Bewegungssteuerung eines vierbeinigen Roboters

Hausarbeit: WP Deeply Embedded

Jannik Beyerstedt Sebastian Szancer

30. Juli 2016

Prof. Dr. Stephan Pareigis

Fakultät Technik und Informatik
Department Informatik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Inhaltsverzeichnis

Abbildungsverzeichnis **3**

Code-Verzeichnis **3**

Abkürzungsverzeichnis **3**

1. Einleitung (Bey) **4**

2. Mechanik und Elektrik (Bey) **5**

2.1. Wahl der mechanischen Plattform 5

2.2. Elektrische Verbindungen 7

2.2.1. Stromversorgung und Anschluss der Servos 7

2.2.2. PWM-Pins auf dem BeagleBone Black 8

2.2.3. Verbindung von zwei BeagleBone Blacks 10

3. Bewegungsablauf (Bey) **11**

3.1. Bewegung eines Beins 11

3.2. Bewegungssequenz aller Beine 13

4. Software (Sza) **16**

4.1. Software-Architektur 16

4.2. Ansteuerung eines Servos 18

4.3. Roboter-Bein 19

4.4. Ansteuerung der Roboter-Beine 21

4.5. Programmierung von Bewegungen 23

A. Roboter (Bey) **25**

B. Quellcode (Sza) **25**

Abbildungsverzeichnis

1. mePed Quadruped Robot (Aufbaubeispiel)[2] 6
2. mögliches Platinenlayout 8
3. PWM-Pins und überlagerte Funktionen 9
4. schematischer Bewegungsablauf 12
5. Bewegungssequenz Variante A 14
6. Bewegungssequenz Variante B 14
7. Schichten der Architektur 16
8. Klassendiagramm 17
9. (Anhang) verwendeter Roboter 25

Code-Verzeichnis

1. Quellcode aus Datei `servo.cpp` der „ResyLib“. 18
2. Konstruktor aus `RobotLeg.cpp`. 19
3. aus `moveServo` Methode zur Bewegung eines Gelenks (aus `RobotLeg.cpp`). 20
4. Konstruktor aus `RobotMovementControl.cpp`. 21
5. Methode zur Bewegung eines Gelenks eines Roboter-Beins (aus `RobotMovementControl.cpp`). 22

Abkürzungsverzeichnis

Bey Autor: Beyerstedt

Sza Autor: Szancer

FDM Fused Deposition Modeling - Ein additives Fertigungsverfahren, das Werkstücke erstellt, indem geschmolzener Kunststoffdraht in Schichten aufgetragen wird.

GPIO general-purpose input/output

PWM Pulsweitenmodulation

UART universal asynchronous receiver/transmitter

1. Einleitung (Bey)

Im Wahlpflichtkurs „Deeply Embedded“ sollte zum Abschluss als kleines Projekt ein Roboter gebaut werden, der auf (insektenartigen) Beinen läuft. Als Plattform für die Steuerung sollte ein BeagleBone Black von BeagleBoard verwendet werden. Dabei war die hauptsächliche Einschränkung, dass der BeagleBone Black nur 8 PWM-Ausgänge zur Verfügung stellt und damit auch nur 8 Servomotoren unabhängig voneinander angesteuert werden können. Daher fiel die Wahl auf einen vierbeinigen Roboter mit jeweils 2 steuerbaren Gelenken pro Bein. Für die Planung und Umsetzung des Projekts waren die letzten vier Wochen der Vorlesungszeit vorgesehen.

Als Servomotoren werden hier kleine Stellantriebe aus dem Modellbau bezeichnet mit denen eine Achse auf eine bestimmte Winkelposition gefahren werden kann. Zur Vorgabe des Winkels wird ein digitales Signal verwendet dessen Einschaltdauer, also die Pulsweite, proportional zum Winkel ist. Dieses Signal wird auch als „pulsweitenmoduliert“ bezeichnet und mit PWM abgekürzt.

In dieser Hausarbeit wird zunächst die Auswahl der mechanischen Elemente und der Elektronik betrachtet. Außerdem musste ein Bewegungsablauf entwickelt werden, mit dem sich der Roboter zunächst vorwärts bewegen kann. Für eine Verfeinerung der Bewegungsabläufe war der gegebene Zeitrahmen leider zu klein.

Im zweiten Teil wird die Abstraktion der Bewegungen des Roboters entwickelt und die sich daraus ergebende Softwarearchitektur erklärt.

Nicht betrachtet wird die konkrete Auswahl von Spannungsversorgung und Umgebungssensoren, da diese Aufgaben von anderen Kursteilnehmern durchgeführt worden sind.

Aufgrund der geringen Zeit und Problemen mit der Hardware konnte in der Vorlesungszeit kein funktionsfähiger Roboter aufgebaut werden. Daher sind einige Betrachtungen dieser Arbeit nur theoretisch, da die entsprechende Hardware nicht mehr umgesetzt wurde.

2. Mechanik und Elektrik (Bey)

Zunächst musste eine geeignete mechanische Plattform entwickelt oder gefunden werden, damit feststeht, ob die Idee eines vierbeinigen Roboters umsetzbar ist und welches Material noch bestellt werden muss.

2.1. Wahl der mechanischen Plattform

Wie in der Einleitung erwähnt ergab die Recherche eines Kommilitonen, dass der BeagleBone Black nur acht PWM Ausgänge zur Verfügung stellt. Da darauf verzichtet werden sollte mehrere BeagleBone Blacks miteinander zu verbinden, um mehr Servos ansteuern zu können, sollte also ein vierbeiniger Roboter mit zwei angetriebenen Gelenken pro Bein gebaut werden.

Im Internet lassen sich unter den Namen „Hexapod“ diverse Roboter auf sechs Beinen finden. Diese haben meist drei angetriebene Gelenke pro Bein, um die Ausrichtung des Fußes auf dem Boden steuern zu können, also werden deutlich mehr als acht Servos benötigt. Vierbeinige Roboter werden als „Quadruped“ bezeichnet und haben zumeist ebenfalls drei Gelenke pro Bein, ein paar Bauarten verzichten aber auch auf einen Servo pro Bein.

„Thingiverse“[1] ist eine Plattform auf der jeder seine 3D-Modelle anderen Nutzern bereitstellen kann. Zumeist sind diese Modelle für den 3D-Druck im Fused Deposition Modeling-Verfahren (FDM) konstruiert. Der Begriff „3D-Druck“ ist eigentlich ein Sammelbegriff für diverse additive Fertigungsverfahren, bezeichnet aber Umgangssprachlich meist das FDM-Verfahren.

