Arbeitskreis SMART MACHINES

Hochschule Kaiserslautern University of Applied Sciences

B.O.T. CHALLENGE

THYMIO II - ASEBA STUDIO

Funktionen und Programmiersprache

INHALT

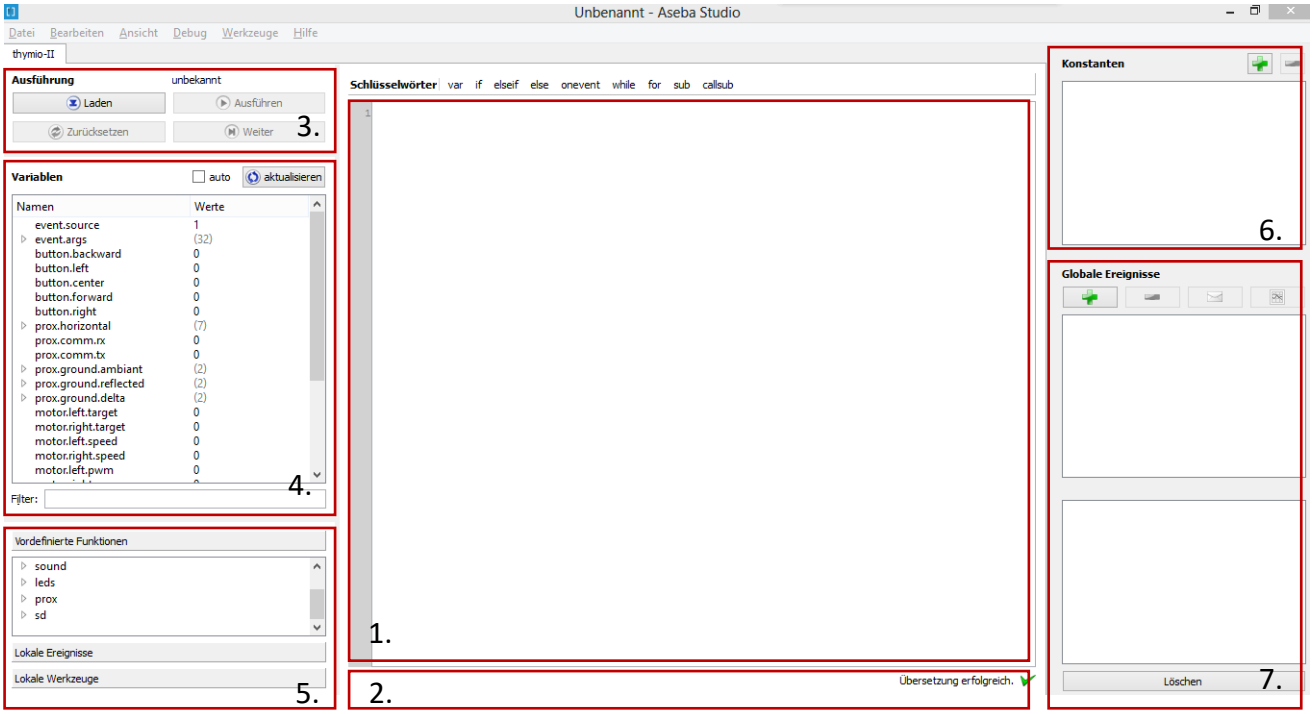
Dieses Tutorial beschreibt zunächst die Oberfläche des Aseba Studios. Anschließend wird die eigene Programmiersprache Aseba erläutert.

Webseite der B.O.T Challenge

Inhalt

Oberfläche

Aseba Studio ist die Entwicklungsumgebung, in welcher der Thymio II programmiert wird. Im Folgenden wird die Oberfläche des Aseba Studios erläutert:



1. Intelligente Eingabe

Die Syntax der Befehle wird im Eingabefeld automatisch farblich hervorgehoben, Absätze eingerückt und Variablen für den Speicher abgekürzt. Außerdem wird die momentane Position im Modus "Schritt für Schritt" laufend aktualisiert und Fehler werden in Rot hervorgehoben.

