Der Raspberry Pi als EEBUS-Knoten in der Programmiersprache Go

**Studienarbeit**

**des Studienganges Informationstechnik**

**an der Dualen Hochschule Baden-Württemberg Stuttgart**

**im 3. Studienjahr**

von

**Volkan Kilic**

**und**

**Tobias Kühn**

16.03.2020

Bearbeitungszeitraum 3. Studienjahr

Matrikelnummer, Kurs 5186319, TINF17IN

8626939, TINF17IN

Studienarbeitsbetreuer Prof. Dr. Karl Friedrich Gebhardt

Selbstständigkeitserklärung

Name, Vorname: Kilic, Volkan

Matrikelnummer: 5186319

Studiengang/Kurs: Informationstechnik / TINF17IN

Name, Vorname: Kühn, Tobias

Matrikelnummer: 8626939

Studiengang/Kurs: Informationstechnik / TINF17IN

Titel der Arbeit: Der Raspberry Pi als EEBUS-Knoten in der Programmiersprache Go

Wir versichern hiermit, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Falls sowohl eine gedruckte als auch elektronische Fassung abgegeben wurde, versichern wir zudem, dass die eingereichte Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum Unterschrift

Inhalt

[1 Einleitung 4](#_Toc34927056)

[1.1 Aufgabenstellung 4](#_Toc34927057)

[1.2 Aufbau der Arbeit 4](#_Toc34927058)

[2 Stand der Technik 5](#_Toc34927059)

[2.1 Grundlagen des EEBUS 5](#_Toc34927060)

[2.2 Ausgangssituation 5](#_Toc34927061)

[2.2.1 EEBUS SHIP 6](#_Toc34927062)

[2.2.2 EEBUS SPINE 6](#_Toc34927063)

[2.3 Grundlagen der Programmiersprache Go 6](#_Toc34927064)

[2.3.1 Go Grundlagen 6](#_Toc34927065)

[2.3.2 Go Methoden und Interfaces 8](#_Toc34927066)

[2.3.3 Go Nebenläufigkeit 8](#_Toc34927067)

[3 Anwendungsfallentwurf 9](#_Toc34927068)

[4 Experimentelle Untersuchungen 9](#_Toc34927069)

[4.1 Untersuchung der Implementierung auf einem beliebigen Rechner 9](#_Toc34927070)

[4.2 Untersuchung der Implementierung auf dem Raspberry Pi 9](#_Toc34927071)

[5 Lösungsentwurf 9](#_Toc34927072)

[6 Entwicklung und Implementierung der Lösung 9](#_Toc34927073)

[7 Literaturverzeichnis 9](#_Toc34927074)

# Einleitung

Hier wird Text stehen. Im Rahmen dieser Studienarbeit wird der EEBUS nach konkreter Aufgabenstellung (Abschnitt 1.1) eruiert.

## Aufgabenstellung

„Ziel der Studienarbeit ist, in erster Linie ein tiefes Verständnis für den EEBUS zu erarbeiten und darzustellen. Darauf aufbauend könnte Software in Golang für einen Raspberry Pi als EEBUS-Knoten oder EEBUS-Gerät entwickelt werden.

* Recherche, Einarbeitung und Darstellung von EEBUS
* Einarbeitung in Golang
* Design der Software oder eines Anwendungsfalles
* Implementierung (erst Implementierung auf dem Laptop)
* Golang auf dem Raspberry Pi und Implementierung auf dem Pi
* Entwicklung einer Demo/Anwendungsfalles“ (Prof. Dr. K. F. Gebhardt)

## Aufbau der Arbeit

Hier wird Text stehen.

# Stand der Technik

In den folgenden Abschnitten werden die Grundlagen zum EEBUS und zu der Programmiersprache Go eruiert. Dabei werden einzelne Punkte in Rücksprache mit dem Studienarbeitsbegleiter ausführlicher oder nur stichpunktartig dokumentiert.

