Hardware. Bei *PTXdist* muss strikt darauf geachtet werden, dass die BSP-Version mit dem Release von *PTXdist* selbst zusammenpasst. Auch dieses System bietet Referenzprojekte in Form von *DistroKit* für einen schnellen Einstieg an. Zusätzlich sind weitere freie Softwarekomponenten, welche von Pengutronix [67] mitentwickelt werden, eingebunden. Dazu gehören z. B. der Update-Client *Robust Auto-Update Controller (RAUC)* [74] oder der Bootloader *Barebox* [8]. Nicht enthaltene Software kann durch die Erstellung eigener Pakete für das Build-System ergänzt wer-

den, dementsprechend muss auch die Kconfig-Oberfläche umkonfiguriert werden. Mit PTXdist lassen sich außerdem verschiedene Kombinationen aus Hardwareplattform und Software-Konfiguration innerhalb eines Projektes abbilden, die sich noch zusätzlich durch sogenannte *Collections* verfeinern lassen. Dadurch wird es möglich, verschiedenste Einsatzszenarien mit relativ geringem Anpassungsaufwand umzusetzen.

Buildroot [18] ist das dritte System dieser Kategorie. Es zeigt viele Ähnlichkeiten zum vorhergehenden PTXdist und wird in abgewandelter Form zur Erstellung der alternativen Router-Firmware im Rahmen des *OpenWrt*-Projektes [65] genutzt. Die Konfiguration erfolgt ebenfalls mit Kconfig bzw. gleichwertigen Schnittstellen, wie *xconfig* oder *gconfig*. Ähnlich ist zudem die Notwendigkeit einer speziellen Toolchain, entweder von Buildroot selbst oder vom Hardware-Hersteller, die separat eingebunden werden muss. Unterschiede gibt es bei der Erstellung eines Projektes in verschiedenen Varianten. In der Dokumentation von Buildroot wird eine Möglichkeit genannt, eigene Anpassungen in einem sogenannten „external tree“ durchzuführen, sodass diese Änderungen gekapselt vom Grundsystem bleiben und bei Bedarf ersetzt oder verändert werden können [19]. Außerdem bietet Buildroot, ähnlich zur Architektur des Yocto Projects, *Layered Customizations*, die Änderungen enthalten, welche über sogenannte *Items* für mehrere Projekte in unterschiedlichem Umfang angewendet werden können [20].

In Tabelle 3.1 sind alle vorgestellten Ansätze als Übersicht zusammengefasst.

Software	Konfiguration	Unterstützer	Besonderheiten
Yocto Project	textuell (Rezepte, Layer)	Linux Foundation	-
PTXdist	grafisch (Kconfig)	Pengutronix GmbH	RAUC, Barebox
Buildroot	grafisch (Kconfig u. Ä.)	-	Nutzung bei OpenWrt

Tabelle 3.1.: Vergleich der Build-Systeme

3.2.2. Netzwerkprotokolle

Wie in Abschnitt 2.1 bereits angedeutet, erfolgt die Kommunikation in einem Sensornetzwerk meist drahtlos. Dafür stehen die verschiedensten Kommunikationsschnittstellen, bspw. Bluetooth, Zigbee oder *Long Range Wide Area Network (LoRaWAN)*, zur Auswahl. Die Möglichkeiten sind in diesem Fall durch die vorgegebene Hardware allerdings auf diejenigen beschränkt, welche auf WLAN-Chips aufsetzen. Diese spalten sich in die traditionelle WLAN-Vernetzung mit externer Infrastruktur in Form von *Access Points (APs)* und in die sogenannte Mesh-Vernetzung, wobei sich jeder Teilnehmer mit allen anderen in seiner Reichweite verbindet. Erstgenannte Variante ist in der Heimvernetzung gang und gäbe, stellt aber für großflächige Netzwerke oft

3. Stand der Technik

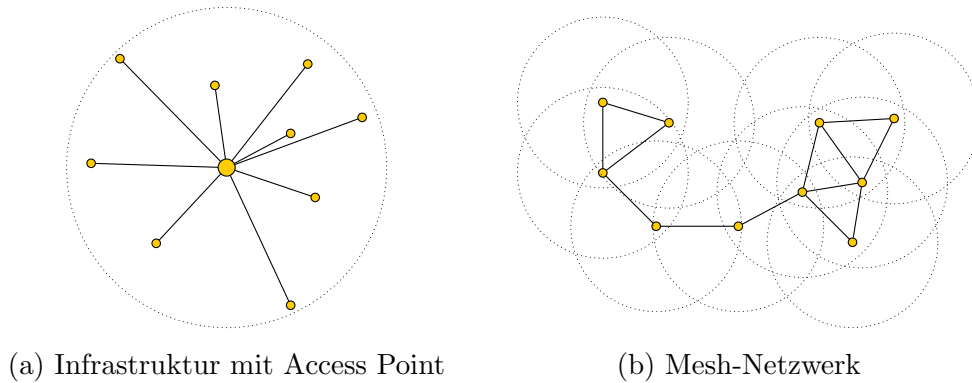


Abbildung 3.1.: Netzwerktopologien

einen erheblichen Mehraufwand an Hardware für die Infrastruktur dar und ist zudem anfälliger im Hinblick auf Ausfälle durch die Zentralität. Diese Art der Vernetzung ist in Abbildung 3.1a grafisch dargestellt. In diesem Fall wird von einem oder mehreren, verbundenen APs ein WLAN-Netz aufgespannt, mit dem sich Sensorknoten innerhalb der Reichweite verbinden und anschließend kommunizieren können. Bei den bereits genannten Mesh-Netzwerken, welche in Abbildung 3.1b abgebildet sind, vernetzen sich die Sensorknoten untereinander. Das bedeutet für den Ausfall eines Knotens, dass die Unterbrechung mit einer entsprechenden Anzahl von anderen Knoten in Reichweite umgangen werden kann. Dazu wird ein dynamisches *Routing*-Verfahren vorausgesetzt, welches die Pakete entsprechend umleiten kann. Für Mesh-Netzwerke auf WLAN-Hardware werden zwei verschiedene Ansätze untersucht, die jeweils auf Schicht 2 des *Open Systems Interconnection (OSI)*-Referenzmodells arbeiten.

B.A.T.M.A.N. Advanced [9], für *Better Approach To Mobile Ad-hoc Networking*, ist ein freies Projekt, welches u. a. von der Freifunk-Initiative entwickelt und verwendet wird. Es setzt WLAN-Hardware voraus, welche den sogenannten *Ad-hoc*-Modus unterstützt. Dieser Modus wird normalerweise verwendet, um genau zwei WLAN-fähige Geräte direkt miteinander zu koppeln. Für die Verwendung muss ein Modul für den Linux-Kernel geladen werden. Anschließend kann das Netzwerk bspw. mit dem *batctl*-Werkzeug konfiguriert werden. Ein solches Mesh-Netzwerk wird auf der Projekt-Webseite mit einer *Emulation eines virtuellen Netzwerk-Switches aller teilnehmenden Knoten* beschrieben.

Der zweite Ansatz ist als 802.11s [100] Teil des sogenannten *WLAN-Standards* 802.11 der *Institute of Electrical and Electronics Engineers (IEEE)*. Der Unterschied liegt auf den ersten Blick darin, dass hier nicht der bestehende Ad-hoc-Modus der Schnittstelle genutzt wird, sondern mehrere eigene Modi bereitgestellt werden [40]. Der wichtigste davon ist der *Mesh Point*-Modus, welcher für innere Knoten des Netzwerks verwendet wird. Außerdem stehen, je nach Hardwareunterstützung, noch die Modi *Mesh AP*, für Knoten, welche einen mit dem Mesh verbundenen AP für nicht mesh-fähige Geräte bereitstellen und *Mesh Portal*, für die Verbindung des Meshs mit

3. Stand der Technik

anderen Netzwerken und somit bspw. dem Internet, zur Verfügung. In der Praxis wurde ein solches Mesh-Netzwerk größeren Ausmaßes zuerst im Rahmen der Initiative *One Laptop Per Child (OLPC)* [62] umgesetzt [29].

Tabelle 3.2 fasst alle vorgestellten Ansätze als Übersicht zusammen.

Kategorie	Variante	Zentralität	Skalierung	Voraussetzung
Infrastruktur	802.11g/n u. Ä.	hoch	aufwändig	Access Point(s)
Mesh	BATMAN adv	niedrig	einfach	Ad-hoc-Modus
	802.11s	niedrig	einfach	Mesh-Hardware

Tabelle 3.2.: Vergleich der Vernetzungsstrategien

3.2.3. Nachrichtenübermittlung

Nachdem in Unterabschnitt 3.2.2 die Grundlagen der Kommunikationsinfrastruktur im Sensornetzwerk behandelt wurden, wird nun auf die Übertragung der Nachrichten eingegangen. Diese nutzt das Netzwerk als Medium und folgt einem Übertragungsprotokoll, für welches wiederum verschiedene Lösungen infrage kommen.

Die erste Möglichkeit besteht darin, ein eigenes Protokoll, basierend auf Sockets mit dem *Transmission Control Protocol (TCP)* bzw. *User Datagram Protocol (UDP)* zu entwickeln. Einerseits sind dadurch sämtliche Freiheiten in der Umsetzung gegeben, andererseits erhöht jede zusätzliche Funktionalität den Mehraufwand bei der Entwicklung. Ein weiteres Augenmerk liegt auf den Verbindungen selbst, denn jede Datenübertragung erfordert die Informationen zum jeweiligen Ziel der Nachricht. Bei anderen Lösungsansätzen existiert ein zentrales Verzeichnis, das die Informationen aller Clients auf Abruf bereitstellt. Analog dazu müssen diese Informationen bei diesem Socket-Ansatz ebenfalls bereitstehen. Zusätzlich ist es erforderlich, ein Datenformat festzulegen, wonach die verschiedenen Informationstypen übermittelt werden und über die Absicherung nachzudenken, sodass keine ungewollten Zugriffe und Manipulationen möglich sind. Des Weiteren müssen Fehlerbehandlungen, beispielsweise bei Verbindungsabbrüchen, berücksichtigt werden. All diese Faktoren fordern ein erhebliches Maß an Entwicklungsaufwand, der im Rahmen dieser Arbeit nicht zweckmäßig wäre. Demnach werden nun umfassendere, bestehende Lösungen vorgestellt.

Das *Constrained Application Protocol (CoAP)* [21] ist ein Protokoll, welches auf den *Representational State Transfer (REST)* aufbaut. D. h., dass es in Form von Methoden, wie *GET* und *POST*, Zugriff auf Ressourcen gibt, die durch *Uniform Resource Locators (URLs)* beschrieben werden. CoAP ist speziell für IoT-Anwendungen konzipiert. Mit geringem Ressourcenbedarf ist es selbst für stark eingeschränkte Hardware, wie Mikrocontroller, geeignet. Das Protokoll unterstützt die Übertragung verschiedener Datenformate, darunter die *Extensible Markup Language (XML)* und die *JavaScript Object Notation (JSON)*, mit einer entsprechenden Kennzeichnung per UDP. Zudem werden sämtliche Verbindungen standardmäßig mit *Datagram Trans-*

3. Stand der Technik

port Layer Security (DTLS) verschlüsselt.

Message Queuing Telemetry Transport (MQTT) [52] stellt einen bekannten und weit verbreiteten Vertreter dieser Kategorie dar. Das Protokoll verfolgt mit der dem *Publisher-/Subscriber*-Prinzip ein anderes Konzept als das vorher genannte CoAP. Hier steht ein sogenannter *Broker* im Mittelpunkt, welcher jegliche Kommunikation vermittelt. Clients werden zum *Publisher*, wenn sie auf einem Kommunikationskanal, genannt *Topic*, Daten veröffentlichen. Clients können Topics abonnieren, werden damit zum *Subscriber* und erhalten damit sämtliche Nachrichten, welche im Topic veröffentlicht werden. Ein Topic kann dabei mehrere Publisher und mehrere Subscriber besitzen und Clients können ebenfalls Publisher und Subscriber mehrerer Topics zugleich sein. Der Broker ist dabei die zentrale Stelle, wenn ein Client Subscriber oder Publisher eines Topics werden möchte. Bei einer Subscribe-Anfrage wird die Adresse des Anfragenden in die Empfängerliste des entsprechenden Topics eingetragen. Bei einer Publish-Anfrage wird die Nachricht dementsprechend an alle Empfänger verteilt. Die Verbindung zwischen Client und Broker wird prinzipiell über die *Transmission Control Protocol/Internet Protocol-Familie (TCP/IP)* hergestellt. Es existiert mit *MQTT-SN* aber noch eine Variante des Protokolls für Netzwerke ohne TCP/IP-Unterstützung, z. B. Zigbee. Auf dem Broker ergibt sich ein Gesamtbild der Infrastruktur, weshalb die Clients weniger komplexe Logik benötigen. Zudem bietet MQTT mit *Quality of Service (QoS)* für Nachrichten, *Retain*-Markierungen und dem *Last Will* einige nützliche Zusatzfunktionen. MQTT unterstützt *Transport Layer Security (TLS)* und gilt als effizient und einfach zu verwenden, weshalb es weit verbreitet ist.

Das *Advanced Message Queuing Protocol (AMQP)* [6] ist ein weiteres, umfangreiches Protokoll, um viele verschiedene Systeme miteinander zu vernetzen. Analog zu MQTT kann ein zentraler Broker oder alternativ *Peer-to-Peer*-Vernetzung genutzt werden. Die Verbindungen basieren auch hier auf TCP/IP, allerdings ist das Protokoll nicht speziell für IoT-Anwendungen konzipiert, sondern für die Verbindung von Geschäftsprozessen. Zusätzlich sind *Application Programming Interfaces (APIs)* enthalten, welche spezifische Konzepte, bspw. den *Java Message Service (JMS)* und die *Windows Communication Foundation (WCF)*, abstrahieren und somit auf verschiedenen Systemen einheitlich bereitstellen. AMQP folgt einem Prinzip, das einem erweiterten Publisher-Subscriber-Modell ähnelt. Es besteht grundlegend aus *Publisher*, *Exchange*, *Bindings* und *Consumer*. In Zusammenhang mit einem Regelsystem ermöglicht es einerseits die Abbildung wesentlich komplexerer Strukturen als die bisher genannten Protokolle. Auf der anderen Seite bringt es dadurch auch einen größeren Verwaltungsaufwand bei der Kommunikation mit sich. Die minimale Nachrichtengröße bei AMQP beträgt 60 Bytes, im Gegensatz dazu sind es bei MQTT nur 2 Bytes.

Die *Open Platform Communications Unified Architecture (OPC UA)* [63] wird als „industrielles M2M-Kommunikationsprotokoll“ beschrieben. Es enthält eigentlich zwei verschiedene Protokolle, die sich aber durch eine einheitliche API kaum von außen unterscheiden. Eines davon ist das Webservice-Protokoll auf Basis von *Simple Object Access Protocol (SOAP)*, welches für Webanwendungen vorgesehen ist. Das

3. Stand der Technik

zweite Protokoll ist das Binärprotokoll, das speziell für eingebettete Geräte empfohlen wird. Es zeichnet sich durch einen geringeren Verwaltungsaufwand aus und benötigt weniger Ressourcen, da keine Unterstützung für XML oder HTTP benötigt wird. Des Weiteren bewirbt die OPC UA maschinenlesbare Datensemantik, Redundanz und mit *UA Security* zudem „Authentifizierung und Autorisierung, Verschlüsselung und Datenintegrität durch Signieren“.

Tabelle 3.3 gibt einen Überblick über alle vorgestellten Ansätze.

Software	Prinzip	Sicherheit	Besonderheiten
TCP/UDP	Sockets	TLS/DTLS	sehr aufwändig
CoAP	REST	DTLS	für stark eingeschränkte Hardware
MQTT	Publisher-Subscriber	TLS, Authentifizierung	unterstützt Quality of Service
AMQP	Publisher-Subscriber (erweitert)	TLS, Authentifizierung	komplexes Grundprinzip
OPC UA	Binär/SOAP	TLS, UA Security	umfasst Binär- und Web-Protokoll

Tabelle 3.3.: Vergleich der Nachrichtenprotokolle

3.2.4. Update-Systeme

Die Knoten des Sensornetzwerks sollen nach den Vorgaben der Aufgabenstellung über eine Möglichkeit für Systemaktualisierungen *over-the-air (OTA)* verfügen. Anforderungen an das Update-System sind vorwiegend Robustheit im Fehlerfall und die Möglichkeit zur zentralen Koordination der Aktualisierungen. Für eingebettete Systeme mit Linux-Betriebssystem, wie sie im Rahmen dieser Arbeit Verwendung finden, existieren bereits Lösungen, die nachfolgend betrachtet werden [87].

Mender [44] ist eine umfassende Komplettlösung für OTA-Systemaktualisierungen. Es besteht aus einem Server, welcher letztendlich mehrere Docker-Container umfasst und einem Client, welcher über einen Yocto Layer direkt in das Betriebssystem eingebunden wird. Möglich ist auch die Verwendung ohne Yocto, dies erfordert allerdings zusätzlichen Entwicklungsaufwand. Der Client sendet in regelmäßigen Abständen von Scripten gesammelte Inventardaten des Systems an den Server. Durch eigene Scripte lässt sich dieser Report um beliebige Informationen erweitern. *Mender* arbeitet nach dem *Double Copy*-Prinzip, d. h., es werden zwei parallele Systempartitionen auf den Geräten eingerichtet. Über den Bootloader wird anschließend konfiguriert, welches System gestartet werden soll. Bei einem Update wird ein komplettes Linux-Abbild, erstellt mit dem Yocto Project, empfangen und direkt in die inaktive Partition installiert. Alternativ können *Standalone Deployments* ohne Netzwerkanbindung, bspw. über USB-Sticks, ausgeführt werden. Nach erfolgreicher Installation erfolgt ein Neustart in das neue System. Falls im Verlauf des Updates ein Fehler auftritt, sodass

3. Stand der Technik

das neue System nicht startet oder keine Verbindung zum Mender-Server zustande kommt, wird automatisch ein sogenannter *Rollback* eingeleitet. Das bedeutet, dass der vorhergehende, lauffähige Systemzustand durch Starten des ursprünglichen Systems auf der zweiten Partition wiederhergestellt wird. Zwei weitere Partitionen enthalten den Bootloader und einen Bereich für persistente Daten, die von Aktualisierungen unberührt bleiben. Da der Mender-Client nach einem Zustandsmodell arbeitet, welches sich durch sogenannte *State-Scripts* erweitern lässt, kann das System an viele Einsatzszenarien angepasst werden. Das Projekt wird zudem von der Europäischen Kommission im Rahmen von *Horizon 2020* gefördert und lässt somit auf weitere Verbesserungen hoffen. Die Einrichtung des Demonstrationssystems funktioniert *out-of-the-box* und ist somit nicht aufwändig. Für den Produktiveinsatz sind entsprechende, sicherheitsrelevante Anpassungen nötig, welche aber sehr ausführlich in der Dokumentation [45] beschrieben sind. Diese umfassen bspw. die Erstellung von Zertifikaten für die sichere Übertragung oder die Einrichtung von Schlüsseln zum Signieren der Update-Pakete, auch *Artefakte* genannt. Ein Nachteil besteht darin, dass bei jedem Update ein komplettes Mender-Artefakt zum Zielgerät übertragen werden muss, allerdings besagen Informationen auf der Projektwebseite, dass *Binary Delta*-Updates geplant sind. Das würde die zu übertragende Datenmenge auf die echten Änderungen im System reduzieren.

Alle nun folgenden Lösungen bestehen, im Gegensatz zu Mender, nur aus Client-Software. Viele davon können in Kombination mit dem *Eclipse Hawkbit*-Server [36] verwendet werden. Dieser steht mittlerweile als Docker-Container zur einfachen Einrichtung bereit. Es muss jedoch berücksichtigt werden, dass sich das Hawkbit-Projekt zu diesem Zeitpunkt noch in einem frühen Entwicklungsstadium befindet. *SWUpdate* [79] ist ein Update-Client, welcher u. a. als Yocto-Layer bereitgestellt wird. Er nutzt standardmäßig die *Single Copy*-Architektur. Updates werden also aus einem kleineren Update-System heraus auf das eigentliche Betriebssystem angewendet. Unterstützt wird aber auch die Verwendung von Double Copy. Ein weiterer Unterschied liegt in der Art der Update-Bereitstellung. *SWUpdate* stellt im *Mongoose*-Modus ein Webinterface bereit, über welches ein Nutzer selbst Updates auf das betreffende Gerät hochladen und installieren kann. Das hat den Nachteil, dass auf dem eingebetteten System Ports geöffnet werden müssen, womit wiederum ein Sicherheitsrisiko einhergeht. Analog zur Funktionsweise von Mender bietet *SWUpdate* allerdings auch den *Suricata*-Modus, womit in regelmäßigen Abständen bei einem Server auf Updates geprüft und ein Report gesendet wird. Die Bereitstellung von Updates über USB-Massenspeicher oder serielle Verbindungen ist auch hier möglich. Zudem werden signierte Aktualisierungen angeboten, die Übertragung scheint jedoch über ungesichertes HTTP stattzufinden. *SWUpdate* erlaubt außerdem die Einbindung benutzerdefinierter Module, um z. B. Mikrocontroller-Firmware auszuliefern. Der Konfigurations- und Anpassungsaufwand ist dementsprechend höher als bei Mender.

Der RAUC [74] wird als freies Open-Source-Projekt, wie das Linux-Build-System PTXdist, hauptsächlich von der Pengutronix GmbH [67] vorangetrieben. Die Client-Software ist sowohl für das Linux-Build-System PTXdist als auch für das Yocto

3. Stand der Technik

Project als Layer verfügbar. Unterstützt werden auch hier Single und Double Copy, allerdings beschrieben als asymmetrisches bzw. symmetrisches System. Aktualisierbare Komponenten des Systems, meist Partitionen, werden als sogenannte *Slots* definiert. Für jeden Slot können entsprechende Inhalte in den Update-Paketen enthalten sein. RAUC bietet zwei verschiedene Formate für Updates: *Bundle*, bestehend aus einer Datei mit dem kompletten System-Update und *Network*, aufgeteilt in Manifest- und Komponentendateien. Quelle der Updates können auch bei dieser Lösung *Provisioning Server*, wie Eclipse Hawkbit, oder externe Datenträger, wie USB-Massenspeicher, sein. Sicherheitsaspekte werden durch die Möglichkeit zum Signieren der Update-Pakete und durch verschlüsselte Übertragungswege gewährleistet. RAUC ist neben der Funktion als Update-Client zudem das Werkzeug zum Erstellen, Inspizieren und Signieren der Updates auf dem Erstellungssystem. Durch die vielfältigen Konfigurationsparameter und die nötige Serveranbindung ist der Anpassungsaufwand vergleichbar mit dem von SWUpdate.

Mit *swupd* [78] wird ein weiterer Ansatz betrachtet, welcher im laufenden System Aktualisierungen auf Dateiebene durchführt. Der Client nutzt dabei vorzugsweise *Binary Delta*-Updates, wobei, wie bereits bei Mender erwähnt, nur Unterschiede auf binärer Ebene aktualisiert werden. *swupd* ist auf Systeme ausgerichtet, deren Entwicklungsstrategie es vorsieht, häufig kleine Änderungen auszuliefern. Anwendung findet es u. a. beim *Clear Linux Project*, einer Linux-Distribution für „Cloud-, Client- und IoT-Einsatzszenarien“. Software wird hier zwar in Paketform erzeugt, aber nur funktional zusammengefasst in sogenannten *Bundles* über Repositories ausgeliefert. Der Aufwand zur Einrichtung dieses Clients ist entsprechend hoch, da der Aufbau und die Wartung eines eigenen Repositories erforderlich ist.

Eine Art der Versionsverwaltung auf Dateiebene bietet *meta-updater* [49]. Wie der Name bereits erkennen lässt, handelt es sich hierbei um einen Yocto-Layer. Dieser Client nutzt die *OSTree*-Bibliothek [66], um jede Datei des Systems, ähnlich zu Systemen wie *git*, zu versionieren. Er ermöglicht atomare Updates inklusive Rollback-Funktion. Der Bandbreitenbedarf liegt etwa zwischen denen von Mender und *swupd*, da nur geänderte Dateien, diese allerdings komplett, übertragen werden müssen. Zudem wird, wie bei *swupd* und gegensätzlich zu symmetrischen Double Copy-Architekturen, nur Speicher für eine Systeminstallation belegt. Bei *meta-updater* wird mit *Aktualizr* [5] beziehungsweise dem *Remote Vehicle Interaction (RVI) SOTA-Client* [75] ein zusätzliches Modul einbezogen, welches „Authentifizierungs- und Bereitstellungsfunktionen“ mit *OSTree* vereint. Dieses funktioniert jedoch nur in Zusammenhang mit einem korrespondierenden *RVI SOTA-Server* [76]. „RVI ist eine Open-Source-Infrastruktur, entwickelt von der GENIVI-Allianz und Jaguar Land Rover, um die nächste Generation vernetzter Fahrzeugdienste anzutreiben“. Die GENIVI-Allianz wiederum ist eine *Non-Profit*-Organisation der Automotive-Industrie, um die breite Anpassung von Open-Source für *In-Vehicle Infotainment (IVI)* voranzutreiben und freie Technologie für vernetzte Fahrzeuge bereitzustellen. Ein gewisses Maß an Sicherheit ist durch die Verwendung von Authentifizierung und *Hypertext Transfer Protocol Secure (HTTPS)* zur Kommunikation gegeben. Der Arbeitsaufwand ist ähnlich hoch einzuschätzen, wie bei *swupd*,

da auch hier ein eigenes Repository für OSTree erstellt und sämtliche Komponenten konfiguriert sowie aufeinander abgestimmt werden müssen. Zur Vereinfachung ist zu Aktualizr ein Werkzeug erhältlich, welches Änderungen aus der Bitbake-Ausgabe von Yocto direkt in ein OSTree-Repository übertragen kann.

Alle vorgestellten Ansätze sind in Tabelle 3.4 als Übersicht zusammengefasst.

Software	Umfang	Prinzip	Rollback
Mender	Server, Client	Double Copy	ja
SWUpdate	Client	Single/Double Copy	ja
RAUC	Client	Single/Double Copy	ja
swupd	Client	Binary Delta	nein
meta-updater	Client	Dateiversionierung	ja

Tabelle 3.4.: Vergleich der Update-Systeme

3.2.5. Bedienoberflächen

Um den aktuellen Systemstatus einsehen und ggf. auch Einfluss darauf nehmen zu können, bietet es sich an, eine Bedienoberfläche zur Verfügung zu stellen. Auch für diesen Punkt existiert bereits Software, die die Entwicklung stark vereinfacht und zum Teil frei verfügbar ist.

Node-RED [53] ist eine dieser Lösungen und zudem die einzige, welche ganzheitlich quelloffen ist. Über eine separate Weboberfläche lassen sich damit sogenannte *Flows* grafisch erstellen. Sie enthalten einzelne Module, die *Nodes*, welche beliebig miteinander verknüpft werden können. Durch Erweiterungspakete lässt sich die Auswahl der verfügbaren Nodes vergrößern. So enthält *node-red-dashboard* [54] eine Sammlung von Nodes zur Erstellung eines „Live-Daten-Dashboards“. Außerdem können eigene Funktionen in *Template-Nodes* mit *JavaScript* und *AngularJS* umgesetzt werden. Außergewöhnlich ist zudem die integrierte Unterstützung für viele verschiedene Kommunikationskanäle. Dazu gehören u. a. TCP, UDP, HTTP, MQTT sowie E-Mail und Twitter.

Geckoboard [35] ist ein kommerzieller Vertreter dieser Kategorie. Beworben wird der Dienst als „flexibles, individuell anpassbares und optisch ansprechendes Management-Dashboard“ für den Business-Bereich. Mit anpassbaren *Widgets* lassen sich Informationen aus verschiedenen Quellen, den eigenen Vorstellungen entsprechend, darstellen. Das Dashboard lässt sich, im Gegensatz zu Node-RED, direkt in der Dashboard-Ansicht anpassen. Als einzige Möglichkeit, Daten in Geckoboard einzuspeisen, ist das JSON-Format über HTTPS angegeben.

Die letzte Lösung in diesem Abschnitt ist *freeboard* [31]. Sie wird als „freie Open-Source-Alternative zu Geckoboard“ [32] angepriesen und ist nur im Browser, als statische Web-Applikation und somit ohne Server lauffähig. Das macht sie auch für die Nutzung auf eingebetteten Systemen interessant. Nachteil ist, dass lediglich die Client-Seite quelloffen verfügbar ist. Sämtliche Funktionen der Server-Seite, wie

3. Stand der Technik

Benutzerverwaltung, Datenbankanbindung und Zugriffskontrolle, sind Teil des proprietären Angebots von *freeboard.io* und somit nicht frei erhältlich. Dadurch sind zwar Freiheiten bei der Entwicklung gegeben, allerdings ist im selben Zug der Entwicklungsaufwand wesentlich höher, als bspw. bei der Verwendung von Node-RED. Daten lassen sich in der Beispielumgebung auch hier nur in teils auf bestimmte Quellen zugeschnittenen JSON-Strukturen zuführen. Die Oberfläche lässt sich durch die Implementierung sogenannter *Widget Plugins* erweitern.

Abschließend fasst Tabelle 3.5 alle vorgestellten Ansätze als Übersicht zusammen.

Software	Anbindung	Erweiterung	Art
Node-RED	HTTP, MQTT uvm.	Flows, Templates	Open Source
Geckoboard	HTTP (JSON)	-	kommerziell
Freeboard	HTTP (JSON)	Widget Plugins	Open Source (Client)

Tabelle 3.5.: Vergleich der Dashboard-Lösungen

3.2.6. Künstliche Intelligenz

In Unterabschnitt 2.3.2 wurden verschiedene Verfahren zum Trainieren neuronaler Netze vorgestellt. Unüberwachtes Lernen kommt für diesen Anwendungsfall nicht infrage, da es im Rahmen dieser Arbeit nicht darum geht, Eingaben verschiedenen, unbekanntenen Klassen zuzuordnen. Durch den Mangel an Trainingsdaten ist überwachtes Lernen ebenfalls keine Option. RL hingegen ist in der Lage, ohne vorheriges Training, direkt durch Interaktion mit der Umwelt zu lernen. Dies ermöglicht den Einsatz einer KI, ohne große Mengen an Trainingsdaten sammeln und vorverarbeiten zu müssen. Im Folgenden werden dementsprechend verschiedene Software-Lösungen vorgestellt, welche die Anwendung von RL ermöglichen. Grundlage dieser Lösungen sind bspw. TensorFlow, das *Microsoft Cognitive Toolkit (CNTK)*, Theano oder Torch. Darauf aufbauend bilden die Bibliotheken eine Abstraktionsschicht für verschiedene Anwendungszwecke, um bspw. RL einfach nutzbar zu machen. Dadurch wird die Anwendung von ML bzw. KI für ein breiteres Spektrum von Entwicklern zugänglich. Die Basis-Frameworks bringen zum Teil mehrere APIs für verschiedene Programmiersprachen, wie C++, C# oder Java, mit. Als gemeinsamer Nenner lässt sich zumeist die Programmiersprache Python unter ihnen finden. Infolgedessen ist auch eine Vielzahl der Bibliotheken, die meist mit verschiedenen dieser Frameworks kombinierbar sind, als Modul für Python verfügbar.

Die zunächst betrachtete Bibliothek ist *keras-rl* [107]. Wie der Name bereits erkennen lässt, baut sie auf die *Keras*-Bibliothek [93] auf und erweitert den Funktionsumfang um RL. Sie ist kompatibel mit TensorFlow, Theano, CNTK und der Simulationsumgebung *OpenAI Gym* [91]. Gym stellt verschiedene virtuelle Umgebungen zum Trainieren und Testen von RL-Algorithmen bereit. In *keras-rl* sind u. a. verschiedene Varianten von Q-Learning implementiert.

Für die Verwendung im Rahmen dieser Arbeit ist die Bibliothek jedoch nur bedingt

3. Stand der Technik

geeignet, da für den Realwelteinsatz der Umweg über eine Gym-Umgebung erforderlich ist.

Eine weitere Lösung stellt *rllab* dar, ein „Framework zur Entwicklung und Evaluation von RL-Algorithmen“ [96]. Es ist verwendbar mit Theano oder TensorFlow und besitzt einige Gemeinsamkeiten mit dem zuvor beschriebenen *keras-rl*. So ist es ebenfalls auf die Simulation mit OpenAI Gym ausgelegt und „beinhaltet Implementierungen für eine breite Spanne von kontinuierlichen Kontrolltasks“. Am 17. Oktober 2018 wurde bekanntgegeben, dass die bisherigen Entwickler ihre Arbeit am Projekt einstellen. Es wird jedoch von einer Forschergruppe verschiedener Universitäten übernommen und unter dem Namen *garage* [34] weiterentwickelt.