2. Sofortige Kompilierung

Die Software kompiliert die Befehle noch während der Benutzer am Schreiben ist.
Das Resultat der Kompilierung (Erfolg oder Beschreibung des Fehlers) wird unter dem Eingabefeld angezeigt. Dies ist äußerst nützlich, da die Fehler sofort behoben werden können und damit die Qualität des Codes erhöht wird.

3. Fehlersucher

Mit diesen Knöpfen kann der Code auf den Roboter geladen und gestarten werden.

Aseba Studio beinhaltet einen Fehlersucher. Für jedes Netzwerkelement wird der aktuelle Ausführungsstatus angezeigt. Die kontinuierliche Ausführung und die Haltepunkte werden Schritt für Schritt angezeigt. Ein Klick mit der rechten Maustaste innerhalb des Befehlseingabefeldes erlaubt es in der ausgewählten Linie, einen Haltepunkt zu setzen oder zu entfernen. Nach jedem Haltepunkt oder Schritt werden die Variablen aktualisiert.

4. Überprüfen der Daten

Das Programm listet die verfügbaren Variablen jedes Netzwerkelements auf. Mit einem einzigen Klick kann diese Liste aktualisiert werden. Diese Liste gibt einen Gesamtüberblick aller momentanen Werte der Netzwerkelemente: die Werte der Sensoren, der Aktoren und der Variablen.

5. Lokale Ereignisse und vorgegebene Funktionen

Es werden die lokalen Ereignisse jedes Netzwerkelements und die vorgegebenen Funktionen angezeigt. In der Werkzeughilfe wird jede vorgegebene Funktion kurz beschrieben.

6. Konstanten definieren

Es ist möglich, Konstanten für alle vorhandenen Netzwerkelemente zu definieren.

7. Netzwerk-weit Ereignisse

Die Namen der Ereignisse können vom Benutzer selbst definiert werden und mit einem Doppelklick auf den Namen wird das korrespondierende Ereignis gesendet. Unter der Liste aller Ereignisnamen, zeigt ein Bildschirm die kürzlich eingetretenen Ereignisse über eine gewisse Zeit mit ihren Parametern an. Dies erlaubt das Überwachen des Verteilungsverhaltens des Netzwerks.

Tipps & Tricks

In Aseba Studio können Stichwörter in den Seitenfenstern angeklickt und ins mittlere Programmierfenster gezogen werden. Das erspart Schreibarbeit.
Außerdem können auch in Aseba Zeilen angewählt und wie bei Windows mit Kopieren und Einfügen (Ctrl-C, Ctrl-V in Windows und Linux, oder Command-C, Command-V in Mac) verschoben werden.

Programmierung mit Aseba

Quelle: https://www.thymio.org/de:asebalanguage

Diese Hilfe finden Sie auch innerhalb des Aseba Studios im Menu Hilfe -> Sprache.

Die Syntax der Programmiersprache Aseba gleicht derjenigen von Matlab (verbreitete wissenschaftliche Programmiersprache). Dies ermöglicht Programmierern mit Vorkenntnissen sich schnell zu orientieren und Aseba in Kürze zu erlernen. Semantisch gesehen, ist Aseba eine einfache, imperative Programmierung mit einem einzelnen Datentyp (16 Bit Ganzzahlen mit Vorzeichen) und Vektoren davon. Diese Einfachheit erlaubt auch Programmierern ohne Vorkenntnisse Mikrokontroller-basierte Roboter zu steuern.

Kommentare

Kommentare erlauben, in den Programmtext Information einzufügen, die vom Compiler ignoriert wird. Dies ist sehr nützlich um Informationen in natürlicher Sprache einzufügen oder zwischenzeitlich Code-Teile auszuschalten.
Kommentare beginnen mit dem Symbol # und enden auf derselben Zeile.