## Grundlagen des EEBUS

Der EEBUS ist eine Kommunikationsschnittstelle, die auf mehreren Standards und Normen basiert. Somit kann der EEBUS von jedem Gerät und jeder technische Plattform unabhängig von Hersteller und Technologie frei genutzt werden. Das Ziel des EEBUS ist eine gemeinsame herstellerübergreifende Sprache für Energiemanagement im Internet of Things zu entwickeln und zu etablieren. Die (technische) EEBUS-Spezifikation ist freizugänglich und wird von der EEBUS Initiative standardisiert. Die Mitglieder des Vereins definieren zusammen konkrete branchenübergreifende Anwendungsszenarien, die dann im EEBUS als Anwendungsfälle standardisiert werden. [Q4] Der EEBUS ist in EEBUS SPINE und EEBUS SHIP unterteilt. In den folgenden Unterpunkten wird die Ausgangssituation, der EEBUS SPINE und der EEBUS SHIP eruiert.

## Ausgangssituation

„Künftige Systeme der elektrischen [Energieversorgung](https://de.wikipedia.org/wiki/Energieversorgung) werden stärker von volatilen Quellen wie [Wind-](https://de.wikipedia.org/wiki/Windenergie) oder [Sonnenenergie](https://de.wikipedia.org/wiki/Sonnenenergie) geprägt und dezentraler organisiert sein als heute. Diese Entwicklung wird einen intensiveren Informationsaustausch in Form von [intelligenten Stromnetzen](https://de.wikipedia.org/wiki/Intelligentes_Stromnetz) zwischen allen Teilnehmern erforderlich machen. Insbesondere [Privathaushalte](https://de.wikipedia.org/wiki/Privathaushalt) werden dabei aktivere Rollen als bisher einnehmen. Derzeit diskutierte Möglichkeiten sind unter anderem:

* Sie treten nicht nur als Verbraucher, sondern auch als Erzeuger auf, z. B. mit eigenen [Photovoltaikanlagen](https://de.wikipedia.org/wiki/Photovoltaikanlage) oder [Blockheizkraftwerken](https://de.wikipedia.org/wiki/Blockheizkraftwerk). Diese Erzeugungsanlagen sind eventuell als [virtuelle Kraftwerke](https://de.wikipedia.org/wiki/Virtuelles_Kraftwerk) organisiert. Das schließt die Möglichkeit ein, den benötigten Strom von vornherein vorzugsweise lokal zu erzeugen und damit zur Netzentlastung beizutragen.
* Sie stellen die Akkukapazität ihrer [am Netz hängenden Elektrofahrzeuge](https://de.wikipedia.org/wiki/Vehicle_to_Grid) als Puffer für den Ausgleich kurzfristiger Produktions- und Verbrauchsschwankungen zur Verfügung oder sie laden das Fahrzeug mit selbsterzeugtem Strom.
* Sie richten ihren Verbrauch an der aktuellen Stromerzeugung aus, indem der Betrieb leistungsstarker Verbraucher in Zeiten großen Stromangebots [verlagert](https://de.wikipedia.org/wiki/Demand_Side_Management) wird. Für diese Lastverschiebung eignen sich alle Haushaltsgeräte, deren Betriebszeitpunkte ohne Funktionseinbuße variiert werden können, z. B. Wärmepumpen, Kältegeräte, Spülmaschinen etc. Laut der [BDEW](https://de.wikipedia.org/wiki/BDEW)-Jahresstatistik aus dem Jahre 2009 entfallen rund 80 % des häuslichen Stromverbrauchs auf diese Kategorie. […]

All diese Szenarien setzen voraus, dass sämtliche Teilnehmer untereinander Informationen zum Energiemanagement austauschen können:

* Lokal können sich alle zum Haushalt gehörenden Erzeuger und Verbraucher untereinander über Produktions- und Verbrauchswerte, [Lastgänge](https://de.wikipedia.org/wiki/Lastprofil) und [-prognosen](https://de.wikipedia.org/wiki/Lastprognose) etc. verständigen und ihre Betriebszustände aufeinander abstimmen.
* Zwischen Energieversorger und Haushalt findet ein bidirektionaler Austausch u. a. von Tarifinformationen und Messwerten statt, der weitgehend automatisiert und ohne persönliches Eingreifen des Nutzers erfolgt.“ [Wikipedia/EEBUS]