TensorForce [108] ist die dritte Software-Lösung dieser Kategorie. Auf der Projektseite wird sie als „TensorFlow-Bibliothek für angewandtes Reinforcement Learning“ beschrieben. Wie der Name bereits andeutet, setzt es TensorFlow als Backend voraus. Die Verknüpfung mit verschiedenen Simulationsumgebungen, u. a. OpenAI Gym, ist auch hier möglich. Zusätzlich ist sie durch eine Modularisierung in Umgebung, *Runner*, Agent und Modell in der Lage, außerhalb von solchen Simulatoren eingesetzt zu werden. Dies ist der Hauptunterschied zu den bisher genannten RL-Bibliotheken. Ähnlich zu *keras-rl* sind auch hier, neben weiteren Agenten, verschiedene Varianten von Q-Learning implementiert. Besonders hervorzuheben ist bei dieser Bibliothek der Agent für *Deep Q-learning from Demonstrations (DQfD)* [99]. Dieser Algorithmus ermöglicht das Vortrainieren der KI mit einer kleinen Menge von Demonstrationsdaten vor dem eigentlichen Einsatz. In Experimenten wurden damit in frühen Lernphasen viel bessere Ergebnisse erzielt als mit herkömmlichen RL-Algorithmen. Dies ist besonders für Realweltanwendungen mit großen Zeitabständen zwischen Lernschritten von enormer Bedeutung. Es muss allerdings betont werden, dass sich *TensorForce* noch in einer frühen Entwicklungsphase befindet. Dementsprechend sollte beachtet werden, dass Fehlfunktionen nicht auszuschließen sind und die Dokumentation ebenfalls unvollständig ist.

Tabelle 3.6 gibt einen Überblick über alle vorgestellten Ansätze.

Software	Ausrichtung	Basis-Framework	Besonderheiten
<i>keras-rl</i>	Simulation	TensorFlow, Theano, CNTK	-
<i>rllab</i>	Simulation	TensorFlow, Theano	-
<i>TensorForce</i>	Simulation, Realwelt	TensorFlow	modularer Aufbau

Tabelle 3.6.: Vergleich der RL-Bibliotheken

3.3. Zusammenfassung

Nach Analyse der verwandten Arbeiten dieser Thematik wird deutlich, dass ähnliche Ansätze bereits mit unterschiedlichen Herangehensweisen untersucht wurden. Ein Teil der vorgestellten Arbeiten beschreibt das Erreichen ähnlicher Zielsetzungen mit Techniken, die für die Gegebenheiten der hier gestellten Aufgabe nicht anwendbar sind. Andere Resultate stammen vorwiegend aus Simulationsumgebungen, welche prinzipbedingt eine gewisse Abweichung von Realwelteinsätzen mit sich bringen. Andererseits bilden sie eine gute Grundlage, auf die im Verlauf dieser Arbeit aufgebaut werden kann. Ziel ist es, ein auf RL basierendes System, mit den in der Theorie gewonnenen Erkenntnissen, in die Praxis umzusetzen. Das bestehende Wissen aus den theoretischen Arbeiten sollen dabei um die für den Realwelteinsatz relevanten Überlegungen und Einflussgrößen erweitert werden. In Kombination mit den in Kapitel 2 vorgestellten Grundlagen, der zur Verfügung gestellten Hardware sowie einer Auswahl der hier betrachteten Softwarelösungen wird dafür zunächst ein Konzept erstellt. Anschließend soll dieses in die Praxis umgesetzt und in verschiedenen Konfigurationen erprobt werden, um die Ergebnisse im Realwelteinsatz zu evaluieren.

4. Konzept

Im vorhergehenden Kapitel erfolgte die Vorstellung und Abgrenzung von thematisch verwandten Arbeiten. Des Weiteren wurden bestehende Lösungsansätze in Bezug auf die Schwerpunkte der Aufgabenstellung präsentiert. In diesem Kapitel wird eine Auswahl aus diesen Ansätzen getroffen und daraus ein Konzept zur Umsetzung abgeleitet. Dies erfolgt in den Abschnitten *Architekturentwurf*, *Anbindung der Klimaanlage*, *Betriebssystem*, *Sensornetzwerk*, *Systemaktualisierung der Sensorknoten*, *Nutzerschnittstelle*, *Datenaufzeichnung*, *KI-Agent* und *Bewertungskriterien*.

4.1. Architekturentwurf

Die Aufgabenstellung fordert zusammengefasst die Umsetzung folgender Punkte:

- Erweiterung der Steuerung einer Klimaanlage
- Erfassung von Umweltdaten durch ein Sensornetzwerk aus Einplatinenrechnern
- Auswertung der Messwerte durch KI
- Systemaktualisierungen *OTA* im Sensornetzwerk

In Abbildung 4.1 ist die zugrundeliegende Architektur zusammengefasst dargestellt. Zur Implementierung des Systems stehen insgesamt drei Einplatinenrechner und

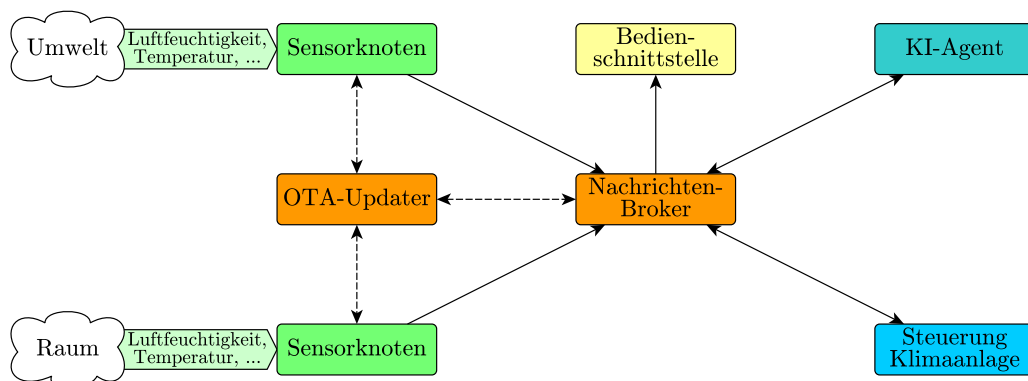


Abbildung 4.1.: Architekturüberblick

ein Desktop-PC zur Verfügung. Dabei sind alle BeagleBones als Knoten des Sensornetzwerks geplant. Die Steuerung der Klimaanlage wird mit dem HTTP-Modul

4. Konzept

ausgestattet und ist mit der Ethernet-Schnittstelle an ein *Local Area Network (LAN)* angebunden, in welchem sich auch der Desktop-PC befindet. Die beiden BeagleBone Black Wireless werden mit den bereitgestellten Sensoren ausgestattet und sollen die Messwerte erfassen. Da der BeagleCore zusätzlich zum WLAN eine Ethernet-Schnittstelle besitzt, bietet sich dieser als Brücke zwischen Sensornetz LAN an. Er soll aufgrund seiner zentralen Position im Netzwerk zudem die Funktion des Nachrichtenbrokers übernehmen und gewährleistet durch die Ethernet-Anbindung eine hohe Erreichbarkeit, auch bei eventuellen Störungen im Sensornetzwerk. Die restlichen Software-Komponenten sind zur Ausführung auf dem Desktop-PC vorgesehen. Dazu zählen die Bedienoberfläche, der OTA-Updater, der KI-Agent sowie die Anbindung der Klimaanlage. Der Updater hat einen hohen Speicherplatzbedarf für die Update-Pakete, weshalb die Umsetzung auf einem der BeagleBones nicht sinnvoll erscheint. Alle anderen genannten Komponenten unterliegen und sollen deshalb auf dem Entwicklungssystem, dem Desktop-PC, ausgeführt werden. Hinzu kommt eine erhöhte Belegung des Arbeitsspeichers durch den KI-Agenten, welcher auf den BeagleBones nicht in dem Maß vorhanden ist. Die verschiedenen Clients sollen einheitlich mit Python 3 realisiert werden, weil sämtliche Funktionalitäten und Bibliotheken als Module hierfür bereitstehen. Auf die einzelnen Komponenten dieses Entwurfs wird in den folgenden Abschnitten detailliert eingegangen.

4.2. Anbindung der Klimaanlage

Um die vorhandene Klimaanlage in das geplante System einzubinden, wird eine Schnittstelle zu deren Steuerung benötigt. Wie bereits in Abschnitt 2.2 erwähnt, bietet der Hersteller der Anlage für den hier verwendeten iTC ein HTTP-Schnittstellen-Modul an. Das ermöglicht die Kommunikation mit dem Gerät von einem Programm ausgehend über die Ethernet-Anbindung. Somit können Informationen des gesamten Klimasystems abgerufen und verändert werden. Das Schnittstellen-Modul stellt ein spezielles Protokoll, welches auf HTTP-POST-Anfragen beruht, bereit. Es existieren insgesamt vier verschiedene Kommandos, die den folgenden Funktionsumfang haben [22]:

- Abrufen aller mit dem iTC verbundener Innengeräte
- Anfordern der Eigenschaften aller konfigurierten Innengeräte, bspw. benutzerdefinierter Name und Anschlussstyp
- Statusabfrage einer Auswahl von Innengeräten, u. a. An/Aus, Betriebsmodus, Raumtemperatur und Temperatursollwert
- Statusänderung einer Auswahl von Innengeräten, z. B. An/Aus, Betriebsmodus und Temperatursollwert

Um die HTTP-Schnittstelle in das Gesamtkonzept der KI-Steuerung einzubinden, wird eine Abstraktion des Protokolls in Form einer API benötigt [113]. Diese soll in

einer weiteren Komponente genutzt werden, um die Funktionalität über das Nachrichtenprotokoll des Sensornetzwerks bereitzustellen und Informationen in beide Richtungen umwandeln zu können.

4.3. Betriebssystem

Im Sensornetzwerk kommen BeagleBone-Einplatinenrechner auf Linux-Basis zum Einsatz. Für diese soll, wie bereits in Unterabschnitt 3.2.1 erläutert, ein angepasstes Linux-Betriebssystem erstellt werden. Damit kann die Aktualisierbarkeit gewährleistet und das System auf die notwendige Software reduziert werden. Aufgrund der Unterstützung der Linux-Foundation und vorhandener Erfahrung im Betreuerumfeld dieser Arbeit soll hierzu das Yocto Project genutzt werden. Dies erfordert zunächst das Erstellen eines neuen Projektes auf Grundlage der Yocto-Referenzdistribution Poky [70]. Anschließend wird dem Projekt ein eigener Layer hinzugefügt. In diesen werden die, zusätzlich zum Grundsystem benötigten, Software-Module sowie Änderungen eingepflegt und somit beim Erstellungsvorgang in das Zielsystem übernommen. Geplante Inhalte dieses Layers sind:

- Module zur Konfiguration und Erstellung des Mesh-Netzwerks
- Broker- bzw. Client-Software für das Nachrichtenprotokoll
- Änderungen zur Aktivierung der Sensorschnittstellen
- eigene Programme zum Auslesen der Sensormesswerte

Das System soll dabei auf die, zum Zeitpunkt der Konzeption empfohlene, Poky-Version 2.4.3 „rocko“ verwendet werden. Das resultierende Betriebssystemabbild soll anschließend auf microSD-Speicherkarten übertragen und die BeagleBones damit gestartet werden. Eine Verwendung des internen Flash-Speichers ist aufgrund von möglichen Komplikationen mit dem OTA-Updater nicht geplant. Um verschiedene Konfigurationen, je nach Funktion des verwendeten Einplatinenrechners, zu verwenden, ist es zudem vorgesehen, die Hardware-Adresse der Netzwerkschnittstellen zur Unterscheidung auszuwerten. Dies soll in einem Startskript umgesetzt werden, welches die passende Software-Konfiguration auf der jeweiligen Hardware startet.

4.4. Sensornetzwerk

Die Gestaltung des Sensornetzwerk zur Erfassung der Umweltdaten ist im Sinne einfacher Skalierbarkeit und Ausfallsicherheit als selbstvernetzendes *Mesh*, ohne weitere Infrastruktur, vorgesehen. Dadurch kann mit einer gewissen Anzahl von Knoten in einer geeigneten Anordnung auch der Ausfall von Knoten und damit eine Leitungsunterbrechung kompensiert werden [118]. Das wird durch dynamisches *Routing*

4. Konzept

möglich, welches Pakete bei Blockaden auf der geplanten Route durch das Netzwerk auf alternative Wege umleitet und somit dennoch zum Ziel führt [112]. Hardwareseitig ist WLAN der gemeinsame Nenner bei allen Einplatinenrechnern, allerdings wird der Ad-hoc-Modus vom Chipsatz des per USB-Stick nachgerüsteten Beagle-Cores nicht unterstützt. Dieser Modus ist die Voraussetzung für ein Netzwerk auf Basis von B.A.T.M.A.N. Advanced, weshalb sich die Auswahl auf die Umsetzung eines Mesh-Netzwerks nach IEEE 802.11s beschränkt. Hier ist die Kompatibilität mit allen verwendeten WLAN-Chips gegeben. Eine Implementierung von 802.11s ist bereits Teil des Linux-Kernels [58] und muss lediglich aktiviert werden. Dazu soll die Kernel-Konfiguration im selbst erstellten Yocto-Layer dementsprechend angepasst werden. Außerdem müssen zur Konfiguration und Absicherung des Mesh-Netzwerks entsprechende Befehle in das Startskript eingetragen bzw. dafür vorgesehene Textdateien eingebunden werden. Unabhängig davon werden die Treiber und ggf. Firmwares der verwendeten WLAN-Hardware benötigt. Diese sollen ebenfalls per Kernel-Anpassung in den Yocto-Layer und damit ins Zielsystem integriert werden.

Die Übertragung der Nachrichten wird über das MQTT-Protokoll stattfinden. Aufgrund der Effizienz sowie des einfachen Prinzips bei gleichzeitig erweitertem Funktionsumfang eignet es sich für diesen Anwendungsfall am besten. Durch die weite Verbreitung sind zudem Bibliotheken für viele verschiedene Programmiersprachen und ebenso Broker für die meisten Betriebssysteme verfügbar. Im Konzept wird der für Linux verfügbare MQTT-Broker *Eclipse Mosquitto* [51] vorgesehen, da dieser bereits als Rezept in Poky enthalten ist und somit ohne großen Aufwand eingebunden werden kann.

Die Bosch BME680-Umweltsensoren können entweder über SPI oder I2C kommunizieren, beides ist auch seitens der BeagleBones möglich. Da I2C eine Verbindungsleitung weniger benötigt als SPI, soll es an dieser Stelle verwendet werden. Die Sensoren sind in der Lage, die Luftqualität in Form eines Widerstandswerts zu erfassen. Diese Messelemente sind beheizt, um die chemischen Reaktionen mit diversen flüchtigen organischen Verbindungen messbar zu machen. Deshalb benötigen sie eine gewisse Aufwärmphase, bis sie stabile Ergebnisse liefern [2]. Nach dieser Phase sollten sie in regelmäßigen Abständen abgefragt werden, um die Messwertstabilität aufrechtzuerhalten. Sie sind sensibel für ein breites Spektrum von Stoffen, wie Methan, Kohlenmonoxid und Ethanol [68]. Für die Platzierung der Sensoren ist es empfehlenswert, eine möglichst vor Zugluft geschützte Position im selben Bereich, für den letztendlich die Sollwerte gelten sollen, zu wählen. Vorgesehen ist zudem die Anbindung von zwei Reed-Sensoren, welche an die *General Purpose Input/Outputs (GPIOs)* der BeagleBones angeschlossen werden. Das Senden aller Sensormessdaten über MQTT soll in einem Intervall von zehn Sekunden nach dem Push-Prinzip geschehen. Diese zeitliche Auflösung eignet sich für die Erfassung des Raumklimas und speziell für die Beobachtung von Änderungen, ohne die Übertragungswege im Sensornetzwerk stark zu belasten. Kürzer andauernde Änderungen, welche innerhalb eines Intervalls liegen und damit nicht erfasst werden können, z. B. Türöffnungen oder -schließungen, sollen abweichend davon unverzüglich übermittelt werden.

4.5. Systemaktualisierung der Sensorknoten

Die Sicherheit des Sensornetzwerks muss, bestärkt durch die in Kapitel 1 aufgeführten Medienberichte, ebenfalls berücksichtigt werden. Im Gegensatz zu vielen, bereits auf dem Markt befindlichen Produkten, soll von vornherein eine Möglichkeit vorgesehen werden, die Knoten des Sensornetzwerks OTA mit Systemaktualisierungen zu versorgen. Dieser Mechanismus erfordert ein robustes Verfahren, sodass im Fehlerfall, nach wie vor, ein funktionierender Zustand vorliegt und das System nicht durch Störungen außer Gefecht gesetzt werden kann.

Von den in Unterabschnitt 3.2.4 vorgestellten Systemen wurde sich hierfür auf Mender als Komplettlösung festgelegt. Mender lässt sich hervorragend mit dem Yocto Project kombinieren, indem der bereitgestellte Layer in das bestehende Projekt eingebunden wird. Dies installiert und konfiguriert den Mender-Client, was das resultierende System ohne großen Aufwand zu kompletten Systemaktualisierungen aus der Ferne befähigt. Im Fehlerfall kann Mender einen Rollback auf die vorherige, funktionierende Betriebssystemversion durchführen und somit einen Ausfall des Knotens verhindern. Es ist zudem möglich, verifizierte Updates durch Erstellung eines Schlüssels und Signieren der Artefakte zu erreichen [46]. Diese Möglichkeit soll im Rahmen dieser Arbeit jedoch nicht genutzt werden, da die Updates ohnehin nur im lokalen Netzwerk und zu Demonstrationszwecken durchgeführt werden. Durch den Mender-Layer wird bei jedem Build-Vorgang des Projektes neben dem eigentlichen Speicherkartenabbild ein zusätzliches Update-Artefakt zur Auslieferung mit dem Mender-Server erstellt. Die Serverseite von Mender soll auf dem Desktop-PC eingerichtet und ausgeführt werden.

4.6. Nutzerschnittstelle

Für die Interaktion mit der erweiterten Klimaanlagesteuerung ist es nötig, eine Bedienoberfläche bereitzustellen. Auf dieser sollen Informationen zum Zustand des gesamten System angezeigt und Bedienelemente zur Änderung der Einstellungen durch den Nutzer untergebracht werden. Dazu gehört z. B. eine Statusübersicht der Klimageräte sowie Elemente zur Änderung der Temperatursollwerte für die betreffenden Räume. Um die Bedienung auf möglichst vielen Plattformen nutzen zu können wird die Umsetzung mittels Web-Oberfläche angestrebt. Von den in Unterabschnitt 3.2.5 vorgestellten Lösungen kommt deshalb die Open-Source-Variante Node-RED inkl. Dashboard-Erweiterung zum Einsatz. Die Konfiguration der Serverseite muss entsprechend angepasst werden und ist anschließend einsatzbereit. Über den webbasierten Editor soll anschließend ein Flow erstellt werden, welcher die Bedienelemente enthält, miteinander verbindet und zudem die Anbindung an das Nachrichtenprotokoll realisiert. Dieser Flow wird letztendlich in eine Web-Darstellung umgewandelt und dem Benutzer angezeigt. Zusätzlich zu den Zustandsinformationen der Klimaanlage und den Einstellungselementen sollen zudem die letzten Messwerte des Sensornetzwerks angezeigt werden. Ein Verlaufsdiagramm der Sensordaten ist zusätzlich

geplant, um die Auswirkungen von Aktionen des KI-Agenten oder äußerer Einflüsse verfolgen zu können.

4.7. Datenaufzeichnung

Um die Ergebnisse der Testszenarien nachvollziehen und auswerten zu können, ist ein entsprechender Aufzeichnungs-Client vonnöten. Die Funktion umfasst die Sammlung sämtlicher Sensor- und Klimasteuerungsdaten, welche über das Nachrichtenprotokoll übermittelt werden. Anschließend sollen die Informationen regelmäßig, in Datensätzen zusammengefasst, in eine Datei geschrieben werden [117]. Das Datenformat dieser Dateien, im Folgenden auch *Record Files* genannt, muss dabei so gestaltet werden, dass es problemlos wieder eingelesen und weiterverwendet werden kann. Außerdem sollte es ein menschenlesbares Format sein, damit eine manuelle Einsichtnahme stets möglich ist. Dementsprechend ist die Speicherung als Textdatei vorgesehen. Hier sind wiederum verschiedene Varianten denkbar, z. B. die Verwendung eines Logging-Clients pro Raum oder eines gemeinsamen für mehrere Räume. Beides ist im Rahmen dieser Arbeit geplant. Dabei bietet es sich an, möglicherweise zukünftig verwertbare Zustandsparameter, wie Wettervorhersagen, mit aufzuzeichnen [115]. Damit kann eine gute Grundlage für weiterführende Experimente und Testszenarien geschaffen werden.

4.8. KI-Agent

Ein RL-Agent lernt entweder direkt in der Umgebung, in der er eingesetzt werden soll oder in einer dieser Umgebung nachempfundenen Simulation. Es muss zudem berücksichtigt werden, dass RL, gerade bei Realweltanwendungen, mit einer langen Trainingsdauer verbunden ist, bis erste Erfolge zu verzeichnen sind. Ursache ist die zeitliche Verzögerung der Trainingsschritte solcher Szenarien im Vergleich zu anderen Lernarten, welche große Trainingsdatensätze stapelweise verarbeiten. Außerdem muss der Agent seine Umgebung zunächst erkunden und durch Versuch und Irrtum eine Strategie für die Aktionswahl entwickeln. Jedoch ist der Bedarf an Rechenleistung für diese Anwendungen auch wesentlich geringer, sodass auf Beschleunigung durch die Grafikkarte verzichtet werden kann und ggf. sogar der Einsatz auf eingebetteten Systemen möglich ist.

Für die Umsetzung des KI-Agenten im Rahmen dieser Arbeit soll TensorFlow in der Version 0.4.3 zum Einsatz kommen. Durch das Konzept des modularen Aufbaus bietet es, von den in Unterabschnitt 3.2.6 vorgestellten Ansätzen, die beste Grundlage für Realweltanwendungen. Im Rahmen der Umsetzung wird entsprechend ein eigener *Runner* benötigt, der im Endeffekt die Schnittstelle zwischen Agent und Umgebung darstellt. Der Agent selbst nutzt einen Algorithmus, um sein Modell in Form eines künstlichen neuronalen Netzes anzupassen. Sowohl der Algorithmus als auch das Modell müssen auf den jeweiligen Anwendungsfall angepasst werden.

4. Konzept

Geplant ist der Einsatz der KI-Steuerung zunächst in zwei Besprechungsräumen. In jedem dieser Räume wird ein BeagleBone inkl. Sensorik, folgend auch mit *SensorBone* bezeichnet, platziert, um das vorherrschende Raumklima zu erfassen. Die Räume sind mit etwa 10 und 20 m² Grundfläche zudem verschieden groß und besitzen ein bzw. zwei Klimageräte. Sie sind werktags unterschiedlich oft und stark sowie unregelmäßig in Benutzung. Die Klimageräte werden nur bei Bedarf manuell über das Bedienfeld aktiviert und deaktiviert. Zunächst soll ein Agent pro Raum eingesetzt werden. Wenn das System funktioniert, sind auch Tests mit einem Agenten für beide Räume vorgesehen. Vorerst wird, aufgrund der Wetterbedingungen, nur der Kühlmodus der Anlage getestet. Der Heizmodus kann evtl. bei niedrigeren Temperaturen genutzt werden. Allerdings ist die Notwendigkeit durch das Vorhandensein einer zusätzlichen Heizung unwahrscheinlich. Die Messung der Außenwerte wird zunächst vernachlässigt, da beide Räume keine Außenwände besitzen. Dadurch sind zudem die Umwelteinflüsse, z. B. durch Sonneneinstrahlung, wesentlich geringer, was wiederum den Lernvorgang beschleunigt. Aufgrund der partiellen Wahrnehmung der Umgebung sind Zustandsänderungen durch externe Einflüsse aus Sicht des Agenten oftmals nicht-deterministisch und hemmen den Lernprozess. Personen, die sich im Raum aufhalten, Sonneneinstrahlung sowie geöffnete Fenster und Türen beeinflussen das Raumklima, ohne dass der Agent dies als Folge seiner Aktionen unbedingt erwartet. Die Exploration ist besonders kurz vor und während der Nutzung des Raums unerwünscht und würde zu unkomfortablem Raumklima führen. Deshalb soll im Rahmen dieser Arbeit der, bereits in Unterabschnitt 3.2.6 vorgestellte, DQfD-Agent verwendet werden. Er ermöglicht das Trainieren eines Modells mit einer kleinen Menge gesammelter Demonstrationsdaten vor dem eigentlichen Einsatz. Im Gegensatz zu herkömmlichen RL-Algorithmen ist dadurch das Verhalten in der Startphase wesentlich besser. Ebendiese Eigenschaft kann helfen, das Risiko eines unkomfortablen Raumklimas durch Exploration des Agenten zu minimieren. Durch einen *priorisierten Replay-Speicher* fließt während der Laufzeit in der echten Umgebung zudem immer ein gewisser Anteil der Demonstrationsdaten mit in den Lernprozess ein. Je höher die Priorität eines Eintrags ist, desto häufiger wird er wiederverwendet. Der Replay-Speicher wird fortwährend mit neuen Daten aktualisiert, wobei ggf. alte Datensätze überschrieben werden. Nach dem *Pretraining*, also dem Vortrainieren, lernt der Agent also weiter durch direkte Interaktion mit seiner Umwelt. Dabei kann er die durch Demonstrationsdaten erlernte Strategie weiter verbessern und infolgedessen bessere Ergebnisse erzielen. Besonders für die Erweiterung eines bestehenden Systems mit KI stellt dies eine gute Lösung dar, ohne die bisherige Funktionsgüte durch Exploration stark zu verschlechtern. Für das Pretraining sind allerdings Demonstrationsdaten notwendig, welche während normaler Nutzungsphasen aufgezeichnet werden sollen. Dazu soll die Funktionalität des Logging-Clients, welcher in Abschnitt 4.7 bereits konzipiert wurde, verwendet werden. Für den KI-Agenten muss zudem der Zustandsraum definiert werden, d. h. welche Eingaben er in welchem Format bekommt, um seine Umwelt und somit den vorherrschenden Zustand zu erfassen. Es ist angedacht, neben der Raumtemperatur als Hauptparameter, zusätzliche Sensormesswerte, wie Luftfeuchtigkeit oder Luftqualität, einzubeziehen.

4. Konzept

Weitere Zustandsparameter könnten von den Reed-Sensoren, bspw. über geöffnete Fenster oder Türen stammen. Denkbar wäre ebenso eine Art Wettervorhersage über den gemessenen Luftdruck, wobei die Reaktionszeit der Klimaanlage für eine weiter vorausschauende Steuerung zu kurz erscheint. Ebenso muss der Aktionsraum festgelegt werden, der bestimmt, wie viele Aktionen der Agent in welchem Format ausgeben kann. Dabei ist eine Anpassung auf den jeweiligen Funktionsumfang der Klimageräte, also mögliche Betriebsmodi oder Begrenzung der Sollwerte, notwendig. Sowohl beim Zustandsraum als auch beim Aktionsraum muss beachtet werden, dass mehr Parameter immer mit steigendem Explorationsaufwand und damit längerer Trainingsdauer verbunden sind. Neben dem Aktionsraum ist es zudem erforderlich, einen passenden Aktionsintervall zu wählen. Er muss einerseits genug Zeit umfassen, bis die Wirkung des Klimageräts einsetzt, um Zustandsänderungen zu erfassen. Andererseits darf er aber nicht zu lang sein, damit eine genaue Regelung noch möglich ist. Weiterer Faktor für diese Entscheidung ist die, laut Inbetriebnahmeanleitung [22], empfohlene Einhaltung von maximal 7000 Änderungen pro Parameter jedes Innengeräts im Jahr. Diese Einstellungen werden im nicht-flüchtigen Speicher der Klimageräte abgelegt und dürfen daher nicht zu oft überschrieben werden. Mit einem gewissen Spielraum rund um einen einstellbaren Temperatursollwert soll es dem Agenten ermöglicht werden, Energie zu sparen und die restlichen Zustandsparameter in einem optimalen Bereich zu halten. Dazu muss sich die Belohnungsfunktion, wenn sich die Raumtemperatur im definierten Fenster um den Sollwert befindet, positiv sein, also eine Belohnung zurückgeben. Außerhalb des Fensters sollte sie dementsprechend keine Belohnung oder einen negativen Return, also eine Bestrafung, ergeben. Zusätzlich sollte sie im Temperatursollbereich zusätzliche Boni für gute „Nebenparameter“, wie Luftfeuchtigkeit oder Luftqualität, erteilen. Abzüglich muss es innerhalb des Temperatursollfensters Bestrafungen für energetisch aufwändige Aktionen, wie Kühlen oder Heizen, geben. Diese Strafe muss für den Lüftungsmodus logischerweise geringer sein. Für energiesparendes Verhalten sollte dementsprechend entweder ein neutraler Return oder eine Belohnung gewährt werden.

4.9. Bewertungskriterien

Um die Resultate des entwickelten KI-Systems mit der herkömmlichen Steuerung zu vergleichen, werden in diesem Abschnitt verschiedene Kriterien definiert. In unterschiedlichen Testszenarien soll damit ermittelt werden, wie sich die Erweiterung der Steuerung mit einer KI im Realwelteinsatz schlägt.

Erstes Kriterium ist der Energiebedarf. Es wird erwartet, dass mit der KI-Steuerung eine Senkung des Energiebedarfs im Vergleich zur herkömmlichen Steuerung erreicht wird. Durch die Gestaltung der Belohnungsfunktion kann darauf Einfluss genommen werden. Die Abschätzung erfolgt dabei durch den Anteil der Zeiträume, in denen das Innengerät eingeschaltet ist, am Gesamtumfang des jeweiligen Testlaufs. Zusätzlich wird der Betriebsmodus während der Aktivitätsphasen dafür ausgewertet. Der Lüftungsmodus ist dabei energetisch weniger aufwändig, als der Kühlbetrieb.

4. Konzept

Da es sich um ein VRV-System handelt, dessen Interna nicht bekannt sind, ist eine genauere Einschätzung nicht möglich.

Zweites Kriterium ist das Raumklima. Hier wird eine Besserung beim Einhalten des Temperatursollbereichs und ebenso eine Besserung der Nebenparameter prognostiziert. Letzteres ist jedoch davon abhängig, welche Nebenparameter in den Zustand des KI-Agenten aufgenommen und wie stark sie in der Belohnungsfunktion berücksichtigt werden. Die Werte hierfür können direkt aus den Messdaten der Sensoren entnommen und miteinander verglichen werden.

Drittes Kriterium ist die Anpassungsfähigkeit des Systems. Je nach Umgebung, in der der KI-Agent trainiert wird, sind verschiedene Verhaltensweisen zu erwarten. Möglicherweise fallen die Unterschiede sehr gering aus. Je mehr die äußeren Einflüsse dieser Umgebungen voneinander abweichen, desto größer sollten diese Unterschiede ausfallen. Die Beurteilung dieses Kriteriums erfolgt durch die Analyse der Aufzeichnungen des Logging-Clients. Dazu werden KI-Agenten bzw. ihre Aktionen in den verschiedenen Räumen analysiert und miteinander verglichen.