Beispiel:

```
# dies ist ein Kommentar
var b    # ein anderer Kommentar
```

Bemerkungen über mehrere Linien sind auch möglich. Sie beginnen mit dem Symbol *#** und enden mit ***#**.

Beispiel:

```
#*
Dies ist ein Kommentar
über mehrere Linien verteilt.
*#
var b    # kurzer Kommentar
```

Skalare

Skalare Variablen sind Variablen, die Nummern enthalten. Sie können in allen Ausdrücken vorkommen, wie zum Beispiel bei der Initialisierung einer Variable, in einem mathematischen Ausdruck oder in einer logischen Bedingung.

Notation

Skalare können in verschiedenen Basen dargestellt werden. Für uns ist die natürlichste Basis die Basis 10, die zum Dezimalsystem führt, bei dem die Ziffern von 0 bis 9 verwendet werden. Eine negative Zahl wird durch ein vorangestelltes - (Minus-) Zeichen gekennzeichnet.

```
i = 42
i = 31415
i = -7
```

Sowohl das Dualsystem als auch das Hexadezimalsystem werden unterstützt. Binäre (Dual-) Zahlen bekommen ein 0b vorangestellt, hexadezimale Zahlen ein 0x.

```
# Binäre Schreibweise
i = 0b110          # i = 6
i = 0b11111111     # i = 255

# Hexadezimale Schreibweise
i = 0x10       # i = 16
i = 0xff       # i = 255
```

Variablen

Variablen beziehen sich entweder auf einfache ganze Zahlen oder Vektoren von ganzen Zahlen. Die zulässigen Werte sind alle ganzen Zahlen zwischen -32768 und 32767, was einem Speicherbereich von 16 Bit (2 Byte) entspricht. Man kann Vektorkomponenten mit eckigen Klammern [Zahl] aufrufen. Die Indexierung der Vektoren beginnt bei Null. Alle benutzerdefinierten Variablen müssen am Anfang des Asebaprogramms definiert werden, vor allen weiteren Programmteilen.

Der Name einer Variablen muss diesen drei Regeln genügen:

- Der Name darf nur groß- und kleingeschriebene Buchstaben enthalten, sowie '_' und '.'.
- Der Name muss mit einem Buchstaben oder einem '_' anfangen.
- Beim Namen wird Groß- und Kleinschreibung unterschieden. Die Variable "foo" ist nicht gleich der Variable "Foo".
- Der Name darf keines der reservierten Schlüsselwörter von Aseba sein.

Variablen dürfen während der Deklaration initialisiert werden, indem das Zuweisungssymbol mit einem beliebigen gültigen mathematischen Ausdruck verbunden wird. Eine Variable, die nicht initialisiert wurde, enthält einen nicht-deterministischen Wert. Dieser ist höchstwahrscheinlich nicht null!

Beispiel:

```
var a
var b = 0
var c = 2*a + b          # Warnung: 'a' wurde nicht initialisiert
```

• Reservierte Schlüsselwörter

Die folgenden Schlüsselwörter dürfen nicht als Namen für Variablen benutzt werden, da sie Bestandteil der Aseba-Sprache sind.

Schlüsselwörter			
abs	call	callsub	do
else	elseif	emit	end
for	if	in	onevent
return	step	sub	then
var	when	while	

• Konstanten

Konstanten können in Aseba Studio im "Konstanten"-Panel definiert werden, aber nicht direkt im Programm-Code. Eine Konstante ist eine Zahl, die überall dort benutzt werden kann, wo eine einfache Zahl direkt benutzt werden könnte. Der Unterschied zu einer herkömmlichen Variable ist der, dass Konstanten nach ihrer Definition nicht mehr verändert werden können. Dies ist nützlich, wenn Sie das Verhalten verschiedener Abläufe an einer zentralen Stelle kontrollieren möchten, wie zum Beispiel einen Schwellwert anpassen, der von mehreren Aseba-Knoten genutzt wird. Eine Konstante muss einen eindeutigen Namen haben, insbesondere nicht den Namen einer bereits bestehenden Variablen. Andernfalls wird bei der Code-Übersetzung ein Fehler bemängelt.

```
# angenommen, wir haben eine Konstante mit dem Namen THRESHOLD definiert
var i = 600
if i > THRESHOLD then
  i = THRESHOLD - 1
end
```

