|  |
| --- |
|  |
| Dokumentation: Gemeinsame API e-Puck und LDVbot |
|  |

# Optimales Modell

Das optimale Modell für eine gemeinsame API zwischen den Robotern e-Puck und LDVbot basiert darauf, dass Algorithmen nur einmal geschrieben werden müssen um sie auf beiden Robotern laufen zu lassen. Um dies komplett zu realisieren, müssten die Roboter die selben Sensoren besitzen. Da jedoch nur einige der Sensoren auf beiden Robotern vorhanden sind, ist eine einheitliche API in diesem Fall noch nicht umzusetzen. Jedoch kann eine Umsetzung für die gemeinsamen Sensoren angestrebt werden. Im optimalen Fall untergliedern wir die API in drei Teile. Der erste Teil sind die gemeinsamen Sensoren, welche auf den Robotern laufen. Diese sollten im optimalen Fall identisch umgesetzt werden um hier eine Benutzung auf e-Puck und LDVbot zu gewährleisten. Der zweite Teil ist die Kommunikation von Code und Roboter. Der dritte und letzte Teil sind dann die zusätzlichen Sensoren, die nicht auf beiden Robotern vorhanden sind.

Abbildung 1: Optimales Modell der gemeinsamen API

## Gemeinsame Sensoren

Wie beschrieben sind die gemeinsamen Sensoren, die Sensoren die sowohl auf dem e-Puck als auch auf dem LDVbot vorhanden sind. Im optimalen Fall sollten diese auf beiden Robotern identisch angesprochen und benutzt werden. In unserem Fall wären der Motor, die LEDs, der Bodensensor und der Näherungssensoren die gemeinsam benutzen Sensoren. Jedoch ist eine Implementierung nach dem optimalen Modell hier schon schwer umzusetzen, da bis auf die LEDs alle anderen Sensoren anders angesprochen werden. Besonders auffällig ist das beim Motor. Zwar besitzen beide Motoren einen Schrittmotor jedoch bietet die Firmware des e-Puck keine Möglichkeit an, den Roboter eine bestimmte Distanz mit einem festgelegten Winkel fahren zu lassen, während dies beim LDVbot umgesetzt ist. Würde man in diesem Fall die API nach dem optimalen Modell implementieren, wären in der Klasse Methoden vorhanden die jeweils nur ein Roboter benutzen kann. Dies würde die Benutzerfreundlichkeit der API sehr stark einschränken, da neue Benutzer in der Dokumentation nachlesen müssten, welche Methoden für welchen Roboter zulässig sind um eine Aneinanderreihung von Fehlermeldungen zu verhindern. Ein weiteres Problem wäre bei dieser Umsetzung, dass es schwer möglich ist einen neuen Roboter in die API zu integrieren, weil dieser alle gemeinsamen Sensoren besitzen muss um einen Umbau der API zu verhindern.

## Kommunikationsmodul

Der zweite Teil des optimalen Modells ist das Kommunikationsmodul. Das Modul soll die Kommunikation zwischen den Methoden und der Firmware auf dem physikalischen Roboter darstellen. Im besten Fall ist die Kommunikation für beide Roboter identisch. Das heißt gemeinsame Sensoren werden mit der selben Byteausgabe bedient. Dies ist jedoch im jetzigen Zustand nicht umgesetzt. Eine Implementierung die nahe an der optimalen Lösung liegt wäre hier jedoch möglich, indem die öffentlichen Methoden die Unterscheidung treffen welcher Roboter gerade in Benutzung ist und diese dann die privaten Methoden, die zur Kommunikation mit dem jeweiligen Roboter dienen ausführen. Da die API in Python geschrieben ist, würde der Benutzer beim Programmieren sehen welche Methoden alle vorhanden sind und könnte versuchen gleich die privaten Methoden auszuführen. Diese Umsetzung würde gleichzeitig auch die Übersicht einschränken, weil dem Benutzer in den meisten Python-Entwicklungsumgebungen alle Methoden des Modules angezeigt werden. Genauso wie bei den gemeinsamen Sensoren stellt hier auch das Integrieren eines dritten Roboters ein weiteres Problem da. Sollte die optimale Lösung einer identischen Kommunikation umgesetzt werden, müsste dieser Roboter auf die selbe Weise kommunizieren, wie die vorherigen. Sollte jedoch die Variation mit den privaten Methoden implementiert werden, würde dies die Übersicht für den Benutzer noch weiter erschweren, weil neue Methoden für den dritten Roboter implementiert werden müssten.

## Getrennte Sensoren

Den letzten Teil stellen die getrennten Sensoren da. In dieser Kategorie werden alle Sensoren implementiert, die einzigartig sind für den jeweiligen Roboter. Deswegen werden im optimalen Fall beide Roboter getrennt und dessen Sensoren werden über diese Roboterklasse angesprochen. Bei dieser Umsetzung würde der Benutzer nur die Klassen und Methoden sehen, die für den ausgewählten Roboter zur Verfügung stehen. Da hier keine Rücksicht genommen werden muss wie andere Roboter diesen Sensor handhaben, stellt dieser Teil kein Problem bei der Integrierung zusätzlicher Roboter da.

# Implementiertes Modell

Da das optimale Modell in der bestehenden Lage starke schwächen aufweist wurde sich entschlossen ein anderes Modell zu implementieren. Eine Überführung in das optimale Modell wäre jedoch möglich, wenn die Schwächen die vor allem in der Firmware der Roboter liegen, behoben werden.