4.10. Zusammenfassung

Das Konzept für die einzelnen Bestandteile des Systems wird in Abbildung 4.2 zusammenfassend dargestellt. Beschrieben wird der Einsatz in zwei verschiedenen

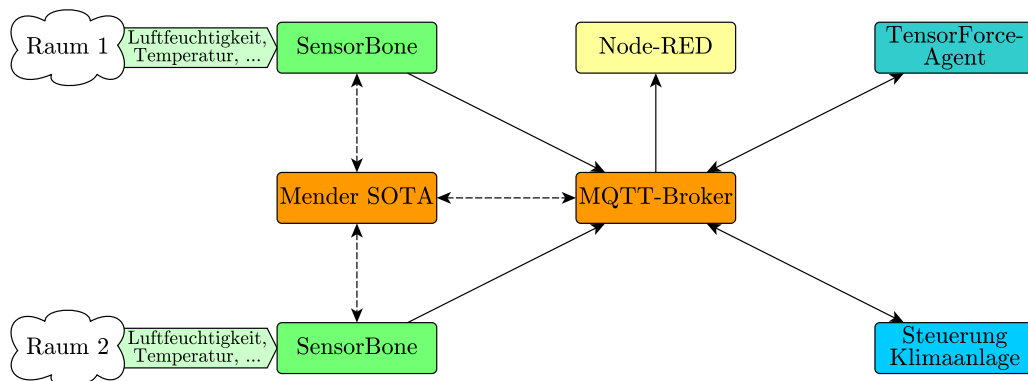


Abbildung 4.2.: Architekturkonzept

Räumen. Darin wird jeweils ein BeagleBone, ausgestattet mit Sensorik zur Erfassung des Raumklimas, platziert. Alle Programme zur Erfassung und Aufzeichnung der Messwerte, Anbindung der Klimaanlage sowie zur Realisierung der KI werden in Python geschrieben. Die Übermittlung sämtlicher Nachrichten erfolgt über das MQTT-Protokoll. Grundlage der Vernetzung ist ein Mesh-Netzwerk nach dem IEEE 802.11s. Der BeagleCore übernimmt dabei die zentrale Rolle des Brokers und Gateways zwischen LAN und Mesh-Netzwerk. Im LAN befindet sich der iTC als Steuerungszentrale der Klimaanlage und ein Desktop-PC als weiterer Teilnehmer. Auf diesem PC werden die Bedienoberfläche, welche durch Flows mit Node-RED

4. Konzept

und der Dashboard-Erweiterung umgesetzt wird sowie der KI-Agent, implementiert mit TensorFlow, ausgeführt. Ein Logging-Client erfasst alle anfallenden Informationen wiederverwertbar in einer Datei. Diese Aufzeichnungen sollen zur Bewertung des Systems, als auch zum Pretraining des KI-Agenten mittels DQfD verwendet werden. Außerdem ist der Desktop-PC die Plattform für den Server des Mender-Update-Server, als auch für die Integration der Klimaanlage. Das Linux-Betriebssystem der BeagleBone wird mit Yocto Project speziell für diese Anwendung erstellt. Über das eingebaute Mender-System können komplette Systemaktualisierungen auf die Einplatinenrechner aufgespielt werden. Im nächsten Kapitel folgen die Details zur konkreten Umsetzung der einzelnen Punkte des Konzepts.

5. Umsetzung

Dieses Kapitel beschreibt die Umsetzung des vorher erarbeiteten Konzepts. Dementsprechend erfolgt hier die genaue Beschreibung der einzelnen Komponenten und Besonderheiten des Systems in den Abschnitten *Inbetriebnahme der Komponenten*, *Anbindung der Klimaanlage*, *Betriebssystem*, *Sensornetzwerk*, *Systemaktualisierung der Sensorknoten*, *Nutzerschnittstelle*, *Datenaufzeichnung* und *KI-Agent*.

Die im vorhergehenden Kapitel geplante Systemarchitektur ist in Abbildung 5.1 grafisch dargestellt. Wie bereits in Abschnitt 4.1 beschrieben, kommt für die Implemen-

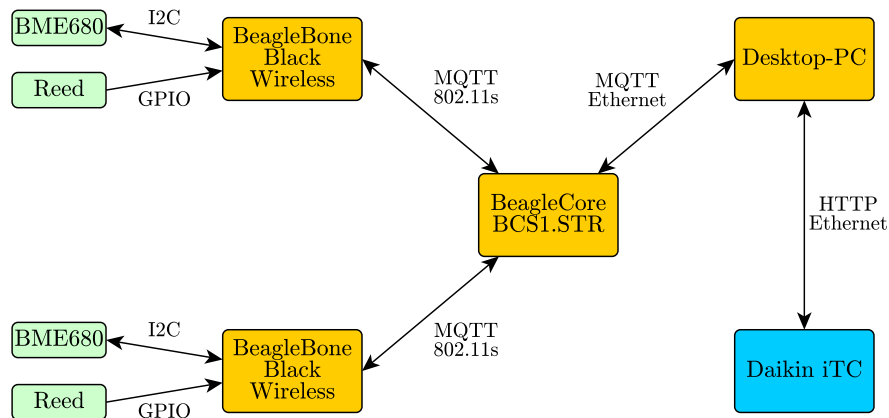


Abbildung 5.1.: Übersicht der Netzwerkstruktur mit Sensornetz und LAN

tierung der Software ausschließlich die Programmiersprache Python zum Einsatz, da alle benötigten Bibliotheken und Module dafür verfügbar sind. Als gemeinsame Grundlage der Clients auf dem Desktop-PC werden ausgewählte Konstanten als globale Definitionen in der Datei *Shared_Definitions.py* hinterlegt. Enthalten sind sowohl die Standardbelegung der Sollwerte sowie deren Toleranzbereiche mit dem Suffix *_EPSILON* als auch der Referenzwert für die Berechnung der Luftqualität. Außerdem beinhaltet die Datei Verbindungsinformationen des MQTT-Brokers und des iTCs. Die Datenstrukturen zur Interaktion mit der HTTP-Schnittstelle des iTC sind als *namedtuples* spezifiziert. Sie haben den Vorteil einer festgelegten Struktur gegenüber normalen *tuples* in Python. Damit wird gewährleistet, dass eine solche Struktur immer alle vorgesehenen Elemente enthält. Hinzu kommt, dass über die Bezeichner auf die Elemente zugegriffen werden kann, sodass es Verwechslungen durch eine abweichende Reihenfolge der Daten ausgeschlossen sind. Die Zuordnung der Rückgabewerte des Daikin iTC zu ihrer Bedeutung wird durch *Dictionaries* dargestellt. Damit kann der zurückgegebene Wert direkt als Schlüssel ver-

5. Umsetzung

wendet und in eine menschenlesbare Meldung übersetzt werden. In den Dictionarys *COMMAND_REQUEST* und *COMMAND_RESPONSE* sind jeweils named-tuples vom Typ *Command_Info* hinterlegt. Diese geben dem HTTP-API-Modul Auskunft darüber, welcher *Identifizier (ID)* für ein Kommando an den iTC gesendet werden muss und wie groß die Anfrage bzw. die Antwort minimal und maximal sein darf. Die Übersetzung der Aktionen, welche der KI-Agent ausgeben darf, in die korrespondierenden Konfigurationen für die Klimageräte ist ebenfalls in einem Dictionary mit dem Bezeichner *ACTIONS* festgelegt. Der darin spezifizierte Wert des *Setpoints*, also des Temperatursollwerts, wird dabei noch auf die aktuell angegebene Wunschtemperatur aufaddiert und ergibt damit erst die korrekte Geräteeinstellung. Für die Verarbeitung der Sensorwerte und Statusinformationen zu Eingaben des KI-Agenten steht die Funktion *PROCESS_STATE()* bereit. Sie erhält den aktuellen Zeitstempel, sämtliche Soll- und Messwerte sowie den Betriebsmodus des Innengeräts und die gewählte Aktion als Parameter. Daraus werden die Soll-Ist-Differenzen für Temperatur und Luftfeuchtigkeit sowie die Luftqualität, relativ zum Referenzwert von *1500000* in *GAS_BASELINE*, ermittelt. Diese Werte werden zusammen mit der Aktion von der Belohnungsfunktion *CALCULATE_REWARD()* in einen Return umgewandelt. Im Falle einer nicht in *ACTIONS* definierten Aktion wird *0* zurückgegeben. Die nun folgende Belohnungsberechnung reflektiert beispielhaft lediglich den letzten Entwicklungsstand. Alle Zwischenschritte bis zu diesem Ergebnis sind in den Ausführungen von Kapitel 6 beschrieben. Wenn die übergebene Aktion gültig ist, der Betrag der Temperaturdifferenz aber nicht mehr unter *TEMPERATURE_AIM_EPSILON* liegt, so hat das eine Bestrafung in der Höhe von *WEIGHT_PENALTY* zur Folge. In allen anderen Fällen erfolgt die Berechnung einer Belohnung als Summe aus *WEIGHT_TEMPERATURE* sowie Anteilen von *WEIGHT_HUMIDITY*, *WEIGHT_QUALITY* und *WEIGHT_ENERGY*. Die Komponente zur Bewertung der Luftfeuchtigkeit entscheidet anhand der Schwellwerte *HUMIDITY_AIM_EPSILON1* und *HUMIDITY_AIM_EPSILON2*, ob eine Belohnung bzw. Bestrafung in Höhe von *WEIGHT_HUMIDITY* oder *0* als Mittelwert vergeben wird. Der Wert der Luftqualität beträgt minimal *0* und maximal *1*. Das muss ggf. durch Anpassung von *GAS_BASELINE* an die örtlichen Gegebenheiten sichergestellt werden. Dieser Wert wird auf einen Wertebereich von *-1* bis *1* abgebildet und bestimmt so als Faktor den Einfluss von *WEIGHT_QUALITY*. Damit werden Werte unter *0,5* bestraft, während Werte über *0,5* eine Belohnung erhalten. Schließlich erfolgt die Auswertung der Energieaufnahme. Dabei wird eine Aktion, bei der das Klimagerät ausgeschaltet ist, mit *WEIGHT_ENERGY* belohnt. Aktionen im Lüftungsmodus werden mit der Hälfte davon und alle verbleibenden, dementsprechend im Kühlmodus, mit dem vollen Umfang von *WEIGHT_ENERGY* bestraft. Die Belohnungsfunktion setzt sich somit aus folgenden Formeln zusammen:

Luftqualität:

$$R_q(q) = 2q - 1$$

Luftfeuchtigkeit:

$$R_h(\Delta_h) = \begin{cases} 1 & \text{wenn } |\Delta_h| < \epsilon_{h1} \\ 0 & \text{wenn } |\Delta_h| < \epsilon_{h2} \\ -1 & \text{sonst} \end{cases}$$

Energieaufnahme:

$$R_e(a) = \begin{cases} 1 & \text{wenn } a.on_off = 0 \\ -\frac{1}{2} & \text{wenn } a.operation_mode = 1 \\ -1 & \text{sonst} \end{cases}$$

Wichtung:

$$R(\Delta_t, \Delta_h, q, a) = \begin{cases} w_t + w_q R_q(q) + w_h R_h(\Delta_h) + w_e R_e(a) & \text{wenn } |\Delta_t| < \epsilon_t \\ -w_p & \text{sonst} \end{cases}$$

q Messwert Luftqualität

w_t Gewicht des Temperatur>Returns

a gewählte Aktion

w_q Gewicht des Luftqualitäts>Returns

Δ_h Soll-Ist-Differenz Luftfeuchtigkeit

w_h Gewicht des Feuchtigkeits>Returns

Δ_t Soll-Ist-Differenz Temperatur

w_e Gewicht des Energie>Returns

w_p Gewicht der Betrafung

Für die Steuerung mehrerer Klimageräte mit nur einem KI-Agenten ist zudem das Äquivalent `PROCESS_STATE_GLOBAL()` zu dieser Funktion vorhanden. Außerdem enthält das Modul die Definition der Funktion `log_message()` zur Ausgabe von Meldungen mit Zeitstempel über die Standardausgabe.

Damit die hier festgelegten Definitionen in den anderen Python-Programmen genutzt werden können, müssen sie sich im selben Verzeichnis befinden und die benötigten Variablen, z. B. mit `from Shared_Definitions import Configuration, ACTIONS`, importieren.

5.1. Inbetriebnahme der Komponenten

Bevor mit der Implementierung der Software begonnen werden kann, erfolgt die Installation des HTTP-Interface-Modul auf dem iTC. Dazu wird die Konfigurationsoberfläche am Gerät aufgerufen und der vorher aktivierte Lizenzschlüssel eingegeben. Nach einem Neustart des iTC kann der Netzwerkport, unter dem die Schnittstelle erreichbar sein soll, konfiguriert und das Modul verwendet werden. Genauere Informationen dazu sind im entsprechenden Handbuch [25] nachzulesen.

5. Umsetzung

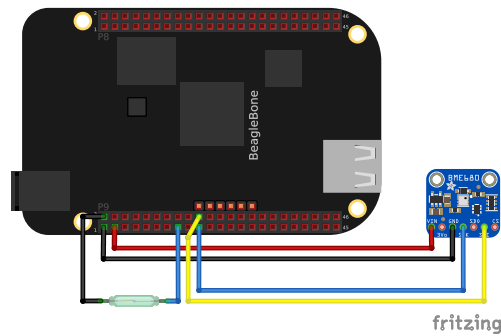


Abbildung 5.2.: Überblick der Sensorschaltung [4]

Die BeagleBones werden mit Speicherkarten, Steckernetzteilen und der BeagleCore zudem mit einem WLAN-USB-Stick ausgestattet. Die nun folgende Beschreibung der Schaltung ist in Abbildung 5.2 grafisch dargestellt. Für den Anschluss der Umweltsensoren müssen an die Breakout-Boards zunächst Pin-Leisten angelötet werden. Danach können sie über Jumper-Kabel am Anschluss *P9* der BeagleBones mit den Pins *1* und *3* zur Stromversorgung sowie mit Pin *19* und *20* verbunden werden, was dem dritten I2C-Bus mit Index *2* entspricht. An die Anschlussleitungen der Reed-Sensoren werden direkt Jumper-Kabel angelötet und diese, ebenfalls an *P9*, zwischen Pin *15* und Masse-Pin *2*, als Schalter verwendet. Hierbei muss beachtet werden, dass die Reed-Sensoren entweder gegen Masse schalten und damit ein *Pull-Up*-Widerstand am GPIO-Pin benötigt wird oder gegen Plus schalten und dementsprechend ein *Pull-Down*-Widerstand am GPIO-Pin benötigt wird. Welche Variante gewählt wird, ist für diesen Anwendungsfall letztendlich nicht entscheidend. Deshalb wurde sich hier für die Umsetzung mittels Pull-Up und Schalten gegen Masse entschieden. Hierzu sollen die entsprechenden internen Widerstände der BeagleBones verwendet werden. Dazu sind Änderungen am *Device-Tree* nötig, die wiederum in den Yocto-Layer einfließen. Der *Device-Tree* ist eine Hardware-Beschreibung, die dem Linux-Kernel übergeben wird, um die korrekte Erkennung aller Geräte und das Laden der zugehörigen Treiber zu ermöglichen.

5.2. Anbindung der Klimaanlage

Die Anbindung des iTC ist aufgeteilt in zwei Komponenten, *Daikin_HTTP_API.py* und *daikinMQTT.py*. Die HTTP-API ist als Python-Modul in *Daikin_HTTP_API.py* implementiert. Hauptbestandteil ist die Klasse *Daikin_iTC*, welche sämtliche Methoden der Schnittstelle implementiert. Sie folgt der Protokollbeschreibung des HTTP-Schnittstellenmoduls [22]. Enthalten sind einerseits statische Methoden zum Erzeugen und Umwandeln der geforderten Datenstrukturen, erkennbar durch den Zusatz *@staticmethod* sowie den fehlenden *self*-Parameter. Andererseits existieren instanzbezogene Methoden zum Ausführen der Steuer- und Abfragebefehle der Schnittstelle. Die URL des Daikin iTC wird dem Konstruktor der Klasse, welcher bei Python

5. Umsetzung

stets der Methode `__init__` entspricht, als Parameter übergeben und pro Instanz gespeichert. Sie ist zentral festgelegt und wird aus dem Modul `Shared_Definitions.py` importiert. Die Methode `execute_command()` bildet die Grundlage zum Senden und Empfangen von Befehlen sowie Daten zum bzw. vom iTC. Sie erwartet den Bezeichner des auszuführenden Befehls, welcher in `COMMAND_REQUEST` aus `Shared_Definitions.py` hinterlegt sein muss. Mit Aufruf von `struct.pack()` wird ein Byte-Objekt in der Variable `payload` erzeugt, welches mit den übergebenen Inhalten im spezifizierten Format befüllt wird. Das angegebene Format `<LL` steht dabei für die Byte-Reihenfolge `little-endian`, die der iTC erwartet und zwei vorzeichenlose, vier Byte umfassende Ganzzahlen. In den ersten vier Byte muss immer die Größe des gesendeten Datenpakets und in den folgenden vier Byte die Befehls-ID angegeben werden. Die Größe umfasst dabei den Kopf, also acht Byte für die Größenangabe sowie die Befehls-ID und den „Rumpf“, welcher der Methode im Parameter `data` übergeben wird. Die Befehls-ID wird aus der in `COMMAND_REQUEST` hinterlegten `Command_Info`-Struktur entnommen, auf welche wiederum mit dem Bezeichner des Befehls als Schlüssel zugegriffen wird. Schließlich wird der Inhalt des übergebenen Parameters `data`, ebenfalls ein Byte-Objekt, an `payload` angehängt. Danach erfolgt die Prüfung, ob die Gesamtgröße des Pakets innerhalb der in der `Command_Info`-Struktur des Befehls vorgegebenen Grenzen liegt. Bei erfolgreicher Prüfung wird die Anfrage als HTTP-POST-Anfrage mit der Methode `requests.post()` an den iTC geschickt. Im Kopf der Anfrage muss dabei `Content-Type` auf `application/octet-stream` gesetzt werden. Die `post()`-Methode gibt die empfangene Antwort in einer Response-Struktur zurück. Es folgt die Überprüfung des HTTP-Status-Codes und der Größe der empfangenen Rumpfdaten gemäß `Command_Info` aus `COMMAND_RESPONSE`. Zusätzlich werden die ersten beiden Vier-Byte-Werte, analog zu `struct.pack()` mittels `struct.unpack_from()`, aus der Antwort entnommen. Sie enthalten ebenfalls die Größe und die Befehls-ID der Antwort, welche mit der tatsächlichen Größe und der erwarteten ID aus `Command_Info` verglichen werden. Sofern alle Werte den Erwartungen entsprechen, wird der Inhalt der Antwort von der Methode zurückgegeben. Im Fehlerfall erfolgt stets die Rückgabe einer leeren Zeichenkette.

Methoden, welche intern Befehle mit `execute_command()` ausführen, erzeugen die jeweils benötigte `data`-Struktur, übergeben diese zusammen mit dem Befehlsbezeichner und werten schließlich die Antwort aus. Bei `get_connection_status()` und `set_status()` beinhaltet die Antwort ab Byte 20 ein Bitfeld, welches die Adressen der Klimageräte widerspiegelt. Dieses wird mit Hilfe der statischen Methode `address_struct_to_addresses(response[20:])` durch Bitmaskierung in eine menschenlesbare Liste der Adressen umgewandelt. Für die Umwandlung in die entgegengesetzte Richtung, die in `get_status()` benötigt wird, ist `addresses_to_address_struct()` zuständig. Darin wird zudem die Hilfsmethode `address_to_byte_bit_index()` verwendet, um die Byte- und Bit-Position einer Adresse im Bitfeld zu berechnen. Die Methoden `get_properties()` und `get_status()` liefern jeweils ein Dictionary mit einer `Unit_Properties`- bzw. `Unit_Status`-Struktur, entsprechend der Geräteadresse als Schlüssel, zurück. Das Dictionary wird bei `get_status()` in ein Tupel eingebettet, welches zusätzlich die Anzahl der angeforderten Geräte enthält. `get_properties()` nutzt

intern die statische Hilfsmethode *label_to_address()*, um die im *long_name* angegebene Bezeichnung des Geräts in eine Adresse umzuwandeln. Das Gegenstück hierzu, *address_to_label*, wird innerhalb der Klasse nicht benötigt, steht aber zur Nutzung in anderen Programmteilen zur Verfügung. In den Auswertungsroutinen der Methoden werden die empfangenen Daten, entsprechend der Protokolldefinition, verschieden interpretiert. Erwähnenswert ist hierbei die Verwendung der Formatangabe *<f* beim Aufruf von *struct.unpack()* an verschiedenen Stellen. An dieser Stelle werden die vier übergebenen Bytes, wieder in *little-endian*-Reihenfolge, als Gleitkommazahl entpackt. Die Erzeugung der zu sendenden Datenstruktur in der Methode *set_status()* ist die komplexeste dieser Klasse. Als Eingabe erhält die Methode ein Dictionary mit *Configuration*-Objekten der zu konfigurierenden Innengeräte. Dieses wird durchlaufen und die Inhalte Schritt für Schritt auf Übereinstimmung mit den möglichen Einstellungen aus Dictionarys, wie *ON_OFF* oder *FAN_DIRECTION*, geprüft. Bei einer plausiblen Einstellung wird der Wert in das Byte-Array *change_struct* dieses Innengeräts eingetragen und in der Variable *set_bits* ein Merkbit für diese Änderung gesetzt. Nach dem Durchlauf eines jeden *Configuration*-Objekts wird *set_bits* ausgewertet, *change_struct* bei mindestens einer Änderung an die Anfragedaten in *data* angehängt und *change_count* inkrementiert. Nur, wenn nach dem Durchlaufen aller *Configuration*-Objekte des Dictionarys mindestens eine Änderung in *change_count* vermerkt ist, wird der *SET_STATUS*-Befehl ausgeführt. Das dient der Vermeidung unnötiger Schreibzugriffe auf den Speicher der Klimageräte, was bereits in Abschnitt 4.8 begründet wurde.

Im Client *daikinMQTT.py* findet die eben beschriebene Klasse *Daikin_iTC* nun Verwendung. Der zugrundeliegende Aufbau des Programms entspricht dabei Abbildung 5.5. Der MQTT-Client verbindet sich in *create_mqtt_connection()* mit Name und Passwort zum Broker. Im *Callback on_connect()* abonniert er die notwendigen Topics aller zu steuernden Innengeräte. Ein Überblick der gesamten Topic-Struktur ist in Abbildung 5.3 dargestellt. Die angesprochenen Innengeräte sind als Tupel in der globalen Variable *UNIT_ADDRESSES* aus *Shared_Definition.py* definiert. Danach beginnt der Client, regelmäßig die verbundenen Klimageräte sowie deren Status vom iTC abzurufen und diesen im jeweiligen Topic des Klimageräts zu publizieren. Das geschieht in der Funktion *publish_data()*, welche indirekt via *Timer* von *start_publish_data()* aufgerufen wird. Der Veröffentlichungsintervall ist in der Variable *PUBLISH_INTERVAL* standardmäßig auf zehn Sekunden festgelegt. Da die Statusabfrage ein Dictionary zurückgibt, dieses jedoch nicht direkt per MQTT-Nachricht versendet werden kann, wird es in eine JSON-Zeichenkette umgewandelt. Eingehende Status- und Aktionsnachrichten werden in der Callback-Funktion *on_message()* ausgewertet, sodass keine unnötigen Schreibvorgänge auf dem iTC stattfinden. Auf den *action*-Topics empfangene Aktionen für Innengeräte werden anschließend in die *Configuration*-Struktur umgewandelt und über die HTTP-API an den iTC gesendet. Nachrichten auf den *set_status*-Topics werden direkt als JSON-Objekt interpretiert und im passenden Format an den iTC übermittelt. Zudem wird der Temperatursollwert von der Bedienoberfläche abonniert, um in der Lage zu sein, die Sollwerte für die Einstellung der Klimageräte entsprechend der Aktionen berech-

5. Umsetzung

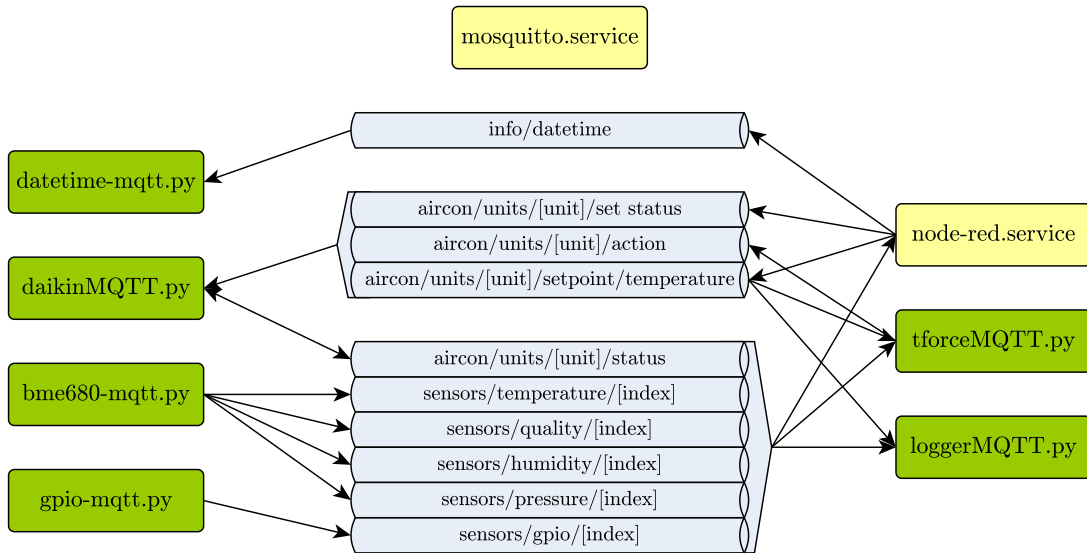


Abbildung 5.3.: Übersicht der MQTT-Topics

nen zu können. Eine Übersicht der kompletten MQTT-Struktur ist in Abbildung 5.3 abgebildet. Detailliertere Informationen zu Aufbau und Funktionsweise der MQTT-Clients können zudem den Ausführungen in Abschnitt 5.8 entnommen werden.

5.3. Betriebssystem

Das Betriebssystem der BeagleBone-Sensorknoten wird mit Yocto erstellt. Zunächst erfolgt die Einrichtung eines Projektes mit dem Herunterladen der Referenzdistribution Poky. Anschließend werden die Layer hinzugefügt. Dies kann entweder über das Tool *bitbake-layers* oder manuell, durch Eintragen der Pfade in `conf/bblayers.conf`, geschehen. Die Konfiguration erfolgt in der Datei `conf/local.conf`. Dort werden bspw. die Hardwareplattform bzw. das BSP mittels *MACHINE*, zusätzliche Softwaremodule in `IMAGE_INSTALL_append` und Einstellungen für die *bitbake*-Umgebung festgelegt. An dieser Stelle wird auch der zusätzliche Nutzer *dev* angelegt und das Passwort für *root* gesetzt. Das geschieht durch Einfügen der entsprechenden Befehle in die Variable `EXTRA_USERS_PARAMS` und das Anhängen von *extrausers* an die Variable `INHERIT`. Für weitergehende Änderungen, die nicht in vorgefertigten Layers zur Verfügung stehen, muss ein Layer selbst erstellt werden. Darin können neue Rezepte erzeugt oder bestehende durch sogenannte *Append-Files* modifiziert werden. Das kann wiederum durch das *recipe-tool* oder per Hand durch Anlegen und Bearbeiten der entsprechenden Dateien erreicht werden. Bei *Append-Files* muss stets darauf geachtet werden, innerhalb des Layers dieselbe Ordnerstruktur und Rezeptkategorie, z. B. *recipes-core*, des Originalrezeptes zu verwenden. Ansonsten kann *bitbake* die Datei nicht auffinden und wird sie beim Erstellen des Systems nicht berücksichtigen. In diesem Fall wurden die existierenden Layer *meta-oe*, *meta-python*

und *meta-networking* aus dem OpenEmbedded Layer Index hinzugefügt. Zusätzlich wurde der Layer *meta-kihvac* für eigene Modifikationen erstellt. Darin enthalten ist die Kategorie *recipes-applications* mit dem Rezept *platform-init.bb*. Dieses installiert das Startskript zur Initialisierung der Plattform, welches, wie in Abschnitt 4.3 beschrieben, die verschiedenen Software-Konstellationen je nach Hardware startet. Das Skript selbst wird beim Systemstart vom ebenfalls im Rezept enthaltenen Dienst ausgeführt. Anpassungen des Linux-Kernels für die Funktion der *Light-Emitting Diodes (LEDs)* auf den Platinen und des USB-Netzwerk-Adapters [13] sind mit sogenannten Konfigurationsfragmenten realisiert. Diese befinden sich in der Kategorie *recipes-kernel* und werden durch das Append-File *linux-yocto_%.bbappend* eingebunden. Ebenfalls über ein Append-File wird der Login des Benutzers *root* über *Secure Shell (SSH)* deaktiviert. Dieses befindet sich unter der Kategorie *recipes-connectivity* im Ordner *openssh*. Damit wird an den Konfigurationsschritt des Pakets ein Befehl angehängt, der die entsprechende Zeile der Konfigurationsdatei *sshd.config* modifiziert. Die Konfiguration des in Poky bereits enthaltenen BSPs der BeagleBones, *meta-yocto-bsp/conf/machine/beaglebone.conf*, muss bei der Nutzung von BeagleBone Black Wireless zudem noch modifiziert werden. Der Device-Tree dieses Modells fehlt in der Variable *KERNEL_DEVICETREE* und muss dort durch Hinzufügen von *am335x-boneblack-wireless.dtb* an erster Stelle ergänzt werden. Wird das nicht getan, so erkennt das Betriebssystem bspw. die WLAN-Hardware nicht und kann diese nicht nutzen.

Nach Fertigstellung der Anpassungen wird das Abbild des gewünschten Systems, in diesem Fall *core-image-full-cmdline*, mit dem Erstellungswerkzeug *bitbake* erzeugt.

5.4. Sensornetzwerk

Die WLAN-Chips, welche bei den BeagleBones verwendet werden, benötigen neben dem entsprechenden Device-Tree auch Treiber und Firmware. Diese sind als Kernel-Module bzw. Pakete verfügbar und werden mit Yocto ins Linux-System eingebunden. Das wird durch das Hinzufügen der Namen dieser Komponenten zur Variablen *IMAGE_INSTALL_append* in der Datei *local.conf* erreicht. Benötigt werden jeweils Kernel-Modul und *linux-firmware*-Paket für *wl18xx* der BeagleBone Black wireless sowie für *rtl8192cu* des WLAN-USB-Sticks und das Konfigurationswerkzeug *iw*. Für die beiden BeagleBone Black Wireless wird zusätzlich das Modul *wlcore-sdio* benötigt. Damit die genannten Module überhaupt erzeugt werden, wird die Konfiguration des Kernels mit den Fragmenten *wlan.cfg* dementsprechend angepasst. Für die Integration der Mesh-Unterstützung nach IEEE 802.11s wird analog das Konfigurationsfragment *mesh_network.cfg* eingebunden. Da es bei der Verbindung des BeagleCore mit dem Mesh teilweise erst nach einigen Neustarts des WLAN-Adapters gelingt, die anderen Teilnehmer zu erreichen, wird hierfür eine Schleife in das Startskript eingebaut. Zur Absicherung des Mesh-Netzwerks stehen verschiedene Möglichkeiten zur Verfügung. Jedoch kann mit keiner der Varianten, welche *auth-sae* [7], *wpa-supPLICANT* [83] bzw. *wpa-supPLICANT-mesh* [84] nutzen, nach Ein-

richtung noch im Mesh kommuniziert werden. Es scheint sich dabei um ein häufig auftretendes Problem zu handeln [59][60]. Tests mit der neueren Poky-Version 2.5 *sumo* statt 2.4 *rocko* endeten mit demselben Fehlerbild. Der Grund hierfür konnte im Verlauf dieser Arbeit nicht eindeutig ermittelt werden, weshalb die Absicherung zugunsten der restlichen Umsetzung zunächst deaktiviert bleiben muss.