• Vektoren

Vektoren entsprechen einem zusammenhängenden Bereich im Speicher, der wie eine einzelne logische Instanz verwendet werden kann. Sie können mit den üblichen eckigen Klammern [] deklariert werden. Die Zahl zwischen den eckigen Klammern legt die Anzahl der in diesem Vektor enthaltenen Elemente fest. Dies ist die Größe des Vektors und kann im Nachhinein nicht verändert werden. Die Größe wird durch einen konstanten Ausdruck angegeben, der auch mathematische Operationen mit Skalaren und Konstanten umfassen darf. Optional können dem Vektor mit dem Vektor-Konstruktor (Beispiel siehe unten) Werte zugewiesen werden. Wird dieser Konstruktor benutzt, so muss die Größe des Vektors nicht mehr angegeben werden.

Beispiel:

```
var a[10]                  # Vektor mit 10 Element
var b[3] = [2, 3, 4]       # Initialisierung
var c[] = [3, 1, 4, 1, 5]  # Größe wird implizit auf 5 gesetzt
var d[3*FOO-1]             # Größe wird durch einen konstanten Ausdruck bestimmt (FOO ist eine Konstante)
```

Auf Vektoren kann auf verschiedene Weise zugegriffen werden:

- Auf ein einzelnes Element kann mit eckigen Klammern zugegriffen werden, die einen einzelnen Wert umschließen. Der kleinste Index des Vektors ist 0. Beliebige mathematische Ausdrücke mit anderen Variablen können als Index verwendet werden.
- Eine Auswahl an Elementen kann durch eckige Klammern getroffen werden, die mehrere konstante Ausdrücke enthalten, die durch einen Doppelpunkt ':' voneinander getrennt werden.
- Werden die eckigen Klammern weggelassen, so greift man auf den gesamten Vektor zu.

Beispiel:

```
var foo[5] = [1,2,3,4,5]
var i = 1
var a
var b[3]
var c[5]
var d[5]

a = foo[0]              # Kopiere das erste Element
a = foo[2*i-2]          # von 'foo' nach 'a'

b = foo[1:3]            # Kopiere das 2., 3. und 4. Element
b = foo[1:2*2-1]        # aus 'foo' nach 'b'

c = foo                 # Kopiere alle 5 Element von 'foo' nach 'c'
d = c * foo             # Multipliziere die Vektoren 'foo' und 'c' ein Element nach dem anderen und speichere das Ergebnis in 'd'
```

• Vektor-Konstruktoren

Mithilfe von Vektor-Konstruktoren können Arrays aus Variablen, anderen Arrays, Skalaren und sogar komplexen Ausdrücken erzeugt werden. Sie kommen vor allem bei der Initialisierung anderer Arrays oder als Operanden in Ausdrücken, Funktionen und Ereignissen vor. Ein Vektor wird mit eckigen Klammern konstruiert, die mehrere Ausdrücke enthalten können, die jeweils durch ein Komma ',' getrennt werden. Die Größe des Konstruktors ist die Summe der Größen aller individuellen Ausdrücke und muss mit der Größe des Vektors, in dem der Konstruktor gespeichert werden soll, übereinstimmen.

Beispiel:

```
var a[5] = [1,2,3,4,5]      # Konstruktor um einen Vektor zu initialisieren
var b[3] = [a[1:2],0]       # b wird mit [2,3,0] initialisiert
a = a + [1,1,1,1,1]         # addiere zu jedem Element aus 'a' eins dazu
a = [b[1]+2,a[0:3]]         # a ist jetzt [5,2,3,4,5]
```

Ausdrücke und Zuweisungen

Ausdrücke erlauben mathematische Berechnungen und sind in einer normalen traditionellen mathematischen Notation (Infix syntax) geschrieben. Zuweisungen benutzen = und weisen das Resultat einer Ausdrucksberechnung einer Variable zu (oder einer Komponente einer Variable, falls die Variable ein Vektor ist). In Aseba gibt es verschiedene Operatoren, welche in der Tabelle unten mit ihrer Priorität aufgeführt sind. Um einen Ausdruck in einer anderen Reihenfolge zu evaluieren, können Klammern benutzt werden.