### EEBUS SHIP

### EEBUS SPINE

## Grundlagen der Programmiersprache Go

Go ist eine Open-Source-Programmiersprache, die ursprünglich von dem Unternehmen Google LLC entwickelt und im Jahr 2012 in der Version 1 freigegeben wurde. [Q1] Go ist ausdrucksstark, prägnant, sauber und effizient. Seine Parallelitätsmechanismen erleichtern das Schreiben von Programmen, die das Beste aus Multicore- und Netzwerkmaschinen herausholen, während das neuartige Typsystem eine flexible und modulare Programmerstellung ermöglicht. Go kompiliert schnell zu Maschinencode und bietet dennoch den Komfort der automatischen Speicherbereinigung und die Möglichkeit der Laufzeitreflexion. Es ist eine schnelle, statisch typisierte, kompilierte Sprache, die wie eine dynamisch typisierte, interpretierte Sprache wirkt. [Q2] In den folgenden Teilabschnitten werden die wichtigsten Go Grundlagen, Methoden, Interfaces und die Behandlung von Nebenläufigkeit in Go beschrieben.

[Q1] <https://blog.golang.org/go-version-1-is-released> aufgerufen: 12.03.2020

[Q2] <https://golang.org/doc/> aufgerufen: 12.03.2020

### Go Grundlagen

Jedes Go-Programm besteht aus Paketen. Go-Programme starten immer in dem *main*-Paket in der *main*-Funktion. Über den *import*-Befehl können andere Pakete importiert werden. Dadurch kann der Benutzer auf *exportierte Namen* der anderen Pakete zugreifen. Auf jegliche nicht exportierte Namen kann von außerhalb des Ursprungspaketes nicht zugegriffen werden. Ein Name wird dann exportiert, wenn er mit einem Großbuchstaben beginnt. [Q3] Im folgenden Codebespiel (Abb. 1) wird ein Programm ausgeführt, welches die Zahl Pi ausgibt.

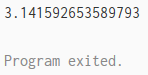
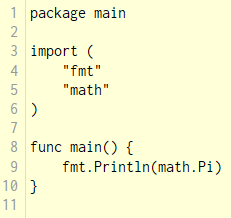


Abbildung 1 Beispielcode: Ausgabe Pi

In dem Codebeispiel (Abb.1) sind die exportierten Namen auf die zugegriffen werden eine Konstante (*Pi*) und eine Funktion (*Println*). Die Konstante *Pi* gibt den Wert von Pi und die Funktion *Println* gibt die Ausgabe von der Konstante *Pi*.

Eine Funktion kann keine oder mehrere Argumente entgegennehmen und eine beliebige Anzahl von Rückgabewerten zurückgeben. In diesem Codebeispiel (Abb.1) nimmt die Funktion eine Konstante entgegen.

Konstanten werden mit dem Schlüsselwort *const* deklariert. Konstanten können Buchstaben, Strings, boolesche oder numerische Werte sein, wie in folgendem Codebeispiel (Abb. 2) ersichtlich wird. [Q3]

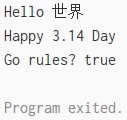
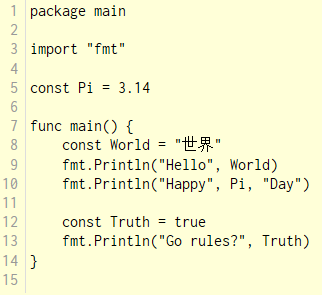


Abbildung 2 Beispielcode: Konstanten

Die Basistypen in Go sind: *bool, string, int, int8, int16, int32, int64, uint, uint8, uint16, uint32, uint64, uintptr, byte* (anderer Name für *uint8*), *rune* (anderer Name für *int32*), *float32, float64, complex 64, complex128*. Die Datentypen *int, uint* und *uintptr* belegen auf 32-bit-Systemen 32 bits und auf 64-bit-Systemen 64 bits. [Q3]

Mit der Anweisung *var* wird eine Liste von Variablen deklariert. Variablen können auch mit einem Initializer (einem pro Variable) versehen werden. Wenn ein Initializer benutzt wird, muss der Variablentyp nicht angegeben werden. Die Variable übernimmt den Typ des Initializers. Go bietet die Möglichkeit innerhalb einer Funktion statt der *var*-Anweisung mit Initializer die *:=* Anweisung für die Deklaration und Zuweisung zu verwenden. Außerhalb einer Funktion ist die := Anweisung nicht verfügbar. Im folgenden Codebeispiel (Abb. 3) wird dies exemplarisch für die Datentypen *int, bool* und *string* vorgeführt. [Q3]