Aufbauend auf das Mesh-Netzwerk kommt das *Internet Protocol Version 4 (IPv4)* zum Einsatz. Die Teilnehmer des Sensornetzwerks werden über die Ethernet-Schnittstelle des BeagleCore als Mesh Portal an ein bestehendes LAN angebunden. In diesem Netzwerk werden die Adressen von einem Server per *Dynamic Host Configuration Protocol (DHCP)* vergeben. Damit die Sensorknoten zwecks des einfacheren Zugriffs über SSH stets dieselbe Adresse zugewiesen bekommen, muss das Client-Programm *dhcpcd* die *clientid* anstatt der standardmäßig festgelegten *duid* verwenden. Diese Anpassung wird über ein Append-File für *dhcpcd* in *recipes-connectivity* durchgeführt. Die Ethernet-Schnittstelle des BeagleCore wird zusammen mit dem WLAN-Adapter in einem *Bridge-Interface* zusammengefasst. Dadurch werden die beiden anliegenden Netze logisch vereint, d. h. alle Geräte sind Teil des bestehenden Netzwerks und somit vom Desktop-PC erreichbar. Die nötige Software zur Erstellung des Bridge-Interfaces wird mit dem Paket *bridge-utils* zur Variablen *IMAGE_INSTALL_append* in der Datei *local.conf* hinzugefügt. Gestartet werden sowohl DHCP-Client, als auch Bridge-Interface durch das Startskript. Der BeagleCore soll zudem die Rolle des MQTT-Brokers übernehmen. Im OpenEmbedded Layer Index ist mit *mosquitto* ein entsprechendes Rezept für diese Funktion verfügbar, welches wiederum via *IMAGE_INSTALL_append* in das Zielsystem installiert wird. Zusätzlich werden die Konfigurationsdateien *mosquitto.conf* und *users.conf* mit einem Append-File in *recipes-connectivity* eingefügt. In erstgenannter Datei werden die Einstellungen *allow_anonymous false* und *password_file /etc/mosquitto/users.conf* festgelegt. Das aktiviert die Authentifizierung für Verbindungen zum Broker mit den in *users.conf* festgelegten Zugangsdaten. Diese werden mit dem dafür vorgesehenen Werkzeug, *mosquitto_passwd*, jeweils aus Benutzername und Passwort generiert. Die Sensorknoten werden durch die Rezepte *python3* und *python3-paho-mqtt* aus dem Layer *meta-python* mit Python 3 und der MQTT-Bibliothek von *Eclipse Paho* ausgerüstet. Zum Auslesen der Sensorik am I2C-Bus und den GPIO-Pins existieren ebenfalls Python-Module, welche mit den Rezepten *python3-adafruit-bb10* und *python3-bme680* aus der Kategorie *recipes-devtools* installiert werden. Da diese noch nirgends bereitgestellt werden, müssen sie selbst erstellt werden. Erstgenanntes Rezept ist für die Nutzung der GPIO-Pins vorgesehen. Die Bibliothek stammt von *Adafruit* [1] und wird aus dem *Python Package Index* abgerufen und installiert, was dem Aufrufen des Kommandos *pip install* bspw. auf Desktop-Systemen entspricht. Dazu muss das Rezept von den Rezeptklassen *pypi* und *setuptools3* erben sowie Prüfsummen für das Paket enthalten, um eine fehlerfreie Installation zu gewährleisten. Dasselbe Vorgehen findet auch beim Rezept für die BME680-Bibliothek *python3-bme680* Anwendung. Aufgrund der zusätzlichen Abhängigkeiten zu anderen Modulen wird hier nicht die Adafruit-Bibliothek [3], sondern eine Alternative von *Pimoroni* [69] verwendet. Sie ist für ähnliche Breakout-

5. Umsetzung

Boards ausgelegt, welche lediglich keine Anbindung über SPI besitzen. Da an dieser Stelle sowieso nur der I2C-Bus verwendet werden soll und die Funktionalität ansonsten gleichwertig ist, kommt diese Variante zum Einsatz. Hierbei muss darauf geachtet werden, dass dem Build-System die bestehende Abhängigkeit zum Modul *smbus* mitgeteilt wird. Das geschieht durch Hinzufügen des Paketnamens zur Variablen *RDEPENDS_\${PN}* zum Rezept. Infolgedessen wird das genannte Modul automatisch mit installiert, um die Abhängigkeit zu erfüllen.

Des Weiteren sind für die Konfiguration der GPIO-Pins Anpassungen notwendig, die wiederum in das Append-File zur Anpassung des Kernels, *linux-yocto_%.bbappend*, eingepflegt werden. Die folgenden Änderungen stammen aus den Quellen des Beagle-Bone Linux-Kernels [11]. Zuerst wird das *Patch-File 0001_bone_pinmux_gpio.patch* betrachtet. Es fügt ein notwendiges *Label* in den Device-Tree ein und inkludiert eine neue Device-Tree-Quelldatei, die den Pin-Multiplexer der BeagleBones mit sämtlichen Funktionen und Pull-Ups sowie -Downs beschreibt [10]. Außerdem werden im Abschnitt *Éocp*, in Referenz auf das zuvor eingefügte Label, sogenannte *Pin Multiplex Helper* registriert. Sie ermöglichen es, die Konfiguration der Pins als normaler Benutzer des Systems zu ändern. Für diese beiden Funktionalitäten sind zudem noch Treiber notwendig, die durch das Patch- und das Append-File als Optionen in die Konfigurationsoberfläche *Kconfig* eingefügt werden [12]. Die korrespondierenden *Makefiles* zum Übersetzen werden im selben Zug angepasst bzw. eingefügt. Der Quelltext der Treiber ist in den Dateien *gpio-of-helper.c* und *bb_bone_pinmux-helper.c* enthalten. Durch das Patch-File werden auch einige Änderungen am Treiber der GPIO-Schnittstelle vorgenommen. Die Aktivierung der, in die Kernel-Konfiguration eingepflegten, Optionen erfolgt schließlich durch ein weiteres Konfigurationsfragment in der Datei *gpio.cfg*.

Da nun alle Voraussetzungen zum Erfassen von Messwerten und deren Übermittlung beschrieben sind, kann auf die Entwicklung der Software eingegangen werden. Im eigenen Yocto-Layer wird dafür unter der Kategorie *recipes-applications* das Rezept *python-mqtt.bb* erzeugt. Es enthält sowohl die beiden MQTT-Clients *bme680-mqtt.py* und *gpio-mqtt.py* für die Bosch BME680-Sensoren und die GPIO-Sensorik, als auch *datetime-mqtt.py* zum Synchronisieren der Systemzeit auf den BeagleBones. Ausschlaggebender Grund für die Notwendigkeit der Zeitsynchronisation sind die Mender-Clients. Deren Verbindungsversuche werden bei zu großer Zeitabweichung durch die Zertifikatsprüfung des Servers abgewiesen. In Anbetracht der Tatsache, dass die BeagleBones keine gepufferte *Real-Time Clock (RTC)* besitzen, somit bei Trennung der Stromversorgung ihre Systemzeit verlieren und nicht immer ein *Network Time Protocol (NTP)*-Server erreichbar ist, wird das als geeignete Lösung angesehen. Zu jedem der drei Clients ist außerdem ein entsprechender Dienst enthalten, der den korrespondierenden Client startet. Die Dienste sind für das *Init-System systemd* ausgelegt, welches als erster Prozess nach dem Start des Betriebssystems für die Verwaltung aller weiteren Prozesse zuständig ist. Die Dienste der beiden Clients zur Sensorauswertung werden mittels sogenannter *Service Templates* umgesetzt, was @ in deren Namen erkennbar ist. Das bietet die Möglichkeit, über den *Instance Identifier*, der beim Start des Dienstes nach dem @ angegeben wird, Parameter

direkt an das Client-Programm durchzureichen. Anwendung findet diese Technik, um den Namen des MQTT-Topics und des zu verwendenden GPIO-Pins einfach im Startskript angeben und ändern zu können. Alle Dienste sind zudem so konfiguriert, dass sie sich zehn Sekunden nach fehlerhafter Beendigung automatisch neu starten. Damit wird sichergestellt, dass die Clients auch bei noch nicht vorhandener Erreichbarkeit des MQTT-Brokers oder anderer temporärer Fehlerzustände zeitnah gestartet werden, sobald alle Voraussetzungen erfüllt sind. Analog zum Rezept *platform-init.bb* aus Abschnitt 5.3 müssen alle Dienste durch Erben der `systemd`-Klasse sowie Hinzufügen des Paketnamens zu `SYSTEMD_PACKAGES` und der Dienstdateien zu `SYSTEMD_SERVICE_${PN}` bekanntgemacht werden. Die Quelltexte der Client-Programme selbst werden im Rezept noch finalisiert. Das bedeutet, dass die Platzhalter für Serveradresse, Port sowie Nutzernamen und Passwort der MQTT-Verbindung erst im `do_compile`-Schritt von `bitbake` durch die gewünschten Werte ersetzt werden. Diese sind im Rezept durch Variablen festgelegt und werden mit dem Programm `sed` über reguläre Ausdrücke ersetzt. Dadurch entsteht der Vorteil, Änderungen dieser Parameter, bspw. durch Verwendung eines anderen Brokers, zentral einmalig durchführen zu können und somit eine bessere Wartbarkeit zu gewährleisten. Schließlich erfolgt die Installation der Client-Programme und Dienstdateien an ihre Zielorte im `do_install`-Schritt des Rezepts. Der grundlegende Aufbau der Client-Programme wird zunächst am übersichtlichsten Quelltext, dem des Zeitsynchronisationsclients *datetime-mqtt.py*, beschrieben. Zusätzlich sind die beiden verwendeten Grundstrukturen in Abbildung 5.4 und Abbildung 5.5 visualisiert. Eintrittspunkt des Programms ist der Aufruf der Funktion `init()` am Ende des Quelltextes. Im Fall dieses Clients wird darin nur eine weitere Funktion, `createMQTTConnection()`, aufgerufen. Diese wiederum instantiiert ein Objekt der MQTT-Client-Klasse, welche vom Modul *python-paho-mqtt* bereitgestellt wird. Anschließend werden dem Objekt einige Callback-Funktionen zugewiesen. Sie ermöglichen es, auf Ereignisse, hier die erfolgreiche Verbindung zum Broker, Abbruch der Verbindung und eingehende Nachrichten zu reagieren. Abschließend werden Nutzernamen und Passwort des Clients für den Broker festgelegt und der Verbindungsaufbau initiiert. Danach wird der MQTT-Client mit der Methode `client.loop_forever()` in eine Endlosschleife versetzt. Sobald die Verbindung zum Broker hergestellt wurde, wird im `on_connect()`-Callback eine Meldung ausgegeben und das Topic `info/datetime` abonniert. Mit dem Rückgabewert `4` werden fehlerhafte Authentifizierungsparameter signalisiert. Infolgedessen wird ein neuer Verbindungsversuch unternommen, da solche Fehler trotz richtiger Angaben in der Startphase des Brokers vorkommen können. Bei Verbindungsabbrüchen gibt dieser Client lediglich eine Nachricht aus. Er ist nur Subscriber, deshalb sind keine weiteren Funktionen zum Veröffentlichen von Daten vorhanden. Eingehende Nachrichten werden konventionsgemäß zunächst nach dem Topic-Namen `info/datetime` gefiltert, auch wenn hier nur ein Topic abonniert wurde. Der Wert wird als Ganzzahl interpretiert und mit einem Referenzdatum aus der Startphase der Arbeit auf Plausibilität geprüft. Letztendlich wird die Angabe von Millisekunden in Sekunden umgewandelt und per `time.clock_settime()` als Systemzeit sowie per Aufruf des Linux-Kommandos `hwclock -w` in die RTC

5. Umsetzung

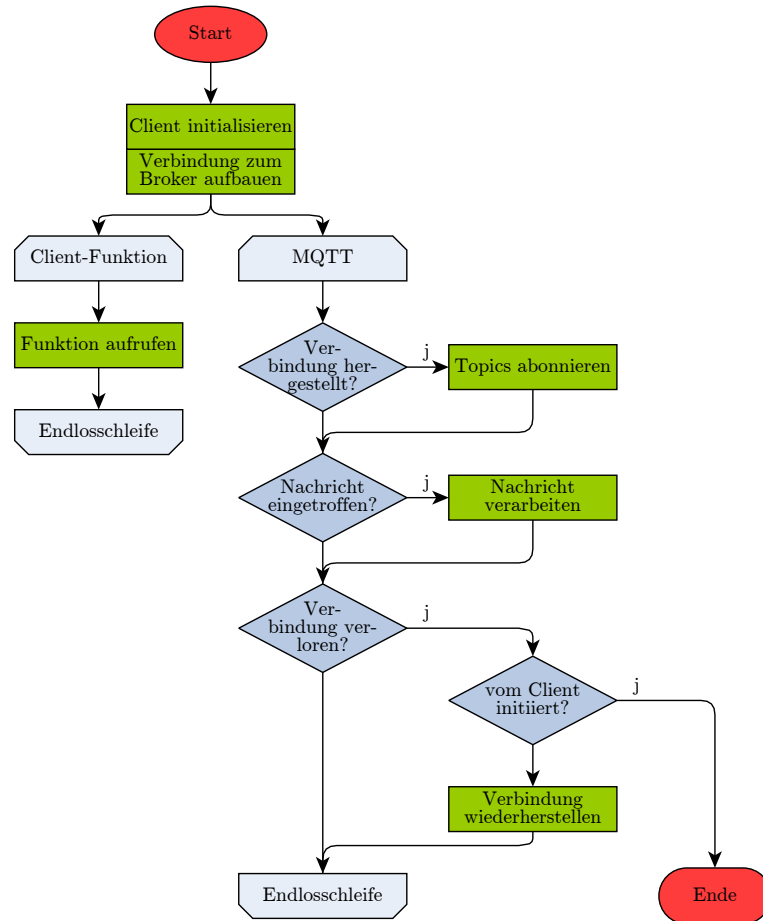


Abbildung 5.4.: MQTT-Client-Architektur mit Sendeschleife

übernommen. Danach ist die Zeitsynchronisation beendet, der Client trennt mit der Methode `disconnect()` die Verbindung, woraufhin dieser auch die Endlosschleife verlässt und das Programm ordnungsgemäß beendet. Damit entspricht dieser Client der in Abbildung 5.5 dargestellten Variante, jedoch ohne die Timer-Funktionen. Der Client zum Auswerten der GPIO-Pins, `gpio-mqtt.py`, hat zusätzlich zur eben beschriebenen, grundlegenden Funktionsweise eines MQTT-Clients die Möglichkeit, Daten auf einem Topic zu veröffentlichen. Dazu erfolgt in der `init()`-Funktion zunächst die Auswertung der beim Aufruf des Programms übergebenen Parameter. Aufgrund der Verwendung von Service Templates mit Instance Identifier kann hierbei nur eine zusammenhängende Zeichenkette ohne Leerzeichen verwendet werden. Deshalb werden die beiden benötigten Parameter, Topic und Bezeichnung des GPIO-Pins, per Komma separiert erwartet und dementsprechend ausgewertet. Der Topic-Parameter wird dabei an `sensors/gpio/` angehängt, sodass alle GPIO-Werte denselben Topic-Pfad besitzen. Bei der Pin-Bezeichnung muss das Schema der verwendeten `Adafruit_BBIO`-Bibliothek befolgt werden. D. h., es muss entweder die Bezeichnung aus Anschluss und Pinnummer, bspw. `P8_10` oder der Name, bspw.

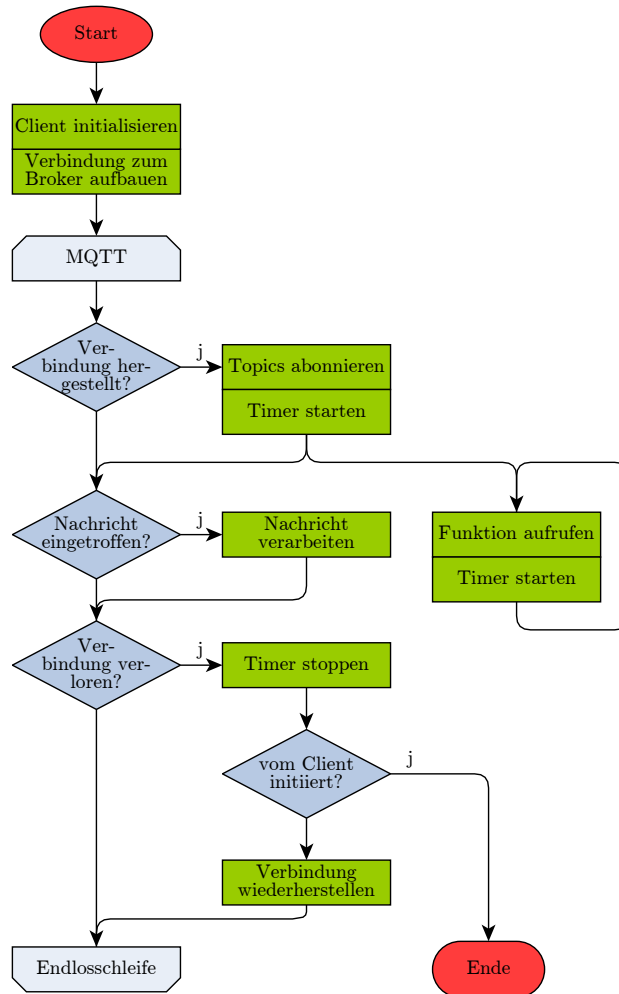


Abbildung 5.5.: MQTT-Client-Architektur mit Timer

GPIO0_26, angegeben werden. Mit dem Aufruf von *GPIO.setup()* wird der Pin als Eingang konfiguriert und der Pull-Up aktiviert. Das wäre ohne die zuvor gemachten Modifikationen am Device-Tree und den Treibern nicht möglich gewesen. Im Gegensatz zu *datetime-mqtt.py* wird hier in *createMQTTConnection()* statt *client.loop_forever()* die Methode *client.loop_start()* genutzt. Der Client spiegelt somit die zweite, in Abbildung 5.4 abgebildete, Architektur wider. Das liegt in der Verwendung eines blockierenden Haupt-Threads in diesem Programm begründet. Während *client.loop_forever()* selbst blockierend ist, startet *client.loop_start()* einen eigenen Hintergrund-Thread, der die Verarbeitung von Netzwerkeignissen gleichermaßen übernimmt. Der Haupt-Thread des Programms wird in der Funktion *publishData()* durch die Endlosschleife blockiert. Diese liest den Status des GPIO-Pins aus und veröffentlicht ihn auf dem Topic. Anschließend wird eine bestimmte Zeit auf eine Flanke im Signal des Pins gewartet, wodurch eine erneute Statusübermittlung vor dem üblichen Intervall eingeleitet werden würde. Nach Ablauf der Wartezeit, welche

5. Umsetzung

in der globalen Variable *PUBLISH_INTERVAL* auf zehn Sekunden festgelegt ist, wird der Status auch ohne Änderung gesendet. Letzter Aufruf innerhalb der Funktion *init()* ist *client.loop_stop()*, was den MQTT-Hintergrund-Thread beendet, sobald die Endlosschleife in *publishData()* unterbrochen wird.

Der dritte und damit letzte MQTT-Client ist dafür zuständig, den BME680-Sensor einzubinden. Das Programm erwartet nur einen Parameter zur Festlegung der Publisher-Topics. Ähnlich zum GPIO-Client wird dieser jeweils an die Pfade beginnend mit *sensors/* und der Bezeichnung des jeweiligen Messwerts, bspw. *temperature/*, angehängt. In der *init()*-Funktion dieses Clients wird zudem der I2C-Bus mit Index 2 über das Modul *smbus* initialisiert. Unter Angabe der Busadresse und der Referenz auf den I2C-Bus wird die BME680-Bibliothek von Pimoroni anschließend eingerichtet. Die Busadresse der Sensoren ist standardmäßig *0x77*. Durch Verbinden des bei I2C unbenutzten Pins *Serial Data Out (SDO)* mit Masse kann sie auf *0x76* geändert werden. Es folgt das Setzen von Sensorparametern, wie *Oversampling*, Filter und Konfiguration des Heizelements. Die zuletzt genannte Konfiguration sollte nur bei instabilen Messergebnissen angepasst werden. Das Oversampling wird hierbei 16-fach und somit maximal gewählt, um kurzzeitige Störungen auszugleichen. Mit *sensor.set_filter(bme680.FILTER_SIZE_7)* wird zusätzlich ein Filter aktiviert, der Signalspitzen aus den Messwerten herausfiltert [68]. Letzter Unterschied zum Quelltext des GPIO-Clients ist die Funktion *publishData()*. Hier ist zunächst eine Aufwärmphase für die Luftqualitätsmessung enthalten. In der globalen Variable *PREHEAT_STEPS* wird die Anzahl von Schritten und damit die ungefähre Sekundendauer dieser Phase angegeben. 600 ist dabei ein Erfahrungswert, nachdem sich die Widerstandswerte für gewöhnlich stabilisiert haben. Je nach Umgebungstemperatur kann er allerdings variieren. Nach der Aufwärmphase folgt die Endlosschleife zur Erfassung und Übermittlung der Messwerte. Demnach folgt dieser Client auch der Schleifenarchitektur aus Abbildung 5.4. Die Messung an sich erfolgt hierbei in Abständen von einer Sekunde, um die Stabilität des beheizten Luftqualitätssensors zu gewährleisten. Die Veröffentlichung hingegen findet nur in jedem zehnten Durchlauf, also ca. alle zehn Sekunden, statt. Dabei werden die Messwerte jedoch nicht direkt versendet, sondern der Mittelwert der letzten 60 Messungen. Das hilft, zusätzlich zum Oversampling und Filter der Sensorbibliothek, Schwankungen zu glätten und somit ein gleichmäßigeres Messergebnis zu erreichen. Die verwendeten Sensoren haben eine Messgenauigkeit von ± 1 °C bei der Temperatur, 3 % bei der Luftfeuchtigkeit und ± 1 hPa beim Luftdruck [3]. Die Messung der Luftqualität ist von Luftbewegungen abhängig, da der Sensor durch diese erst mit der Umgebungsluft in Berührung kommt. Die Temperatur- und Luftfeuchtigkeitsmessung hat zudem eine hohe Empfindlichkeit gegenüber Luftströmungen, weshalb es wichtig ist, den Sensor vor direkter Zugluft aus den Klimageräten zu schützen. Infolgedessen werden die Sensoren mit einem Papierröhrchen versehen. Einerseits macht das die Luftqualitätssmessung zwar etwas ungenauer, da Luftbewegungen den Sensor nicht mehr so gut erreichen, andererseits wird die Temperaturmessung aber stabiler. Da die Temperatur für diese Anwendung höher priorisierte Parameter ist, wird der Kompromiss auf Kosten der ungenaueren Luftqualitätssmessung eingegangen.

5.5. Systemaktualisierung der Sensorknoten

Die OTA-Updates mit Mender lassen sich, wie bereits in Abschnitt 4.5 erwähnt, sehr gut in Yocto-Projekte integrieren. Für die Verwendung im lokalen Netzwerk wird im Rahmen dieser Arbeit die Demonstrationsumgebung verwendet. Der Mender-Client wird durch Hinzufügen der Layer *meta-mender-core* und *meta-mender-demo* zum Yocto-Projekt eingebunden. Der demo-Layer darf nicht für den Produktiveinsatz verwendet werden, da dort aus Sicherheitsgründen neue Zertifikate und Schlüssel notwendig sind. Nach dem Hinzufügen des Layer müssen in *conf/local.conf* des Projekts noch einige Variablen gesetzt werden. *MENDER_ARTIFACT_NAME* legt den Namen des Artefakts fest, welcher eindeutig sein muss, da gleichnamige Artefakte vom Client als identisch angesehen und somit nicht aktualisiert werden. An *INHERIT* wird *mender-full* angehängt, um den Update-Client zu installieren. Die Einträge von *DISTRO_FEATURES_append* bis *VIRTUAL-RUNTIME_initscripts* dienen der Aktivierung des Init-Systems *systemd*, welches für Mender benötigt wird. Mit *ARTIFACTIMG_FSTYPE* wird der Dateisystemtyp auf *ext4* festgelegt. Die Gesamtgröße des verfügbaren Speicherplatzes für die Systempartitionen wird mit *MENDER_STORAGE_TOTAL_SIZE_MB* angegeben. Daraus wird die Aufteilung der einzelnen Partitionen bei der Erstellung automatisch errechnet. Die Adresse des Mender-Demo-Servers wird in der Variable *MENDER_DEMO_HOST_IP_ADDRESS* definiert. Beim Produktiveinsatz ohne den demo-Layer muss dafür stattdessen *MENDER_SERVER_URL* genutzt werden. Zusätzlich existieren Variablen, mit denen beispielsweise die Intervalle für Update-Prüfungen und den Report von Inventardaten verändert werden können. Diese sind mit Standardwerten befüllt und in der Mender-Dokumentation genauer erläutert [48].

Der Layer *meta-mender-demo* fügt zwei *systemd*-Konfigurationsdateien für drahtgebundene Netzwerkschnittstellen zum Betriebssystem hinzu, um einen reibungslosen Demo-Betrieb zu gewährleisten. Da die Konfiguration in diesem Fall durch das Startskript und *dhcpcd* ausgeführt wird, muss die beim BeagleCore zutreffende Datei *eth.network* entfernt werden. Das geschieht mittels eines Append-Files in der Kategorie *recipes-core* im eigens erstellten Yocto-Layer. Angehängt an den *do_install*-Schritt wird die Datei aus dem entsprechenden Pfad im Zielsystem gelöscht und steht somit nicht mehr in Konflikt mit dem Startskript. Des Weiteren besteht die Möglichkeit, den Inventarreport der Mender-Clients zu erweitern, sodass in der Administrationsoberfläche genauere Informationen von den Clients einsehbar sind. Dazu wird beispielhaft ein zusätzliches Skript angelegt, welches einige Kommandos ausführt und die Ausgaben nach dem von Mender vorgegebenen Muster erzeugt. Es trägt den Namen *mender-inventory-custom* und liefert Informationen über das aktuelle Datum sowie die Uhrzeit, angemeldete Benutzer, Startparameter des Linux-Kernels und die Betriebszeit. Durch ein Append-File an das Mender-Rezept in *recipes-mender* wird es mit den entsprechenden Zugriffsrechten in den *inventory*-Ordner des Mender-Clients installiert. Interessant ist beim Mender-Client außerdem, dass er zustandsbasiert arbeitet. Es gibt insgesamt neun verschiedene Zustände, deren Übergänge durch sogenannte *State scripts* beeinflusst werden können. Damit ist

5. Umsetzung

es möglich, bspw. beim Übergang zwischen *Sync* und *Update*, also vor einer Aktualisierung, eine Bestätigung vom Nutzer einzufordern.

Der Mender-Server setzt zunächst die *Docker Engine* und *Docker Compose* voraus, da er in Form von Docker Containern ausgeliefert wird. Um den Server bereitzustellen, hier auf dem Desktop-PC mit Debian-Betriebssystem, muss der Quelltext heruntergeladen werden. Durch Ausführen des enthaltenen Skripts *up* wird das Herunterladen der eigentlichen Container initiiert. Gleichermäßen existiert das Skript *stop* zum Beenden der Serverdienste und *reset* zum Zurücksetzen des Mender-Servers. Nachdem die verschiedenen Komponenten bereit sind, werden sie gestartet und zusätzlich Einträge in der Datei */etc/hosts* hinzugefügt, um alle Dienste auf die Docker-Netzwerkstruktur anzupassen. Sobald die Administrationsoberfläche unter *https://localhost/* erreichbar und das Demo-Zertifikat des *Mender Gateways* akzeptiert wurde, erscheint der Login-Bildschirm. Nun kann durch den Aufruf des *mender-useradm* entsprechend der Mender-Dokumentation ein Benutzer angelegt werden [47]. Mit den für *username* und *password* angegebenen Werten kann anschließend der Login durchgeführt werden. Durch das Einbinden der Mender-Layer wird bei jedem Build-Vorgang mit bitbake ein Update-Artefakt für Mender erstellt. Diese Artefakte tragen die Dateiendung *.mender* und können in der Web-Oberfläche unter *Artifacts* hochgeladen werden. Sofern sich bereits Mender-Clients beim Server gemeldet haben, können diese im Tab *Pending* akzeptiert werden. Alternativ besteht auch die Möglichkeit, Clients im Vorhinein per Authentifizierungsdatei zuzulassen. Anschließend werden die Geräte im Tab *Device groups* aufgelistet. Per Mausklick auf die Einträge können z. B. die Inventardaten eingesehen werden, sofern der Client bereits einen Report gesendet hat. Außerdem ist es in dieser Ansicht möglich, die Clients zu gruppieren und zu filtern. Das ist hilfreich, um die drahtlos angebotenen Sensorknoten als Gruppe, unabhängig vom BeagleCore, aktualisieren zu können. Somit lassen sich Verbindungsabbrüche durch den Neustart des Mesh Portals als Bindeglied zum LAN vermeiden. Diese Zustände würden während Aktualisierungsvorgängen zu Fehlern führen, in Folge Updates scheitern lassen und ggf. einen *Rollback* auf den vorherigen Zustand auslösen. Hochgeladene Artefakte können für einzelne Geräte in der *Devices*-Übersicht oder für Gruppen unter *Deployments* zur Auslieferung freigegeben werden. Daraufhin starten die Clients das Herunterladen der Aktualisierung und entpacken diese, ohne zusätzlichen Speicher zu belegen, in die inaktive Systempartition. Der Fortschritt des Deployments kann während des Vorgangs in der Web-Oberfläche verfolgt werden. Nach erfolgreicher Installation wird ein Neustart in das aktualisierte System ausgeführt. Wenn keine Probleme auftreten, meldet sich der Client beim Server zurück und der Update-Prozess gilt als erfolgreich beendet. Sollte hingegen ein Fehler auftreten, so wird der Ausgangszustand durch einen Rollback auf das vorherige System wiederhergestellt und eine entsprechende Statusmeldung an den Server gesendet. Das Ergebnis eines abgeschlossenen Deployments bleibt im Tab *Finished* der Ansicht *Deployments* einsehbar.

5.6. Nutzerschnittstelle

Für die Umsetzung der Bedienoberfläche ist Node-RED vorgesehen. Es handelt sich dabei um eine serverseitigen JavaScript-Umgebung, die auf *Node.js* basiert. Zur Installation auf dem Debian-System des Desktop-PCs muss zunächst die *NodeSource*-Paketquelle eingebunden werden [55]. Anschließend wird über die Paketverwaltung *apt* das Paket *nodejs* installiert, welches den *node package manager (npm)* enthält. Mit npm ist es nun möglich, das Modul *node-red* sowie die Erweiterung *node-red-dashboard* zu installieren. Im Anschluss kann node-red mit dem gleichnamigen Kommando gestartet werden. Um es direkt beim Systemstart bzw. als Hintergrundprozess starten zu können, gibt es verschiedene Möglichkeiten. Entweder wird *PM2*, ein Prozessmanager, verwendet oder ein *systemd*-Dienst erstellt. An dieser Stelle wird die letztgenannte Variante gewählt, da hierfür bereits alle Abhängigkeiten erfüllt sind. Dazu können bestehende Dienstdateien, bspw. aus den Node-RED-Quellen für den *Raspberry Pi*, angepasst und genutzt werden [56]. Vor dem Starten von Node-RED wird die Konfigurationsdatei *.node-red/settings.js* im Benutzerverzeichnis angepasst. Änderungen umfassen den Dateinamen des Flows im Attribut *flowFile*, die URL des Editors in *httpAdminRoot*, die URL des Dashboards in *ui* sowie Authentifizierungsdaten für Editor und Client-API in *adminAuth* bzw. *httpNodeAuth* und *httpStaticAuth*. Die Passwörter werden dabei mit *bcrypt* verschlüsselt [57]. Nach dem Starten von Node-RED ist die Web-Oberfläche des Editors unter *http://127.0.0.1:1880/admin* erreichbar. Darin können Flows grafisch erstellt werden, um gewünschte Funktionen zu realisieren. Es existieren viele vorgegebene Nodes, die einfach per *Drag-and-Drop* in den Flow hineingezogen werden können. Dennoch kommt die Auswahl bei speziellen Anforderungen an ihre Grenzen, weshalb mit dem Template-Node eigene Funktionen implementiert werden müssen. Die Flows, welche die Funktionalitäten für die Umsetzung dieser Arbeit enthalten, sind aufgegliedert in *Informationen*, *Klimageräte*, *Sensorwerte* und je eine Unterkategorie pro gesteuertem Raum. Die grafischen Elemente dieser Flows werden von der Dashboard-Erweiterung auf zwei Tabs, *Anzeige* und *Steuerung*, des Dashboards dargestellt. In Abbildung 5.6 und Abbildung 5.7 sind Bildschirmfotos beider Ansichten abgebildet. Sie folgen funktional ihrer Benennung und sind wiederum untergliedert in Gruppen. Beispiele aus dem Anzeige-Tab sind die Gruppen *Klimageräte*, *Sensor 1* und *Verlauf*. Flows enthalten in der Regel Eingabedaten, hier über MQTT-Inputs, wandeln diese ggf. um, reagieren darauf und geben eine Rückmeldung, bspw. über Dashboard-Elemente oder MQTT-Outputs. Adresse und Zugangsdaten des MQTT-Brokers sind dabei global für alle Flows festgelegt. Die Sensordaten werden direkt in *Gauge*-Elemente zur Anzeige und in regelmäßigen Abständen in die Verlaufsdiagramme geleitet. Die Verwendung aller ankommenden Messwerte in den Diagrammen führt bei steigender Datenmenge zu immer trägerem Verhalten der Dashboard-Oberfläche. Zur Behebung dieses Problems wird deshalb die Datenzufuhr der Verlaufsdiagramme auf einen Wert pro Minute reduziert. Die fehlende Auflösung hat dabei keine Nachteile, da die Diagramme durch ihre Skalierung bereits vorher nicht alle Datenpunkte anzeigen konnten. Zusätzlich findet eine



Abbildung 5.6.: Anzeige-Tab der Node-RED-Oberfläche

Vorverarbeitung der Sensordaten statt, sodass die Dezimalstellen einheitlich und zweckmäßig begrenzt werden. Die Statusanzeige der Klimageräte in beiden Tabs ist in Template-Nodes implementiert. Sie gibt Auskunft über den Verbindungsstatus, An/Aus, Betriebsmodus, Filterwarnung, Sollwert, Ansaugtemperatur sowie Gebläsestufe und -richtung der einzelnen Innengeräte. Die Informationen hierzu werden vom *daikinMQTT.py*-Client auf den MQTT-Topics veröffentlicht und über die MQTT-Inputs der Flows empfangen. In den Template-Nodes erfolgt, unter Nutzung von *AngularJS*, die Umwandlung in *Hypertext Markup Language (HTML)* und *Cascading Style Sheets (CSS)*. Die verwendeten Symbole stammen aus dem *FontAwesome*-Paket, welches bereits Teil von Node-RED Dashboard ist. Mit *AngularJS* werden dynamisch CSS-Klassen zugewiesen, sodass aktive Symbole farbig und inaktive grau dargestellt werden. Alle Bezeichnungen der Klimageräte stammen aus der Konfiguration des iTC und sind mit dem *getProperties()*-Befehl der HTTP-API abrufbar. Diese Informationen werden jedoch nicht über MQTT veröffentlicht, da sie sich normalerweise nur einmal bei der Einrichtung der Anlage ändern. Daher sind sie im Template-Node als Dictionary mit dem Namen *unit_name* lokal hinterlegt. Im Steuerungs-Tab wird das Prinzip dieses Template-Nodes wiederverwendet, jedoch in reduzierter Form, sodass nur die pro Raum gesteuerten Geräte angezeigt werden. Unterhalb der Statusanzeigen befinden sich die Bedienelemente zum Ein-/Ausschalten, zum Einstellen des Temperatursollwerts und zum Zurücksetzen der Gerätekonfiguration auf Standardwerte. Bei der Konfiguration des Bedienelements für den Sollwert muss jedoch die Begrenzung der Klimaanlage beachtet werden. Da dem KI-Agenten Spielraum um die Sollwerte herum gegeben werden soll, muss der

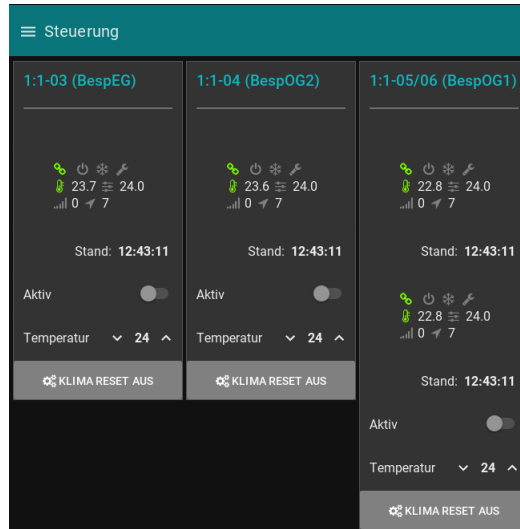


Abbildung 5.7.: Steuerungs-Tab der Node-RED-Oberfläche

Wertebereich entsprechend kleiner ausfallen, damit keine ungültigen Steuerbefehle außerhalb dieser Begrenzung versendet werden können. Außerdem ist die Einstellung so ausgelegt, dass der Ausgang des Steuerelements einer Verzögerung unterliegt. Damit wird der eingestellte Wert erst zehn Sekunden nach der letzten Änderung versendet. Das hilft, die Übertragung von kurzzeitigen Zwischenschritten zu vermeiden. Die Verlaufsdiagramme zeigen pro Parameter jeweils die Werte beider Sensorknoten in verschiedenen Farben an. Dabei entscheidet der zuerst eintreffende Wert nach dem Start von Node-RED, welches Topic welcher Farbe zugeordnet wird. Die Farben wurden aus Gründen der Übersichtlichkeit in die Gruppen des Dashboards als horizontale Linie eingebaut. Alternativ kann die Zuordnung nur in der Legende, welche durch Bewegen des Mauszeigers über die Diagramme angezeigt wird, eingesehen werden. Da die Zuordnung vom zuerst eintreffenden Messwert abhängt, muss die Reihenfolge beim Starten der Sensorknoten bzw. beim Starten von Node-RED berücksichtigt werden.