Priorität	Operator	Beschreibung	Assoziativität	Stelligkeit
1	()	Gruppieren eines Unter-Ausdrucks		unär
2	* /	Multiplikation, Division		binär
	%	Modulo		binär
3	+ -	Addition, Subtraktion		binär
4	<< >>	Linksverschiebung, Rechtsverschiebung		binär
5	|	binäres oder (or)	links assoziativ	binär
6	^	binäres, exklusives oder (xor)	links assoziativ	binär
7	&	binäres und	links assoziativ	binär
8	-	unäres minus		unär
9	~	binäres nicht		unär
10	abs	absoluter Wert		unär
11	=	Zuweisung		binär
	|= ^= &=	Zuweisung durch binäres oder, xor, und		binär
	~=	Zuweisung durch binäres nicht		binär

	*= /=	Zuweisung durch Produkt und Quotient		binär
	%=	Zuweisung durch modulo		binär
	+= -=	Zuweisung durch Summe und Differenz		binär
	<<= >>=	Zuweisung durch Links- oder Rechtsverschiebung		binär
	++ --	unäres Inkrement und Dekrement		unär

Die *Zuweisung durch* Versionen der binären Operatoren funktioniert, indem sie den Operator auf eine Variable anwenden und das Resultat in derselben Variablen speichern. Zum Beispiel bedeutet A* = 2 das Gleiche wie A = A*2. Diese Abkürzungen sollen den Code einfacher lesbar machen.

Beispiel:

```
a = 1 + 1
# Resultat: a = 2
a *= 3
# Resultat: a = 6
a++
# Resultat: a = 7

b = b + d[0]
b = (a - 7) % 5
c[a] = d[a]
c[0:1] = d[2:3] * [3,2]
```

Benutzung

Mathematische Ausdrücke sind ein sehr allgemeines Werkzeug. Sie können in vielen verschiedenen Situationen benutzt werden, zum Beispiel:

- auf der rechten Seite einer Zuweisung
- als Index beim Zugriff auf Element eines Vektors
- in Funktions-Aufrufen
- als Argumente beim Auslösen von Ereignissen

Ablaufsteuerung

Bedingungen

Aseba stellt zwei Arten von Bedingungen zur Verfügung: if und when. Vergleichende Operatoren sind ==, !=, >, >=, <, und <=; beliebige Ausdrücke können verglichen werden. Vergleiche können gruppiert werden mit Hilfe von and (logische Konjunktion), or (logische Disjunktion), und not (logische Negation) Operatoren sowie mit Klammern.
Ein Vergleich besteht aus einem Operator und zwei Operanden und kann entweder richtig

oder falsch sein. Die Operanden können beliebige Ausdrücke sein. Die folgende Tabelle listet die Vergleichsoperatoren auf:

Operator	Wahrheitswert
==	richtig, wenn die Operanden gleich sind (gleich)
!=	richtig, wenn die Operanden unterschiedlich sind (ungleich)
>	richtig, wenn der erste Operand grösser ist als der zweite (grösser)
>=	richtig, wenn der erste Operand grösser oder gleich gross ist wie der zweite (grösser-gleich)
<	richtig, wenn der erste Operand kleiner ist als der zweite (kleiner)
<=	richtig, falls der erste Operand kleiner oder gleich gross ist wie der zweite (kleiner-gleich)

Sowohl if als auch when führen den darauffolgenden Befehlsblock aus, falls die Bedingung nach if bzw. when erfüllt ist. Allerdings führt when den darauffolgenden Befehlsblock ausschließlich dann aus, wenn die letzte Auswertung der Bedingung falsch war und die momentane Bedingung richtig ist. Dies ermöglicht eine effizientere Arbeitsweise, da der Befehlsblock nur dann ausgeführt wird, wenn sich etwas geändert hat.