Das implementierte Modell benutzt die Umsetzung der zusätzlichen Sensoren des optimalen Modells. Es wird hier also gleich zu Beginn unterschieden welcher Roboter benutzt wird. Somit stehen dem Benutzer später nur die Klassen und Methoden zu Verfügung, die der Roboter bedient.

Um die Überführung in das optimale Modell zu vereinfachen und eine mögliche Benutzung des selben Codes für beide Roboter zu ermöglichen, wurden die Methoden von gemeinsamen Sensoren identisch benannt. Möchte man jetzt zum Beispiel LED 1 setzen, benutzt man bei beiden Robotern die Methode *setLED()*. Auch kann ein weiterer Roboter in diesem Modell einfach integriert werden, da die Unterscheidung welcher Roboter benutzt wird gleich am Anfang gemacht wird.

Abbildung 2: Implementiertes Modell

## e-Puck

Der e-Puck hat in diesem Modell 11 Untermodule die alle Sensoren und Motoren und die Kommunikation darstellen. Jedoch sollte von außerhalb nie direkt auf die Sensoren zugegriffen werden. Um Sensoren und Motoren anzusprechen benutzt man die jeweilige *get()* Funktion des gewünschten Sensors um mit diesem zu kommunizieren. Auch wenn die Methoden für die Kamera in dieser API implementiert sind, kann diese nur benutzt werden wenn man sich über Bluetooth verbindet, da die Firmware nur so auf die Anfragen reagiert. Auch besteht beim e-Puck noch ein Fehler bei der Benutzung von *setPosition()* in der Motor Klasse, der den e-Puck einfrieren lässt.

## LDVbot

Der LDVbot hat 9 Untermodule, die den Hauptteil der vorherigen Funktionalitäten zur Verfügung stellen, um Benutzern der alten API einen schnellen Umstieg zu gewährleisten. Neben Motor, LEDs, Bodensensor und Annäherungssensoren kann beim LDVbot noch auf die Batterie zugegriffen werden um den jeweiligen Status abzufragen. Außerdem dem LDVbot eine Nummer zugewiesen werden, über die die Kommunikation stattfindet. Jeder vorhandene LDVbot hat jedoch schon eine Standardnummer, wodurch eine Änderung meistens nicht notwendig ist. Das Infrarotmodul hat keine Änderungen zum Vorgänger erfahren, da es ein experimentelles Modul war und eine Änderung neben der Benutzung auch Änderungen in der Firmware mit sich bringen könnte. Der analoge Zugriffe auf einige der Sensoren wurde im Modul *Analog* implementiert. Beim LDVbot wurden die Methoden Blink und Spin entfernt, da diese nicht geeignet waren für eine API und bei Benutzung selbst über die zur Verfügung stehenden Methoden implementiert werden können.

# Verbesserungsvorschläge

Da das Ziel die Benutzung des selben Codes für beide Roboter war, sollte in Zukunft an der API und der jeweiligen Firmware gearbeitet werden um dies zu erreichen und die API in das optimale Modell überzuführen. Als erstes sollte versucht werden eine einheitliche Kommunikation aufzubauen. Der zweite Schritt wäre die Benutzung gemeinsamer Sensoren soweit wie möglich zu vereinheitlichen. So sollte der e-Puck Schrittmotor auf die selbe Weise angesprochen werden wie der LDVbot Motor. Einzelne Methoden können dann immer noch im selben Module implementiert werden und mit Hilfe der Benennung als Methode für den jeweiligen Roboter identifiziert werden. Da sowohl beim e-Puck als auch beim LDVbot nicht festgestellt werden kann wenn die Verbindung fehlgeschlagen oder zusammengebrochen ist, sollte eine Challenge-Response Abfrage implementiert werden, die immer wieder abgesendet wird um den Verbindungsstatus abzufragen. Besonders für den LDVbot wäre dies notwendig, da dieser anders als der e-Puck den Code nicht auf dem Roboter selbst ausführt sondern dieser auf dem jeweiligen Computer läuft. Das beim e-Puck einige Änderungen in der Benutzung stattgefunden haben, sollte hier die Firmware angepasst werden, da überschüssige Methoden entfernt werden können. Auch können die Kameramethoden geändert werden um eine Benutzung außerhalb von Bluetooth zu gewährleisten.

API

[Modul: robot.py 8](#_Toc443488030)

[Klasse: robot 8](#_Toc443488031)

[Modul: epuck.py 9](#_Toc443488032)

[Klasse: robot 9](#_Toc443488033)

[Modul: eAccelerometer.py 15](#_Toc443488034)

[Klasse: acclerometer 15](#_Toc443488035)

[Modul: eCamera.py 16](#_Toc443488036)

[Modul: eCommunication.py 16](#_Toc443488037)

[Enumeration: DIC-MSG 16](#_Toc443488038)

[Klasse: communication 16](#_Toc443488039)

[Modul: eFirmware.py 19](#_Toc443488040)

[Klasse: firmwareVersion 19](#_Toc443488041)

[Modul: eFloorsensor.py 20](#_Toc443488042)

[Klasse: floorsensor 20](#_Toc443488043)

[Modul: eLed.py 21](#_Toc443488044)

[Enumeration: led\_status 21](#_Toc443488045)

[Klasse: led 21](#_Toc443488046)

[Modul: eLightsensor.py 22](#_Toc443488047)

[Klasse: lightsensor 22](#_Toc443488048)

[Modul: eMicrophone.py 23](#_Toc443488049)

[Klasse: microphone 23](#_Toc443488050)

[Modul: eSelector.py 24](#_Toc443488051)

[Klasse: selector 24](#_Toc443488052)

[Modul: eMovement.py 25](#_Toc443488053)