5.7. Datenaufzeichnung

Zur Erfassung der Demonstrationsdaten zum Vortrainieren des KI-Agenten und zur Auswertung der Ergebnisse ist der Logging-Client in *loggerMQTT.py* zuständig. Der grundlegende Aufbau entspricht dem der Timer-Architektur in Abbildung 5.5, welche zudem in Abschnitt 5.8 detaillierter beschrieben ist. Das Programm erwartet den Dateinamen für die Aufzeichnungen sowie den Sensorindex des betreffenden Sensors als Parameter. In der Funktion *check_log()* wird analog zum KI-Client überprüft, ob alle benötigten Daten vorliegen und aktuell sind. Ist das der Fall, so wird mit *log()* ein Schritt in die angegebene Log-Datei, hier auch *Record-File* genannt, aufgezeichnet. Der Intervall zwischen zwei Schritten ist in *LOG_INTERVAL* auf 580 Sekunden fest-

gelegt. Dieser Intervall sollte etwas kürzer sein, als der Lernintervall des KI-Agenten. Das gewährleistet, dass die Aufzeichnung vor der nächsten Aktion des Agenten stattfindet. Deshalb wird hier der doppelte Abstand zweier Statusmeldungen des Clients *daikinMQTT.py* als Differenz gewählt. Bei Erkennung einer Statusänderung des überwachten Innengeräts wird diese durch die Variable *unit_configuration_changed* markiert und mittels *reset_check_log()* direkt ein neuer Log-Schritt eingeleitet. Aufgezeichnet werden sämtliche Sensordaten, auch die in diesem Moment nicht relevanten, um für spätere Verwendungen alle Möglichkeiten beizubehalten. Zusätzlich zu den Messwerten der Sensoren werden die aktuellen Sollwerte, der Betriebsmodus des Innengeräts sowie eine JSON-Zeichenkette des kompletten Gerätestatus aufgezeichnet. Im ersten Schritt nach Programmstart oder einer Statusänderung ist *step* gleich *-1*, weshalb noch keine Aktion durchgeführt wurde und somit keine Aufzeichnung stattfinden kann. In allen weiteren Schritten wird die gewählte Aktion rückwärts, also durch Auswertung der Gerätekonfiguration, aus *unit_configuration* ermittelt. Dabei wird in bestimmten Fällen mit Hilfe des Dictionaries *ACTIONS* die Nummer der Aktion bestimmt. Sollte keiner der Einträge auf die aktuelle Konfiguration passen, so wird das mit dem Wert *-1* gekennzeichnet. Eine direkte Auswertung des *action*-Topics der Innengeräte würde unter Umständen zu Inkonsistenzen führen, da sich die Konfiguration von außen bspw. durch das Bedienteil verändern lässt und die übermittelte Aktion dadurch nicht mehr zutreffend sein könnte. Im Anschluss wird der gesammelte Datensatz als Meldung ausgegeben sowie durch die Hilfsfunktion *append_file()*, versehen mit Datum und Uhrzeit, in die angegebene Datei geschrieben. Die Funktion *log()* endet mit der Übernahme der ggf. neuen Sollwerte und dem Inkrementieren des Schrittzählers *step*.

Das Datenformat des Record-Files folgt einem eigens definierten Muster. Die Aufzeichnung im *Comma-Separated Values (CSV)*-Format wurde getestet, aber die JSON-Zeichenketten bereiteten beim Import Probleme. Es kam zu Konflikten zwischen den im JSON-Format enthaltenen Kommata und denen des CSV-Formats. Deshalb werden die Daten nun, getrennt durch Pipe-Zeichen, jeweils auf einer Zeile pro Datensatz, gespeichert. Dabei ist jeder Datensatz wie folgt aufgebaut:

- Datum,
- Uhrzeit,
- Ausgangszustand,
- Index der ausgeführten Aktion und
- Folgezustand.

Die Zustände umfassen jeweils mehrere Felder für:

- Temperaturmesswerte und -sollwert,
- Luftfeuchtheitsmesswerte und -sollwert,
- Luftqualitätsmesswerte,
- Luftdruckmesswerte,
- Status der GPIO-Sensoren,
- Betriebsmodus des Klimageräts und
- JSON-Status des Klimageräts.

Durch diesen Aufbau werden zwar einige Informationen doppelt gespeichert, da der Folgezustand eines Satzes dem Ausgangszustand des nächsten entspricht, allerdings wird dadurch der Import vereinfacht. Die Datei kann zeilenweise eingelesen werden, ohne Daten zeilenübergreifend vorhalten zu müssen. Zudem verhindert dieses Vorgehen, dass ein fehlerhafter Datensatz die Auswertung seines Vorgängers oder Nachfolgers beeinträchtigt.

In der Datei *loggerMQTT_global.py* befindet sich die Implementierung eines weiteren Aufzeichnungs-Clients. Dieser dient der Erfassung und Aufzeichnung sämtlicher anfallenden Daten. Dazu werden die eintreffenden Statusmeldungen, entsprechend der Adresse des Innengeräts, in Dictionaries abgelegt. Ebenso werden die Sollwerte für die in *UNIT_ADDRESSES* spezifizierten Adressen auf diese Weise zwischengespeichert. Die Speicherung der Sensordaten erfolgt analog zur Implementierung aus *loggerMQTT.py* in einem Dictionary mit dem jeweiligen Topic-Namen als Schlüssel. Sobald sich der Status bei einem der Innengeräte ändert, wird durch Rücksetzen des *step*-Zählers unverzüglich ein neuer Log-Abschnitt begonnen. Das hat zur Folge, dass mehrere, zeitlich geringfügig versetzte Änderungen zu einer längeren Unterbrechung der Erfassung führen können. In der Praxis ändern sich meist alle Innengeräte einer Zone gleichzeitig. Selten kommen einzelne, zeitlich aber ausreichend distanzierete Änderungen vor, sodass dies kein großes Problem darstellt. In der *log()*-Funktion werden zusätzlich Daten zum Wetter und eine Wettervorhersage von der *OpenWeatherMap* abgerufen. Das geschieht mit dem bereits für die HTTP-API genutzten Python-Modul *requests*. Beim Aufruf der URL werden die geografischen Koordinaten, das gewünschte Einheitensystem sowie ein Schlüssel und optional Proxy-Einstellungen angegeben. Um einen Zugangsschlüssel für diese API zu erhalten, muss eine Registrierung bei dem Dienst erfolgen. Die Antworten befinden sich, wie die Statusinformationen der Klimageräte, im JSON-Format, sodass sie direkt in das Zustands-Tupel übernommen werden. Statusinformationen erfasst dieser Client nicht nur von einem, sondern von allen verfügbaren Klimageräten. Deshalb ist auch die Rückwärtssuche nach der gewählten Aktion und die Sollwertübernahme für jedes der in *UNIT_ADDRESSES* angegebenen Geräte durchzuführen. Ähnlich zum

einfachen Logging-Client werden die gesammelten Daten als Meldung ausgegeben und in eine Datei geschrieben. Dabei umfasst ein Datensatz wieder die gleichen Bestandteile, wobei die Zustände nun jeweils elf Felder beinhalten. Der Client *loggerMQTT-global.py* ist dazu gedacht, Demonstrationsdaten in größerem Umfang als Grundlage für spätere Einsatzszenarien aufzuzeichnen und somit weitere Erkenntnisse sammeln zu können.

5.8. KI-Agent

Das Programm, welches letztendlich die KI implementiert, baut auf das Python-Modul TensorFlow auf. TensorFlow besitzt eine Abhängigkeit zu TensorFlow, Version 1.5 oder höher. Dieses muss zuvor mit dem Kommando *pip install tensorflow* installiert werden. Um TensorFlow mit Beschleunigung der Grafikkarte zu nutzen, kann stattdessen das Paket *tensorflow-gpu* verwendet werden. Das funktioniert jedoch nur mit einer passenden Nvidia-Grafikkarte, dem proprietären Treiber, dem *Compute Unified Device Architecture (CUDA)*-Toolkit sowie dem *cuDNN-Software Development Kit (SDK)*. Für diesen Fall ist jedoch keine solche Beschleunigung vonnöten, weshalb die erste Variante Anwendung findet. Anschließend wird das *tensorforce*-Modul selbst und die ebenfalls benötigte MQTT-Bibliothek *paho-mqtt* mit *pip* installiert. Durch die modulare Struktur von TensorFlow wird es möglich, den KI-Agenten losgelöst von Simulationsumgebungen wie *OpenAI Gym* zu verwenden. Dabei wird die Umgebung des Agenten von den Sensorknoten erfasst und kann durch Befehle an die Klimasteuerung beeinflusst werden. Der *Runner*, welcher die Schnittstelle des Agenten zu seiner Umgebung verkörpert, wird im KI-Client-Programm implementiert. In *tforceMQTT_DQFD.py* ist die Variante für den Einzelbetrieb und in *tforceMQTT_DQFD-global.py* diejenige zur parallelen Steuerung mehrere Klimageräte enthalten. Die Grundidee dazu stammt aus dem Beispiel der Projektseite im Abschnitt *Create and use agents* [108] und ist in Abbildung 5.8 dargestellt. Es erfolgt zuerst die Erstellung des Agenten, inklusive Spezifikation des künstlichen neuronalen Netzes im Attribut *network*. Danach wird das Pretraining des DQfD-Agents mit Demonstrationsdaten durchgeführt. Anschließend geht das Programm in die Hauptschleife über. In einem Durchlauf werden die benötigten Sensor- und Statusdaten gesammelt und auf deren Grundlage ein Lernschritt ausgeführt. Abschließend wird eine bestimmte Zeit gewartet, bevor der nächste Durchlauf beginnt. Ein Lernschritt besteht dabei aus den folgenden Komponenten:

- Zustandserfassung,
- Aktionswahl,
- Ausführung der Aktion und
- Beobachtung des Returns.

5. Umsetzung

Im Quelltext von *tforceMQTT_DQFD.py* ist diese Struktur nicht auf den ersten Blick erkennbar, da sie in die MQTT-Client-Funktionen eingebettet ist. Diese folgen dem in Abbildung 5.5 abgebildeten Muster.

Nach Programmstart erfolgt zuerst die Initialisierung des KI-Agenten. Dazu wird, je nach gewünschtem Lernalgorithmus, eine der Agent-Klassen der TensorForce-Bibliothek instantiiert. Hier findet DQfD Verwendung, weshalb ein *DQFDAgent* erstellt und der globalen Variable *agent* zugewiesen wird. Die Parameter *states*, *actions*, *batched_observe* und *batching_capacity* stammen von der übergeordneten Klasse *Agent* und sind somit für alle erbenenden Klassen gleichermaßen gültig. Der Zustandsraum wird als Dictionary in *states* angegeben. Dieses ist durch die Variable *TF_STATES* in *Shared_Definitions.py* definiert und wird importiert. Zum Testen verschiedener Zustandsräume kann durch den Wert des Schlüssels *shape* das „Aussehen“, d. h. die Dimensionierung, angepasst werden. Der Datentyp selbst wird mit dem Schlüssel *type* im Dictionary hinterlegt. Hierbei sind *float* und *int* für Fließkomma- bzw. Ganzzahlen gängige Werte. Der Aktionsraum wird ebenfalls als Dictionary mit dem Parameter *actions* übergeben. Die Definition der zugewiesenen Variable *TF_ACTIONS* findet sich wieder in der gemeinsamen Definitionsdatei. Mit den Schlüsseln *type* und *shape* können auch hier Datentyp und -format der Aktionsausgabe des KI-Agenten festgelegt werden. Zusätzlich kann die Anzahl beim Typ *int* mit *num_actions* und der Wertebereich beim Typ *float* mit *min_value* sowie *max_value* angegeben werden. Es ist auch möglich, mehrere Aktionsdefinitionen, mit beliebigen Schlüsseln in einem übergeordneten Dictionary zusammenzufassen. Somit können verschiedene Aktionstypen in einem Schritt ausgegeben werden. Bei der Anwendung des DQfD-Agenten werden Aktionen mit dem Typ *float* jedoch nicht unterstützt. Der Einsatz dieses Aktionstyps, bspw. um den Temperatursollwert als variable Ausgabegröße darstellen zu können, bleibt damit verwehrt. Durch die Parameter *batched_observe* und *batching_capacity* kann auf die Verarbeitung der Belohnung Einfluss genommen werden. Die erstgenannte Option bestimmt, ob eine Stapelverarbeitung der gewonnenen Erfahrungen durchgeführt wird und letztere, wie viele Erfahrungen dafür zusammengefasst werden. Durch die zeitlich großen Abstände zwischen zwei Aktionen ist das im Rahmen dieser Umsetzung allerdings nicht sinnvoll, da die gemachten Erfahrungen dadurch noch später Wirkung zeigen würden. Über den *network*-Parameter der TensorForce-Klasse *LearningAgent*, welche von *Agent* erbt und selbst von *DQFDAgent* geerbt wird, wird das künstliche neuronale Netz spezifiziert. Hier wird die Variable *TF_NETWORK* übergeben, welche, wie der Zustands- und Aktionsraum, in *Shared_Definitions.py* definiert ist. Der Konstruktor der *DQFDAgent*-Klasse erwartet für diesen Parameter eine Liste von Dictionaries. Jeder Listeneintrag repräsentiert dabei einen Layer des neuronalen Netzes. Die Dictionaries besitzen wiederum einen *type*-Schlüssel, unter dem die Art des Layers, bspw. *dense* oder *lstm*, angegeben wird. Die Anzahl der Neuronen pro Layer wird mit dem, unter *size* abgelegten, Wert festgelegt. Weitere Konfigurationseinträge dieser Kategorie sind vom Typ des Layers abhängig. Über den Parameter *saver* kann die periodische Erstellung von Speicherpunkten nach einer gewissen Zeit oder Anzahl von Schritten aktiviert werden. Da das Speichern allerdings manuell in

5. Umsetzung

der *learn()*-Funktion durchgeführt wird, ist dieser Parameter nicht nötig. Ähnlich zum *saver* können, durch Spezifizieren des Parameters *summarizer*, in bestimmten Abständen Protokolldateien des Modells für *TensorBoard* erzeugt werden. Mit diesem Werkzeug des TensorFlow-Frameworks können verschiedene Interna des Lernprozesses visualisiert werden. Deren Bezeichner können unter dem Schlüssel *labels* als Liste angegeben werden. TensorForce liefert jedoch nur wenige Informationen für die Protokollierung, weshalb diese Möglichkeit nicht weiter verfolgt wird. Eine Vorverarbeitung der Eingabezustände kann durch *states_preprocessing* aktiviert werden. Dabei ist es möglich einen einzelnen Schritt oder eine Sequenz anzugeben, um bspw. Bilder zu skalieren oder den Wertebereich zu normalisieren. Für die Vorverarbeitung der Rewards besteht diese Möglichkeit gleichermaßen. Im Rahmen dieser Arbeit wird allerdings keine der beiden Vorverarbeitungsarten genutzt, da sowohl die Zustände, als auch die Rewards in einem geeigneten Format in den jeweiligen Funktionen generiert werden. Eine sehr wichtige Einstellung des Agenten ist die *actions_exploration*. Dadurch wird bestimmt, in welchem Maß Aktionen nach den bisherigen Erfahrungen oder durch Erkundung unbekannter Wege gewählt werden. Diese Abwägung wird auch mit den Begriffen *Exploration* und *Exploitation* beschrieben. Hier muss ein Kompromiss gefunden werden, denn um eine Strategie verbessern zu können, muss der Agent Aktionen ausprobieren. Nur so kann er dafür Rückmeldung durch die Belohnung erhalten und lernen, ob eine Aktion in einem bestimmten Zustand Erfolg verspricht oder nicht. In diesem Fall wird die *Epsilon Decay*-Strategie verwendet, welche zu Beginn durch einen hohen ϵ -Wert, hier $0,5$, einen entsprechend hohen Anteil an Erkundung zulässt. Bis zum Erreichen des Zielwerts, hier 0 , nach der angegebenen Anzahl von Schritten, sinkt der Wert im Zeitverlauf. Das hat den Vorteil, dass zu Beginn viele Aktionen probiert werden und somit ein gewisser Erfahrungsschatz aufgebaut wird. Dieser kann nach der anfänglichen Erkundungsphase genutzt werden, um eine möglichst optimale Strategie zu finden. Ein weiterer entscheidender Punkt ist der *optimizer*. Durch ihn wird bestimmt, wie und in welchen Größenordnungen die Gewichte des künstlichen neuronalen Netzes verändert werden. Dazu wird mit *type* angegeben, nach welcher Strategie die Anpassung stattfindet. Im Rahmen dieser Umsetzung wird der populäre *Adam*-Optimizer verwendet. Zusätzlich wird dem Optimizer eine Lernrate übergeben, die sich je nach Anwendungsfall stark unterscheiden kann. Sie bestimmt, wie groß die Änderungen an den Gewichten sind, die der Optimizer durchführt. Kleinere Werte bewirken einerseits eine feiner gestufte Annäherung an ein Optimum und sind damit möglicherweise in der Lage, eine bessere Lösung zu finden. Andererseits dauert der Weg zu dieser Lösung länger und der Agent bleibt unter Umständen in lokalen Optima hängen. Größere Werte hingegen ermöglichen es, sich einem Optimum schneller zu nähern. Sie bergen aber die Gefahr, durch zu große Anpassungen über das Ziel hinauszuschießen oder in eine Schwingung um eine Lösung zu geraten. Außerdem besteht die Möglichkeit, dass Optima durch ein zu großes Raster übersprungen und infolge gar nicht gefunden werden. Die Menge der vorhandenen Trainingsdaten ist für die Festlegung dieses Werts ebenfalls von Bedeutung, damit die vorhandenen Möglichkeiten überhaupt ein sichtbares Ergebnis bewirken. Im Rahmen dieser Arbeit werden des-

5. Umsetzung

halb zwei verschiedene Lernraten getestet. Zu Beginn wird der Vorgabewert $0,01$ und später, in Anlehnung an das Paper [121], eine Lernrate von $0,2$ genutzt. Mit dem Parameter *discount* wird die, in Unterabschnitt 2.3.2 bereits erläuterte, *Discount Rate* spezifiziert. Sie dient der Bestimmung der „Weitsicht“ des Agenten bei seiner Aktionswahl bzgl. der erwarteten Rewards. Je höher der Wert ist, desto mehr wird die Aktionswahl auf lange Sicht optimiert. Im Gegensatz dazu bewirken kleinere Werte eine Ausrichtung der Strategie auf die naheliegende Zukunft. Auch hier wird anfangs der TensorForce-Vorgabewert $0,99$ erprobt und später, ebenfalls dem Paper [121] entsprechend, auf $0,7$ reduziert. Der Parameter *memory* stellt die Option bereit, dem Agenten einen Speicher für Datensätze, bestehend aus Zustand, Aktion, Reward, Episodenende und ggf. Folgezustand, bereitzustellen. Damit kann das sogenannte *Experience Replay*, also das wiederholte Einbeziehen früherer Erfahrungen in den Lernprozess, genutzt werden. Das führt zur effizienteren Ausnutzung gemachter Erfahrungen und zu schnellerem Lernerfolg. Hierbei gibt es verschiedene Varianten, die bspw. bestimmte Erfahrungen priorisieren, zufällige auswählen oder stets die aktuellsten wiederholen. TensorForce sieht bei DQfD standardmäßig einen Speicher vom zufällig wählenden Typ *replay* mit der 1000-fachen Batch-Größe vor, der auch weiterhin verwendet wird. Die folgenden Parameter sind spezifisch für den DQfD-Agenten:

- *expert_margin* bestimmt den Abstand der Q-Werte aus Demonstrationsdaten vor den im Betrieb erlernten Q-Werten [99], standardmäßig $0,5$
- *demo_memory_capacity* gibt die Größe des Speichers für die Demonstrationsdaten vor, standardmäßig 10000
- *demo_sampling_ratio* definiert die *Sample-Rate* der Demonstrationsdaten zur Laufzeit, mit Vorgabewert $0,2$

Bei diesen drei zuletzt genannten Parametern werden jeweils die vorgegebenen Standardwerte der TensorForce-Bibliothek verwendet.

Nach der Initialisierung des Agents folgt der restliche Aufbau dem gleichen Muster, wie bereits beschriebenen MQTT-Clients. So beginnt die eigentliche Programmausführung mit der *init()*-Funktion, in der die Kommandozeilenparameter ausgewertet werden. In diesem Fall sind die Adresse des zu steuernden Klimageräts, der Index des zu verwendenden Sensorknotens sowie ein Offset des Lernintervalls, welcher später erläutert wird, zwingend anzugeben. Optional kann ein *Record-File* mit zugehörigem Sensorindex für das Pretraining spezifiziert werden. Die Notwendigkeit des Sensorindexes liegt im Datenformat der Record-Files, welche jeweils die Messwerte beider Sensorknoten beinhalten, begründet. Sind die beiden optionalen Parameter vorhanden, so wird damit anschließend die Funktion *pretrain_file()* aufgerufen. Darin wird der Agent durch *agent.reset()* zunächst zurückgesetzt, die angegebene Log-Datei geöffnet und schließlich Zeile für Zeile eingelesen. Die Zeilen werden, dem Log-Format aus Abschnitt 5.7 entsprechend, aufgetrennt und die Anzahl der Elemente geprüft. Danach erfolgt das Wiederherstellen der Python-Datenstrukturen

5. Umsetzung

aus den Textelementen. Ggf. wird der Aktionsraum früherer Entwicklungsstadien in das aktuelle Format konvertiert. Darauf folgend wird die jeweilige Aktion durch Abgleich mit dem Dictionary *ACTIONS* auf Validität geprüft und Status sowie Reward aus den importierten Informationen berechnet. Die Variable *ACTIONS* wird, neben den anderen gemeinsamen Festlegungen, zu Beginn des Quelltextes aus der Datei *Shared_Definitions.py* importiert. Durch die Abfrage, welche die Variable *terminal* setzt, wird jede fünfte Zeile als das Ende einer Epoche markiert. Das entspricht der gewählten Episodenlänge und muss somit beim Aufzeichnen mit dem Logging-Client beachtet werden. Am Ende eines jeden Schleifendurchlaufs werden die eingelesenen Daten als Dictionary an die Liste *demonstrations* angehängt. Nachdem alle Zeilen der Datei verarbeitet wurden wird die Anzahl der Pretraining-Schritte, falls noch nicht manuell geschehen, auf die Zahl Demonstrationsdatensätze in *demonstrations* festgelegt. Schlussendlich erfolgt der Import der Demonstrationsdaten, das darauf basierende Pretraining sowie ein abschließender Reset des Agenten.

Anfänglich wurde auch der Versuch unternommen, ein Pretraining unter Zuhilfenahme der Simulationsumgebung OpenAI Gym in der Funktion *pretrain_gym()* umzusetzen [71]. Dazu muss eine eigene Umgebung für Gym erstellt werden, die dann eine Raunklimatisierung, welche sich möglichst identisch zur realen Umgebung verhält, simuliert. Der Versuch wurde allerdings aufgrund von zu großen Schwierigkeiten bei der Umsetzung wieder verworfen. Die Entwicklung einer möglichst realitätsnahen Umgebung setzt eine sehr komplexe Modellierung voraus, damit der Agent in der echten Umgebung gleichermaßen funktioniert. Auch unter Einbindung einer Gebäudesimulationssoftware, wie das in einigen Papers genutzte *EnergyPlus* [30] bzw. *OpenStudio* [64], resultiert dies in viel mehr Aufwand, als es im Rahmen dieser Arbeit möglich wäre. Dennoch ist der Quelltext des Ansatzes auf dem beigelegten Datenträger, siehe Abschnitt A.1, enthalten.

Werden dem Programm die beiden optionalen Parameter nicht übergeben, so versucht es, den letzten gespeicherten Speicherpunkt wiederherzustellen. Der Dateipfad der Speicherpunkte, auch *Checkpoints* genannt, ist relativ zum Arbeitsverzeichnis und setzt sich zusammen aus *models/*, gefolgt vom Präfix *unit* und der Adresse des Klimageräts. Die Bezeichnung des letzten Checkpoints wird automatisch in der Datei *checkpoints* abgelegt. Nach dem Pretraining bzw. Laden des Checkpoints ist der letzte Aufruf in der Funktion *init()*, wie von den anderen MQTT-Clients bekannt, *create_mqtt_connection()*. Die Implementierung des KI-Agenten ist nicht-blockierend, weshalb der MQTT-Client mit *client.loop_forever()* in seine Hauptschleife versetzt wird. Sobald die Verbindung zum Broker hergestellt wurde, werden im Callback *on_connect()* wiederum alle benötigten Topics abonniert. Besonderheit ist hier die Verwendung eines Platzhalters im Topic-Namen. MQTT bietet zwei verschiedene Platzhalter bzw. *Wildcards*:

- *+* wählt sämtliche Topics einer einzelnen Ebene aus und
- *#* schließt alle Topics der aktuellen Ebene im Pfad sowie darunterliegende ein.

Im Fall eines Authentifizierungsfehlers, gekennzeichnet durch *rc == 4*, wird der Ver-

5. Umsetzung

bindungsversuch wiederholt. Nach erfolgreichem Verbindungsaufbau mit $rc == 0$ wird durch `reset_check_learn()` in die eigentliche KI-Funktionalität übergegangen, welche in Abbildung 5.8 als Ablaufplan dargestellt ist. Die Funktion ruft wieder-

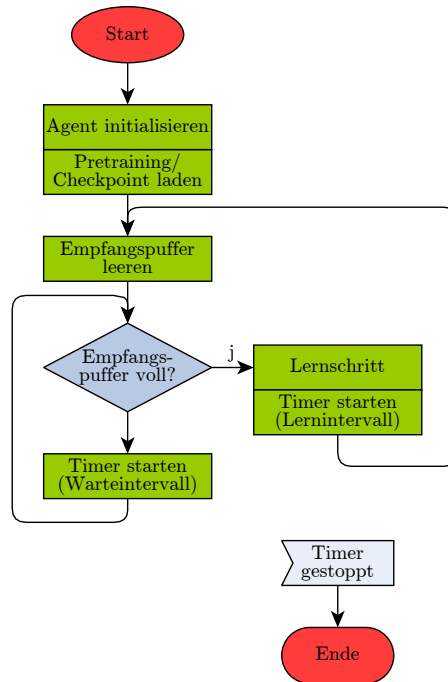


Abbildung 5.8.: Lernschleife des KI-Agenten

um `reset_vars()` auf, welche die Variable `unit_configuration` und das Dictionary der Sensorwerte, `sensor_data`, für die folgende Auswertung in `check_learn()` leert. Die Funktion `check_learn()` gewährleistet, dass alle zuvor rückgesetzten Strukturen wieder mit aktuellen Sensormesswerten und Umgebungsinformationen befüllt wurden, bevor ein Lernschritt der KI mit der Funktion `learn()` ausgeführt wird. Sofern die benötigten Daten noch unvollständig sind, wird `check_learn()` über einen Timer nach einer in `WAIT_INTERVAL` angegebenen Wartezeit in Sekunden wieder aufgerufen. Verläuft die Vollständigkeitsprüfung erfolgreich, so erfolgt die Ausführung eines Lernschritts durch die Funktion `learn()`. Ebenfalls unter Nutzung eines Timers wird darauffolgend die in `LEARN_INTERVAL` spezifizierte Zeit in Sekunden gewartet, wobei dieser Wert durch das Intervall-Offset beim Aufruf des Programms verlängert werden kann. Nach Ablauf des Timers wird der Kreis des KI-Lernverfahrens mit erneutem Einstieg in die Funktion `reset_vars()` geschlossen. Der Grundaufbau dieses Programms entspricht somit dem des Logging-Clients. Der jeweils aktive Timer wird in der globalen Variable `timer` hinterlegt, sodass er bei einem Verbindungsabbruch im Callback `on_disconnect()` gestoppt werden kann. Ohne diese Implementierung würde nach einem Verbindungsabbruch und anschließender Wiederherstellung in `on_connect()` ein zweiter Timer-Kreis beginnen, während der erste noch in `check_learn()` auf Daten wartet. Letztendlich würden mehrere solcher Kreise parallel

laufen, was zu unvorhersehbarem Verhalten führt und somit nicht erwünscht ist. Die Sensordaten werden im Callback *on_message()* empfangen und in das Dictionary *sensor_data* eingetragen. Als Schlüssel dafür dient der Name des Topics, auf dem der Messwert empfangen wurde. Eintreffende Sollwerte werden in Puffervariablen zwischengespeichert, aus denen sie nach der Auswertung des nächsten Lernschritts der KI übernommen werden. Die initialen Sollwerte stammen wiederum aus globalen Variablen, welche aus *Shared_Definitions.py* importiert werden. Statusnachrichten des Klimageräts, welchem der jeweilige KI-Client per übergebenem Parameter zugewiesen wurde, werden vom Client *daikinMQTT.py* im JSON-Format übermittelt. Demnach müssen sie zunächst mittels *decode()*-Methode von der Byte-Darstellung in eine Zeichenkette umgewandelt und diese durch *json.loads()* in ein JSON-Objekt geladen werden. Anschließend wird eine *Configuration*-Struktur erzeugt, welche die zur Statusüberwachung notwendigen Informationen des Klimageräts beinhaltet. Durch die Variable *unit_configuration_changed* wird angezeigt, ob sich der Status des Innengeräts gegenüber der letzten Statusnachricht geändert hat. Dazu wird *last_unit_configuration* bei Programmstart mit *None* initialisiert. Bei Empfang der ersten Statusnachricht in *on_message()* wird der Variable die Konfiguration aus *unit_configuration* zugewiesen. Folglich sind die beiden Variablen an dieser Stelle gleich. Sollte sich die aktuelle Konfiguration von der vorhergehenden unterscheiden, erfolgt eine Prüfung, ob die Änderung erwartet, also durch den KI-Agenten selbst oder unerwartet, bspw. durch das Bedienfeld des Geräts, ausgelöst wurde. Im erstgenannten Fall wird beim Senden einer Aktion die Variable *expect_configuration_change* in *learn()* auf *True* gesetzt. Das hat zur Folge, dass die geänderte Konfiguration in *last_unit_configuration* übernommen und *expect_configuration_change* wieder auf *False* zurückgesetzt wird. Als Bestätigung einer eingetroffenen, erwarteten Statusänderung wird der Buchstabe *e* hinter der gewählten Aktion ausgegeben. Sollte eine erkannte Änderung jedoch nicht erwartet worden sein, so wird dies in *unit_configuration_changed* hinterlegt. Die Variable *external_override* wird zusätzlich auf *True* gesetzt, wenn das Innengerät an- bzw. ausgeschaltet oder der Sollwert verändert wird. Diese Änderungen deuten an, dass eine Person unzufrieden mit der aktuellen Klimatisierung war und das Gerät am Bedienfeld umkonfiguriert hat. Anschließend wird auch hier der neue Status in *last_unit_configuration* übernommen und zusätzlich der globale Timer gestoppt. Mit *reset_check_learn()* wird der Lernkreislauf, im Gegensatz zur Handhabung erwarteter Änderungen, direkt neugestartet.