Beispiel:

```
if a - b > c[0] then          #wenn a - b > [0] dann
   c[0] = a
elseif a > 0 then             #falls die erste Bedingung (a - b > c[0]) falsch war und (a>0)
richtig, dann
      b = a
else                          #falls keine der vorhergehenden Bedingungen richtig war
      b = 0
end                           #Ende des if Befehlsblocks

when a > b do                 #wenn a > b mach
   leds[0] = 1
end                           #Ende des when Befehlsblocks
```

Hier wird der Befehlsblock when nur ausgeführt, falls a größer als b ist.

▪ Schleifen

Zwei Konstruktionen ermöglichen Schleifen: *while* und *for*.

Eine *while* Schleife wiederholt einen Befehlsblock solange wie die verlangte Bedingung am Ende zutrifft. Die Funktion hat dieselbe Form wie wenn man if benutzt.

Beispiel:

```
while i < 10 do                  #während i < 10 führe den folgenden Befehlsblock aus
   v = v + i * i
```

```
  i = i + 1
end                              #Ende des while Blocks
```

Eine *for*-Schleife erlaubt es, eine Variable über einen Bereich ganzer Zahlen laufen zu lassen, optional mit einer Schrittweite.

Beispiel:

```
for i in 1:10 do                 #für i in 1,2,3,...,9,10 führe aus
  v = v + i * i
end
for i in 30:1 step -3 do         #für i in 30,27,24,21,...,6,3 führe aus
  v = v - i * i
end
```

Blöcke

• Unterprogramme

Falls dieselben Sequenzen von Operatoren an zwei oder mehreren Orten innerhalb des Codes auftreten, genügt es, ein Unterprogramm zu schreiben und dieses von diversen Orten her aufzurufen. Ein Unterprogramm kann mit dem Stichwort *sub* gefolgt vom Namen des Unterprogramms definiert werden. Ein Unterprogramm kann mit dem Stichwort *callsub* aufgerufen werden. Unterprogramme müssen immer erst definiert werden, bevor sie aufgerufen werden können. Unterprogramme können weder Argumente beinhalten noch rekursiv sich selbst aufrufen, weder direkt noch indirekt. Allerdings haben Unterprogramme Zugriff auf alle Variablen.

Beispiel:

```
var v = 0
sub toto                 #Unterprogramm Namens toto
v = 1
onevent test             #falls Ereignis Namens test eintritt, dann
callsub toto             #rufe Unterprogramm toto auf
```

• Ereignisse

Aseba besitzt eine ereignisbasierte (*event-based*) Architektur, was eine asynchrone Ereignisauslösung ermöglicht. Extern können Ereignisse zum Beispiel von einem anderen Aseba-Netzwerkelement ausgelöst werden. Intern können Ereignisse zum Beispiel von einem Sensor mit aktualisierten Daten ausgelöst werden.

Die Wahrnehmung eines Ereignisses kann einen Codeblock aktivieren, falls dieser zuvor mit dem Stichwort *onevent* und der Bezeichnung des Ereignisses definiert worden ist. Der Code am Anfang des Befehls bestimmt, wann die darauf folgenden Befehle ausgeführt oder zurückgesetzt werden. Befehle können auch Ereignisse senden mit dem Stichwort *emit*, gefolgt von der Bezeichnung des Ereignisses und gegebenenfalls den zu sendenden

Variablen. Falls eine Variable gegeben wird, muss die Größe des Ereignisses der Größe des zu schickenden Arguments entsprechen. Ereignisse erlauben Befehle an anderen Netzwerkelementen auszulösen oder mit einem externen Programm zu kommunizieren.