Nun konnte auch ein Modell für einen Quadruped-Roboter[2] auf Thingiverse gefunden werden, das unseren Anforderungen entspricht. Dieses ist zwar eigentlich für Plattenmaterial konzipiert, das mit einem Lasercutter ausgeschnitten wird, kann aber auch mit FDM hergestellt werden. Dieses Verfahren musste gewählt werden, da in dem kurzen zur Verfügung stehenden Zeitraum nur ein FDM-Drucker zur Verfügung stand. Der Bausatz wurde auch deshalb verwendet, da im Labor Modellbau-Servos der Baugröße Mini vorhanden waren und diese in die Bauteile passen.

[1] http://www.thingiverse.com
[2] http://www.thingiverse.com/thing:756681

Ein Roboterbein eines insektenartigen Roboters hat meist drei Gelenke, die zwei Freiheitsgrade an der „Hüfte“ und einen Freiheitsgrad am „Knie“ ausbilden. Die Hüfte schwenkt das Bein in der horizontalen Ebene und dient dazu das Bein auf und ab bewegen. Das Knie sorgt dann dafür, dass der Fuß senkrecht auf dem Boden steht. Unser Roboter kann auf ein angetriebenes Gelenk verzichten, da für das dritte Gelenk eine Zwangsführung eingesetzt wird und somit die Hüfte das Knie mit bewegt. Im Folgenden wird der Servo für das Heben und Senken als Knie bezeichnet. Die Grundplatte des Roboters ist quadratisch und dient der Befestigung der Beine, sowie von Sensoren und Controller. Die Beine sind an den Ecken angebracht, sodass der Roboter symmetrisch aufgebaut ist, wie in Abb. 1 dargestellt. Der konkrete im Kurs verwendete Roboter ist in Anhang A abgebildet, dort sind allerdings die einzelnen Bauteile schlechter zu erkennen.

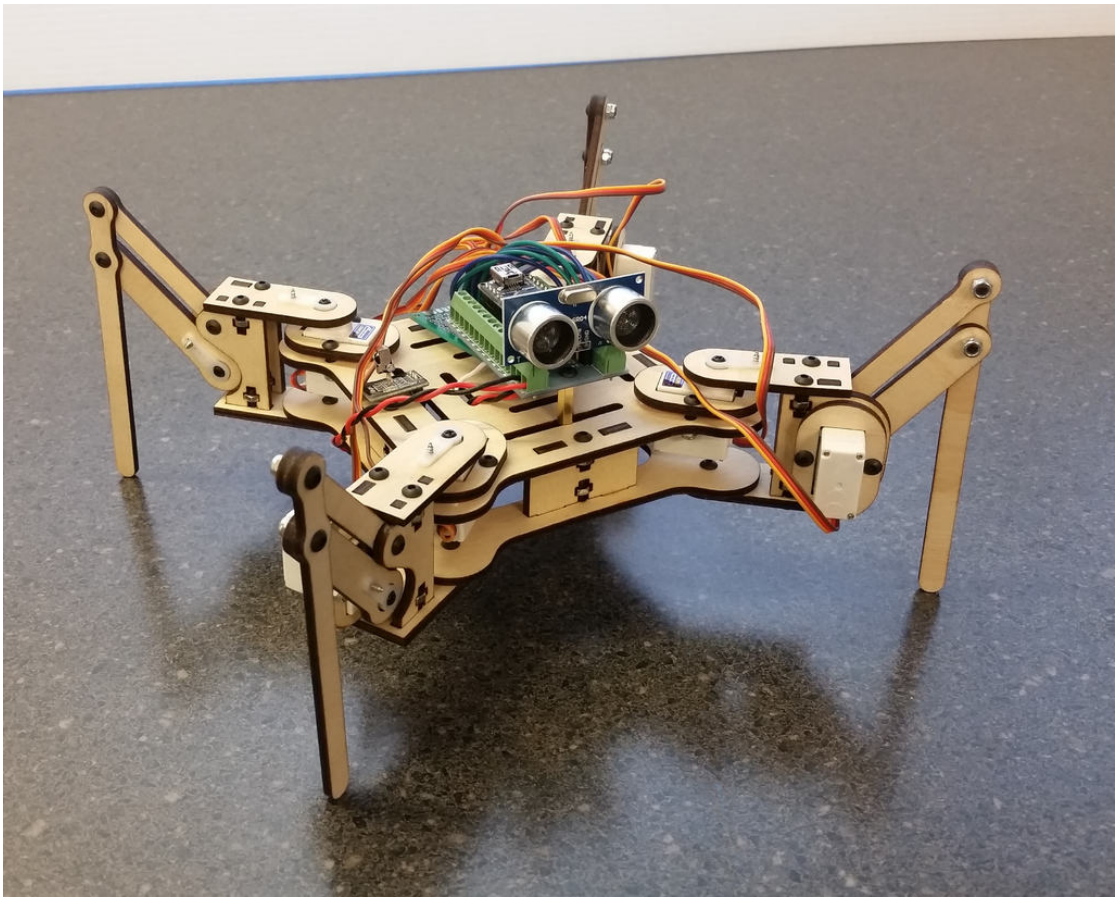

Abbildung 1: mePed Quadruped Robot (Aufbaubeispiel)[2]

Das Weglassen eines Antriebs lässt zwar nicht mehr zu den Winkel des Fußes zum Boden zu steuern, bedeutet aber auch, dass diese Berechnung nicht durchgeführt werden muss und somit das Modell der Bewegung einfacher ist. Der Bewegungsablauf wird in Kap. 3 beschrieben.

2.2. Elektrische Verbindungen

2.2.1. Stromversorgung und Anschluss der Servos

Die verwendeten Modellbau-Servos sollten an ca. 5–6 V betrieben werden. Da für den BeagleBone Black 5 V Betriebsspannung erforderlich sind, kann diese Spannungsquelle auch für die Servos verwendet werden. Die Servos sollten dabei nicht von den 5 V Pins der BeagleBone Black gespeist werden, da dieser die benötigte Leistung über diese Pins nicht aufbringen kann. Daher muss eine Lösung entwickelt werden, um die Spannung an die Servos zu verteilen.

Wenn schon eine Verteilung der Spannungsversorgung aufgebaut wird, kann diese auch dafür genutzt werden, die Servos einfacher an den BeagleBone Black anzuschließen, da der BeagleBone Black keine direkte Möglichkeit für Servo-Stecker bietet. Da die Servo-Stecker im gängigen Rastermaß von 2,45 mm ausgeführt sind, können einfache Stiftleisten anstelle von speziellen Verbindern verwendet werden, die auf einer Lochrasterplatine angebracht werden.

Über diese Platine können dann auch die Datenpins der Servos an die dafür ausgewählten Pins des BeagleBone Black angeschlossen werden. Dafür kann die Platine entweder als Cape gebaut werden, sodass alle Pins der Header des BeagleBone Black direkt auf der Platine zur Verfügung stehen. Alternativ kann die Platine auch mit einzelnen Kabeln oder zwei großen Steckern, die auf die zwei Header des BeagleBone Black passen, mit diesem verbunden werden. Die einzelnen Kabel hätten den Vorteil, dass die Belegung der Pins einfach geändert werden kann, wenn dies durch Erweiterungen erforderlich ist. Die Platine kann dann außerdem für die Spannungsversorgung von Sensoren verwendet werden.

Ein mögliches Platinenlayout ist in Abb. 2 dargestellt. Dabei ist das Batteriefach nur als Symbol für eine beliebige Spannungsversorgung zu verstehen.

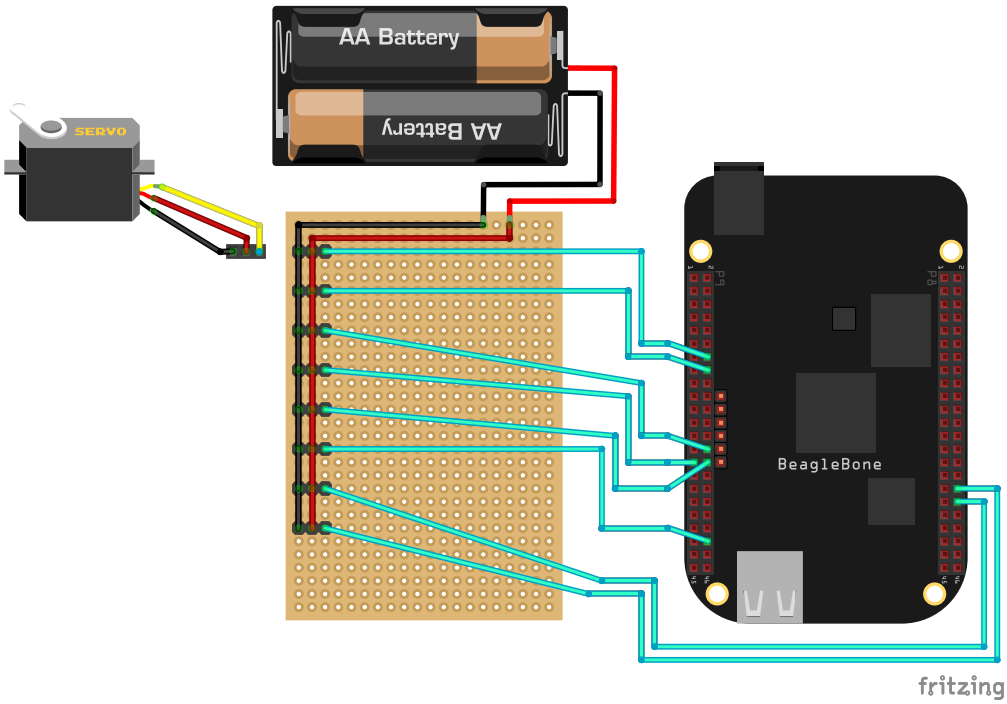

Abbildung 2: mögliches Platinenlayout

2.2.2. PWM-Pins auf dem BeagleBone Black

Im Verlauf des Projektes stelle es sich heraus, dass es aus ungeklärten Gründen nicht möglich ist alle 8 PWM-Ausgänge des BeagleBone Black gleichzeitig zu benutzen. Ein Grund dafür könnte sein, dass die als PWM-Ausgänge benutzen Pins zum Teil auch für andere Funktionen, wie UART und SPI, genutzt werden und daher eventuell auftretende Kollisionen beachtet werden müssen.

Auf der Beagleboard-Webseite[3] sind die GPIO-Pins des BeagleBone Black tabellarisch dargestellt. Dort werden auch die acht PWM-Ausgänge und vier Timer erwähnt, es sind aber auch auch 14 Pins mit „PWM“ beschriftet.

Der BeagleBone Black verwendet als Hauptprozessor einen ARM-Chip, auf dem neben der CPU diverse Peripheriemodule zur Verfügung stehen. Zur Erzeugung von PWM-Signalen werde dabei in der Regel die Timer-Module verwendet. Ein Timer ist dabei eigentlich nur ein Zähler, der mit einem bestimmten Takt hoch- oder runterzählt. Da

[3] http://beagleboard.org/support/bone101#headers

der Prozessor mit einem relativ hohen Takt arbeitet, kann dieser mit dem sog. Prescaler verringert werden, sodass der Takt zur gewünschten Anwendung passt. Wenn der Timer periodisch verwendet wird, ist die Periodendauer von der Taktfrequenz und dem Startwert des Timers abhängig, da Timer meist abwärtszählend verwendet werden. Sobald der Timer den Wert 0 erreicht, wird wieder mit dem Startwert begonnen.

Um ein PWM-Signal zu erzeugen wird nun auch noch der sog. Match-Value verwendet. Sobald der Timer also auf den Startwert zurückgesetzt wird, wird der Ausgangspin auf HIGH geschaltet und bei Erreichen des Match-Values wieder auf LOW. Die Pulsweite des PWM-Signal kann also über den Match-Value eingestellt werden. Außerdem muss der Timer natürlich mit einem GPIO-Pin verbunden werden, auf dem das Signal ausgegeben wird.

Eine weitere Besonderheit der Timer ist, dass pro Timer-Modul meist zwei Timer zur Verfügung stehen. Da die Module von 0 an durchnummeriert sind, werden die zwei Timer eines Moduls mit A und B bezeichnet. Dadurch besteht die Einschränkung, dass die A- und B-Timer nicht vollständig unabhängig voneinander verwendet werden können.

P9

		1	2		
		3	4		
		5	6		
		7	8		
		9	10		
		11	12		
		13	14	EHRPWM1A	
		15	16	EHRPWM1B	
		17	18		
		19	20		
UART2_TXD	EHRPWM0B	21	22	EHRPWM0A	UART2_RXD
		23	24		
		25	26		
		27	28	ECAPPWM2	
	EHRPWM0B	29	30		
	EHRPWM0A	31	32		
		33	34		
		35	36		
		37	38		
		39	40		
		41	42	ECAPPWM0	UART3_TXD
		43	44		
		45	46		

P8

		1	2		
		3	4		
		5	6		
	TIMER4	7	8	TIMER7	
	TIMER5	9	10	TIMER6	
		11	12		
	EHRPWM2B	13	14		
		15	16		
		17	18		
	EHRPWM2A	19	20		
		21	22		
		23	24		
		25	26		
		27	28		
		29	30		
		31	32		UART5_RTS
		33	34	EHRPWM1B	UART3_RTS
		35	36	EHRPWM1A	UART3_CTS
		37	38		UART5_RXD
		39	40		
		41	42		
		43	44		
	EHRPWM2A	45	46	EHRPWM2B	

Abbildung 3: PWM-Pins und überlagerte Funktionen

Somit lassen sich auch die Beschriftungen der Pins in Abb. 3 erklären. Die „EHRPWM“-Pins können alle auf zwei verschiedene Pins auf den Headern P8 und P9 geschaltet

werden können. Außerdem sind für die PWM-Signale jeweils der A- und der B-Timer verwendbar. Trotzdem war es in der Vorlesung leider nicht möglich von einem Timer-Modul beide Timer gleichzeitig zu betreiben.

Die mit „ECAPPWM“ gekennzeichneten Pins werden vom sog. Enhanced Capture Modul des Mikrocontrollers bereitgestellt. Daher muss dafür ein weiteres Device Tree Overlay geladen werden. Danach sollten diese aber eigentlich wie die anderen Timer benutzbar sein. (vgl. [1])

In Abb. 3 ist noch einmal eine Übersicht der PWM-Pins und der eventuell damit kollidierenden anderen Funktionen dargestellt. Um dieselben PWM-Timer einfacher identifizieren zu können sind diese mit derselben Farbe markiert.

2.2.3. Verbindung von zwei BeagleBone Blacks

Der BeagleBone Black stellt zwar theoretisch acht PWM-Ausgänge für die Ansteuerung der Servos zur Verfügung, die Ansteuerung der Pins konnte jedoch nicht für alle acht Pins gleichzeitig erfolgen. Daher müssen nun doch zwei BeagleBone Blacks miteinander verbunden werden, auch wenn dies durch die Auswahl eines Roboters mit nur acht Servos verhindert werden sollte.

Um das System einfach zu halten, wurde ein Master-Slave-System geplant. Die Software ist darauf ausgelegt, dass in einer möglichst gleichmäßig getakteten Schleife die Sensoren ausgewertet und die Aktoren gestellt werden. Wenn die Bewegungssteuerung nun aber auf zwei BeagleBone Blacks aufgeteilt ist, müssen deren Schleifen synchronisiert werden. Eine einfache Möglichkeit ist es, die beiden Boards über eine serielle Schnittstelle, z. B. UART, miteinander zu verbinden und den Master ein Synchronisationssignal zu Beginn der Hauptschleife aussenden zu lassen. Der Slave wartet dann zu Beginn seiner Hauptschleife auf das Signal, sodass beide Schleifen in Rahmen der Latenz der UART-Verbindung synchron laufen.

Auf dem Slave muss dabei nur eine Hälfte der Beine gesteuert werden, die Auswertung der Sensoren kann vollständig am Master erfolgen, sodass nur die Bewegungssteuerung synchronisiert werden muss. Damit auch ein Wechsel der Bewegungsrichtung möglich ist, sollten zusätzlich zum Synchronisationssignal Steuercodes übertragen werden, die den internen Zustand der Bewegungssteuerung ändern. Details zum Aufbau der Software folgen in Kap. 4.

3. Bewegungsablauf (Bey)

Nachdem nun die mechanische Plattform ausgewählt wurde und damit auch die konkreten Bewegungsräume der Beine feststehen, muss ein Bewegungsablauf entwickelt werden, mit dem der Roboter sich vorwärts bewegt. Dabei besteht die Einschränkung, dass die Beine sich nicht vollständig unabhängig voneinander bewegen dürfen, da diese ansonsten kollidieren würden.

Zunächst wird jedoch die Bewegung eines Beins betrachtet um danach aus diesen Bewegungen eine Laufbewegung aller Beine zusammenzustellen.

3.1. Bewegung eines Beins

Da der Roboter vier Beine hat, liegt es nahe einen Bewegungsablauf zu planen, der auf vier Takten basiert. Damit kann die Gesamtbewegung zusammengesetzt werden, indem jedes Bein um einen Takt zum vorherigen Bein versetzt ist.

Damit der Roboter nicht umfällt, müssen zu jeder Zeit mindestens drei Beine auf dem Boden stehen. Dabei dürfen die Beine natürlich auch nicht alle auf einer Seite des Schwerpunktes des Roboters stehen, da dieser sonst umkippt. Diese Einschränkung wird dann aber erst im nächsten Kapitel betrachtet.

Um eine Bewegung zu erreichen, muss ein Bein sich zuerst auf dem Boden bewegen und am Ende des Bewegungsraumes wieder an den Anfang zurückgesetzt werden. Da sich maximal ein Bein zur Zeit in der Luft befinden darf, steht nur ein Takt für die Rücksetzbewegung zur Verfügung.

Wenn also nun jeder Takt noch einmal in drei Teilschritte unterteilt wird besteht ein Zyklus aus 12 Teilschritten (4 Takte). Damit soll sich das Bein in 3 Takten von „vorne“ nach „hinten“ bewegen und im letzten Takt wieder nach vorne. Für das Zurücksetzen wird im ersten Teilschritt das Bein angehoben, im nächsten nach vorne bewegt und im letzten Teilschritt wieder abgesenkt. Es ergibt sich also in vereinfachter Darstellung eine Rechteckbewegung, die der Fuß in den vier Takten durchläuft. In Abb. 4 ist die Bewegung noch einmal schematisch mit den Startpositionen der vier Takte dargestellt.

In der Realität wird der Fuß anstelle einer linearen Bewegung ein Kreissegment auf dem Boden beschreiben. Dies liegt an der Aufhängung des Beines am „Hüftgelenk“ und am

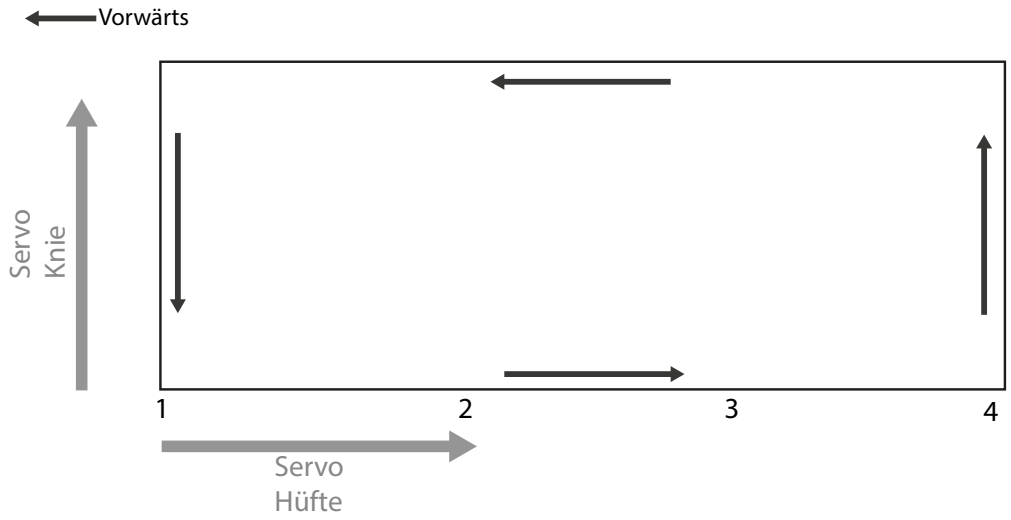

Abbildung 4: schematischer Bewegungsablauf

Fehlen eines dritten Gelenkes, mit dem dieser Effekt ausgeglichen werden könnte. In den folgenden Betrachtungen wird jedoch zur Vereinfachung weiterhin angenommen, dass der Fuß eine lineare Bewegung auf dem Boden ausführt.

Damit der Roboter sich kontrolliert vorwärts bewegt, muss der oben beschriebene Bewegungsablauf eines Beines zu einer Bewegungssequenz aller Beine zusammengesetzt werden.

3.2. Bewegungssequenz aller Beine

Zur Vereinfachung wird erst einmal eine Bewegungssequenz erstellt, mit der der Roboter nach vorne laufen soll. Da der Roboter quadratisch ist, wird einfach eine Seite als „Vorne“ festgelegt, sodass es vordere und hintere, sowie linke und rechte Beine gibt.

Wie anfangs erwähnt, kann ein hinteres Bein nur seine vorderste Position einnehmen, wenn das vordere Bein noch genug Abstand hat, sich also gerade nicht hinten befindet. Daher muss immer zuerst das vordere Bein nach vorne zurückgesetzt werden, bevor das hintere Bein der selben Seite ebenfalls diese Bewegung machen kann.

Eine Bewegungssequenz kann dabei erst einmal für eine Seite des Roboters entwickelt werden und danach auf die andere Seite mit einem gewissen zeitliche Versatz übertragen werden. Dadurch ergeben sich zwei Möglichkeiten:

(a) Das hintere Bein eilt dem vorderen Bein der selben Seite um einen Takt nach. Dadurch ergibt sich ein zeitlicher Versatz zwischen den beiden Seiten von zwei Takten.

(b) Das hintere Bein eilt um zwei Takte nach. In dieser Variante beträgt der Versatz einen Takt, die beiden Seiten wechseln sich also mit den Schritten ab.

Da ein Nacheilen von drei Takten äquivalent zu einem Voreilen von einem Takt ist, was in der Vorbedingung ausgeschlossen wurde, wurde diese Möglichkeit nicht betrachtet.

Die zwei möglichen Bewegungssequenzen sind in den Abbildungen 5 und 6 dargestellt. Dabei ist am jeweiligen Bein die aktuelle Position im Bewegungsablauf aus Abb. 4 notiert. Außerdem ist das Bein, welches sich nach vorne zurück bewegen wird, markiert.

Im vorherigen Kapitel (Kap. 3.1) wurde nur kurz darauf eingegangen, dass zusätzlich zu möglichen Kollisionen der Beine beachtet werden muss, dass der Roboter nicht umkippt. Dafür darf der Schwerpunkt des Roboters nicht außerhalb der von den Beinen aufgespannten Fläche befinden. Da sich immer ein Bein in der Luft befindet, ergibt sich ein Dreieck aus den drei verbleibenden Beinen. Da der Schwerpunkt des Roboters unbekannt ist und von der Beladung mit Sensoren abhängt, kann erst einmal nur angenommen werden, dass sich dieser in der Mitte befindet.

In der Abbildung lässt sich bei Variante A nun erkennen, dass die Konfigurationen oben links und unten rechts definitiv kippen werden.

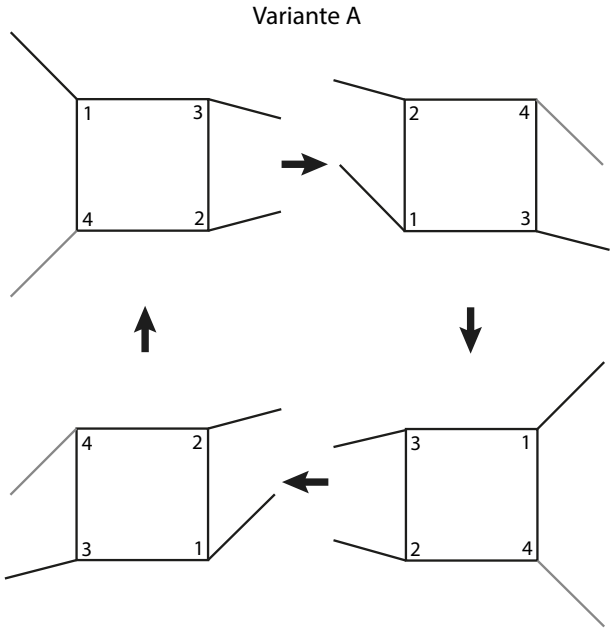

Abbildung 5: Bewegungssequenz Variante A

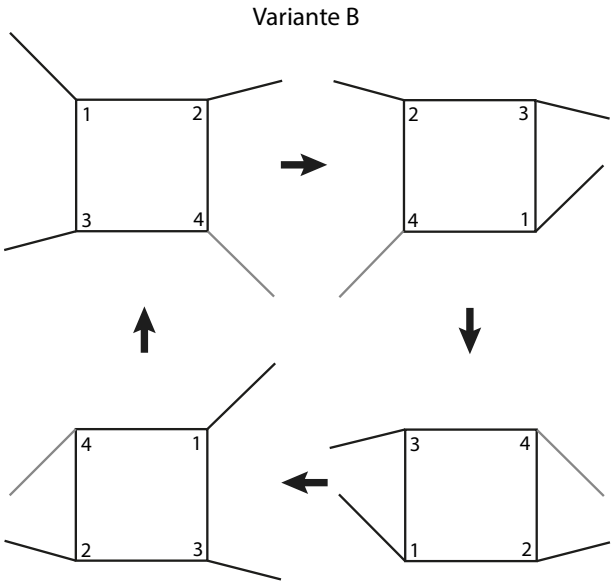

Abbildung 6: Bewegungssequenz Variante B

In der Variante B sieht es schon etwas besser aus, auch wenn hier die oben rechts dargestellte Konfiguration kritisch ist. In den Zeichnungen wurde ein Bewegungsbereich von 90° für das Hüftgelenk gewählt. Daher könnte die eben genannte Konfiguration durch einen verkleinerten Bewegungsbereich des Hüftgelenkes dahingehend verändert werden, dass der Roboter nicht kippt. Dies gilt auch für die oben links und unten rechts dargestellten Konfigurationen, die zwar nicht definitiv kippen werden, aber dennoch nur gerade eben die Stabilitätsbedingung erfüllen.

4. Software (Sza)

Zur Realisierung von Bewegungen des Roboters, wie z. B. das in Kapitel 3 beschriebene vorwärts Laufen, oder das hoch- und runter Bewegen des Körpers, ist eine entsprechende koordinierte Ansteuerung der Servos der Roboter-Beine nötig. In diesem Kapitel wird die dafür entwickelte Software beschrieben.

4.1. Software-Architektur

Die Software für die Steuerung der Bewegung des Roboters besteht aus 4 Schichten (Siehe: Abb. 7).

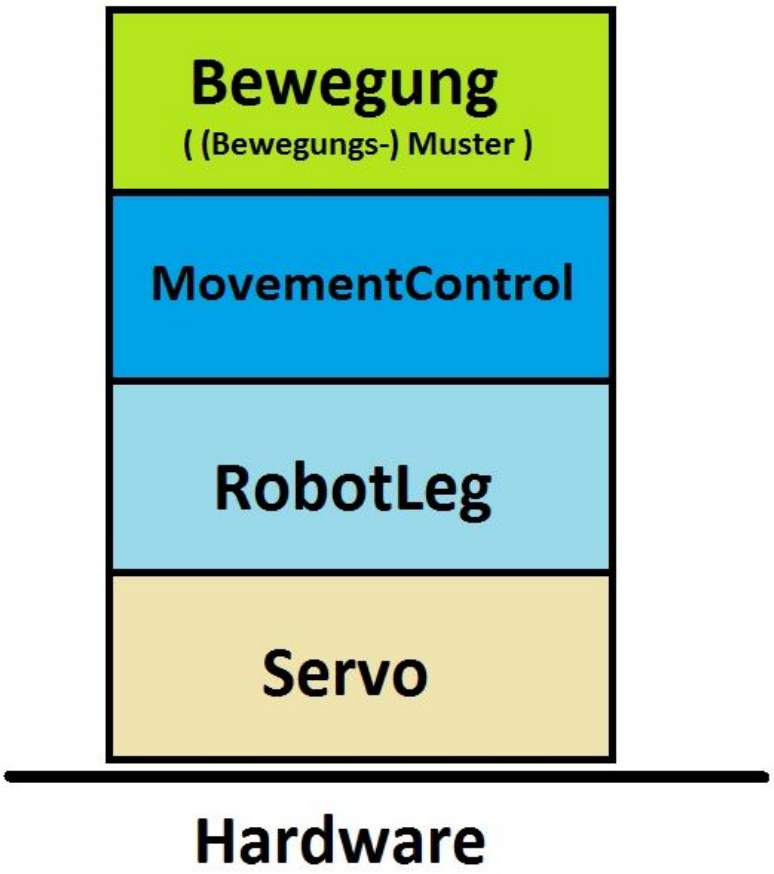

Abbildung 7: Schichten der Architektur

Die direkte Ansteuerung der Hardware (Servomotoren) passiert in einer Servo-Klasse, welche durch die „ResyLib“ bereitgestellt wird.

Die nächste höhere Schicht ist die Klasse `RobotLeg`, welche ein ganzes Roboter-Bein repräsentiert. Um mehrere Roboter-Beine koordiniert ansteuern zu können, bietet die darüber liegende Schicht, die Klasse `RobotMovementControl` den gezielten Zugriff auf die Beine des Roboters. Über die `RobotMovementControl` ist es möglich, die auf der höchsten Schicht `RobotMovement` definierten Bewegungsmuster auszuführen.

Die Software Architektur „auf einen Blick“ sieht man im Klassendiagramm (Siehe: Abb. 8).

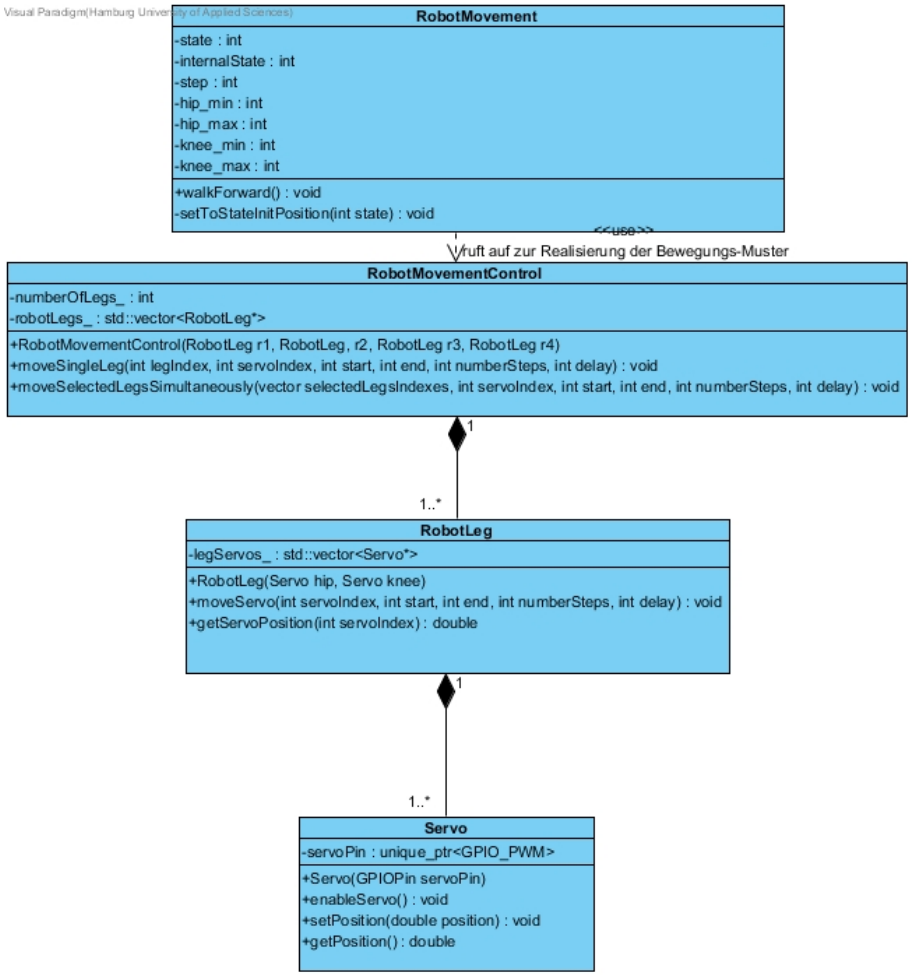

Abbildung 8: Klassendiagramm

4.2. Ansteuerung eines Servos

Die Ansteuerung eines Servos geschieht über einen festgelegten PWM-Pin. Der Servomotor fährt die über das Tastverhältnis (Duty Cycle) eingestellte Position an. (Siehe: Code 1) In unserem Fall sind für die Positionsangabe double-Werte zwischen 0.0 und 1.0 zulässig.

```
1 void Servo::setPosition(double position) {
2   currentPosition = position;
3
4   // todo check for min max value
5   servoPin->setDuty(1000000 + (1000000 * position));
6 }
```

Code 1: Quellcode aus Datei `servo.cpp` der „ResyLib“.

Damit ein Servo benutzt werden kann, muss er immer zuerst „enabled“ werden.

4.3. Roboter-Bein

Ein Roboter-Bein wird durch die Klasse `RobotLeg` repräsentiert. Ein Bein besteht bei unserem Roboter aus zwei Gelenken („Hüfte“ und „Knie“), wobei ein Gelenk jeweils ein Servo ist (Siehe: Code 2) und bewegt werden kann.

Aus Gründen der Erweiterbarkeit der Software und eines leichten Zugriffs besitzt ein `RobotLeg` einen `std-vector` von `Servos`. So braucht bspw. beim Einsatz der Software auf einem Roboter mit 3 Gelenken in jedem Bein nur ein weiteres Element zum `std-vector` von Gelenken hinzugefügt werden und der Zugriff über den Index bleibt prinzipiell gleich.

```
1  /**
2  * Constructor.
3  */
4  RobotLeg::RobotLeg(Servo* hip, Servo* knee){
5    legServos_.push_back(hip);
6    legServos_.push_back(knee);
7    //Enable servos:
8    legServos_[0]->enableServo();
9    legServos_[1]->enableServo();
10 }
```

Code 2: Konstruktor aus `RobotLeg.cpp`.

Zur Bewegung der Gelenke stellt die Klasse eine public-Methode `moveServo` zur Verfügung. Dabei muss, neben dem Gelenk, eine Start- und Zielposition (Angabe in Grad) für die Bewegung vorgegeben werden, sowie die Anzahl der Schritte in der die Bewegung ausgeführt werden soll. Daraus wird eine Sequenz von Positionen ermittelt, die dann nacheinander vom Servo angefahren werden (Siehe: Code 3, nächste Seite).

```
1  //Stelle Sequenz von Positionen zusammen:
2  std::vector<int> positionSequence;
3
4  double stepSize = (end - start) / double(numberSteps);
5
6  for (int i = 0; i <= numberSteps; i++) {
7      positionSequence.push_back( start + (stepSize*i) );
8  }
9
10 //Fahre die Positionen an:
11 for (auto j : positionSequence){
12     // convert from 0..180 deg => 0.0..1.0
13     legServos_[servoIndex]->setPosition(j / 180.0);
14     sleep(delay);
15 }
```

Code 3: aus `moveServo` Methode zur Bewegung eines Gelenks (aus `RobotLeg.cpp`).

Ergänzende Überlegungen:

Aus Sicherheitsgründen wäre es zudem noch sinnvoll den Bewegungsradius eines Gelenks (softwareseitig) des Roboter-Beins einzuschränken, damit für die Bewegung eines Gelenks nicht der vollständige Arbeitsbereich des Servos zur Verfügung steht, da dieser in der Regel nicht für die natürliche Bewegung des Gelenks nötig ist und seine Ausnutzung unter Umständen zur Beschädigung von Hardware führen könnte (z. B. Kollision von Beinen bei fehlender Einschränkung des Bewegungsradius der „Hüftgelenke“).

4.4. Ansteuerung der Roboter-Beine

Die Klasse `RobotLeg` ermöglicht die Bewegung eines Gelenks des Beins. Nun soll im Kontext der Bewegung nicht mehr auf der Ebene eines Beins gedacht, sondern der Roboter als Ganzes betrachtet werden. Die Klasse `RobotMovementControl` erlaubt die Steuerung aller Beine des Roboters. Ähnlich wie bei den Gelenken des Roboter-Beins, wird auch hier ein `std-vector` für die Beine des Roboters verwendet (Siehe: Code 4).

```
1  /**
2  * Constructor.
3  */
4  RobotMovmentControl::RobotMovmentControl(RobotLeg* r1, RobotLeg* r2,
       RobotLeg* r3, RobotLeg* r4){
5    numberOfLegs_ = 0;
6    robotLegs_.push_back(r1);
7    numberOfLegs_++;
8    robotLegs_.push_back(r2);
9    robotLegs_.push_back(r3);
10   robotLegs_.push_back(r4);
11   numberOfLegs_ = 4;
12 }
```

Code 4: Konstruktor aus `RobotMovementControl.cpp`.

In Hinblick auf die Ansteuerung der Beine kann man zwischen zwei Arten von Bewegungen unterscheiden:

1. Eine Bewegung bei der beliebige Gelenke alle nacheinander bewegt werden, also die Bewegung eine Sequenz aus einzelnen Gelenk-Rotationen ist, d. h. es muss nur ein Servo zeitgleich eine Position anfahren (z. B. ein Bein nach vorne setzen).
2. Eine Bewegung bei der mehrere Gelenke gleichzeitig bewegt werden, d. h. es müssen mehrere Servos zeitgleich eine Position anfahren (z. B. den Körper runter bewegen).

Für die erste Art von Bewegung reicht es aus, eine Methode bereitzustellen, welche die gezielte Bewegung eines Gelenks eines Beins erlaubt (Siehe: Code 5).
Mit einer Sequenz von Methoden-Aufrufen lassen sich dann alle möglichen Bewegungen realisieren.

```
1 /**
2 * Bewegt einen Servo eines Roboter-Beins.
3 */
4 void RobotMovmentControl::moveSingleLeg(int legIndex, int servoIndex, int
      start, int end, int numberSteps, int delay){
5   robotLegs_[legIndex]->moveServo(servoIndex, start, end, numberSteps,
       delay);
6 }
```

Code 5: Methode zur Bewegung eines Gelenks eines Roboter-Beins (aus `RobotMovementControl.cpp`).

Für die zweite Art kann das Bewegen eines Gelenks schrittweise gesteuert werden, sodass die einzelnen Schritte der verschiedenen Gelenke abwechselnd erfolgen (Idee für eine Single-Thread Lösung).
Sollen bspw. alle Knie-Gelenke von 60 Grad auf 50 Grad, werden zuerst alle nacheinander angesteuert von 60 Grad auf 58 Grad zu fahren, danach von 58 Grad auf 56 Grad usw. Die in `RobotMovementControl.cpp` implementierte Methode zur Bewegung der Gelenke mehrerer Beine, erlaubt nur die simultane Bewegung von entweder Knie- oder Hüft-Gelenken im selben Winkel.

Ein mögliches Problem dieser Lösung ist, dass u. U. eine solche Bewegung des Roboters nicht flüssig ist.

Um zukünftig weitere Bewegungsmuster umsetzten zu können, muss die Klasse `RobotMovementControl` eventuell um weitere Methoden zur simultanen Ansteuerung mehrerer Servos erweitert werden.

4.5. Programmierung von Bewegungen

Die eigentliche Definition der Bewegung passiert in der Klasse `RobotMovement`, einer Bibliothek von (Bewegungs-) Mustern, die Bewegungen wie „vorwärts laufen“ aus einzelnen Gelenk-Bewegungen zusammensetzen (momentan besteht `RobotMovement` nur aus „vorwärts laufen“). Das „vorwärts Laufen“ (Siehe: Kapitel 3) wurde hier mit Hilfe einer State-Machine implementiert.

Die Klasse `RobotMovement` kann in Zukunft um weitere Bewegungsmuster erweitert werden, sodass der Roboter bspw. rückwärts und seitwärts laufen, oder den Körper neigen (hoch- und runter bewegen) kann.

Literatur

[1] AJAYKUMARKANNAN: *Explanation of the PWM Modules on the Beaglebone.* https://makingaquadrotor.wordpress.com/2012/09/06/explanation-of-the-pwm-modules-on-the-beaglebone/. Version: 2012. – [Online; abgerufen 17.07.2016]

[2] PIERCE, Scott: *mePed Quadruped Robot.* http://www.thingiverse.com/thing:756681. Version: 2015. – [Online; abgerufen 17.07.2016]

Anhänge

A. Roboter (Bey)

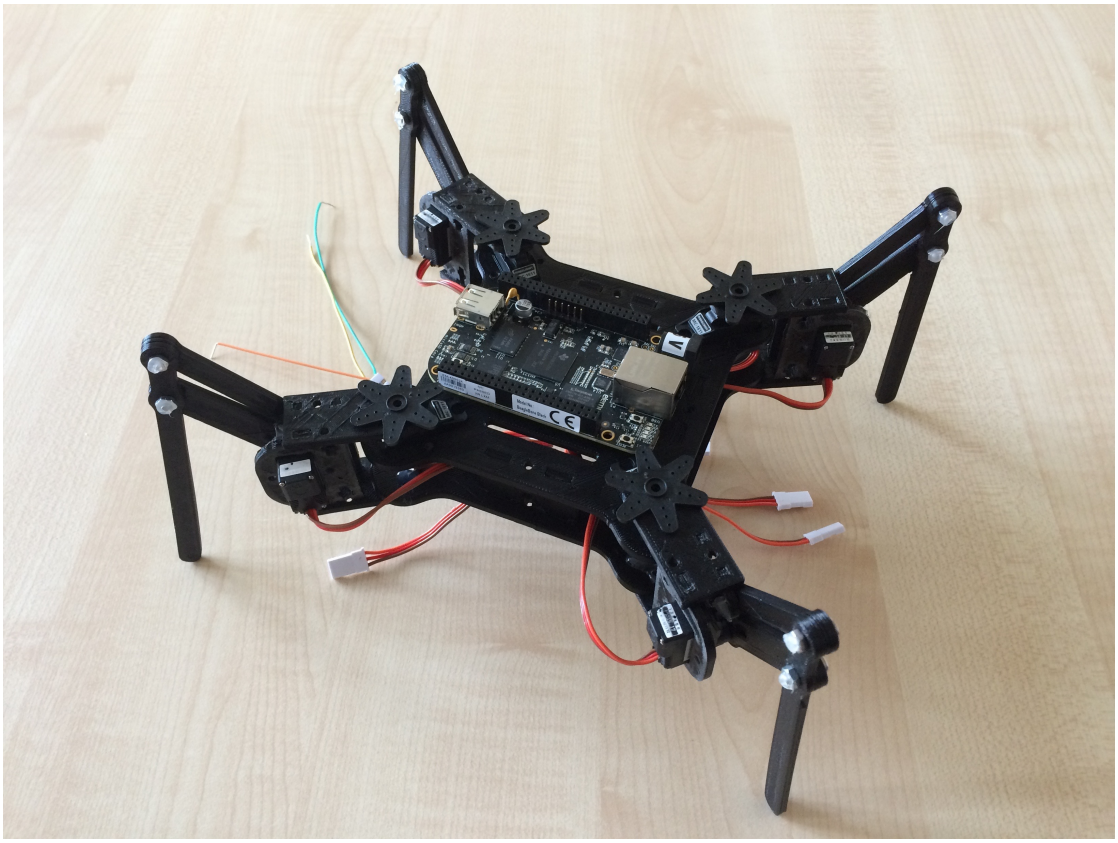

Abbildung 9: (Anhang) verwendeter Roboter

B. Quellcode (Sza)

Die Quellcode-Dateien sind in der zugehörigen ZIP-Datei angefügt.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 30. Juli 2016	
Ort, Datum	Unterschrift

Hamburg, 30. Juli 2016	
Ort, Datum	Unterschrift