Die Funktion *learn()* führt, wie bereits erwähnt, den eigentlichen Lernschritt mit der TensorFlow-Bibliothek aus. Dazu wird zunächst die aktuelle Zeit in der Variable *timestamp* erfasst. Anschließend folgt die Behandlung von Statusänderungen. Sofern in *on_message()* eine Änderung erkannt und diese als *external_override* eingestuft wurde, wird die Funktion verlassen und somit in *check_learn()* für die Dauer eines Lernintervalls gewartet. Damit wird dem Änderungswunsch des Bedieners Folge geleistet. Die Dauer dieses Intervalls wurde von drei zunächst auf fünf und schließlich auf zehn Minuten erhöht, um den in Abschnitt 4.8 genannten Ansprüchen zu genügen. In der Praxis dauert es bis zu sechs Minuten, bis nach Aktivierung eines

5. Umsetzung

Klimageräts der gewünschte Effekt eintritt. Nach dieser Pause erfolgt im nächsten Aufruf von *learn()* die Behandlung normaler Statusänderungen. Sie besteht darin, den Reward auf 0 zu setzen und dem Agenten somit weder Belohnung noch Bestrafung für die zuletzt gewählte Aktion zu geben. Der aktuelle Schritt innerhalb einer Episode wird in *step* gespeichert. Zu Beginn des Programms ist die Variable *θ*, d. h. es findet noch keine Bewertung in *learn()* statt. Das ist ebenso in der Behandlung der Statusänderungen berücksichtigt. Hier wird *step*, sofern größer als *θ*, dekrementiert, sodass der aktuelle Lernschritt als Wiederholung dargestellt wird. Im Falle von *step* gleich 1 hätte das zur Folge, dass keine Bewertung stattfinden würde, obwohl im Schritt zuvor eine Aktion ausgeführt wurde. An dieser Stelle würde TensorFlow eine Fehlermeldung ausgeben. Deshalb wird die Beobachtung bei *step* gleich 1 vor dem Dekrementieren, mit dem Aufruf von *agent.observe()*, ausgeführt. Mit dem Zurücksetzen von *unit_configuration_changed* auf *False* ist die Behandlung der Statusänderungen abgeschlossen. Wenn keine Statusänderung vorliegt, wird der Reward für die letzte Aktion des Agenten zu Beginn von *learn()* mit *PROCESS_STATE* aus den gesammelten Sensor- und Statusdaten sowie den Sollwerten berechnet. *PROCESS_STATE* ist dabei ein Import aus *Shared_Definitions.py*. Der Index des zu verwendenden Sensors, welcher dem Programm als Parameter übergeben werden muss, wird an den Topic-Pfad des jeweiligen Messwerts angehängt, z. B. *sensors/temperature/1*. Wie bereits zuvor erwähnt ist *step* beim ersten Aufruf von *learn()* 0, d. h. es werden lediglich die ggf. neuen Sollwerte aus den Puffervariablen übernommen und der aktuelle Zustand erfasst. Danach leitet der Agent mit *agent.act()* eine Aktion daraus ab. Der Parameter *deterministic* erlaubt es hierbei, die anfangs definierte Exploration an- oder abzuschalten. Die resultierende Aktion wird auf dem Topic *aircon/units/[unit_address]/action* veröffentlicht und auf der Gegenseite vom Client *daikinMQTT.py* an die Klimasteuerung übermittelt. Schließlich erfolgt über das Dictionary *ACTIONS* und die aktuellen Gerätekonfiguration *unit_configuration* eine Prüfung, ob eine Statusänderung zu erwarten ist. Ggf. wird die dafür vorgesehene Variable gesetzt. Abschließend wird der Lernschrittindex inkrementiert. In den nächsten Schritten wird dadurch zusätzlich der Bewertungs-zweig der *learn()*-Funktion durchlaufen. Hier werden die Rewards der aktuellen Episode zu statistischen Zwecken aufsummiert und ausgegeben. Mit der globalen Variable *STEPS_PER_EPISODE*, welche in *Shared_Definitions.py* definiert ist, wird die Anzahl der Lernschritte pro Episode festgelegt. Der Startwert hierfür beträgt fünf Schritte, um den zeitlichen Horizont des Agenten, in Kombination mit dem zehnminütigen Lernintervall, auf unter eine Stunde zu beschränken. Liegt *step* noch unterhalb dieser Beschränkung, so wird *agent.observe()* mit dem zuvor berechneten Reward und *terminal* gleich *False* parametrisiert. Falls sich die Sollwerte ändern, wird der Agent zudem zurückgesetzt, um in Kombination mit dem gewählten Zustandsraum keine fehlerhaften Annahmen zum Zustandsverlauf zuzulassen. Sobald *step* die Begrenzung von *STEPS_PER_EPISODE* erreicht, wird die aktuelle Episode beendet und eine neue begonnen. Dazu wird der Episodenzähler *episode*, der aufsummierte Reward, als auch der maximale Reward aller bisherigen Episoden ausgegeben und aktualisiert. Die Variable *step* wird wieder auf 0 zurückgesetzt und

5. Umsetzung

dem KI-Agenten mit *terminal* gleich *True* beim Aufruf von *agent.observe()* das Ende der Episode signalisiert. Der Lernschrittindex bleibt jedoch nur bis zum Ende der Funktion *0*, denn dort wird er wieder inkrementiert, um für die gewählte Aktion im nächsten Schritt eine Bewertung zu erhalten. Damit wird gewährleistet, dass nur nach dem Programmstart durch *step* gleich *0* einmal keine Beobachtung durchgeführt wird, da es zu diesem Zeitpunkt noch keine ausgeführte Aktion gibt. Danach wird der entsprechende Programmzweig in jedem weiteren Schritt durchlaufen. Mit *agent.save_model()* wird an dieser Stelle ein Checkpoint des Agenten erstellt, der bei erneuten Programmstarts wiederhergestellt werden kann. Wichtig ist, dass der Aufruf dieser Methode nur nach einer abgeschlossenen Beobachtung und vor der folgenden Aktion stattfindet. Ansonsten kann es zu Fehlern bei wiederhergestellten Checkpoints kommen, weil noch eine Aktion ohne Bewertung offen ist, das Programm zu Beginn aber nur eine Aktion ohne Beobachtung ausführt. TensorFlow gibt in einer solchen Situation eine Fehlermeldung aus, da auf Aktionen ohne die *independent*-Markierung eine Beobachtung folgen muss. Nach der Erstellung des Checkpoints besteht die Möglichkeit, das Programm mittels der anfangs erfassten Variable *timestamp* zum gewählten Zeitpunkt automatisch zu beenden oder Pausenzeiten zu definieren. In beiden Fällen wird zuvor das betreffende Klimagerät mit der Aktion *0* ausgeschaltet. Bei Beendigung des Programms wird die Variable *shutdown* auf *True* gesetzt und schließlich mit *client.disconnect()* die MQTT-Verbindung getrennt. Da dies kurze Zeit dauern kann, wird in *check_learn()* ebenfalls eine Abfrage von *shutdown* getätigt und ggf. kein neuer Timer gestartet. Falls der KI-Agent pausieren, also eine Zeit lang keine Aktionen ausgeben soll, wird dies durch *pause = True* gekennzeichnet. Zu Beginn jedes Aufrufs von *check_learn()* findet eine Überprüfung statt, ob die Pause vorüber ist und ggf. wird *pause* wieder auf *False* gesetzt. Die Pausenzeit wird durch Angabe der Beginn- und Endstunde in den globalen Variablen *PAUSE_MODE_START* sowie *PAUSE_MODE_END* bestimmt. Während dieser Zeit werden keine Aktionen ausgeführt, da in *check_learn()* stets die erste Bedingung zutrifft und demzufolge immer wieder der *WAIT_INTERVAL*-Timer ausgelöst wird.

Der bereits erwähnte Lern- bzw. Aktionsintervall bestimmt, in welchem Zeitabstand der KI-Agent den Zustand erfasst und eine Aktion daraus ableitet. Er wird in der globalen Variable *LEARN_INTERVAL* in Sekunden angegeben und kann zusätzlich über den Programmparameter *learn_interval_offset* vergrößert werden. Diese Möglichkeit wird benötigt, um bei mehreren gleichzeitig agierenden KI-Agenten mögliche Überschneidungen zwischen erwarteten und unerwarteten Statusänderungen der Klimageräte zu vermeiden. Da die Statusmeldungen der Innengeräte vom Client *daikinMQTT.py* im Abstand von zehn Sekunden abgerufen und veröffentlicht werden, können mehrere Änderungen innerhalb dieses Abstands stattfinden. Eine solche Situation tritt bspw. ein, wenn ein Agent eine Aktion tätigt, die einen Wechsel der Betriebsmodi innerhalb einer ganzen Zone, also eines logischen Zusammenschlusses mehrerer Innengeräte, zur Folge hat. Führt nun ein anderer KI-Agent in dieser Zone zeitgleich eine Aktion aus, so können sich die beiden Änderungen vermischen und ggf. eine davon verloren gehen. Das wiederum könnte Auswirkungen

auf die korrekte Behandlung der Änderungen haben und somit bspw. eine Pause durch einen externen Eingriff am Bedienteil überspringen. Zu einem Wechsel der Betriebsmodi einer ganzen Zone kommt es, wenn der *Master* der Zone seinen Betriebsmodus ändert. Master ist das Innengerät, welches an erster Stelle der Liste im iTC steht und somit die Entscheidungsgewalt besitzt. Die untergeordneten Innengeräte übernehmen automatisch den Betriebsmodus des Masters, was logischerweise Konsequenzen für die Aktionswahl des KI-Agenten haben kann. Ein solcher Wechsel kann durch Aufsummierung des Lernintervall-Offsets über mehrere Schritte ohne Statusänderung zu beliebiger Zeit innerhalb eines Intervalls des KI-Agenten auftreten. Deshalb wird in einer solchen Situation, wie bereits erwähnt, der aktuelle Lernschritt mit einem Reward von 0 bewertet und mit neuer Aktionswahl direkt wiederholt. In der TensorFlow-Bibliothek gibt es keinen Weg, eine Aktion nachträglich als *independent* zu markieren und somit von der Beobachtung auszuschließen. Der Reward von 0 soll deshalb bewirken, dass der Agent für die unterbrochene Aktion weder eine Belohnung, noch eine Bestrafung erhält. Trotzdem wird es eine, wenn auch geringfügige, Auswirkung in Form einer Gewichtsanzpassung nach sich ziehen, wenn der erwartete Return dieser Aktion in diesem Zustand von 0 abweicht.

5.9. Zusammenfassung

Ergebnis der Implementierungsphase ist ein Gesamtsystem aus Sensornetzwerk, welches das Raumklima zweier Räume erfasst, KI-Agent, der die Daten verarbeitet sowie Schnittstelle zum Klimacontroller, welche Aktionen zu den Klimageräten weiterreicht. Wie in Abbildung 5.1 dargestellt, werden die Bosch BME680-Sensoren über

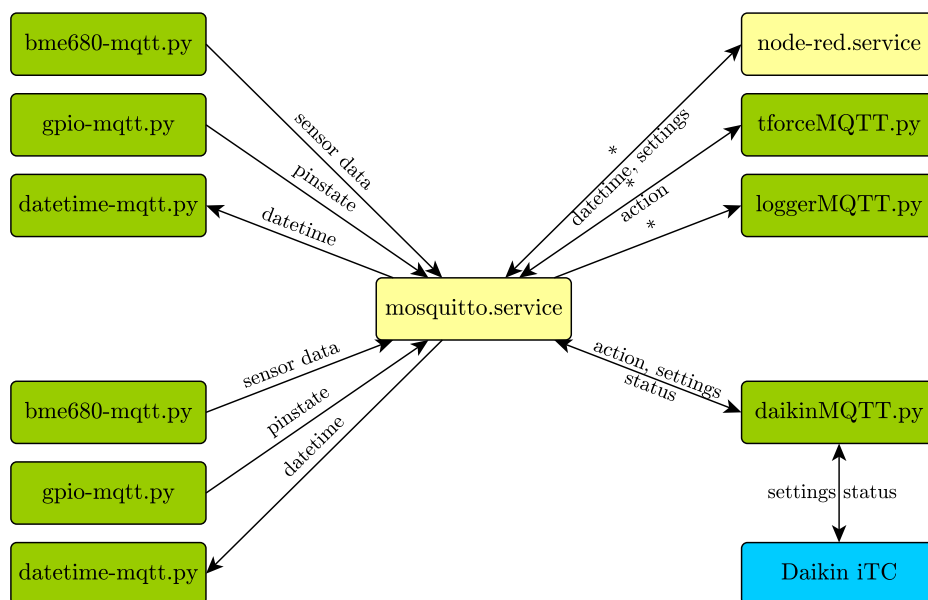


Abbildung 5.9.: Datenflussdiagramm

5. Umsetzung

den I2C-Bus und die Reed-Schalter durch GPIO-Pins ausgelesen. Mittels MQTT-Protokoll sowie Netzwerkinfrastruktur aus 802.11s-Mesh und Ethernet werden Messwerte, Statusnachrichten als auch Aktionen unter den Teilnehmern ausgetauscht. Der BeagleCore dient dabei als Brücke zwischen den beiden Netzwerken sowie als Nachrichten-Broker. Abbildung 5.9 zeigt die einzelnen Softwarekomponenten, welche anhand der Position ihrer Hardwareplattform aus Abbildung 5.1 angeordnet sind. Ergänzend dazu ist eine Übersicht des Datenflusses in Form der gesendeten und empfangenen Informationen an den Verbindungspfeilen abgebildet. Das Sternsymbol * steht dabei für die Gesamtheit der verfügbaren Informationen. Die genaue Struktur der zur Übertragung genutzten MQTT-Topics ist in Abbildung 5.3 visualisiert. Jede Röhre steht dabei für ein Topic, die Publisher sind durch eingehende, die Subscriber durch ausgehende Verbindungen gekennzeichnet. Die Ausgänge einiger Topics, deren Verbindungen in gemeinsamen Empfängern münden, werden dabei zusammengefasst dargestellt. Topics, deren Pfad sich lediglich durch einen Index unterscheidet, werden durch Angabe desselben in eckigen Klammern ebenfalls vereinigt. Mittig über den Topics steht der Mosquitto-Dienst, welcher als MQTT-Broker sämtliche Topics verwaltet.

6. Ergebnis

Das im vorhergehenden Kapitel implementierte System wird nun in verschiedenen Konfigurationen und Szenarien erprobt. Dazu werden Referenzmessungen im Normalbetrieb als Basis erfasst und ein Vergleich mit den Testresultaten angestellt. Es muss jedoch erwähnt werden, dass keine Laborbedingungen bestehen und die Messwerte einer gewissen Toleranz unterliegen. Dennoch wird darauf geachtet, die Bedingungen während der Testläufe so ähnlich wie möglich zu gestalten. In Tabelle 6.1 bis Tabelle 6.4 sind alle verwendeten Konfigurationen, kategorisiert und in chronologischer Reihenfolge, als Übersicht aufgelistet.

	Beschreibung
I	Soll- und Ist-Werte von Temperatur, Luftqualität, Luftfeuchtigkeit, Ist-Wert Luftdruck
II	Soll-Ist-Differenz der Temperatur
III	Soll-Ist-Differenzen von Temperatur und Luftfeuchtigkeit
IV	Betriebsmodus, Soll-Ist-Differenzen von Temperatur und Luftfeuchtigkeit
V	Betriebsmodus, Soll-Ist-Differenzen von Temperatur und Luftfeuchtigkeit, Ist-Wert Luftqualität
VI	Betriebsmodus, Soll-Ist-Differenz von Temperatur, Ist-Wert Luftqualität

Tabelle 6.1.: Zustandsmodelle

Folgend wird der Weg zur Findung eines geeigneten Systems über die dort aufgelisteten Informationen beschrieben. Ausgangssituation ist ein Zustandsraum, der jeweils Soll- und Ist-Werte der Temperatur, Luftqualität, Luftfeuchtigkeit sowie den Luftdruck enthält. Der entsprechende Aktionsraum umfasst alle möglichen Konfigurationen, die an einem Innengerät der Klimaanlage gesetzt werden können. Dazu gehören sämtliche Kombinationen aus Ein-/Aus-Zustand, Lüftungs- bzw. Kühlmodus, drei Setpoints mit maximal ± 1 °C Abstand um den Wunschwert herum, zwei Gebläsestufen und sechs verschiedene Gebläserichtungen. Der Return enthält die Temperaturbelohnung mit einer Wichtung von $0,5$, die Luftqualitäts- und die Luftfeuchtigkeitsbelohnung jeweils mit Wichtung $0,1$ sowie die Energiebelohnung, gewichtet mit Faktor $0,3$. Wertebereich der einzelnen Bestandteile ist jeweils zwischen 0 und 1 , linear ansteigend in Abhängigkeit der Entfernung vom Sollwert, sodass, zusammen mit der Wichtung, stets ein Return im selben Wertebereich entsteht. Die Luftqualität hat hierbei keinen Sollwert in dem Sinn, sodass der komplette Bereich zwischen

6. Ergebnis

Aktionen	Erklärung
73	Ein-/Aus-Zustand, Lüftung/Kühlung, drei Setpoints (± 1 °C), zwei Gebläsestufen, sechs Gebläseerichtungen
9	Ein-/Aus-Zustand, Lüftung/Kühlung, drei Setpoints (± 1 °C), zwei Gebläsestufen, eine Gebläseerichtung
5	Ein-/Aus-Zustand, Lüftung/Kühlung, drei Setpoints (± 1 °C), eine Gebläsestufe, eine Gebläseerichtung
3	Ein-/Aus-Zustand, Lüftung/Kühlung, ein Setpoint (-1 °C), eine Gebläsestufe, eine Gebläseerichtung
4	Ein-/Aus-Zustand, Lüftung/Kühlung/Heizung, ein Setpoint (± 1 °C), eine Gebläsestufe, eine Gebläseerichtung
4	Ein-/Aus-Zustand, Lüftung/Kühlung/Heizung, ein Setpoint (± 2 °C), eine Gebläsestufe, eine Gebläseerichtung

Tabelle 6.2.: Aktionsmodelle

Modell	Wichtung				
	Temperatur	Feuchtigkeit	Qualität	Energie	Strafe
0-1 linear	0,5	0,1	0,1	0,3	-
0-1 quadratisch	0,5	0,1	0,1	0,3	-
1/0	0,5	0,1	0,1	0,3	-
	4	2	2	2	
	50	1	1	10	
1/-1	50	1	1	10	1
			30	25	50
		30	35	15	100

Tabelle 6.3.: Belohnungsmodelle

Minimal- und Maximalwert auf den Wertebereich zwischen 0 und 1 skaliert wird. Das künstliche neuronale Netz der KI umfasst zwei *dense*-Layer mit jeweils 64 Neuronen und *relu*-Aktivierung. Mit einer Lernrate von $0,01$ und einer Discount-Rate von $0,99$ wird das neuronale Netz vom DQfD-Agent, zunächst auf einem Datensatz von 1500 Beispielaktionen, ebenso viele Schritte vortrainiert. Im Anschluss erfolgt die Beobachtung einiger Episoden. Die resultierende Aktionswahl erscheint zufällig, bei zu niedriger Temperatur wählt der Agent trotzdem eine Kühllaktion und im Wunschtemperaturbereich schaltet er das Klimagerät nicht ab.

Eine stufenweise Erhöhung der Neuronenzahl auf 128 , 256 bzw. 512 pro Layer bringt keine Besserung. Als Ursache wird deshalb der umfangreiche Aktionsraum vermutet, der für die vorhandene Menge an Trainingsdaten schlichtweg zu groß zu sein scheint. Dementsprechend erfolgt die Festlegung des Parameter für die Gebläseerichtung auf

6. Ergebnis

Hidden Layers			Pretraining		Lernrate	Discount-Rate
Anzahl	Typ	Neuronen	Datensätze	Schritte		
2	dense	64	1500	1500	0,01	0,99
		128				
		256				
		512				
1	dense	512	1500	1500	0,01	0,99
		64				
		32				
		16				
	8	3000	3000	0,2	0,7	
	lstm					32
						16
8						

Tabelle 6.4.: KI-Konfigurationen

den Wert 7, wodurch Richtungsklappen automatisch schwenken. Außerdem wurde die Setpoint-Einstellung für den Lüftungsmodus entfernt, da sie an dieser Stelle ohnehin keine Auswirkung auf die Funktion hat. Der Umfang des Aktionsraums verringert sich dadurch auf nur noch neun Aktionen. Die Änderung brachte jedoch nicht die gewünschte Besserung, die Aktionswahl ist immer noch nicht zufriedenstellend.

Im nächsten Schritt wird die Belohnungsfunktion angepasst, um dem Agenten deutlicher zu machen, welche Zustände erwünscht sind und welche nicht. Dazu werden die Bestandteile nicht mehr linear, sondern quadratisch ansteigend, wiederum auf Grundlage der Entfernung zum Zielwert, vergeben. Da auch nach dieser Änderung keine wesentlichen Veränderungen eintreten, wird die Reward-Gestaltung grundlegend verändert.

Solche ansteigenden bzw. zum Ziel hinführenden Belohnungen sollen dem Agenten zwar die Annäherung an das gewünschte Verhalten erleichtern, gelten aber als verzögernd für das exakte Finden einer Lösung. Deshalb werden nun Belohnungen verwendet, die nur im Wunschbereich 1 und außerhalb stets 0 zurückgeben. Für die Bewertung Luftqualität wurde dementsprechend ein Schwellwert definiert, ab dem die Belohnung mit 1 erfolgt. Eine solche Reward-Strategie bedeutet, dass der Agent den Zielbereich durch Exploration zufällig finden muss, da er außerhalb keinerlei Hinführung bekommt. Dieser Zielbereich ist für den Agenten in diesem Fall bereits durch die Demonstrationsdaten erkennbar. Zusammen mit der bestehenden Wichtung bringt aber auch diese Änderung keine Besserung des Verhaltens. Infolgedessen wird der Aktionsraum durch die Festlegung der niedrigeren Gebläsestufe und die Anpassung der Setpoint-Konfigurationen im Kühlmodus reduziert. Daraus resultieren

6. Ergebnis

zunächst fünf und nach dem zweiten Schritt noch drei mögliche Aktionen für den Agenten: Innengerät ausschalten, Lüftungsmodus und Kühlmodus mit Setpoint auf nun -1 °C unter dem Wunschwert des Bedieners. Diese Setpoint-Einstellung liegt in Problemen mit der Steuerungshysterese begründet, die trotz Kühlmodus zum Teil keine Kühlung stattfinden lässt. Anscheinend sorgt die extreme Trägheit des internen Temperaturwerts der Steuerung in Verbindung mit der Hysterese dafür, dass der Sollwert, aus Sicht der Steuerung, zu nahe am Istwert liegt und deshalb kaum Kühlleistung angefordert wird. Dennoch haben die gemachten Änderungen keine nennenswerten Auswirkungen auf das Lernergebnis des KI-Agenten.

Um den Return eindeutiger zu formulieren, wird nun die Wichtung der einzelnen Bestandteile verändert. Die Faktoren werden zuerst auf eine Wichtung von 4 für Temperatur und 2 für alle anderen Parameter erhöht. Danach erfolgt ein Test von 50 für die Temperatur, und je 1 für Luftqualität und Luftfeuchtigkeit sowie 10 für energiesparendes Verhalten. Der resultierende Wertebereich der Belohnung von 0 bis 62 soll den Lernprozess stärker beeinflussen. Auch diese Änderung ist jedoch nicht von Erfolg gekrönt.

Den Durchbruch bringen schließlich zwei weitere Veränderungen. Die erste davon besteht darin, einen der beiden Layer des neuronalen Netzes zu entfernen. Die zweite umfasst die Reduktion des Zustandsraums. Mit nur einem, 64 Neuronen umfassenden Layer und lediglich einem Zustandsparameter, der Differenz zwischen Soll- und Ist-Wert der Temperatur, tätigt der Agent erstmals Aktionen, die den Erwartungen entsprechen. Die Reward-Anteile der nicht berücksichtigten Parameter werden dazu über die Wichtungen auf 0 gesetzt. Die Anzahl der Neuronen kann im weiteren Verlauf zunächst auf 32 , anschließend 16 und später sogar auf acht Neuronen reduziert werden, ohne eine Beeinträchtigung des Ergebnisses beobachten zu können. Allerdings treten zeitweise Fälle auf, die der Agent scheinbar nicht richtig einschätzen kann. So kommt es vor, dass der Agent, trotz zu hoher Raumtemperatur, keine Kühlung einleitet.

Aufgrund dessen erfolgt die Erweiterung des Returns um Bestrafungen für unerwünschte Zustände. Statt außerhalb des Sollfensters der einzelnen Klimaparameter 0 zurückzugeben, wird nun -1 verwendet. Das bewirkt, dass der Agent diese Zustände aktiv vermeidet und sich somit stets innerhalb der Sollbereiche aufhält. Aufgrund der starken Wichtung wird hierbei die Einhaltung des Temperaturwerts vom Agenten priorisiert. Das hat zur Folge, dass der Agent auch innerhalb des Sollbereichs eigentlich unnötige Kühlphasen ausführt, die einen erhöhte Energiebedarf nach sich ziehen. Die Wichtung des Returns für die Energieaufnahme wird auf 25 erhöht, um ein energiesparenderes Verhalten zu fördern. Der Agent schaltet das Innengerät nun innerhalb der Temperatursollgrenzen meist ab und aktiviert es erst bei Überschreitung derselben wieder.

Um die Möglichkeiten des Systems auszuloten, wird der Zustandsraum nach der anfänglichen Reduktion nun wieder schrittweise erweitert. Zunächst wird, analog zur Temperatur, die Luftfeuchtigkeit als Differenz zwischen Soll- und Ist-Wert hinzugefügt. Der volle Return wird dabei zunächst innerhalb eines Radius von 10 % relativer Luftfeuchte um den Sollwert herum gewährt. Aufgrund des zu großen Spiel-

6. Ergebnis

raums wird schließlich ein dreistufiger Return mit Rückgabe 1 bei unter 5% Abweichung, 0 bei unter 10% und -1 in allen anderen Fällen eingeführt. Die Wichtung des entsprechenden Anteils im Return wird dabei, mit verbesserungsfähigen Ergebnissen bei Faktor 10 und 20 , letztendlich auf 30 gesetzt.

Da es sich bei der Klimaanlage um ein HVAC-System handelt, kann es nicht nur zum Lüften und Kühlen, sondern auch zum Heizen der Räume genutzt werden. Der Aktionsraum wird zum Einbinden dieser Funktion um eine Aktion erweitert, die das Innengerät in den Heizmodus mit Setpoint $+1\text{ °C}$ versetzt. Im selben Zug wird der Zustandsraum um den Betriebsmodus des Innengeräts erweitert. Durch diesen soll der KI-Agent lernen zu erkennen, ob der Modus des Klimageräts bspw. durch eine Zonenumschaltung geändert wurde.

Da die Aktionswahl außerhalb des gewünschten Temperaturfensters noch nicht zielstrebig genug bzgl. der Rückführung in den Zielbereich erscheint, wird die Belohnungsfunktion nochmals entsprechend verändert. Die bisherigen Belohnungen werden nur noch innerhalb des Sollbereich der Temperatur vergeben, außerhalb wird stets eine Bestrafung von -1 , multipliziert mit einer Wichtung von 50 bzw. später 100 , vergeben. Diese Änderung soll bewirken, dass die Einhaltung des Temperaturbereichs höchste Priorität erhält und nur, wenn diese Bedingung gegeben ist, Returns für die weiteren Parameter vergeben werden.

Weiterhin wurde der Zustandsraum um die Luftqualität erweitert, sodass er nun den Betriebsmodus, die Temperaturabweichung, die Luftfeuchtigkeitsabweichung und die Luftqualität enthält. Der Wert des neuen Parameters liegt zwischen 0 und 1 . Er entspricht dem gemessenen Sensorwert im Verhältnis zur Referenz, welche durch die Variable `GAS_BASELINE` in `Shared_Definitions.py` definiert ist. Die Reward-Berechnung hierfür wird zudem verändert, indem der Luftqualitätswert auf einen Wertebereich von -1 bis 1 abgebildet wird. Durch das Hinzufügen des weiteren Parameters verschlechtert sich die Aktionswahl des Agenten wieder merklich. Eine Erhöhung der Schritte sowie Datensätze für das Pretraining auf je 3000 bringt keine nennenswerte Verbesserung. Selbiges gilt auch für die Verwendung einer niedrigeren Discount- und einer höheren Lernrate aus einer der verwandten Arbeiten [121]. Deshalb wird folgend stets entweder die Luftfeuchtigkeit oder die Luftqualität in den Zustand einbezogen.

Als letzte Änderung werden die Dense-Layer des neuronalen Netzes durch Layer aus zunächst 32 , später 16 und letztendlich acht LSTM-Zellen ersetzt. Durch diese Änderung ist die Güte der Aktionswahl nicht mehr direkt einschätzbar, da LSTM-Zellen statt eines einzelnen Zustands einen Verlauf auswerten. Allerdings verspricht dieser Schritt noch intelligenteres und somit energiesparenderes Verhalten, weil bei kurzen Überschreitungen der Sollwertgrenzen ggf. nicht sofort in den Kühlmodus gewechselt wird.