[Klasse: movement 25](#_Toc443488054)

[Modul: eProximitysensor.py 27](#_Toc443488055)

[Klasse: proximitysensor 27](#_Toc443488056)

[Modul: ldvbot.py 28](#_Toc443488057)

[Klasse: robot 28](#_Toc443488058)

[Modul: lAnalog.py 33](#_Toc443488059)

[Klasse: analog 33](#_Toc443488060)

[Modul: lBattery.py 36](#_Toc443488061)

[Klasse: battery 36](#_Toc443488062)

[Modul: lBotUD.py 37](#_Toc443488063)

[Klasse: robotID 37](#_Toc443488064)

[Modul: lCommunication.py 38](#_Toc443488065)

[Klasse: communication 38](#_Toc443488066)

[Modul: lFloorsensor.py 40](#_Toc443488067)

[Klasse: floorsensor 40](#_Toc443488068)

[Modul: lLed.py 42](#_Toc443488069)

[Enumeration: led\_status 42](#_Toc443488070)

[Klasse: led 42](#_Toc443488071)

[Modul: lMisc.py 43](#_Toc443488072)

[Modul: lMovement.py 44](#_Toc443488073)

[Klasse: movement 44](#_Toc443488074)

[Modul: lProximitysensor.py 46](#_Toc443488075)

[Klasse: proximitysensor 46](#_Toc443488076)

[Modul: lInfrared.py 48](#_Toc443488077)

[Klasse: infrared 48](#_Toc443488078)

## Modul: robot.py

### Klasse: robot

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, rNumber=None)**  Konstruktor setzt Konfiguration für das Logging. Die Loggingdatei ist "robot.log". Der Modus ist auf "w" gesetzt und überschreibt die Loggingdatei jedes mal beim erneuten Benutzen der API. Das Level ist auf "DEBUG" gesetzt. Benutzt wird die Python Bibliothek Logging (https://docs.python.org/2/library/logging.html) |

#### Methoden:

|  |
| --- |
| getEpuck |
| **getEpuck(self)**  Getter Methode um eine neues Objekt des e-Puck zurückzugeben.  Für die Benutzung des e-Puck:  *Import:*  from robot.robot import robot  *Benutzung:*  ePuck = robot().getEpuck()  ***Return:***  Objekt der Klasse robot des Moduls epuck.py |

|  |
| --- |
| getLdvBot |
| **getLdvBot(self, rNumber)**  Getter Methode um eine neues Objekt des LDVbot zurückzugeben.  Für die Benutzung des LDVbot:  *Import:*  from robot.robot import robot  *Benutzung:*  ldvbot = robot().getLdvBot(rNumber)  ***Parameter:***  rNumber - Roboternummer des benutzten LDVbots  ***Return:***  Objekt der Klasse robot des Moduls ldvbot.py |

## Modul: epuck.py

### Klasse: robot

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_port | Serieller Port (default: None) |
| \_connectionStatus | Status der Verbindung (default: False) |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |
| \_movement | Objekt des Motors  Modul: eMovement  Klasse: movement |
| \_proximity | Objekt des Annäherungssensors  Modul: eProximity  Klasse: proximity |
| \_floor | Objekt des Bodensensors  Modul: eFloorsensor  Klasse: floorsensor |
| \_led1 | Objekt LED 1  Modul: eLED  Klasse: led |
| \_led2 | Objekt LED 2  Modul: eLED  Klasse: led |
| \_led3 | Objekt LED 3  Modul: eLED  Klasse: led |
| \_led4 | Objekt LED 4  Modul: eLED  Klasse: led |
| \_led5 | Objekt LED 5  Modul: eLED  Klasse: led |
| \_led6 | Objekt LED 6  Modul: eLED  Klasse: led |
| \_led7 | Objekt LED 7  Modul: eLED  Klasse: led |
| \_led8 | Objekt LED 8  Modul: eLED  Klasse: led |
| \_led9 | Objekt LED 9  Modul: eLED  Klasse: led |
| \_accelerometer | Objekt des Beschleunigungssensors  Modul: eAccelerometer  Klasse: accelerometer |
| \_camera | Objekt der Kamera  Modul: eCamera  Klasse: camera |
| \_firmware | Objekt der Firmware  Modul: eFirmware  Klasse: firmwareVersion |
| \_lightsensor | Objekt des Lichtsensors  Modul: eLightsensor  Klasse: lightsensor |
| \_microphone | Objekt des Mikrophones  Modul: eMicrophone  Klasse: microphone |
| \_selector | Objekt des Selektors  Modul: eSelector  Klasse: selector |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self)**  Konstruktor initialisiert alle privaten Variablen |

#### Methoden:

|  |
| --- |
| getMotor |
| **getMotor(self)**  Getter Methode: Motor  ***Return:***  Private Variable: \_movement |

|  |
| --- |
| getFloorSensor |
| **getFloorSensor(self)**  Getter Methode: Bodensensor  ***Return:***  Private Variable: \_floor |

|  |
| --- |
| getProximitySensor |
| **getProximitySensor (self)**  Getter Methode: Annäherungssensor  ***Return:***  Private Variable: \_proximity |

|  |
| --- |
| getLED |
| **getLED(self, ledID)**  Getter Methode: LED  ***Parameter:***  ledID - ID des LEDs [1,9]  ***Return:***  Private Variable: \_led + ledID |

|  |
| --- |
| getAccelerometer |
| **getAccelerometer(self)**  Getter Methode: Beschleuningungssensor  ***Return:***  Private Variable: \_accelerometer |

|  |
| --- |
| getCamera |
| **getCamera(self)**  Getter Methode: Kamera  ***Return:***  Private Variable: \_camera |