Um die Ausführung von ähnlichen Codes bei neuen Ereignissen zu ermöglichen, dürfen die Befehle nicht blockieren und dürfen deshalb keine Endlosschlaufen enthalten. In der Robotik bedeutet dies, dass ein traditionelles Robotersteuerungsprogramm gewisse Vorgänge in einer Endlosschlaufe durchführt, während die Befehlssprache Aseba dieselben Vorgänge nur in einem auf Sensoren bezogenen Ereignisse ausführt.

Beispiel:

```
var run = 0
onevent start                                   #falls das Ereignis 'start' eintritt, starte
run = 1
onevent stop                                    #falls das Ereignis 'stop' eintritt, halte an
run = 0
onevent ir_sensors
if run == 1 then                                #wenn==1, dann
    emit sensors_values proximity_sensors_values                    #Gib das Ereignis
'sensors_values' aus mit den Variablen 'proximity_sensors_values'
end                                             #Ende des 'if' blocks
```

- Return-Befehl

Mit einem *return*-Befehl ist es möglich ein Unterprogramm früh zu verlassen oder die Ausführung eines Ereignisses abzubrechen.

Beispiel:

```
var v = 0
sub toto
if v == 0 then
    return
end
v = 1
onevent test
callsub toto
return
v = 2
```

Senden externer Ereignisse

Das Programm kann externe Ereignisse versenden, indem das Schlüsselwort *emit* benutzt wird. Auf das Schlüsselwort folgen der Name des Ereignisses und der Name der Variablen,

falls vorhanden. Falls eine Variable zur Verfügung steht, muss die Größe des Ereignisses der Größe des Arguments entsprechen, das herausgegeben wurde. Anstelle einer Variablen können für komplexere Situationen auch Vektor-Konstruktoren und mathematische Formeln verwendet werden. Ereignisse erlauben dem Programm die Ausführung des Programmcodes eines weiteren Knotens auszulösen oder um mit einem externen Programm zu kommunizieren.

```
onevent ir_sensors
emit sensors_values proximity_sensors_values
```

Vorgegebene Funktionen

Die Befehlssprache Aseba wurde entwickelt mit dem Ziel, auch Programmierern ohne Vorkenntnisse zu ermöglichen, die einfachen Befehle schnell zu verstehen und effizient in Mikrosteuerungen zu implementieren. Um komplexe oder stark Ressourcen verbrauchende Prozesse zu programmieren, werden gewisse vorprogrammierte Funktionen zur Verfügung gestellt. Zum Beispiel gibt es in Aseba eine vorgegebene Funktion, die das Skalarprodukt berechnet.

Vorgegebene Funktionen sind sicher, weil sie die Argumente spezifizieren und überprüfen. Argumente können Konstanten, Variablen oder Feldzugänge sein. Später können auf ganze Felder, einzelne Elemente oder Teilfelder zugegriffen werden. Vorgegebene Funktionen sehen ihre Argumente als Referenz an und können ihre Inhalte verändern, aber keine Werte generieren. Durch das Stichwort *call* können vorgegebene Funktionen aufgerufen werden.

Beispiel:

```
var a[3] = 1, 2, 3
var b[3] = 2, 3, 4
var c[5] = 5, 10, 15
var d
call math.dot(d, a, b, 3)
call math.dot(d, a, c[0:2], 3)
call math.dot(a[0],c[0:2],3)
```

Quelle: https://www.thymio.org/de:asebastdnative

Vorgegebene Funktionen Standard-Bibliothek

Aseba bringt eine Standard-Bibliothek mit, die von Haus aus bereits viele Funktionen enthält. Die meisten Geräte liefern diese Standard-Bibliothek mit.[1] Falls die Firmware Ihres Gerätes schon etwas älter ist, unterstützt sie möglicherweise noch nicht neu hinzugekommene Funktionen. Seit Aseba 1.1 sind folgende Funktionen verfügbar[2]:

`math.copy(A,B)`
: Kopiert eines nach dem anderen jedes Element des Vektors B in den Vektor A : $A_i = B_i$.

`math.fill(A,c)`
: Schreibt die Konstante c in jedes Element des Vektors A : $A_i = c$.

`math.addscalar(A, B, c)`
: Berechne $A_i = B_i + c$, wobei c ein Skalar ist.

`math.add(A, B, C)`
: Berechne $A_i = B_i + C_i$, wobei$A$, B und C drei gleichgrosse Vektoren sind.

`math.sub(A, B, C)`
: Berechne $A_i = B_i - C_i$, wobei A, B und C drei gleichgrosse Vektoren sind.

`math.mul(A, B, C)`
: Berechne $A_i = B_i \cdot C_i$, wobei A, B und C gleichgrosse Vektoren sind. *Beachte: Dies ist kein Skalarprodukt!*

`math.div(A, B, C)`
: Berechne $A_i = B_i / C_i$, wobei A, B und C drei gleichgrosse Vektoren sind. *Eine Division durch Null führt zu einer Ausnahmesituation.*

`math.min(A, B, C)`
: Speichere für jedes Element das Minimum der Vektoren B und C in A, wobei A, B und C drei Vektoren gleicher Grösse sind: $A_i = \min(B_i, C_i)$.

`math.max(A, B, C)`
: Speichere für jedes Element das Maximum der Vektoren B und C in A, wobei A, B and C drei Vektoren gleicher Grösse sind: $A_i = \max(B_i, C_i)$.

`math.dot(r, A, B, n)`
: Berechne das Skalarprodukt zweier gleichgrosser Vektoren A und B : $r = \frac{\sum_i (A_i \cdot B_i)}{2^n}$

`math.stat(V, min, max, mean)`
: Berechne das Maximum, den Mittelwert und das Minimum des Vektors V.

`math.argbounds(A, argmin, argmax)`
: Bestimme die Indizes *argmin* und *argmax*, die dem kleinsten und dem grössten Wert aus A entsprechen.

`math.sort(A)`
: Sortiere den Vektor A in situ.

`math.muldiv(A, B, C, D)`
: Berechne die Multiplikations-Division mit 32-bit-Genauigkeit. $A_i = \frac{B_i \cdot C_i}{D_i}$. *Eine Division durch Null führt zu einer Ausnahmesituation.*

`math.atan2(A, Y, X)`
: Berechne $A_i = \arctan\left(\frac{Y_i}{X_i}\right)$, wobei die Zeichen von Y_i und X_i den Ausgabe-Quandranten festlegen. A, Y und X müssen Vektoren gleicher Grösse sein. *Beachte, dass $X_i = 0$ und $Y_i = 0$ zu dem Ergebnis $A_i = 0$ führen.*.

`math.sin(A, B)`
: Berechne $A_i = \sin(B_i)$, wobei A und B zwei Vektoren gleicher Grösse sind.

`math.cos(A, B)`
: Berechne $A_i = \cos(B_i)$, wobei A und B zwei Vektoren gleicher Grösse sind.

`math.rot2(A, B, Winkel)`
: Rotiere den Vektor B um *Winkel* und speichere das Ergebnis in A. *Beachte: A und B müssen beide genau 2 Elemente enthalten.*

`math.sqrt(A, B)`
: Berechne $A_i = \sqrt{B_i}$, wobei A und B zwei Vektoren gleicher Grösse sind.

`math.nzseq(a, B, m)`
: Speichere in a den mittleren Index der längsten Aufeinanderfolge von Zahlen in B, die nicht Null sind oder -1 falls es keine solche gibt oder dessen Länge kleiner als m ist.

`math.rand(v)`
: Bestimme einen zufälligen Wert v aus dem zwischen -32768 und 32767.

Footnotes

1. Geräte mit besonders wenig Flash-Speicher bieten unter Umständen nur eine Teilmenge oder gar keine dieser Funktionen an.
2. In die trigonometrische Funktionen -32768,32767 entsprechen Winkeln von [-pi,pi[.