Der Einsatz des *global*-Clients zur parallelen Steuerung zweier Räume wird nach kurzer Testphase nicht weiter verfolgt. Es treten hier dieselben Probleme auf, wie beim Einzel-Client zu Beginn. Vermutlich liegt das im wiederum zu großen Zustandsraum, der für zwei Räume benötigt wird, begründet. Das Ergebnis der Erprobung verschiedener neuronaler Netze ist demnach ein Netz, welches einen versteckten Layer mit

6. Ergebnis

dieser Umstand im höheren Luftvolumen des Raums begründet, das durch einen einzelnen Sensor schlechter erfasst werden kann. Die Raumluft wird erst bei Betrieb des Klimageräts umgewälzt und führt dabei oft zu einer niedrigeren Luftqualität. Dieses Verhalten ist möglicherweise dafür ausschlaggebend, dass der Agent im großen Raum, selbst unter Einbeziehung der Luftqualität in den Zustand, nie den Lüftungsmodus nutzt. Im kleinen Raum hingegen ist das unter gleichen Bedingun-

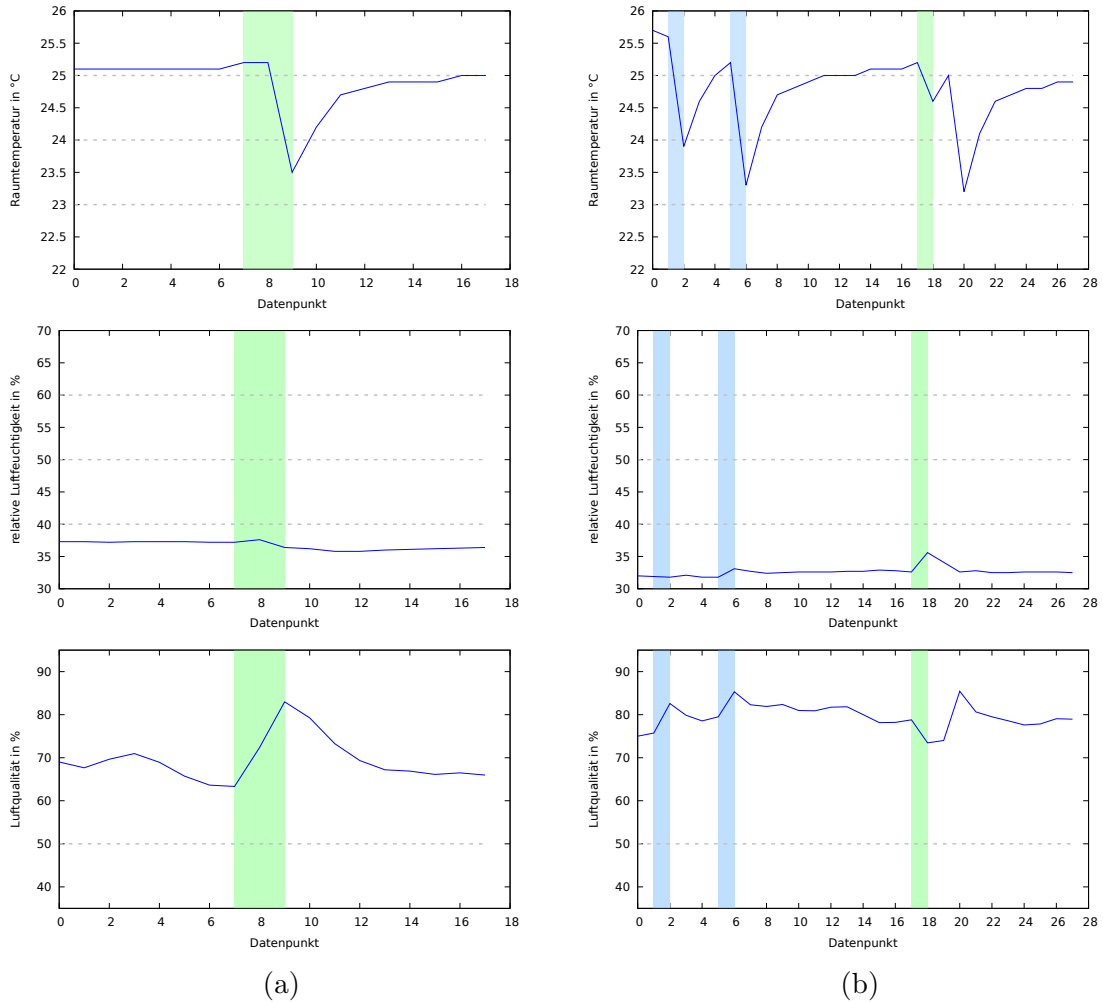


Abbildung 6.2.: Agent mit Kenntnis der Luftqualität im kleinen Raum (1/2)

gen der Fall, wie in den Diagrammen in Abbildung 6.2 und Abbildung 6.3 erkennbar ist. Die Umwälzung wird energetisch geringer bestraft als der Kühlmodus und wirkt sich hier zudem meist positiv auf die Luftqualität aus. Selbst bei anfänglichem Absinken wird bei anhaltendem Betrieb meist eine Verbesserung erreicht. Die relative Luftfeuchtigkeit steigt im Kühlbetrieb durch das Absinken der Temperatur an. Kältere Luft kann weniger Wasser aufnehmen, wodurch die relative Feuchtigkeit bei unveränderter Wassermenge zunimmt. Einige Stellen in den Messwerten, bspw. in Abbildung 6.1, spiegeln trotz abnehmender Temperatur eine Senkung wider. Als

6. Ergebnis

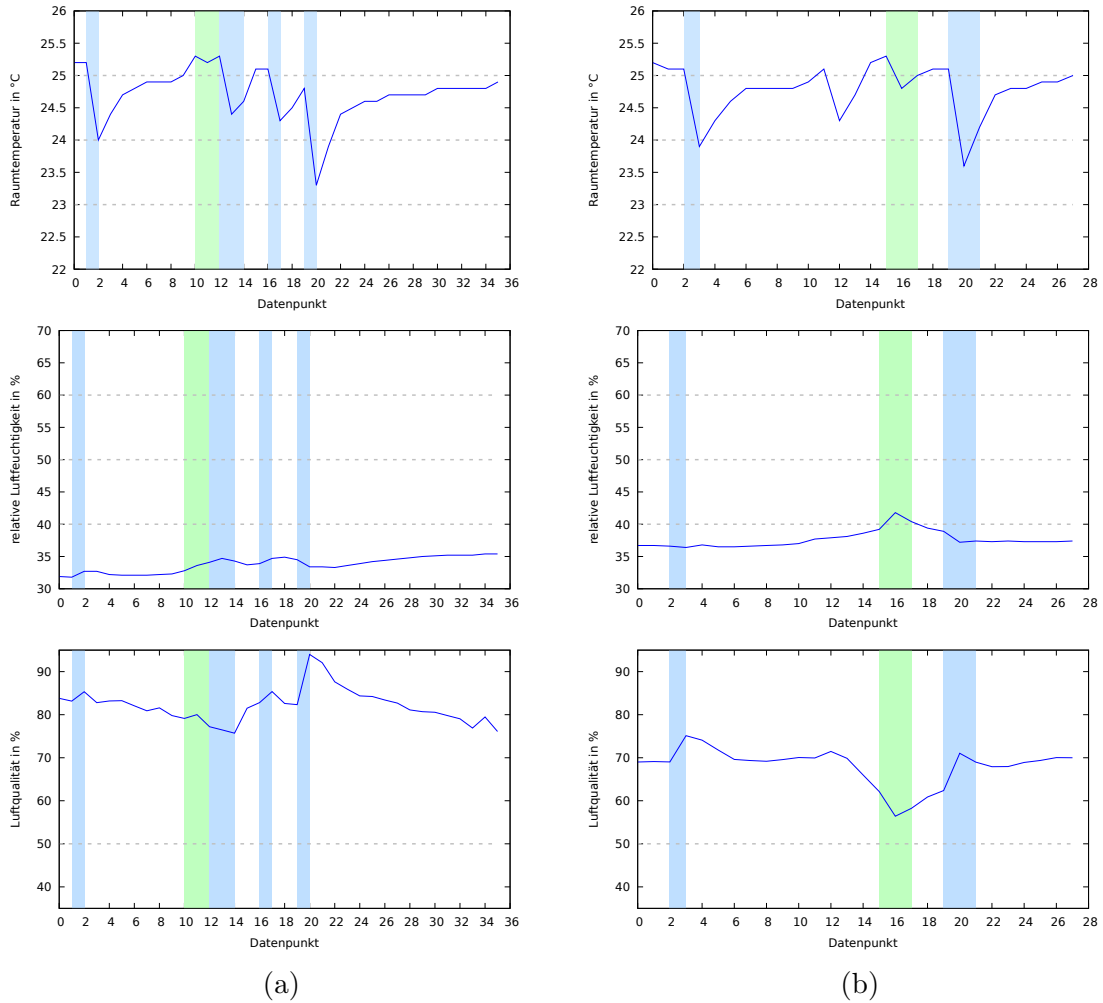


Abbildung 6.3.: Agent mit Kenntnis der Luftqualität im kleinen Raum (2/2)

Grund dafür wird ebenfalls die zuvor besprochene Umwälzung der Luft, ggf. in Kombination mit geöffneten Türen, angenommen. Abbildung 6.3a und Abbildung 6.4b zeigen, dass der KI-Agent bei Temperaturüberschreitungen nicht zwingend direkt in den Kühlmodus schaltet. Bedingt wird dieses Verhalten einerseits durch ggf. fehlende Strategieoptimierung des Agenten, andererseits durch die Betrachtung einer Zustandsfolge über die LSTM-Zellen im neuronalen Netz. Möglicherweise beobachtet der Agent zunächst den Verlauf, um bei einer Temperatursenkung durch externe Einflüsse, wie dies in Abbildung 6.2b und Abbildung 6.3b zu sehen ist, keine zusätzlichen Maßnahmen einleiten zu müssen. Im großen Raum führt dieses Abwarten allerdings dazu, dass der Sollbereich der Temperatur um fast 1 °C verlassen wird. Beispiele dafür sind die Verläufe in Abbildung 6.4 und Abbildung 6.5. Wie bereits beschrieben, nutzt der Agent im kleinen Raum, unter Einbeziehung der Luftqualität in die Eingabe, häufig den Lüftungsmodus. Die Diagramme in Abbildung 6.2 und Abbildung 6.3 zeigen, dass er damit in allen dargestellten Fällen mindestens eine kleine, wenn auch

6. Ergebnis

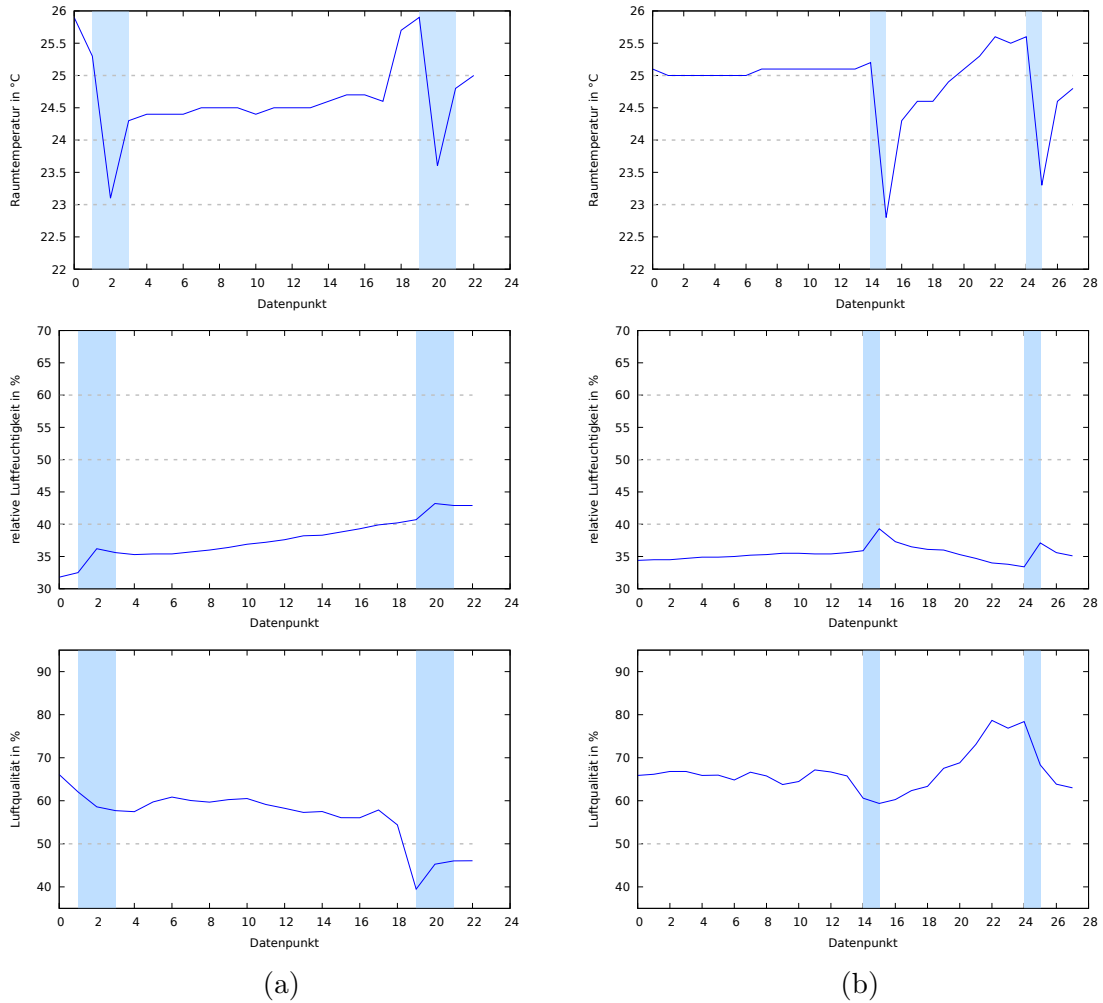


Abbildung 6.4.: Agent mit Kenntnis der Luftfeuchtigkeit im großen Raum

zum Teil nicht ausreichende Absenkung der Temperatur bewirkt. Erklärung dafür könnte sein, dass sich der Agent die Nutzung noch vorhandener Restkälte in den Leitungen und Klimageräten oder die nochmalige Umwälzung zuvor gekühlter Luft angeeignet hat. Erst, wenn die Aktion nicht ausreicht, um wieder in den Sollbereich zurückzukehren, wird in den Kühlbetrieb gewechselt. Dieses energiesparende Vorgehen wird durch die Belohnungsfunktion gefördert, welche den Lüftungsmodus innerhalb des Temperaturfensters weniger hart bestraft als den Kühlmodus. Deshalb versucht der Agent, nur mit Hilfe der Lüftung in den Sollbereich zurückzukehren und durch die demzufolge geringere Strafe seinen Reward zu maximieren. Allerdings ist diese Vorgehensweise nicht bei allen Temperaturüberschreitungen zu beobachten, was wiederum in mangelnder Lernaufreife oder entdeckten Abhängigkeiten von anderen Parametern begründet liegen kann. Aktionen, die innerhalb der Temperaturgrenzen getätigt werden, um die Nebenparameter Luftfeuchtigkeit oder -qualität zu beeinflussen, können lediglich zu einem Zeitpunkt in Abbildung 6.3a beobachtet

6. Ergebnis

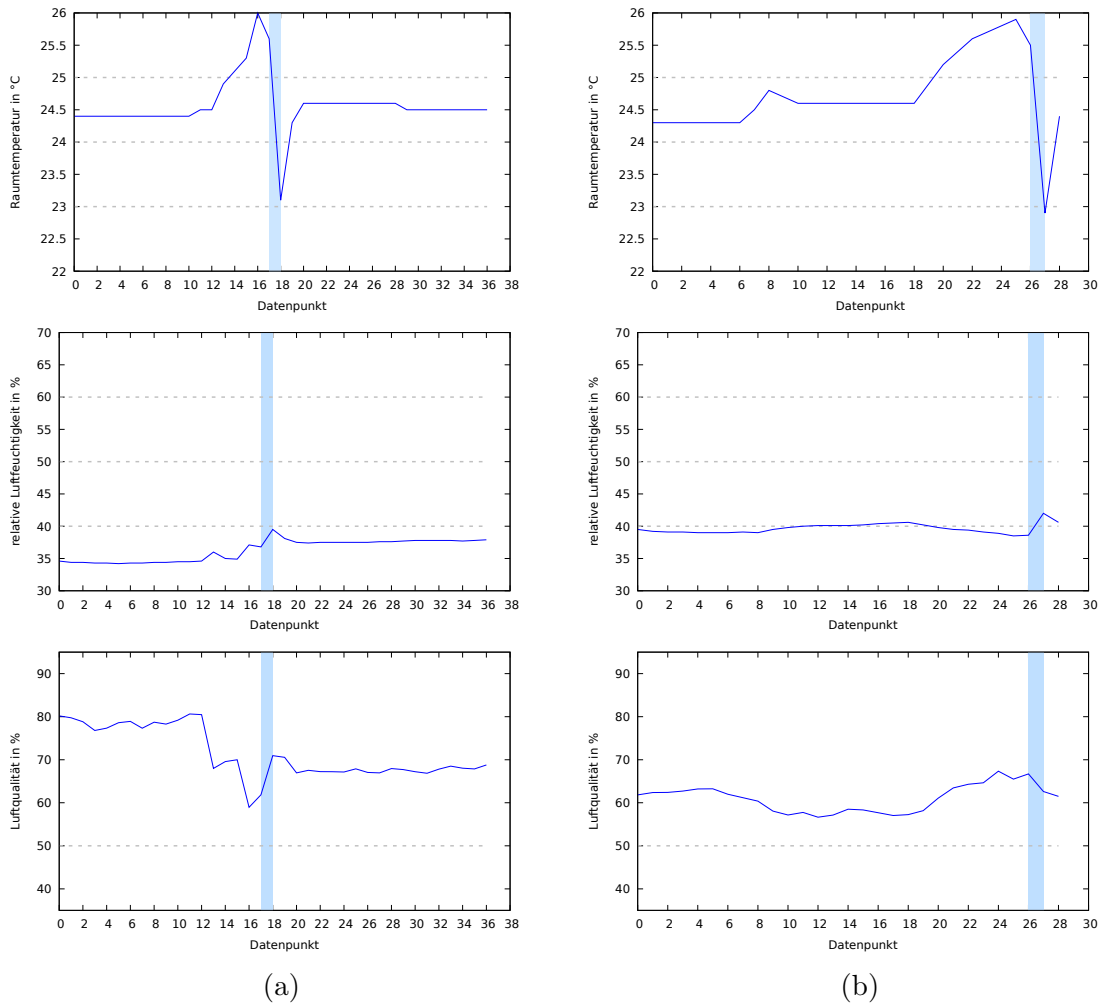


Abbildung 6.5.: Agent mit Kenntnis der Luftqualität im großen Raum

werden. Doch selbst dort könnte der Kühlbetrieb auf die ansteigende Temperatur oder die sinkende Luftqualität zurückzuführen sein. Vermutlich hat die Klimaanlage zu geringen Einfluss auf die beiden Nebenparameter, als dass sie entscheidende Veränderungen bei gleichzeitiger Einhaltung der Temperaturgrenzen bewirken könnte. Hinzu kommt die Gestaltung der Belohnungsfunktion, welche selbst in der letzten Ausführung für diese Aktionen noch zu hohe Bestrafungen bzgl. der Energieaufnahme vergibt.

Die Messwerte vom Normalbetrieb der Klimageräte, d. h. vom dauerhaften Kühlbetrieb, sind in Abbildung 6.6 und Abbildung 6.7 dargestellt. Sie wurden um 1 bzw. $0,5$ °C nach unten korrigiert, um die Abweichungen zu den internen Temperaturwerten der Klimageräte zu kompensieren. In Abbildung 6.6a ist klar ersichtlich, dass der Sollwert von 24 °C über große Strecken sehr strikt eingehalten wird. Dennoch kommt es in Abbildung 6.6a und Abbildung 6.7b stellenweise zu Abweichungen von fast 1 °C, die dann erst nach einem großen Überschwinger mit ca. 2 °C Unterschied

6. Ergebnis

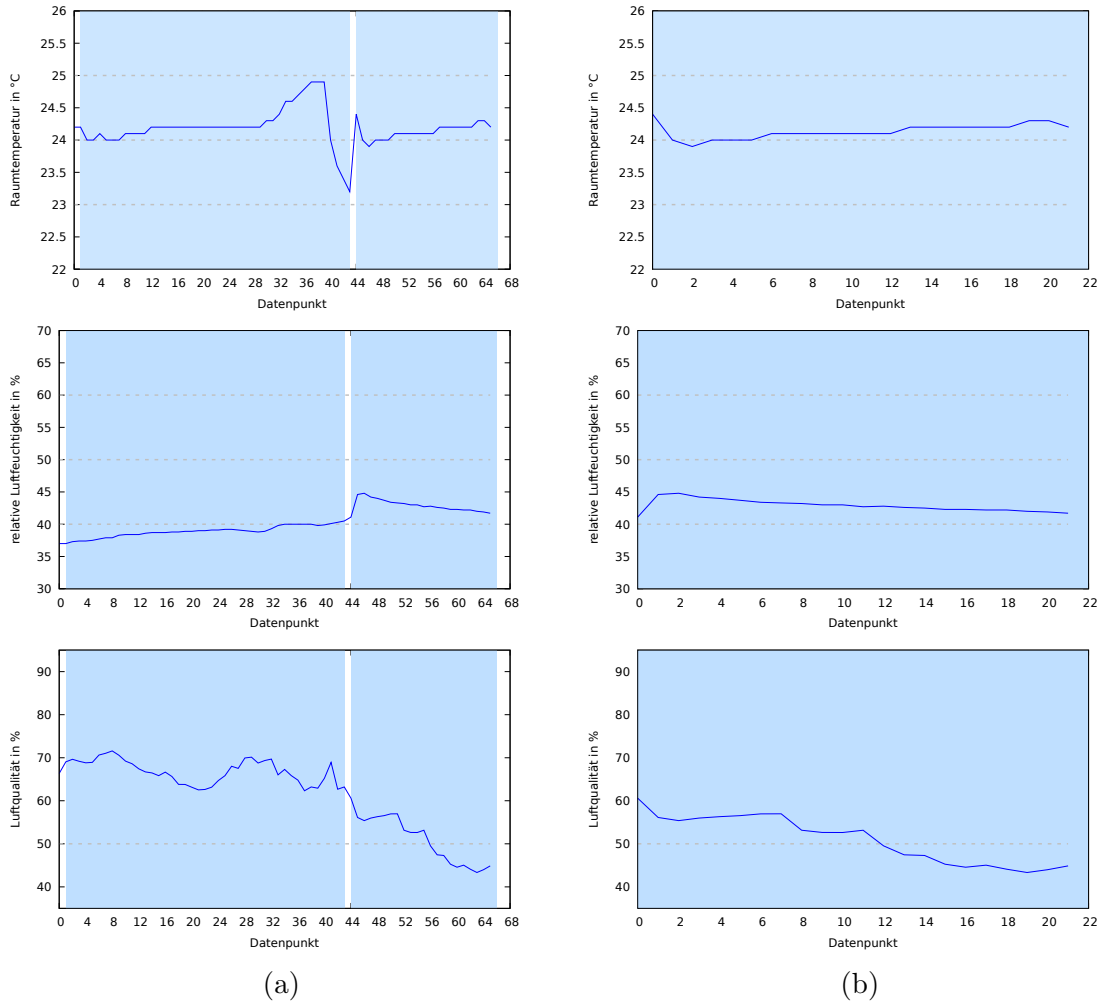


Abbildung 6.6.: Referenzbetrieb ohne KI im kleinen Raum

wieder ausgeglichen werden. Im erstgenannten Beispiel führte der plötzliche Temperaturunterschied sogar dazu, dass das Innengerät durch eine Person im Raum für kurze Zeit deaktiviert wurde. Die Schwankungen während des KI-Betriebs im kleinen Raum sind selten derart hoch. Im großen Raum hingegen tritt dies durch den gemeinsamen Betrieb von zwei Innengeräten und der damit höheren Kühlleistung öfter auf. Durch Konflikte mit der Hysterese der Klimageräte, bspw. erkennbar in Abbildung 6.4a, erfolgt manchmal keine Kühlung, obwohl sich das Gerät im Kühlmodus befindet. Das sorgt für nicht-deterministisches Verhalten aus Sicht des Agenten und bremst somit den Lernvorgang. Er erwartet eine Temperatursenkung, registriert aber manchmal einen Anstieg der Temperatur. Das beeinflusst wiederum die Anpassung seiner Bewertung dieser Aktion in die falsche Richtung.

Zusammenfassend kann gesagt werden, dass der KI-Agent durch die selbstständig erlernte Strategie zur Aktivierung der Innengeräte und Wahl der Betriebsmodi ein intelligentes Verhalten entwickelt hat. Die Einhaltung des festgelegten Temperatur-

6. Ergebnis

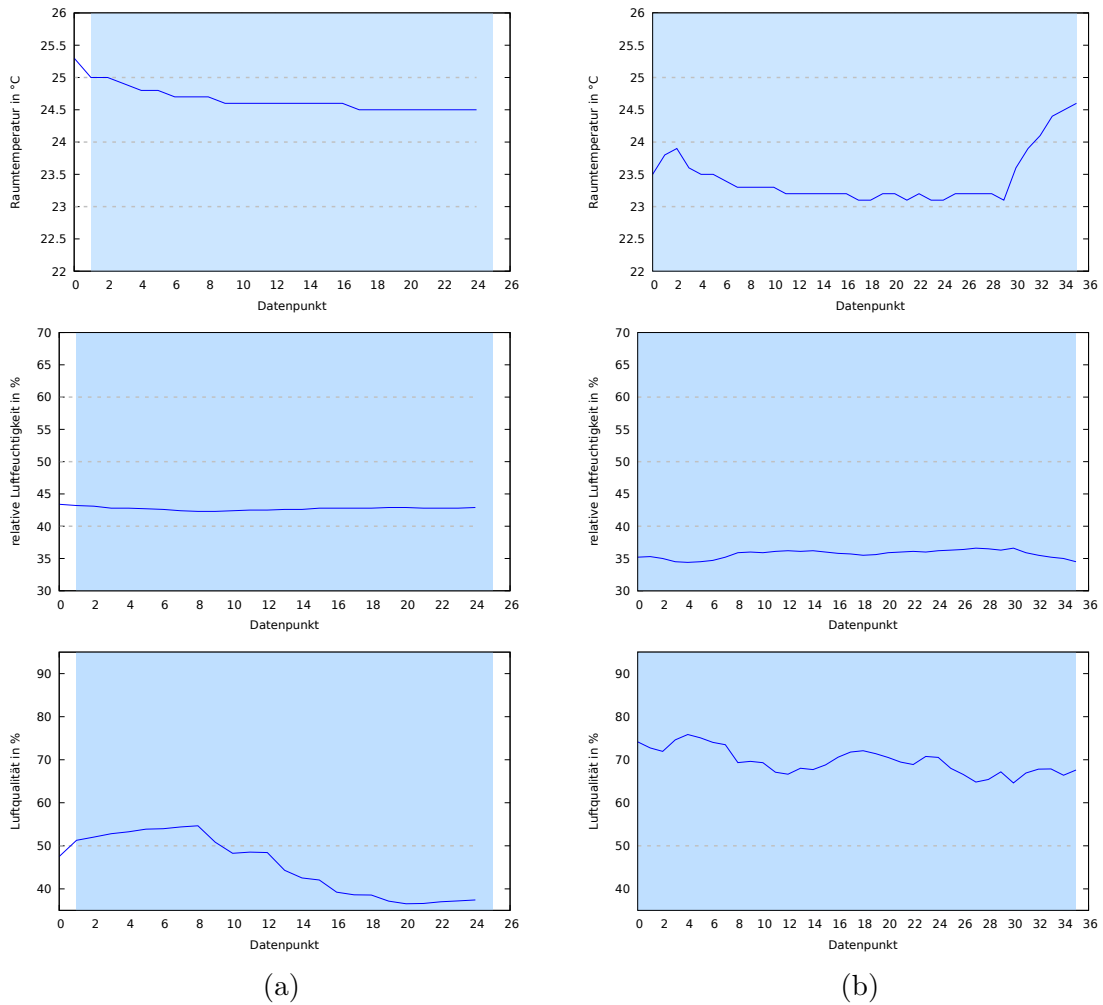


Abbildung 6.7.: Referenzbetrieb ohne KI im großen Raum

bereichs bei gleichzeitiger Reduktion des Energieverbrauchs gelingt dem Agenten bereits gut. Im Vergleich zum Normalbetrieb der Klimageräte spart das entwickelte System Energie durch die automatische Abschaltung und Umschaltung zwischen Kühlung und Lüftung sowie durch den Setpoint-Bereich. Die Statistiken zu den Testläufen können in Tabelle 6.5 nachvollzogen werden. Während des Dauerbetriebs, welcher im Normalfall genutzt wird, liegt eine Aktivitätsrate des Innengeräts von 100 % im Kühlmodus vor. Es wird somit fast dauerhaft eine geringe Kühlleistung angefordert, die von einem Außengerät bereitgestellt werden muss. Die KI-Steuerung hingegen war bei den Tests in der Lage, die Aktivitätsrate auf maximal 16 % durch den Spielraum von 1 °C um den Sollwert herum zu senken. Beim Testlauf unter Einbeziehung der Luftqualität im kleinen Raum entfielen davon zudem ca. 41 % auf den Lüftungsmodus. Die Lüftung bzw. Kühlung erfolgt stets in zehnmütigen Schritten. Damit allein wird bereits ein großer Teil der Energieaufnahme des Gebläses, welches bei eingeschaltetem Klimagerät dauerhaft aktiviert ist, gespart. Hinzu kommt die

6. Ergebnis

Raum	Variante	Schritte		
		Gesamt	davon Lüftung	davon Kühlung
klein	Luftfeuchtigkeit	32	0	3
	Luftqualität	110	7	10
	ohne KI	86	0	85
groß	Luftfeuchtigkeit	52	0	6
	Luftqualität	68	0	2
	ohne KI	58	0	57

Tabelle 6.5.: Ergebnisstatistik der Testläufe

stoßweise Anforderung der Kühlleistung, für deren Bereitstellung ein Außengerät nur kurze Zeit aktiv sein muss. Welche der beiden Varianten für die Energieaufnahme, den Betrieb der Außengeräte sowie Übertragungsverluste günstiger ist, lässt sich durch die unbekanntenen Interna der VRV-Technik nicht ermitteln. Außerdem ist die Energieaufnahme der KI-Steuerung durch den Setpoint von -2 °C zur Umgehung der Hysterese höher, als sie eigentlich für einen Kühlimpuls sein müsste. Weitere Verbesserungen sind durch Fortführen des Trainings und ggf. Anpassung der Systemparametern zu erwarten. Luftfeuchtigkeit und Luftqualität wurden als Nebenparameter vom Agenten bisher kaum berücksichtigt. Einerseits kann der mangelnde Einfluss der Klimaanlage auf diese Größen und andererseits die kurze Lerndauer ursächlich dafür sein. Erkennbar sind zudem die Unterschiede bzgl. des erlernten Verhaltens zwischen den Agenten der beiden Räume, insbesondere bei der Nutzung des Lüftungsmodus. Wie bereits zu Beginn dieses Kapitels erwähnt, muss beachtet werden, dass die Testläufe zwar auf ähnlichen, aber nicht den gleichen Bedingungen beruhen und eine gewisse Toleranz der Resultate somit berücksichtigt werden sollte.

7. Fazit und Ausblick

Nachdem das entstandene System im vorherigen Kapitel einigen Praxistests unterzogen wurde, wird die Arbeit nun abschließend resümiert. Dabei werden Probleme, die bei der Entwicklung auftraten, genannt und Lösungsmöglichkeiten diskutiert. Außerdem ist ein Ausblick enthalten, der Informationen für mögliche weiterführende Projekte bereitstellt.