|  |
| --- |
| getFirmware |
| **getFirmware(self)**  Getter Methode: Firmware  ***Return:***  Private Variable: \_firmware |

|  |
| --- |
| getLightsensor |
| **getLightsensor(self)**  Getter Methode: Lichtsensor  ***Return:***  Private Variable: \_lightsensor |

|  |
| --- |
| getMicrophone |
| **getMircophone(self)**  Getter Methode: Microphone  ***Return:***  Private Variable: \_microphone |

|  |
| --- |
| getSelector |
| **getSelector(self)**  Getter Methode: Selektor  ***Return:***  Private Variable: \_selector |

|  |
| --- |
| getConnectionStatus |
| **getConnectionStatus(self)**  Getter Methode: Verbindungsstatus  ***Return:***  Private Variable: \_connectionStatus |

|  |
| --- |
| getPort |
| **getPort(self)**  Getter Methode: Port  ***Return:***  Private Variable: \_port |

|  |
| --- |
| disconnect |
| **disconnect(self)**  Wenn eine Verbindung besteht (\_connectionStatus=True) führt stop() aus, schließt den Port \_port und setzt \_connectionStatus=False  ***Exception:***  raise Exception ("Connection close problem")  ***Return:***  False - wenn keine Verbindung bestand  True - wenn der Port geschlossen wurde |

|  |
| --- |
| isConnected |
| **isConnected(self)**  Gibt den Verbindungsstatus zurück (sollte ausgebaut werden wenn die Challenge-Repsonse Abfrage implementiert wurde)  ***Return:***  Private Variable: \_connectionStatus |

|  |
| --- |
| reset |
| **reset(self)**  Resettet den e-Puck. Sendet Befehl "R"  ***Exception:***  raise Exception ("No connection available")  ***Return:***  True - wenn der reset geklappt hat |

|  |
| --- |
| stop |
| **stop(self)**  Setzt Motorgeschwindigkeit auf 0 und schaltet alle LEDs aus. Sendet Befehl "S"  ***Exception:***  raise Exception ("No connection available")  ***Return:***  True - wenn stop geklappt hat  False - wenn ein Fehler aufgetreten ist |

|  |
| --- |
| connect |
| **connect(self)**  initialisiert \_port mit baudrate=230400, port='/dev/ttyS0', timeout=0.05 und öffnet diesen. Setzt \_connectionStatus=True und führt einen reset() des e-Puck aus  ***Exception:***  raise Exception ("Connection problem") |

## Modul: eAccelerometer.py

### Klasse: acclerometer

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |
| \_acclerometerFiltered | Boolean (default=False) |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot. Initialisiert \_communication und \_acclerometerFiltered |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht ('A', 12, '@III') wenn \_accelerometerFiltered=True sonst ('a', 6, '@HHH') um die Werte des Beschleunigungssensors zu bekommen  ***Exception:***  raise Exception ("No connection available")  ***Return:***  False - wenn die Antwort kein Tuple ist oder der erste Wert kein Integer  Tuple (0, 0, 0, 0, 0, 0) - \_accelerometerFiltered=True  Tuple (0, 0, 0) - acclerometerFiltered=False |

|  |
| --- |
| setAccelerometerFiltered |
| **setAccelerometerFiltered(self, accelerometerFilter=False)**  setzt \_acclerometerFiltered auf den Wert des Parameters  ***Parameter:***  acclerometerFilter - Boolean (default: False) |

## Modul: eCamera.py

Wird nicht in der API beschrieben, da eine Benutzung zum jetzigen Zeitpunkt noch nicht möglich ist.

## Modul: eCommunication.py

### Enumeration: DIC-MSG

erwartete Länge der Antworten

"v": 2

"\n": 23

"\x0c": 2

"k": 2

"R": 2

### Klasse: communication

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot |

#### Methoden:

|  |
| --- |
| send\_binary\_mode\_epuck |
| **send\_binary\_mode\_epuck(self, parameters)**  Sendet die Nachricht übergeben in **parameters** im binär Modus zum e-Puck und wartet auf Antwort  ***Parameter:***  parameters - zu sendende Nachricht  ***Return:***  False - wenn die Länge der Antwort falsch ist oder keine Antwort für 150 Schritte empfangen wurde  Response - Antwort des e-Puck. Typ ist nicht definiert |

|  |
| --- |
| write\_actuators\_epuck |
| **write\_actuatos\_epuck(self, order)**  Setzt Variablen, Motoren und LEDs. Es wird keine Antwort für die LEDs, die Motorgeschwindigkeit und Motorposition erwartet  ***Parameter:***  order - zu sendende Nachricht |

|  |
| --- |
| send |
| **send(self, msg)**  Sendet Nachricht **msg** über den seriellen Port \_port des Robotors \_robot zum e-Puck  ***Parameter:***  msg - zu sendende Nachricht  ***Return:***  False - sollte ein Fehler auftreten  Response - nach pySerial 2.5 die Länge der Nachricht davor 0 |

|  |
| --- |
| receive |
| **receive(self, n=128)**  Empfängt n Bytes des e-Puck über den Port \_port des Roboters \_robot  ***Paramter:***  n - Bytes zu empfangen (default: 128)  ***Return:***  line - empfangene Nachricht |

|  |
| --- |
| send\_receive |
| **send\_receive(self, msg)**  Sendet Nachricht **msg** über den seriellen Port \_port des Robotors \_robot zum e-Puck. Und versucht 5 mal eine Antwort zu empfangen.  ***Parameter:***  msg - zu sendende Nachricht  ***Return:***  reply - empfangene Nachricht |

## Modul: eFirmware.py

### Klasse: firmwareVersion

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getVersion |
| **getVersion(self)**  Gibt die Version der Firmware des e-Puck zurück  ***Exception:***  raise Exception ("No connection available")  ***Return:***  Response - String der Version der Firmware |

## Modul: eFloorsensor.py

### Klasse: floorsensor

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |
| \_i2cbus | i2c Bus |
| \_i2cdev | i2c Dev |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot. Initialisiert \_communication und die Bus Variablen \_i2cbus und \_i2cdev |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachrichten über den i2cbus um die Werte der Bodensensoren zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  Tuple (0, 0, 0) - Werte der Bodensensoren |

## Modul: eLed.py

### Enumeration: led\_status

ON: 1

OFF: 0

### Klasse: led

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |
| \_status | Status der LED (default: OFF) |
| \_ledID | ID der LED |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot, ledID)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und \_ledID mit ledID. Initialisiert \_communication und \_status |

#### Methoden:

|  |
| --- |
| setLED |
| **setLED(self, status)**  setzt das LED festgelegt durch die \_ledID auf den übergebenen **status**  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  status - led\_status |

## Modul: eLightsensor.py

### Klasse: lightsensor

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht ('O', 20, '@HHHHHHHHHH') um die Werte der Lichtsensoren zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  False - Wenn die Länge der Nachricht nicht übereinstimmt mit den erwarteten Länge  Tuple (0, 0, 0, 0, 0, 0, 0, 0, 0, 0) - Werte der Lichtsensoren |

## Modul: eMicrophone.py

### Klasse: microphone

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht ('u', 6, '@HHH') um die Werte des Mikrophones zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  False - Wenn die Länge der Nachricht nicht übereinstimmt mit den erwarteten Länge  Tuple (0, 0, 0) - Werte des Mikrophones |

## Modul: eSelector.py

### Klasse: selector

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht ('c') um die Werte des Selektors zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  Response - Werte des Selektors |

## Modul: eMovement.py

### Klasse: movement

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| setPosition |
| **setPosition(self, l\_wheel, r\_wheel)**  setzt Position der Reifen. Sendet Nachricht ('P', l\_wheel, r\_wheel)  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  l\_wheel - Position des linken Reifens  r\_wheel - Position des rechten Reifens |

|  |
| --- |
| getPosition |
| **getPosition(self)**  empfängt Position der Reifen. Sendet Nachricht ('Q', 4, '@HH'')  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  False - wenn die Länge der Nachricht nicht der gewünschten Länge entspricht  Tuple (0, 0) - Positionen der Reifen |

|  |
| --- |
| setSpeed |
| **setPosition(self, l\_motor, r\_motor)**  setzt Geschwindigkeit der Motoren. Sendet Nachricht ('D', l\_motor, r\_motor)  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  l\_motor - Geschwindigkeit des linken Reifens  r\_motor - Geschwindigkeit des rechten Reifens |

|  |
| --- |
| getSpeed |
| **getSpeed(self)**  empfängt Geschwindigkeit der Motoren. Sendet Nachricht ('E, 4, '@HH'')  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  False - wenn die Länge der Nachricht nicht der gewünschten Länge entspricht  Tuple (0, 0) - Geschwindigkeit der Motoren |

|  |
| --- |
| driveForward |
| **driveForward(self, speed)**  setzt Geschwindigkeit der Motoren auf den selben Wert **speed** um geradeaus zu fahren. Benutzt setSpeed(**speed**, **speed**)  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  speed - Geschwindigkeit der Motoren |

|  |
| --- |
| stop |
| **stop(self)**  setzt Geschwindigkeit der Motoren auf den selben Wert **0** um anzuhalten. Benutzt setSpeed(**0**, **0**)  ***Exception:***  Raise Exception ("No connection available") |

## Modul: eProximitysensor.py

### Klasse: proximitysensor

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht ('N', 20, '@HHHHHHHHHH') um die Werte der Annäherungssensoren zu erhalten  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  False - Wenn die Länge der Nachricht nicht übereinstimmt mit den erwarteten Länge  Tuple (0, 0, 0, 0, 0, 0, 0, 0, 0, 0) - Werte der Annäherungssensoren |

|  |
| --- |
| calibrate |
| **calibrate(self)**  sendet Nachricht ('k') um die Annäherungssensoren zu kalibrieren. Im Umkreis von 10 cm sollte kein anderen Objekt stehen bei Benutzung dieser Methode  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  False - beim Auftreten eines Fehlers  True - nach erfolgreichen Kalibrieren |

## Modul: ldvbot.py

### Klasse: robot

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_rNumber | Roboternummer (default: None) |
| \_port | Serieller Port (default: None) |
| \_connectionStatus | Status der Verbindung (default: False) |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |
| \_movement | Objekt des Motors  Modul: lMovement  Klasse: movement |
| \_proximity | Objekt des Annäherungssensors  Modul: lProximity  Klasse: proximity |
| \_floor | Objekt des Bodensensors  Modul: lFloorsensor  Klasse: floorsensor |
| \_led1 | Objekt LED 1  Modul: lLED  Klasse: led |
| \_led2 | Objekt LED 2  Modul: lLED  Klasse: led |
| \_led3 | Objekt LED 3  Modul: lLED  Klasse: led |
| \_battery | Objekt des Beschleunigungssensors  Modul: lBattery  Klasse: battery |
| \_botID | Objekt der Kamera  Modul: lBotID  Klasse: robotID |
| \_infrared | Objekt der Firmware  Modul: lInfrared  Klasse: infrared |
| \_analog | Objekt des Lichtsensors  Modul: lAnalog  Klasse: analog |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, rNumber=None)**  Konstruktor initialisiert alle privaten Variablen |

#### Methoden:

|  |
| --- |
| getMotor |
| **getMotor(self)**  Getter Methode: Motor  ***Return:***  Private Variable: \_movement |

|  |
| --- |
| getFloorSensor |
| **getFloorSensor(self)**  Getter Methode: Bodensensor  ***Return:***  Private Variable: \_floor |

|  |
| --- |
| getProximitySensor |
| **getProximitySensor (self)**  Getter Methode: Annäherungssensor  ***Return:***  Private Variable: \_proximity |

|  |
| --- |
| getLED |
| **getLED(self, ledID)**  Getter Methode: LED  ***Parameter:***  ledID - ID des LEDs [1,3]  ***Return:***  Private Variable: \_led + ledID |

|  |
| --- |
| getBattery |
| **getBattery(self)**  Getter Methode: Batterie  ***Return:***  Private Variable: \_battery |

|  |
| --- |
| getRobotID |
| **getRobotID(self)**  Getter Methode: RobotID  ***Return:***  Private Variable: \_botID |

|  |
| --- |
| getInfrared |
| **getInfrared(self)**  Getter Methode: Infrared Modul  ***Return:***  Private Variable: \_infrared |

|  |
| --- |
| getAnalog |
| **getAnalog(self)**  Getter Methode: Analog  ***Return:***  Private Variable: \_analog |

|  |
| --- |
| getRNumber |
| **getRNumber(self)**  Getter Methode: Roboternummer  ***Return:***  Private Variable: \_rNumber |

|  |
| --- |
| getConnectionStatus |
| **getConnectionStatus(self)**  Getter Methode: Verbindungsstatus  ***Return:***  Private Variable: \_connectionStatus |

|  |
| --- |
| getPort |
| **getPort(self)**  Getter Methode: Port  ***Return:***  Private Variable: \_port |

|  |
| --- |
| disconnect |
| **disconnect(self)**  Wenn eine Verbindung besteht (\_connectionStatus=True) führt stop() aus, schließt den Port \_port und setzt \_connectionStatus=False  ***Exception:***  raise Exception ("Connection close problem")  ***Return:***  False - wenn keine Verbindung bestand  True - wenn der Port geschlossen wurde |

|  |
| --- |
| isConnected |
| **isConnected(self)**  Gibt den Verbindungsstatus zurück (sollte ausgebaut werden wenn die Challenge-Repsonse Abfrage implementiert wurde)  ***Return:***  Private Variable: \_connectionStatus |

|  |
| --- |
| stop |
| **stop(self)**  Setzt Motorgeschwindigkeit auf 0 und schaltet alle LEDs aus. Sendet Befehl "S"  ***Exception:***  raise Exception ("No connection available") |

|  |
| --- |
| connect |
| **connect(self, mPort='/dev/ttyACM0')**  initialisiert \_port mit baudrate=57600, port= **mPort**, timeout=None, stopbits=STOPBITS\_ONE, parity=PARITY\_NONE, öffnet diesen und setzt \_connectionStatus=True  ***Exception:***  raise Exception ("Connection problem") |

## 

## Modul: lAnalog.py

### Klasse: analog

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| setdestroy |
| **setdestroy(self, d1=0, d2=0, d3=0, d4=0, d5=0, d6=0, d7=0, d8=0, g1=0, g2=0, g3=0)**  Setzt analogen destroy für Sensoren  ***Exception:***  raise Exception ("No connection available")  ***Parameter:***  d1 = 0  d2 = 0  d3 = 0  d4 = 0  d5 = 0  d6 = 0  d7 = 0  d8 = 0  g1 = 0  g2 = 0  g3 = 0  ***Return:***  'fail' - sollte ein Fehler auftreten |

|  |
| --- |
| getdestroy |
| **getdestroy(self,)**  Sendet Nachricht [rNumber, 3, 208] um zu Empfangen welche Sensoren mit "High" simuliert werden  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  Tuple |

|  |
| --- |
| analogread |
| **analogread(self, sensor)**  Sendet analoge Nachricht [rNumber, 4, 5, **sensor**] um Werte des Sensors zu erhalten  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  reponse - Werte des angefragten Sensors |

|  |
| --- |
| analogreadbattery |
| **analogreadbattery(self)**  Sendet analoge Nachricht [rNumber, 3, 205] um die analog Werte der Batterie zu empfangen  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  Integer - Volt/1000 Batteriestatus des Roboters |

|  |
| --- |
| analogreaddistanceleft |
| **analogreaddistanceleft(self)**  Sendet analoge Nachricht [rNumber, 4, 6, 1] um die analog Werte des linken Abstandssensors zu erhalten  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  Tuple (0, 0, 0, 0) - Werte des linken Abstandssensors |

|  |
| --- |
| analogreaddistanceright |
| **analogreaddistanceright(self)**  Sendet analoge Nachricht [rNumber, 4, 6, 2] um die analog Werte des rechten Abstandssensors zu erhalten  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  Tuple (0, 0, 0, 0) - Werte des rechten Abstandssensors |

## 

## Modul: lBattery.py

### Klasse: battery

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  Sendet Nachricht [rNumber, 3, 204] um die digital Werte der Batterie zu empfangen  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  Integer - Batteriestatus des Roboters [0,1] 0 wenn sich die Batterie dem Ende zuneigt |

## Modul: lBotUD.py

### Klasse: robotID

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| setRobotID |
| **setRobotID(self, r\_type)**  Sendet Nachricht [rNumber, 5, 203, r\_type] um die ID des Roboters zu setzen  ***Exception:***  raise Exception ("No connection available")  ***Parameter:***  r\_type - ID des Roboters (Integer)  ***Return:***  'fail' - sollte ein Fehler auftreten |

|  |
| --- |
| getRobotID |
| **setRobotID(self)**  Sendet Nachricht [rNumber, 3, 4] um die ID des Roboters zu bekommen. Um einen Wert zu erhalten muss die ID vorher gesetzt werden  ***Exception:***  raise Exception ("No connection available")  ***Return:***  'fail' - sollte ein Fehler auftreten  Integer - ID des Roboters (falls vorher gesetzt) |

## Modul: lCommunication.py

### Klasse: communication

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot |

#### Methoden:

|  |
| --- |
| send |
| **send(self, msg)**  Sendet Nachricht **msg** über den seriellen Port \_port des Robotors \_robot zum LDVbot  ***Parameter:***  msg - zu sendende Nachricht  ***Return:***  'fail' - sollte als Antwort nicht "transmitted " zurückkommen  0 - sonst |

|  |
| --- |
| receive |
| **receive(self)**  Empfängt Antwortzeile des LDVbot über den Port \_port des Roboters \_robot  ***Return:***  line - empfangene Nachricht |

|  |
| --- |
| send\_receive |
| **send\_receive(self, msg)**  Sendet Nachricht **msg** über den seriellen Port \_port des Robotors \_robot zum LDVbot. Und versuch 8 mal eine Antwort zu empfangen.  ***Parameter:***  msg - zu sendende Nachricht  ***Return:***  lines - empfangene Nachricht |

## 

## Modul: lFloorsensor.py

### Klasse: floorsensor

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot. Initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht [rNumber, 4, 3, 2] um die Werte der Bodensensoren zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Tuple (0, 0, 0) - Werte der Bodensensoren |

|  |
| --- |
| setSensitivity |
| **setSensitivity(self, sens)**  sendet Nachricht [rNumber, 5, 200, sens\_int1, sens\_int2] um die Sensitivität der Bodensensoren zu setzen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  sens - Sensitivtät der Bodensensoren [0, 1023]  ***Return:***  'fail' - Beim Auftreten eines Fehlers |

|  |
| --- |
| getSensitity |
| **getSensitivity(self)**  sendet Nachricht [rNumber, 3, 206] um die Sensitivität der Bodensensoren zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Integer [0, 1023] - Sensitivität der Bodensensoren |

## Modul: lLed.py

### Enumeration: led\_status

ON: 1

OFF: 0

### Klasse: led

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des e-Puck  Modul: epuck.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: eCommunication  Klasse: communication |
| \_status | Status der LED (default: OFF) |
| \_ledID | ID der LED |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot, ledID)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und \_ledID mit ledID. Initialisiert \_communication und \_status |

#### Methoden:

|  |
| --- |
| setLED |
| **setLED(self, status)**  setzt das LED festgelegt durch die \_ledID auf den übergebenen **status**. Sendet Nachricht [rNumber, 5, 2, \_ledID, **status**]  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  status - led\_status |

## Modul: lMisc.py

#### Methoden:

|  |
| --- |
| fail |
| **fail(data)**  gibt je auf den Fehler einen unterschiedlichen Rückgabewert zurück,  ***Return:***  output - 198, 199 Rückgabe der Werte  output - 55 Rückgabe des Wertes ("WARNING: No Response from remote device")  output - 20 gibt 'fail' zurück ("WARNING: NRF-transmission failed")  output - 15 ("ERROR: Received unexpected value") |

|  |
| --- |
| to\_bin |
| **to\_bin(decimal, bits)**  Wandelt dezimal Werte in binär Werte um  ***Parameter:***  decimal - dezimal Wert  bits - Anzahl bits  ***Return:***  tmp\_value - Binär Wert |

## Modul: lMovement.py

### Klasse: movement

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| drive |
| **drive(self, speed, dist, angle)**  lässt den LDVbot in einer Geschwindigkeit **speed** eine bestimmte Reichweite **dist** in einem bestimmten Winkel **angle** fahren  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  speed - Geschwindigkeit des Roboters [-100, 100]  dist - Distanz zu fahren [0, 65500]  angle - Winkel der Gefahren werden soll [-360, 360] |

|  |
| --- |
| turnLeft |
| **turnLeft(self)**  dreht den Roboter um 90° auf der Stelle nach links |

|  |
| --- |
| turnRight |
| **turnRight(self)**  dreht den Roboter um 90° auf der Stelle nach rechts |

|  |
| --- |
| turnAround |
| **turnAround(self)**  dreht den Roboter um 180° auf der Stelle |

|  |
| --- |
| driveForward |
| **driveForward(self, speed)**  setzt Geschwindigkeit der Motoren auf den selben Wert **speed** um geradeaus zu fahren. Benutzt drive(-**speed**, 65000, 0)  ***Parameter:***  speed - Geschwindigkeit des Motors [-100, 100] |

|  |
| --- |
| stop |
| **stop(self)**  setzt Geschwindigkeit der Motoren auf den selben Wert **0** um anzuhalten. Benutzt drive(**0**, **0**, **0**) |

|  |
| --- |
| isRunning |
| **isRunning(self)**  Sendet Nachricht [rNumber, 3, 209] um zu überprüfen ob der Roboter noch fährt  ***Return:***  Integer - [0,1] 0 wenn der Roboter nicht mehr fährt, ansonsten 1 |

## Modul: lProximitysensor.py

### Klasse: proximitysensor

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getValues |
| **getValues(self)**  sendet Nachricht [rNumber, 4, 3, 1] um die Werte der Annäherungssensoren zu erhalten  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Tuple (0, 0, 0, 0, 0, 0, 0, 0) - Werte der Annäherungssensoren alle {0,1} |

|  |
| --- |
| setSensitivity |
| **setSensitivity(self, sens)**  sendet Nachricht [rNumber, 5, 201, sens\_int1, sens\_int2] um die Sensitivität der Bodensensoren zu setzen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  sens - Sensitivtät der Bodensensoren [0, 1023]  ***Return:***  'fail' - Beim Auftreten eines Fehlers |

|  |
| --- |
| getSensitity |
| **getSensitivity(self)**  sendet Nachricht [rNumber, 3, 207] um die Sensitivität der Bodensensoren zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Integer [0, 1023] - Sensitivität der Bodensensoren |

## Modul: lInfrared.py

### Klasse: infrared

#### Variablen:

|  |  |
| --- | --- |
| Private Variablen |  |
| \_robot | Objekt des LDVbot  Modul: ldvbot.py  Klasse: robot |
| \_communication | Objekt der Kommunikationsklasse  Modul: lCommunication  Klasse: communication |

#### Konstruktor:

|  |
| --- |
| \_\_init\_\_ |
| **\_\_init\_\_(self, bot)**  Konstruktor setzt \_robot mit dem übergebenen Objekt bot und initialisiert \_communication |

#### Methoden:

|  |
| --- |
| getID |
| **getID(self, robotNr)**  sendet Nachricht [rNumber, 3, 9] um die aktuelle Infrarot ID des Roboters zu bekommen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  robotNr - Roboternummer  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Integer [0, 7] - Infrarot ID des Roboters |

|  |
| --- |
| setID |
| **setID(self, robotNr, infra\_id)**  sendet Nachricht [rNumber, 4, 10, **infra**] um die aktuelle Infrarot ID des Roboters zu setzen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  robotNr - Roboternummer  infra\_id - Infrarot ID [0,7]  ***Return:***  'fail' - Beim Auftreten eines Fehlers |

|  |
| --- |
| getSensitivity |
| **getSensitivity(self, robotNr)**  sendet Nachricht [rNumber, 3, 12] um die aktuelle Infrarot Sensitivität des Roboters zu bekommen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  robotNr - Roboternummer  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Integer [0, 1023] - Infrarot ID des Roboters |

|  |
| --- |
| setSensitivity |
| **setSensitivity (self, robotNr, sens)**  sendet Nachricht [rNumber, 5, 13, sens\_int1, sens\_int2] um die aktuelle Infrarot Sensitivität des Roboters zu setzen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  robotNr - Roboternummer  sens - Sensitivität [0, 1023]  ***Return:***  'fail' - Beim Auftreten eines Fehlers |

|  |
| --- |
| setReceiveMode |
| **setID(self, robotNr)**  sendet Nachricht [rNumber, 3, 8] um den Roboter auf ReceiveMode zu setzen  ***Exception:***  Raise Exception ("No connection available") |

|  |
| --- |
| send |
| **send(self, sender\_rbn, receiver\_id, data\_byte)**  sendet Nachricht [**sender\_rbn**, 5, 7, **receiver\_id**, **data\_byte**] um Bytes vom Roboter **sender\_rbn** an den Infrarotsensor mit der ID **receiver\_id** zu senden  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  sender\_rbn - Roboternummer  receiver\_id - Infrarot ID [0,7] des Infrarotempfängers  data\_bytes - Daten [0, 255]  ***Return:***  'fail' - Beim Auftreten eines Fehlers |

|  |
| --- |
| getLastValues |
| **getLastValues(self, robotNr)**  sendet Nachricht [rNumber, 3, 11] um die letzten Infrarotwerte zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  robotNr - Roboternummer  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Tuple - letzte Infrarot Werte |

|  |
| --- |
| getSensorWord |
| **getSensorWord(self, robotNr, sensor)**  sendet Nachricht [**robotNr**, 4, 14, **sensor**] um eine Nachricht des jeweiligen Sensors zu empfangen  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  robotNr - Roboternummer  sensor - Infrarotsensor [1, 8]  ***Return:***  'fail' - Beim Auftreten eines Fehlers  Tuple - Rückmeldung des Sensors |

|  |
| --- |
| init |
| **init(self, sender\_rbn, receiver\_rbn, sender\_id, receiver\_id, sensitivity)**  setID(**sender\_rbn**, **sender\_id**)  setID(**receiver\_rbn**, **receiver\_id**)  setSensitivty(**sender\_rbn**, **sensitivity**)  setSensitivty(**receiver\_rbn**, **sensitivity**)  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  sender\_rbn - Roboternummer  receiver\_rbn - Roboternummer  sender\_id - Infrarot ID [0,7] des Infrarotsenders  receiver\_id - Infrarot ID [0,7] des Infrarotempfängers  sensitivty - [0,1023] |

|  |
| --- |
| sendWord |
| **sendWord(self, sender\_rbn, receiver\_rbn, receiver\_id, byte)**  setReceiveMode(**receiver\_rbn**)  send(**self, sender\_rbn, receiver\_id, byte**)  lastData = getLastValues(**receiver\_rbn**)  ***Exception:***  Raise Exception ("No connection available")  ***Parameter:***  sender\_rbn - Roboternummer  receiver\_rbn - Roboternummer  receiver\_id - Infrarot ID [0,7] des Infrarotempfängers  byte - [0,255] |