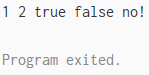
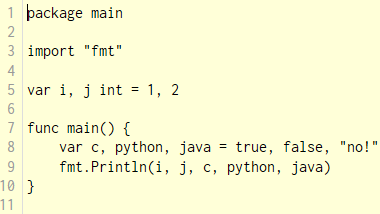


Abbildung 3 Codebeispiel: Variablen

Variablen, die ohne expliziten Wert deklariert werden, werden implizit mit ihrem jeweiligen *Nullwert* initialisiert. *Nullwert* für numerische Typen ist *0*, für boolesche Werte ist *false* und für *strings* ist “ “ (Leerstring). [Q3]

Go bietet die Kontrollstrukturen *for, if, else* und *switch* an. Außerdem können mit der *defer*-Anweisung Funktionen solange mit der Ausführung verzögert werden, bis die umgebenden Funktionen ausgeführt wurden. [Q3]

Weitere wichtige Typen die Go anbietet sind: *Pointer, Structs, Arrays, Slices* und *Maps*. Ein *Pointer* hält die Adresse einer Variablen im Speicher, ähnlich wie in C. Im Gegensatz zu C gibt es in Go keine Pointerarithmetik. Ein *struct* ist ein Verbund von Elementen. Ein *Array* hat eine fixe Größe, ein *Slice* dagegen hat eine dynamische Größe und bietet einen flexiblen Zugriff auf die Elemente eines Arrays. Eine *Map* ordnet Schlüsseln Werte zu. Folgendes Codebeispiel (Abb. 4) zeigt eine Anwendung für eine *Map*. [Q3]

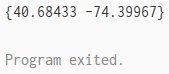
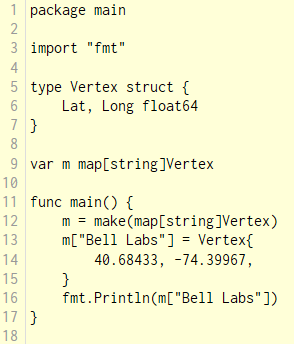


Abbildung 4 Codebeispiel: Map

[Q3] [https://go-tour-de.appspot.com/basics /](https://go-tour-de.appspot.com/basics%20/) aufgerufen: 12.03.2020

### Go Methoden und Interfaces

### Go Nebenläufigkeit

# Anwendungsfallentwurf

In diesem Kapitel wird ein exemplarischer Anwendungsfallentwurf für den EEBUS präsentiert. Der folgende Anwendungsfall wurde mit den zur Verfügung stehenden Ressourcen und in Rücksprache mit dem Studienarbeitsbegleiter entworfen.

Anwendungsfall: „LED“

In diesem Anwendungsfall soll über eine Webseite eine LED über den EEBUS gesteuert werden. Dabei fungiert ein Raspberry Pi (i.F. EEBUS-Knoten genannt) als EEBUS-Knoten und ein weiterer Raspberry Pi (i.F. LED-Steuereinheit genannt) als LED-Steuereinheit. Auf dem EEBUS-Knoten wird ein Webserver laufen gelassen, über den der Benutzer die LED an- und ausschalten kann. An die GPIO der LED-Steuereinheit wird eine LED elektrisch verbunden. Der Webserver und die zwei Raspberry Pi´s kommunizieren über den EEBUS. Folgende Abbildung (Abb, x) stellt den beschriebenen Anwendungsfall grafisch dar.

# Experimentelle Untersuchungen

## Untersuchung der Implementierung auf einem beliebigen Rechner

## Untersuchung der Implementierung auf dem Raspberry Pi

# Lösungsentwurf

In diesem Kapitel wird ein Lösungsentwurf für den beschriebenen Anwendungsfall (Kapitel 3) dargestellt.

# Entwicklung und Implementierung der Lösung

# Literaturverzeichnis

[Q4] <https://de.wikipedia.org/wiki/EEBUS>