Im Großen und Ganzen lässt sich die Arbeit und das entstandene System als Erfolg verzeichnen. Die zu Beginn gesteckten Ziele wurden größtenteils erreicht. Weitere Verbesserungen sind durch Fortführen des Trainings mit den KI-Agenten zu erwarten. Dabei könnte zudem der bereits eingebundene Heizmodus verwendet und bewertet werden. Das resultierende System ist modular und kann mit wenig Aufwand auf weitere Räume oder ein ganzes Gebäude ausgeweitet werden. Dazu muss lediglich ein weiterer Sensorknoten pro Raum und in Reichweite des restlichen Netzwerks platziert werden. Anschließend kann er durch Starten eines weiteren KI-Clients in das System eingebunden werden. Unter Umständen muss dazu der Aktionsraum auf die Betriebsmodi des jeweiligen Innengeräts der Klimaanlage angepasst werden. Das System kann zudem auf andere, ähnlich funktionierende Klimatisierungslösungen angewendet werden, indem die Schnittstelle zum Daikin iTC mit einer passenden Alternative ersetzt wird. Die Erstellung des maßgeschneiderten Linux-Betriebssystems für die Sensorknoten war ein voller Erfolg. Es mussten zwar viele kleinere Änderungen, insbesondere am Device-Tree, eingepflegt werden, dabei kam es aber nicht zu Problemen. Die ausführliche Dokumentation des Yocto Projects [86] war dabei eine große Hilfe. In Kombination mit dem Mender-Updater, dessen Client sich sehr gut in Yocto integrieren und dessen Server sich ebenso leicht einrichten lässt, sind Systemaktualisierungen ohne Schwierigkeiten over-the-air durchführbar. Lediglich der Bandbreitenbedarf der Update-Artefakte ist für mobile Einsätze etwas zu hoch. Da das Projekt Binary Delta-Updates bereits angekündigt hat und außerdem von der Europäischen Kommission gefördert wird, bleibt der weitere Entwicklungsverlauf interessant. Im Vergleich zu einigen der verwandten Arbeiten aus Abschnitt 3.1 tauchten auch praxisbezogene Probleme auf, die in Simulationsumgebungen nicht existent sind. Dazu gehört die Umsetzung des Mesh-Netzwerks, welches durch die Aktivierung der Verschlüsselung auf verschiedene Arten außer Gefecht gesetzt wurde. Andere Projekte, wie OpenWRT, verwenden veränderten Quelltext, um funktionierende Implementierungen zu erhalten. Diese müssten genau analysiert und auf die hier verwendete Yocto-Umgebung angepasst werden, was viel Arbeit mit sich bringt. Im Endeffekt musste das Sensornetzwerk hier unverschlüsselt betrieben werden, was aufgrund der geringen Reichweite im Gebäude letztendlich nicht kritisch war. Dies ist der nächste Nachteil des Mesh-Netzwerks bzw. der verwendeten Hardware. Zum

7. Fazit und Ausblick

Teil brachen die Verbindungen schon beim Versuch der Kommunikation durch eine Wand hindurch zusammen. Personen im Raum verschlechterten die Situation zusätzlich, sodass die Kommunikation zwischen den BeagleBones nur nach Experimentieren mit der Position und Ausrichtung einigermaßen stabil funktionierte. Die Umsetzung des KI-Agenten mit TensorFlow war durch die unvollständige Dokumentation stellenweise kompliziert. Durch die freie Verfügbarkeit der Quelltexte hat sich die Bibliothek aber letztendlich als beherrschbar erwiesen. Das Training in der realen Welt war, wie erwartet, sehr langwierig. Im Verhältnis zu den verwandten Arbeiten, in denen von mehreren Trainingsjahren in der Simulation gesprochen wird, stand hier lediglich ein Bruchteil der Zeit zu Verfügung. Dies wurde durch das Anpassen der Lernrate und der Reduktion von Zustands- und Aktionsraum versucht zu kompensieren. In Verbindung mit dem Pretraining des DQfD-Agenten konnte zumindest die anfängliche Hürde bis zu den ersten sichtbaren Erfolgen verkürzt werden. Mit einer längeren Trainingsphase können ggf. bessere Ergebnisse erreicht und zur weiteren Verbesserung komplexere Zustände ermöglicht werden. Das wiederum kann dem Agenten zu einem vollständigeren Verständnis seiner Umwelt und somit zu einer intelligenteren Strategie verhelfen. Aufgrund der Phase zur Findung einer geeigneten KI-Konfiguration ist viel Zeit verstrichen, bis überhaupt Testläufe durchgeführt werden konnten. Damit war die Zeit zur Evaluation weiterer Szenarien bzw. zur Weiterführung der bisherigen leider beschränkt. Ebendieser Zeitraum war dennoch wichtig, um eine Datenbasis für das Pretraining des Agenten zu schaffen. Durch die unbekanntenen Interna des Daikin HVAC-Systems und der VRV-Technik blieb die Schätzung des Energiebedarfs leider sehr abstrakt. Die Wahl eines Setpoints von -2 °C im Kühlmodus ist zudem nicht optimal bzgl. Energieaufnahme und Komfort, war aber zur Vermeidung von nicht-deterministischem Verhalten durch die Hysterese der bestehenden Steuerung notwendig. Für zukünftige Arbeiten könnte diesbezüglich eine automatische Angleichung des Setpoints an die Außentemperatur stattfinden. Zusätzlich werden dadurch neue Möglichkeiten zum Energiesparen eröffnet. Denkbar wäre auch die Einbeziehung eines Fenstersteuerungssystem zur Kombination mit natürlicher Belüftung, wie dies bereits in einer verwandten Arbeit [92] theoretisch beleuchtet wird. Durch die Ausstattung der BeagleBones mit weiterer Sensorik, wie Bewegungsmelder oder Helligkeitssensoren, könnte man das System noch intelligenter machen. Es wäre damit möglich, die KI mit Hilfe der Belohnungsfunktion so zu trainieren, dass sie Aktionen in Abhängigkeit der Raumbellegung, des Beleuchtungszustands oder auch der Uhrzeit tätigt. Zusätzliche Wetterdaten und -vorhersage können das System zudem darauf ausrichten, vorausschauend zu handeln. Dazu hat der Logging-Client in der Variante *global* bereits Aufzeichnungen entsprechender Daten durchgeführt. Dabei muss aber berücksichtigt werden, dass diese Erweiterungen wiederum einen größeren Lernaufwand mit sich bringen und die benötigte Lerdauer beim Realwelteinsatz demzufolge erhöhen. Zusammenfassend ist die Praxistauglichkeit des entwickelten Systems innerhalb gewisser Grenzen gegeben. Es muss jedoch stets berücksichtigt werden, dass hier eine bestehende Klimatisierungslösung erweitert wurde. Diese schränkt die Möglichkeiten des KI-Agenten, welcher im Endeffekt nur vorgeschaltet wird, sichtbar ein. Ein möglicherweise besser funktionierender An-

7. *Fazit und Ausblick*

satz, welcher die genannten Probleme prinzipbedingt umgeht, umfasst das komplette Ersetzen der bestehenden Steuerung mit einer Neuentwicklung auf Grundlage einer KI. Eine solche Entwicklung umfasst jedoch wesentlich tiefere Eingriffe in die Klimatisierungstechnik, die bis zur Funktionsunfähigkeit führen können und somit im Rahmen dieser Arbeit nicht in Betracht gezogen wurden.

Literaturverzeichnis

- [1] Adafruit BeagleBone IO, . URL <https://github.com/adafruit/adafruit-beaglebone-io-python>. abgerufen am 15.12.2018.
- [2] Adafruit BME680 Dokumentation, . URL <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bme680-humidity-temperature-barometric-pressure-voc-gas.pdf>. abgerufen am 05.12.2018.
- [3] Adafruit BME680 Bibliothek, . URL https://github.com/adafruit/Adafruit_BME680. abgerufen am 05.12.2018.
- [4] AdaFruit Fritzing Library, . URL <https://github.com/adafruit/Fritzing-Library>. abgerufen am 15.12.2018.
- [5] Aktualizr. URL <https://github.com/advancedtelematic/aktualizr>. abgerufen am 03.12.2018.
- [6] Advanced Message Queuing Protocol. URL <https://www.amqp.org>. abgerufen am 03.12.2018.
- [7] Simultaneous Authentication of Equals (SAE). URL <https://github.com/cozybit/authsae>. abgerufen am 15.12.2018.
- [8] Barebox. URL <https://www.barebox.org>. abgerufen am 03.12.2018.
- [9] B.A.T.M.A.N. Advanced. URL <https://www.open-mesh.org/projects/batman-adv/wiki>. abgerufen am 03.12.2018.
- [10] BeagleBone Device-Tree Pinmux, . URL <https://github.com/beagleboard/linux/blob/4.14/arch/arm/boot/dts/am335x-bone-common-universal.dtsi>. abgerufen am 15.11.2018.
- [11] BeagleBone Kernel Repository, . URL <https://github.com/beagleboard/linux>. abgerufen am 15.12.2018.
- [12] BeagleBone Pinmux Helper, . URL <https://github.com/beagleboard/linux/commit/2dcc1b4e3eb9987870eb5f4cb01a3549b396badc>. abgerufen am 15.12.2018.
- [13] BeagleBone USB-Netzwerk, . URL https://elinux.org/BeagleBone_Usb_Networking. abgerufen am 15.12.2018.

LITERATURVERZEICHNIS

- [14] BeagleBone Black Wireless. URL <https://beagleboard.org/black-wireless>. abgerufen am 15.12.2018.
- [15] BeagleCore BCS1 Starter-Kit. URL <http://beaglecore.com/products/bcs1str.html>. abgerufen am 15.12.2018.
- [16] Use Case Qualification Worksheet. URL https://daks2k3a4ib2z.cloudfront.net/574f4712c868ed543a989231/59c08a896eb47c000139509d_V1%20Bonsai%20Use%20Case%20Qualification%20Worksheet.pdf. abgerufen am 15.12.2018.
- [17] Bosch Sensortec BME680. URL https://www.bosch-sensortec.com/bst/products/all_products/bme680. abgerufen am 05.12.2018.
- [18] Buildroot, . URL <https://www.buildroot.org>. abgerufen am 03.12.2018.
- [19] Buildroot External Tree, . URL <https://buildroot.org/downloads/manual/manual.html#outside-br-custom>. abgerufen am 03.12.2018.
- [20] Buildroot Layered Customizations, . URL <https://buildroot.org/downloads/manual/manual.html#customize-dir-structure>. abgerufen am 03.12.2018.
- [21] Constrained Application Protocol. URL <http://coap.technology>. abgerufen am 03.12.2018.
- [22] intelligent Touch Controller Http Interface Option Commissioning Manual, . URL [http://www.tinlavr.ro/parteneri/2_VRV/4_Control_Systems/03_intelligentTouchController/Documents/Installation_Manuals/DCS007A51_ver4.41.00_HTTPComissioningManual\(CB08A081A\)_tcm135-140696.pdf](http://www.tinlavr.ro/parteneri/2_VRV/4_Control_Systems/03_intelligentTouchController/Documents/Installation_Manuals/DCS007A51_ver4.41.00_HTTPComissioningManual(CB08A081A)_tcm135-140696.pdf). abgerufen am 04.12.2018.
- [23] Abbildung: Multi-Split-System, . URL https://www.daikin.com/products/ac/lineup/split_multi_split/images/pic_overview_02.jpg. abgerufen am 07.01.2019.
- [24] Abbildung: Single-Split-System, . URL https://www.daikin.com/products/ac/lineup/split_multi_split/images/pic_overview_01.jpg. abgerufen am 07.01.2019.
- [25] intelligent Touch Controller Startup Manual, . URL https://www.daikin.de/content/dam/document-library/operation-manuals/ctrl/DCS601C51%20-%20startup%20manual%20_Operation%20manuals_English.pdf. abgerufen am 14.12.2018.
- [26] DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. URL <https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40>. abgerufen am 03.12.2018.

LITERATURVERZEICHNIS

- [27] Definition künstlicher Intelligenz. URL <https://wirtschaftslexikon.gabler.de/definition/kuenstliche-intelligenz-ki-40285/version-263673>. abgerufen am 15.12.2018.
- [28] Eintrag im Duden zu Intelligenz. URL <https://www.duden.de/rechtschreibung/Intelligenz>. abgerufen am 15.12.2018.
- [29] Artikel zu 802.11s von Elektronik-Kompendium. URL <http://www.elektronik-kompendium.de/sites/net/1408051.htm>. abgerufen am 13.12.2018.
- [30] EnergyPlus Gebäudeenergiesimulation. URL <https://energyplus.net>. abgerufen am 15.12.2018.
- [31] freeboard, . URL <https://freeboard.io>. abgerufen am 03.12.2018.
- [32] freeboard auf GitHub, . URL <https://github.com/Freeboard/freeboard>. abgerufen am 08.01.2019.
- [33] Kritische Lücken in FreeRTOS gefährden Embedded-Systeme und IoT-Geräte, . URL <https://www.heise.de/security/meldung/Kritische-Luecken-in-FreeRTOS-gefaehrden-Embedded-Systeme-und-IoT-Geraete-4199583.html>. abgerufen am 11.12.2018.
- [34] garage-Framework. URL <https://github.com/rlworkgroup/garage>. abgerufen am 03.12.2018.
- [35] Geckoboard. URL <https://www.geckoboard.com>. abgerufen am 03.12.2018.
- [36] Eclipse Hawkbit. URL <https://www.eclipse.org/hawkbit>. abgerufen am 03.12.2018.
- [37] Artikel zu neuronalen Netzen. URL <https://epaper.heise.de/download/archiv/262094635d97/ct.16.06.130-135.pdf>. abgerufen am 15.12.2018.
- [38] Vorlesung Einführung in die KI (HU Berlin). URL <https://www.informatik.hu-berlin.de/de/forschung/gebiete/ki/lehre/ws0405/vlki-skript/eki0405NeuronaleNetze.pdf>. abgerufen am 15.12.2018.
- [39] Studie zum Energiebedarf für Kältetechnik. URL <https://www.baulinks.de/webplugin/2011/0746.php4>. abgerufen am 12.12.2018.
- [40] 802.11s im Linux Wireless Wiki. URL <https://wireless.wiki.kernel.org/en/developers/Documentation/ieee80211/802.11s>. abgerufen am 15.12.2018.
- [41] Das Drama um Deutschlands Klimaanlage, . URL <https://www.welt.de/sonderthemen/ish-messe/article138114169/Das-Drama-um-Deutschlands-Klimaanlagen.html>. abgerufen am 11.12.2018.

LITERATURVERZEICHNIS

- [42] Artikel des Umweltbundesamts zur Gebäudeklimatisierung, . URL <https://www.umweltbundesamt.de/themen/wirtschaft-konsum/produkte/fluorierte-treibhausgase-fckw/anwendungsbereiche-emissionsminderung/gebäudeklimatisierung>. abgerufen am 11.12.2018.
- [43] Klimageräte im Sommer sparsam einsetzen, . URL <http://www.bfe.admin.ch/energie/00588/00589/00644/index.html?lang=de&msg-id=49667>. abgerufen am 11.12.2018.
- [44] Mender OTA-Updater, . URL <https://mender.io>. abgerufen am 03.12.2018.
- [45] Mender Produktiveinsatz, . URL <https://docs.mender.io/1.5/administration/production-installation>. abgerufen am 03.12.2018.
- [46] Mender Signierung und Verschlüsselung, . URL <https://docs.mender.io/1.5/artifacts/signing-and-verification>. abgerufen am 15.12.2018.
- [47] Mender Testumgebung, . URL <https://docs.mender.io/1.5/getting-started/create-a-test-environment>. abgerufen am 03.12.2018.
- [48] Mender Konfiguration, . URL <https://docs.mender.io/1.5/artifacts/variables>. abgerufen am 03.12.2018.
- [49] meta-updater. URL <https://github.com/advancedtelematic/meta-updater>. abgerufen am 03.12.2018.
- [50] Kriminelle bieten Mirai-Botnetz mit 400.000 IoT-Geräten zur Miete an. URL <https://www.heise.de/security/meldung/Kriminelle-bieten-Mirai-Botnetz-mit-400-000-IoT-Geraeten-zur-Miete-an-3504584.html>. abgerufen am 11.12.2018.
- [51] Eclipse Mosquitto Broker. URL <https://mosquitto.org>. abgerufen am 15.12.2018.
- [52] Message Queuing Telemetry Transport. URL <http://mqtt.org>. abgerufen am 03.12.2018.
- [53] Node-RED, . URL <https://nodered.org>. abgerufen am 03.12.2018.
- [54] Node-RED Dashboard, . URL <https://flows.nodered.org/node/node-red-dashboard>. abgerufen am 15.12.2018.
- [55] Node.js, . URL <https://github.com/node/source/distributions>. abgerufen am 05.12.2018.
- [56] Node-RED Autostart, . URL <https://nodered.org/docs/getting-started/running>. abgerufen am 05.12.2018.

LITERATURVERZEICHNIS

- [57] Node-RED Passwortverschlüsselung, . URL <https://nodered.org/docs/security.html#generating-the-password-hash>. abgerufen am 15.12.2018.
- [58] open80211s Implementierung, . URL <https://github.com/o11s/open80211s/wiki/HOWTO>. abgerufen am 15.12.2018.
- [59] Problem mit Mesh-Verschlüsselung bei OpenWRT, . URL <https://github.com/openwrt/mt76/issues/72>. abgerufen am 15.12.2018.
- [60] Problem mit Mesh-Verschlüsselung bei open80211s, . URL <https://github.com/o11s/open80211s/issues/50>. abgerufen am 15.12.2018.
- [61] OpenEmbedded Layer Index. URL <http://layers.openembedded.org>. abgerufen am 03.12.2018.
- [62] Initiative *One Laptop Per Child*. URL <http://one.laptop.org>. abgerufen am 04.12.2018.
- [63] Open Platform Communications Unified Architecture. URL <https://www.opcfoundation.org>. abgerufen am 03.12.2018.
- [64] OpenStudio Software-Umgebung für EnergyPlus, . URL <https://www.openstudio.net>. abgerufen am 15.12.2018.
- [65] OpenWrt Buildroot-Nutzung, . URL <https://openwrt.org/de/doc/howto/buildroot.exigence>. abgerufen am 03.12.2018.
- [66] libostree. URL <https://github.com/ostreedev/ostree>. abgerufen am 03.12.2018.
- [67] Pengutronix e.K. URL <https://www.pengutronix.de>. abgerufen am 03.12.2018.
- [68] Pimoroni BME680 Tutorial, . URL <https://learn.pimoroni.com/tutorial/sandyj/getting-started-with-bme680-breakout>. abgerufen am 05.12.2018.
- [69] Pimoroni BME680 Bibliothek, . URL <https://github.com/pimoroni/bme680-python>. abgerufen am 15.12.2018.
- [70] Yocto-Referenzdistribution Poky. URL <https://www.yoctoproject.org/software-item/poky/>. abgerufen am 15.12.2018.
- [71] TensorForce Pretraining. URL https://github.com/reinforceio/tensorforce/blob/master/tensorforce/tests/test_dqfd_agent.py. abgerufen am 15.12.2018.
- [72] Abbildung: Prinzip einer Klimaanlage. URL <https://www.energielexikon.info/img/split-klimageraet.png>. abgerufen am 08.01.2019.

LITERATURVERZEICHNIS

- [73] PTXdist. URL <https://www.ptxdist.org>. abgerufen am 03.12.2018.
- [74] Robust Auto-Update Controller. URL <https://www.rauc.io>. abgerufen am 03.12.2018.
- [75] RVI SOTA-Client, . URL https://github.com/genivi/rvi_sota_client. abgerufen am 03.12.2018.
- [76] RVI SOTA-Server, . URL https://github.com/genivi/rvi_sota_server. abgerufen am 03.12.2018.
- [77] Artikel zu Reinforcement Learning. URL http://scholarpedia.org/article/Reinforcement_learning. abgerufen am 15.12.2018.
- [78] swupd-Client, . URL <https://github.com/clearlinux/swupd-client>. abgerufen am 03.12.2018.
- [79] SWUpdate-Agent, . URL <https://github.com/sbabic/swupdate>. abgerufen am 03.12.2018.
- [80] TUCool – Adaptive Klimasteuerung für luftgekühlte Rechenzentren. URL <https://www.tu-chemnitz.de/urz/projekte/tucool.html>. abgerufen am 03.12.2018.
- [81] Daikin VRV-Technik. URL https://www.daikin.de/de_de/ueber-daikin/fuehrende-technologien/variables-kaeltemittelvolumen.html. abgerufen am 15.12.2018.
- [82] Artikel zu schwacher und starker KI. URL http://www.informatik.uni-oldenburg.de/~iug08/ki/Grundlagen_Starke_KI_vs._Schwache_KI.html. abgerufen am 15.12.2018.
- [83] Linux WPA/WPA2/IEEE 802.1X Supplicant, . URL http://w1.fi/wpa_supplicant. abgerufen am 15.12.2018.
- [84] WPA Supplicant mit Mesh-Support, . URL https://github.com/cozybit/wpa_supplicant-o11s-legacy. abgerufen am 15.12.2018.
- [85] Yocto Project, . URL <https://www.yoctoproject.org>. abgerufen am 03.12.2018.
- [86] Yocto Project Mega Manual, . URL <https://www.yoctoproject.org/docs/2.5/mega-manual/mega-manual.html>. abgerufen am 15.12.2018.
- [87] Artikel zu Systemaktualisierungen im Yocto Project Wiki, . URL https://wiki.yoctoproject.org/wiki/System_Update. abgerufen am 15.12.2018.

LITERATURVERZEICHNIS

- [88] René Bergelt, Wolfram Hardt, and Matthias Vodel. Verfahren zur optimierung der energieeffizienz in drahtlosen, energieautarken sensornetzen. In *Informatiksymposium Chemnitz*, pages 3–14. TU Chemnitz, Fachbereich Informatik, July 2012. Chemnitzer Informatik-Berichte CSR-12-01.
- [89] René Bergelt, Matthias Vodel, and Wolfram Hardt. Generische datenerfassung und aufbereitung im kontext verteilter, heterogener sensor-aktor-systeme. Technical report, TU Chemnitz, August 2012.
- [90] René Bergelt, Matthias Vodel, and Wolfram Hardt. Energy-efficient handling of big data in embedded, wireless sensor networks. In *Proceedings of the Sensors & Applications Symposium (SAS2014)*, Queenstown, Neuseeland, February 2014. IEEE Computer Society. ISBN 978-1-4799-2180-5.
- [91] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- [92] Yujiao Chen, Leslie K. Norford, Holly W. Samuelson, and Ali Malkawi. Optimal control of hvac and window systems for natural ventilation through reinforcement learning. *Energy and Buildings*, 169:195 – 205, 2018. ISSN 0378-7788. doi: <https://doi.org/10.1016/j.enbuild.2018.03.051>. URL <http://www.sciencedirect.com/science/article/pii/S0378778818302184>.
- [93] François Chollet et al. Keras, 2015. URL <https://keras.io>.
- [94] K. Dalamagkidis, D. Kolokotsa, K. Kalaitzakis, and G.S. Stavrakakis. Reinforcement learning for energy conservation and comfort in buildings. *Building and Environment*, 42(7):2686 – 2698, 2007. ISSN 0360-1323. doi: <https://doi.org/10.1016/j.buildenv.2006.07.010>. URL <http://www.sciencedirect.com/science/article/pii/S0360132306001880>.
- [95] Chris Delnero. Neural networks and pi control using steady state prediction applied to a heating coil. 12 2018.
- [96] Yan Duan, Xi Chen, Rein Houthoofd, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016. URL <http://arxiv.org/abs/1604.06778>.
- [97] Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84(1):137–169, Jul 2011. ISSN 1573-0565. doi: 10.1007/s10994-011-5235-x. URL <https://doi.org/10.1007/s10994-011-5235-x>.
- [98] Jeffrey Heaton. *Introduction to Neural Networks for Java*. 01 2008. ISBN 1604390085.

LITERATURVERZEICHNIS

- [99] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017. URL <http://arxiv.org/abs/1704.03732>.
- [100] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke. Ieee 802.11s: The wlan mesh standard. *IEEE Wireless Communications*, 17(1):104–111, February 2010. ISSN 1536-1284. doi: 10.1109/MWC.2010.5416357.
- [101] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.
- [102] R. M. Kretchmar, P. M. Young, C. W. Anderson, D. C. Hittle, M. L. Anderson, and C. C. Delnero. Robust reinforcement learning control. In *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, volume 2, pages 902–907 vol.2, June 2001. doi: 10.1109/ACC.2001.945833.
- [103] R. Matthew Kretchmar, Peter M. Young, Charles W. Anderson, Douglas C. Hittle, Michael L. Anderson, and Christopher C. Delnero. Robust reinforcement learning control with static and dynamic stability. *International Journal of Robust and Nonlinear Control*, 11(15):1469–1500. doi: 10.1002/rnc.670. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.670>.
- [104] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007. URL <http://www.dkriesel.com>.
- [105] Daniel Kriesten. Systementwurf eingebetteter heterogener rekonfigurierbarer Systeme mit Linux-Betriebssystem am Beispiel einer modularen Plattform zur Erfassung und Verarbeitung von Sensordaten, 01 2015. URL <http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-154966>.
- [106] Pat Langley. The changing science of machine learning. *Machine Learning*, 82(3):275–279, Mar 2011. ISSN 1573-0565. doi: 10.1007/s10994-011-5242-y. URL <https://doi.org/10.1007/s10994-011-5242-y>.
- [107] Matthias Plappert. keras-rl, 2016. URL <https://github.com/keras-rl/keras-rl>.
- [108] Michael Schaarschmidt, Alexander Kuhnle, and Kai Fricke. Tensorforce: A tensorflow library for applied reinforcement learning. Web page, 2017. URL <https://github.com/reinforceio/tensorforce>.
- [109] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010. ISBN 1608454924, 9781608454921.

LITERATURVERZEICHNIS

- [110] A.R. Trott and T. Welch. 28 - air-conditioning methods. In A.R. Trott and T. Welch, editors, *Refrigeration and Air Conditioning (Third Edition)*, pages 297 – 315. Butterworth-Heinemann, Oxford, third edition edition, 2000. ISBN 978-0-7506-4219-4. doi: <https://doi.org/10.1016/B978-075064219-4/50028-5>. URL <http://www.sciencedirect.com/science/article/pii/B9780750642194500285>.
- [111] Matthias Vodel. Funkstandardübergreifende Kommunikation in Mobilen Ad Hoc Netzwerken, 07 2010. URL <http://nbn-resolving.de/urn:nbn:de:bsz:ch1-201001164>.
- [112] Matthias Vodel. Energieeffiziente Kommunikation in verteilten, eingebetteten Systemen, 09 2015. URL <http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-131891>.
- [113] Matthias Vodel and Marc Ritter. Adaptive sensor data fusion for efficient climate control systems. In *Proceedings of the 17th International Conference on Human-Computer Interaction*, pages 582–593. Springer Verlag, August 2015. ISBN 978-3-319-20680-6.
- [114] Matthias Vodel, Mirko Lippmann, and Wolfram Hardt. Dynamic channel management for advanced, energy-efficient sensor-actor-networks. In *Proceedings of the World Congress on Information and Communication Technologies (WICT2011)*, pages 419–424, Mumbai / India, December 2011. IEEE Computer Society. ISBN 978-1-4673-0125-1.
- [115] Matthias Vodel, René Bergelt, Matthias Glockner, and Wolfram Hardt. Synchronised data logging, processing and visualisation in heterogeneous sensor networks. In *Proceedings of the International Conference on Data Engineering and Internet Technology*, pages 1070–1074, Kuta / Indonesien, January 2012. Springer Verlag. ISBN 978-3-642-28806-7.
- [116] Matthias Vodel, René Bergelt, and Wolfram Hardt. A generic data processing framework for heterogeneous sensor-actor-networks. *International Journal On Advances in Intelligent Systems*, 5(3-4):483–492, December 2012. ISSN 1942-2679.
- [117] Matthias Vodel, René Bergelt, and Wolfram Hardt. Grease framework - generic reconfigurable evaluation and aggregation of sensor data. In *Proceedings of the 2nd International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies (InfoSys2012)*, St. Maarten / Netherlands, March 2012. IARIA. ISBN 978-1-61208-189-2.
- [118] Matthias Vodel, Mirko Lippmann, and Wolfram Hardt. Resource management for advanced, heterogeneous sensor-actor-networks. In *Proceedings of the 11th International Conference on Networks (ICN2012)*, Reunion Island / France, February 2012. IARIA. ISBN 978-1-61208-183-0.

LITERATURVERZEICHNIS

- [119] Yuan Wang, Kirubakaran Velswamy, and Biao Huang. A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems. *Processes*, 5(3), 2017. ISSN 2227-9717. doi: 10.3390/pr5030046. URL <http://www.mdpi.com/2227-9717/5/3/46>.
- [120] Tianshu Wei, Yanzhi Wang, and Qi Zhu. Deep reinforcement learning for building hvac control. In *Proceedings of the 54th Annual Design Automation Conference 2017, DAC '17*, pages 22:1–22:6, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4927-7. doi: 10.1145/3061639.3062224. URL <http://doi.acm.org/10.1145/3061639.3062224>.
- [121] Yohei Yamaguchi, Noritaka Shigei, and H Miyaiima. Air conditioning control system learning sensory scale based on reinforcement learning. *Lecture Notes in Engineering and Computer Science*, 1:1–6, 03 2015.

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

- CSR-10-01** Maximilian Eibl, Jens Kürsten, Robert Knauf, Marc Ritter , Workshop Audiovisuelle Medien, Mai 2010, Chemnitz
- CSR-10-02** Thomas Reichel, Gudula Rünger, Daniel Steger, Haibin Xu, IT-Unterstützung zur energiesensitiven Produktentwicklung, Juli 2010, Chemnitz
- CSR-10-03** Björn Krellner, Thomas Reichel, Gudula Rünger, Marvin Ferber, Sascha Hunold, Thomas Rauber, Jürgen Berndt, Ingo Nobbers, Transformation monolithischer Business-Softwaresysteme in verteilte, workflowbasierte Client-Server-Architekturen, Juli 2010, Chemnitz
- CSR-10-04** Björn Krellner, Gudula Rünger, Daniel Steger, Anforderungen an ein Datenmodell für energiesensitive Prozessketten von Powertrain-Komponenten, Juli 2010, Chemnitz
- CSR-11-01** David Brunner, Guido Brunnett, Closing feature regions, März 2011, Chemnitz
- CSR-11-02** Tom Kühnert, David Brunner, Guido Brunnett, Betrachtungen zur Skelettextraktion umformtechnischer Bauteile, März 2011, Chemnitz
- CSR-11-03** Uranchimeg Tudevdayva, Wolfram Hardt, A new evaluation model for eLearning programs, Dezember 2011, Chemnitz
- CSR-12-01** Studentensymposium Informatik Chemnitz 2012, Tagungsband zum 1. Studentensymposium Chemnitz vom 4. Juli 2012, Juni 2012, Chemnitz
- CSR-12-02** Tom Kühnert, Stephan Rusdorf, Guido Brunnett, Technischer Bericht zum virtuellen 3D-Stiefeldesign, Juli 2012, Chemnitz
- CSR-12-03** René Bergelt, Matthias Vodel, Wolfram Hardt, Generische Datenerfassung und Aufbereitung im Kontext verteilter, heterogener Sensor-Aktor-Systeme, August 2012, Chemnitz
- CSR-12-04** Arne Berger, Maximilian Eibl, Stephan Heinich, Robert Knauf, Jens Kürsten, Albrecht Kurze, Markus Rickert, Marc Ritter , Schlussbericht zum InnoProfile Forschungsvorhaben sachsMedia - Cooperative Producing, Storage, Retrieval and Distribution of Audiovisual Media (FKZ: 03IP608), September 2012, Chemnitz
- CSR-12-05** Anke Tallig, Grenzgänger - Roboter als Mittler zwischen der virtuellen und realen sozialen Welt, Oktober 2012, Chemnitz

Chemnitzer Informatik-Berichte

- CSR-13-01** Navchaa Tserendorj, Uranchimeg Tudevtagva, Ariane Heller, Grenzgänger - Integration of Learning Management System into University-level Teaching and Learning, Januar 2013, Chemnitz
- CSR-13-02** Thomas Reichel, Gudula Rüniger, Multi-Criteria Decision Support for Manufacturing Process Chains, März 2013, Chemnitz
- CSR-13-03** Haibin Xu, Thomas Reichel, Gudula Rüniger, Michael Schwind, Softwaretechnische Verknüpfung der interaktiven Softwareplattform Energy Navigator und der Virtual Reality Control Platform, Juli 2013, Chemnitz
- CSR-13-04** International Summerworkshop Computer Science 2013, Proceedings of International Summerworkshop 17.7. - 19.7.2013, Juli 2013, Chemnitz
- CSR-13-05** Jens Lang, Gudula Rüniger, Paul Stöcker, Dynamische Simulationskopplung von Simulink-Modellen durch einen Functional-Mock-up-Interface- Exportfilter, August 2013, Chemnitz
- CSR-14-01** International Summerschool Computer Science 2014, Proceedings of Summerschool 7.7.-13.7.2014, Juni 2014, Chemnitz
- CSR-15-01** Arne Berger, Maximilian Eibl, Stephan Heinich, Robert Herms, Stefan Kahl, Jens Kürsten, Albrecht Kurze, Robert Manthey, Markus Rickert, Marc Ritter, ValidAX - Validierung der Frameworks AMOPA und XTRIEVAL, Januar 2015, Chemnitz
- CSR-15-02** Maximilian Speicher, What is Usability? A Characterization based on ISO 9241-11 and ISO/IEC 25010, Januar 2015, Chemnitz
- CSR-16-01** Maxim Bakaev, Martin Gaedke, Sebastian Heil, Kansei Engineering Experimental Research with University Websites, April 2016, Chemnitz
- CSR-18-01** Jan-Philipp Heinrich, Carsten Neise, Andreas Müller, Ähnlichkeitsmessung von ausgewählten Datentypen in Datenbanksystemen zur Berechnung des Grades der Anonymisierung, Februar 2018, Chemnitz
- CSR-18-02** Liang Zhang, Guido Brunnett, Efficient Dynamic Alignment of Motions, Februar 2018, Chemnitz
- CSR-18-03** Guido Brunnett, Maximilian Eibl, Fred Hamker, Peter Ohler, Peter Protzel, StayCentered - Methodenbasis eines Assistenzsystems für Centerlotsen (MACeLot) Schlussbericht, November 2018, Chemnitz
- CSR-19-01** Johannes Dörfelt, Wolfram Hardt, Christian Rosjat, Intelligente Gebäudeklimatisierung auf Basis eines Sensornetzwerks und künstlicher Intelligenz, Februar 2019, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz