PlayBotix

Handleiding Programmeren

Deze handleiding helpt je op weg met het programmeren van robots in Python en JavaScript. We wensen je veel plezier bij het coderen en we zijn heel benieuwd naar wat voor gave programma's jij gaat maken!

Inhoudsopgave:

Voorbereiding

Voordat we aan de slag kunnen gaan met het programmeren, moeten we eerst je computer gereed maken om te kunnen verbinden met jullie robots. Welke programmeertaal je ook kiest, er moeten nog wat dingetjes worden ingesteld voordat we echt aan de slag kunnen gaan. Mocht je even niet weten welke programmeertaal voor jouw geschikt is, kijk dan in onze FAQ voor wat handige tips.

Tijdens de voorbereiding en vooral daarna zullen we wat termen gebruiken die wat meer uitleg nodig hebben:

- **Realm (omgeving):** als we het hebben over een realm, dan hebben we het over een soort afgesloten omgeving waar de robot zich in bevindt. Denk bijvoorbeeld aan een straat met huizen. Elk huis heeft zijn eigen adres. Als je bijvoorbeeld een brief wilt sturen aan nummer 17, dan schrijf je op de envelop de straat waar hij of zij woont en het huisnummer. Dit huisnummer komt in zekere zin overeen met de realm. Als je met de robot wilt communiceren, gebruik je zijn huisnummer (realm) om er berichtjes naar toe te sturen.
- **Session (sessie):** Via deze sessie kan de robot met jouw communiceren en kan jij aan de robot opdrachten geven.

In dit document laten wij voorbeeldcodes zien van Python en JavaScript. Beide programmeertalen zien er qua structuur ongeveer hetzelfde uit, maar er zitten wel belangrijke verschillen tussen beide talen. Om duidelijk te maken welke programmeertaal we gebruiken tijdens de voorbeelden, maken we gebruik van een tabel met een kleurtje. Voor Python code maken we gebruik van een licht gele kleur en voor JavaScript van een licht blauwe/paarse achtergrond kleur:

```
print("Dit is Python code")
# Deze regel is commentaar en wordt niet uitgevoerd
```

```
console.log("Dit is JavaScript code")
// Deze regel is commentaar en wordt niet uitgevoerd
```

Python installatie

Wanneer je met Python aan de slag wilt gaan, zijn er een aantal stappen die je moet doen. Deze stappen kunnen per besturingssysteem iets verschillen. Voor dit stappenplan gaan we uit van een Windows besturingssysteem:

1. Zorg ervoor dat de robot, waar je mee aan de slag wilt gaan, aan staat;
2. Open een opdrachtprompt en typ daarin het commando: python. Als **Python 3.6 of hoger** al geïnstalleerd is, dan zie je in de tekst na je commando het versienummer staan. Mocht Python nog niet geïnstalleerd zijn of je maakt gebruik van een oudere Python versie, dan kun je Python downloaden via de officiële website: https://www.python.org/downloads/
3. De volgende stap is dat we pip downloaden en installeren op je computer. Pip zorgt ervoor dat we straks andere Python packages kunnen downloaden die we nodig hebben om te kunnen verbinden met de robot:
    a. Open een nieuwe opdrachtprompt en controleer of pip geïnstalleerd is met de volgende commando:

    ```
    pip3 help
    ```

    Als pip al geïnstalleerd is, krijg je uitleg hoe je pip moet gebruiken en kan je direct naar stap 4 toe gaan. Mocht de opdrachtprompt zeggen dat de commando niet herkend wordt, dan dien je verder te gaan met stap 3.b;
    b. Download de pip installatie script: https://bootstrap.pypa.io/get-pip.py;
    c. Open een nieuwe opdrachtprompt en ga middels de commando cd naar de plek toe waar je de get-pip.py file hebt gedownload. Mocht die in je Downloads folder staan, dan is het commando dat je moet uitvoeren:

    ```
    cd Downloads
    ```

    d. Typ in je opdrachtprompt het volgende om pip te installeren:

    ```
    python get-pip.py
    ```

    e. Mocht de opdrachtprompt zeggen dat ze pip niet kunnen installeren omdat je niet de rechten ertoe hebt. Voer dan stap 3.c en 3.d opnieuw uit, maar start dit keer de opdrachtprompt als administrator;
    f. Controleer of pip succesvol is geïnstalleerd met de volgende commando:

    ```
    pip3 help
    ```

    Als pip succesvol geïnstalleerd is, krijg je uitleg hoe je pip moet gebruiken.
4. Installeer Autobahn en OpenSSL via pip:

    ```
    pip3 install autobahn[twisted,serialization] pyopenssl
    ```

5. Creëer op een handige plek op je computer een nieuwe bestand genaamd demo.py;
6. Open dit bestand in jou favoriete Python editor door er dubbel op te klikken;

7. In de Python editor, voeg de volgende regels toe:

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep

@inlineCallbacks
def main(session, details):
    yield session.call("rie.dialogue.say",
        text="Hallo, ik ben succesvol verbonden!")
    session.leave() # Sluit de verbinding met de robot

# Create wamp connection
wamp = Component(
    transports=[{
      "url": "ws://wamp.robotsindeklas.nl",
      "serializers": ["msgpack"],
      "max_retries": 0
    }],
    realm="VUL HIER JE REALM IN",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

Let op: als je de code hierboven direct kopieert en plakt in je Python editor, dan moet je zelf nog even handmatig tabs toevoegen bij bijvoorbeeld `yield session.call`. Hiermee voorkom je errors zoals '`IndentationError: expected an indented block`'

8. Vul in het stukje tekst waar "VUL HIER JE REALM IN" staat, de realm van de robot in. Deze realm kun je terugvinden als je naar de robot pagina gaat in portal.robotsindeklas.nl of portal.robotsindezorg.nl. Vervolgens zoek je de robot op waarmee je aan de slag wilt gaan en druk je op het hamburger menu. Binnen het menuutje klik je op het klembord en onderin het scherm dat nu verschijnt staat de realm. Een voorbeeld realm: rie.5e1312363dbf49eed032e123;
9. Voer demo.py uit door op Run te klikken;
10. Als alles goed is ingesteld, zal de robot zeggen: 'Hallo, ik ben succesvol verbonden!'. Mocht de robot niks zeggen, controleer dan of de robot online is en of je de realm in zijn volledigheid hebt gekopieerd. Mocht de robot nog steeds niks zeggen en je ziet de error bericht: `TLS failure: certificate verify failed`, dan betekent het dat `pyopenssl` nog niet geïnstalleerd is. Voer nogmaals stap 4 uit.

JavaScript installatie

Om met JavaScript aan de slag te gaan hoeven we niet veel op te zetten. Met een paar simpele stappen kun je zo aan de slag gaan met het programmeren:

1. Zorg ervoor dat de robot, waar je mee aan de slag wilt gaan, aan staat;
2. Creëer op een handige plek op je computer een nieuw bestand genaamd demo.html;
3. Vervolgens open je dit bestand met bijvoorbeeld notepad. Let op: programma's zoals Microsoft Word zullen hierbij niet werken;
4. In het nog lege document voeg je de volgende stukje code toe:

```
<script
src="https://cdn.jsdelivr.net/npm/autobahn-browser@19.7.3/autobahn.min.js"></script>
<span>Welkom bij het programmeren van JavaScript met Robots in de Klas</span>
<script>
  let wamp = new autobahn.Connection({
    url: "wss://wamp.robotsindeklas.nl",
    realm: "VUL HIER JE REALM IN",
    protocols: ["wamp.2.msgpack"]
  })

  wamp.onopen = async (session) => {
    await session.call("rie.dialogue.say", [],
          {text: "Hallo, ik ben succesvol verbonden!"})
    wamp.close() // Sluit de verbinding met de robot
  }

  wamp.open()
</script>
```

5. Vul in het stukje tekst waar "VUL HIER JE REALM IN" staat, de realm van de robot in. Deze realm kun je terugvinden als je naar de robot pagina gaat in portal.robotsindeklas.nl of portal.robotsindezorg.nl. Vervolgens zoek je de robot op waarmee je aan de slag wilt gaan en druk je op het hamburger menu. Binnen het menuutje klik je op het klembord en onderin het scherm dat nu verschijnt staat de realm. Een voorbeeld realm: rie.5e1312363dbf49eed032e123;
6. Ga nu terug naar je verkenner en klik met de rechter muisknop op demo.html. Vervolgens klik je op, 'openen met…' en selecteer je daar een browser;
7. Als alles goed is ingesteld, zal de browser een nieuwe tab openen met daarin de tekst: 'Welkom bij het programmeren van JavaScript met Robots in de Klas' en zal de robot zeggen: 'Hallo, ik ben succesvol verbonden!'. Mocht de robot niks zeggen, controleer dan of de robot online is en of je de realm in zijn volledigheid hebt gekopieerd;
8. Tip: als je op F12 drukt terwijl je in de browser bent, dan open je een extra venster in je scherm waarin je makkelijk code kunt testen en kijken of er fouten in je code staan.

Robot

Met ons platform is het mogelijk om op een slimme en makkelijke manier robots aan te sturen en interacties te ontwerpen. Zo kan je met maar een paar regels code, de robot laten wachten totdat je iemand voor de robot ziet staan, om vervolgens de robot die persoon te laten groeten. In dit hoofdstuk leggen we stap voor stap uit hoe het verbinden en het aanroepen van de robot werkt.

Verbinden

Om met de robot te kunnen communiceren, dien je twee stappen uit te voeren. De eerste stap is dat we de robot moeten pauzeren zodat het standaard programma dat op de robot draait, jouw programma niet in de weg gaat zitten. Je kunt dit programma pauzeren door naar het robots overzicht te gaan en daar op het hamburger menu te klikken. Vervolgens klik je op de pauzeknop en de robot zegt dan ‘agent is gepauzeerd’. Nu dit stukje programma gepauzeerd is, kun je jouw programma afspelen door een verbinding te starten met de robot. Voor Python is het volgende stukje code nodig om te verbinden:

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks

@inlineCallbacks
def main(session, details):
    # PLAATS HIER JE CODE

# Create wamp connection
wamp = Component(
    transports=[{
      "url": "ws://wamp.robotsindeklas.nl",
      "serializers": ["msgpack"],
      "max_retries": 0
    }],
    realm="VUL HIER JE REALM IN",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

Voor JavaScript heb je het volgende stukje code nodig om te verbinden:

```
let wamp = new autobahn.Connection({
    url: "wss://wamp.robotsindeklas.nl",
    realm: "VUL HIER JE REALM IN",
    protocols: ["wamp.2.msgpack"]
})
wamp.onopen = async (session) => {
    // PLAATS HIER JE CODE
}
wamp.open()
```

Zodra je een succesvolle connectie hebt, wordt de functie `main` voor Python aangeroepen en `wamp.onopen` wordt voor JavaScript aangeroepen.

Commando's geven

Nadat je succesvol bent verbonden met de robot, kun je de robot opdrachten geven om bepaalde taken uit te voeren. Zo kan je bijvoorbeeld aan de robot vragen of die met zijn rechterarm wilt gaan zwaaien. Zodra je verbonden bent, heb je toegang tot een stukje code dat `session` heet. `session` is als het ware het stukje code dat we nodig hebben om met de robot te kunnen communiceren. Zo kunnen we met `session` een bepaalde opdracht aanroepen. Dit doen we door op `session` een `call` uit te voeren, zoals hieronder getoond:

```
session.call(...)
```

Als we dit aanroepen, dan weet de robot dat die iets moet gaan doen, maar hij weet nog niet wat die moet doen. Om de robot instructies te geven van wat die moet gaan doen, moeten we ook nog het stukje van de drie puntjes invullen. Als eerst moeten we dus de robot vertellen wat het commando is. Een commando voor een robot noemen we ook wel een RPC[1]. Een RPC is een ingewikkelde term voor een commando, je vertelt hier dus eigenlijk mee: "Hey robot, ik wil dat je een beweging uitvoert".

Nu weet de robot: 'Okay, ik heb de opdracht gekregen om te gaan bewegen, maar ik weet nog niet welke beweging ik moet gaan uitvoeren'. Nu is het aan ons om de robot precies te vertellen welke beweging die moet gaan uitvoeren. Hiervoor maken we gebruik van argumenten die de robot precies vertellen wat wij willen. Voor bijvoorbeeld de opdracht om met de rechterarm te gaan zwaaien, geven we de robot de volgende opdracht:

```
session.call("rom.optional.behavior.play",
             name="BlocklyWaveRightArm")
```

[1] Remote Procedure Call: de technische term voor de commando die we willen dat de robot uitvoert.

En in JavaScript ziet dat er als volgt uit:

```
session.call("rom.optional.behavior.play", [],
             {"name":"BlocklyWaveRightArm"})
```

Nu heeft de robot alle informatie die hij nodig heeft om de opdracht uit te voeren en zal die met zijn rechterarm gaan zwaaien. Op deze manier kun je dus de robot opdrachten geven om bepaalde taken te gaan uitvoeren. In het eerst stukje van de call geef je de opdracht (de RPC) en in het tweede gedeelte geef je verduidelijking van je opdracht. Het tweede gedeelte is niet altijd noodzakelijk, er zijn opdrachten waar het eerste gedeelte al duidelijk genoeg is voor de robot. Kijk zo maar eens bij onze voorbeeldcodes en probeer zelf te ontdekken welke opdrachten direct duidelijk zijn voor de robot en welke opdrachten nog extra informatie nodig hebben.

Luisteren naar sensoren

Naast dat we de robot allerlei opdrachten kunnen geven, kunnen we ook gebruik gaan maken van de sensoren die de robot heeft. Sensoren zijn kleine, slimme componenten op de robot. Zo kan een robot bijvoorbeeld met zijn camera rondkijken en wachten totdat die een gezicht ziet. De camera op de robot is een voorbeeld van een sensor. Een ander voorbeeld van een soort sensor zijn de aanrakingssensoren op de robot. Stel, we willen graag weten of het hoofd van de robot is aangeraakt. Om hierachter te komen moeten we twee stappen uitvoeren. Allereerst willen we een stukje code hebben dat wordt aangeroepen elke keer dat de robot zegt dat er nieuwe sensor data beschikbaar is. Hiervoor maken we weer gebruik van de `session`. Vervolgens melden we ons aan om door te geven dat we graag op de hoogte willen worden gehouden zodra er nieuwe sensor data beschikbaar komt. Dit doen we door op de `session`, `subscribe` aan te roepen:

```
session.subscribe(...)
```

De volgende stap is om aan de robot te vertellen in welke sensor data we geïnteresseerd zijn. Dit doen we door in de subscribe te vertellen welk topic voor ons interessant is. Een topic is een soort van TV kanaal waarop alleen maar één zender kan worden uitgezonden. Wij vertellen dus aan de robot wij willen graag deze zender zien. Vervolgens, willen we natuurlijk ook wat doen met deze data en daarvoor geven we aan de `subscribe` ook een functie mee die steeds wordt aangeroepen zodra er nieuwe data beschikbaar komt.

```
def aangeraakt(frame):
  print(frame)

session.subscribe(aangeraakt, "rom.sensor.touch.stream")
```

We hebben ons ons nu aangemeld voor aanrakings data. Dus zodra het hoofd wordt aangeraakt, dan wordt de functie `aangeraakt` aangeroepen. We moeten nu nog maar één stap uitvoeren voordat we ook daadwerkelijk de informatie binnen krijgen. We moeten namelijk de robot vertellen dat we zijn aangemeld en dat we graag nu de informatie willen verzamelen. Dit doen we door op `session` een `call` te doen die aan de robot vertelt: “Hey robot, ik wil graag dat je al je aanrakings data doorstuurt naar je aanrakings kanaal”. Hieronder zie je een volledig voorbeeld van hoe je dit Python zou moeten doen:

```
def aangeraakt(frame):
  print(frame)

session.subscribe(aangeraakt, "rom.sensor.touch.stream")
session.call("rom.sensor.touch.stream")
```

En in JavaScript ziet dat er als volgt uit:

```
aangeraakt = function(frame){
  console.log(frame)
}
session.subscribe("rom.sensor.touch.stream", aangeraakt)
session.call("rom.sensor.touch.stream")
```

Let hier vooral even op het verschil in volgorde van opdrachten in de `session.subscribe`. In Python vertel je als eerst welke functie moet worden aangeroepen en daarna vertel je op welk kanaal we ons willen aanmelden. In JavaScript is deze volgorde omgedraaid. Dus als eerst geef je het kanaal op en daarna de functie.

In de bijlage staan nog een paar uitgebreide voorbeelden voor Python en JavaScript hoe je aan de slag kunt gaan met het geven van opdrachten en het luisteren naar nieuwe data.

Lichaamsdelen (UBI)

Om de lichaamsdelen van de robot te kunnen gebruiken, maakt ons platform gebruik van een systeem dat heet UBI's (Uniform Body Identifiers). Met deze UBI's kun je bijvoorbeeld aangeven dat je de linker arm wilt laten bewegen. UBI's maken gebruik van een hiërarchische notatie en begint altijd met **body**. De daaropvolgende lichaamsdeel wordt gescheiden met een punt. Dus als je bijvoorbeeld de linkerhand wilt gebruiken, dan gebruik je de UBI: `body.arms.left.hand`

De boomstructuur van lichaamsonderdelen (UBI's):

```
body
|-  head
|    |-  eyes
|    |    |-  right
|    |    |-  left
|    |-  ears
|    |    |-  right
|    |    |-  left
|-  arms
|    |-  right
|    |    |-   upper
|    |    |-   lower
|    |    |-   hand
|    |-  left
|    |    |-   upper
|    |    |-   lower
|    |    |-   hand
|-  torso
|-  legs
|    |-  right
|    |    |-   upper
|    |    |-   lower
|    |    |-   foot
|    |-  left
|    |    |-   upper
|    |    |-   lower
|    |    |-   foot
|-  tail
```

Voorbeeldcode

Ter inspiratie van hoe je aan de slag kunt gaan met het programmeren van de robots, geven we je in dit hoofdstuk een paar voorbeelden. We beginnen met een simpel voorbeeld die een Kerstradio afspeelt op de robot. De code voorbeelden worden vervolgens steeds moeilijker. Als dit de eerste keer is dat je aan de slag gaat met het programmeren in Python of JavaScript, dan raden we je zeker aan om deze code te kopiëren en af te spelen. Vergeet dan vooral niet om de realm van de voorbeelden aan te passen, deze is namelijk voor elke robot anders.

Alle code voorbeelden zijn in Python geschreven, maar je kunt dezelfde code voor JavaScript terugvinden in portal.robotsindeklas.nl. Ga daar naar het robot overzicht en klik op het hamburger menu van een robot die online is. Vervolgens klik je daar op het klembord en je zult daar al deze code voorbeelden zien in Python en in JavaScript.

Ja-knikken

Met de actuator modaliteit kunnen we commando's naar de robot sturen om zijn armen en hoofd te bewegen. De voorbeeldcode hieronder laat je zien hoe je met een paar regels code het hoofd van een robot ja kunt laten knikken. Het enige wat je hoeft te doen is `rom.actuator.motor.write` aan te roepen en te vertellen hoe de beweging eruit moet zien. In dit voorbeeld hoeft de robot alleen met zijn hoofd een knikkende beweging te maken en duurt de hele beweging 2400 milliseconde (= 2.4 seconde).

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks

@inlineCallbacks
def main(session, details):
    # Ja-knikken
    yield session.call("rom.actuator.motor.write",
       frames=[{"time":  400, "data": {"body.head.pitch":  0.1}},
               {"time": 1200, "data": {"body.head.pitch": -0.1}},
               {"time": 2000, "data": {"body.head.pitch":  0.1}},
               {"time": 2400, "data": {"body.head.pitch":  0.0}}],
       force=True)
    session.leave() # Sluit de verbinding met de robot

# Create wamp connection
wamp = Component(
    transports=[{
      "url": "ws://wamp.robotsindeklas.nl",
      "serializers": ["msgpack"],
      "max_retries": 0
    }],
    realm="rie.5e1312363dbf49eed032e123",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

Reageren op aanraking

Naast dat we Kerstmuziek kunnen afspelen, kunnen we ook wachten totdat iemand op het hoofd van de robot heeft gedrukt. Hieronder zie je een voorbeeld van hoe je kan reageren op wanneer iemand het hoofd van de robot aanraakt. Het eerste wat je moet doen is zorgen dat je bent aangemeld op een topic waar straks alle aanraking data naar toe wordt gestuurd. Vervolgens start je met `rom.sensor.touch.stream`, een stream van aanraking data. Elke keer dat iemand één van de aanraking sensoren aanraakt, wordt de functie `aangeraakt` aangeroepen en kan je in die functie kijken welke sensor is aangeraakt. Afhankelijk van je toepassing kan je de robot iets laten zeggen of doen als iemand het hoofd aanraakt. In het voorbeeld hieronder printen we dat het hoofd is aangeraakt zodra iemand de hoofd sensoren heeft aangeraakt.

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks

def aangeraakt(frame):
    if ("body.head.front" in frame["data"] or
            "body.head.middle" in frame["data"] or
            "body.head.rear" in frame["data"]):
        print("Hoofd is aangeraakt!")

@inlineCallbacks
def main(session, details):
    yield session.subscribe(aangeraakt, "rom.sensor.touch.stream")
    yield session.call("rom.sensor.touch.stream")

# Create wamp connection
wamp = Component(
    transports=[{
      "url": "ws://wamp.robotsindeklas.nl",
      "serializers": ["msgpack"],
       "max_retries": 0
    }],
    realm="rie.5e1312363dbf49eed032e123",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

Muziek streamen

Met een onderdeel van de actuator modaliteit kun je muziek afspelen op de robot. Zo kun je een gave interactie maken terwijl de robot geluid van een radio afspeelt. Het starten van de radio stream doe je door `rom.actuator.audio.stream` aan te roepen en in de parameter `url` vertel je de robot waar die de muziek vandaan moet halen. In dit voorbeeld start die bijvoorbeeld een Kerst radiozender :-). Vervolgens laadt de robot de stream en kun je die stoppen door op een knop op je toetsenbord te drukken. Zodra je op het toetsenbord hebt gedrukt, dan zeggen we tegen de robot dat die moet stoppen met het afspelen van de muziek door `rom.actuator.audio.stop` aan te roepen. Let hierbij op dat hier spraken is van een andere stream dan dat je zag bij het voorbeeld 'Reageren op aanraking'. Hier vragen we aan de robot om een bepaalde actie uit te voeren. Bij het 'Reageren op aanraking' voorbeeld, vragen we aan de robot om ons data op te sturen van een bepaalde sensor op de stream `rom.sensor.touch.stream`.

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks

@inlineCallbacks
def main(session, details):
    yield session.call("rom.actuator.audio.stream",
       url = "https://stream.qmusic.nl/qmusic/mp3",
       sync=False)
    print("Het laden van de radio kan even duren")
    print("Druk op je toetsenbord om de muziek te stoppen!")
    input()
    yield session.call("rom.actuator.audio.stop")
    session.leave() # Sluit de verbinding met de robot

# Create wamp connection
wamp = Component(
    transports=[{
       "url": "ws://wamp.robotsindeklas.nl",
       "serializers": ["msgpack"],
       "max_retries": 0
    }],
    realm="rie.5e1312363dbf49eed032e123",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

PlayBotix

Gezichten vinden en volgen

Als een robot een camera op zijn hoofd heeft zitten, dan is die bij ons in staat om een gezicht te vinden en vervolgens ook te volgen. Dus stel jij gaat schuin tegenover de robot zitten en je kijkt de robot aan, dan laat je met de voorbeeldcode hieronder de robot op zoek gaan naar jouw gezicht en zodra de robot je ziet, zal die je proberen te volgen met zijn hoofd.

Het eerste wat we tegen de robot zeggen is dat die moet gaan opstaan, en dat doen we door `rom.optional.behavior.play` aan te roepen en daarbij te zeggen dat we willen opstaan (`name="BlocklyStand"`). Vervolgens staat de robot op en vertellen we tegen de robot dat die op zoek moet gaan naar gezichten. Dit doen we door `rie.vision.face.find` aan te roepen. De robot begint vervolgens rond te kijken totdat die een gezicht ziet en hem recht aankijkt. Als de robot je recht aankijkt, dan begroeten we je door iets te zeggen. Vervolgens zeggen we tegen de robot dat die je gezicht moet volgen totdat de robot je niet meer ziet. Dit doen we door `rie.vision.face.track` aan te roepen. Zodra we niet meer een gezicht zien, laten we de robot wat zeggen en hem vervolgens laten we hem zitten.

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks

@inlineCallbacks
def main(session, details):
    yield session.call("rom.optional.behavior.play",
        name="BlocklyStand")
    yield session.call("rie.vision.face.find")
    yield session.call("rie.dialogue.say", text="Hoi!")
    yield session.call("rie.vision.face.track")
    yield session.call("rie.dialogue.say",
        text="Ik zie je niet meer!")
    yield session.call("rom.optional.behavior.play",
        name="BlocklyCrouch")
    session.leave() # Sluit de verbinding met de robot

# Create wamp connection
wamp = Component(
    transports=[{
       "url": "ws://wamp.robotsindeklas.nl",
       "serializers": ["msgpack"],
       "max_retries": 0
    }],
    realm="rie.5e1312363dbf49eed032e123",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

Belangrijke tips

Yield & Await

In de voorbeeld codes hierboven heb je waarschijnlijk gezien dat we veel gebruik maken van `yield` (Python) en `await` (JavaScript). Met deze twee woorden zeg je tegen je computer dat je wilt wachten op een resultaat of totdat een bepaalde handeling is uitgevoerd. Als we bijvoorbeeld de `yield` weghalen uit `yield session.call("rie.dialogue.say", text="Hoi!")`, dan wachten we niet meer totdat de robot klaar is met praten, maar gaat de code direct door met het uitvoeren van de regels daaronder. Wij raden aan, zeker als dit de eerste keer is dat je aan de slag gaat met dit soort code, om standaard `yield` voor `session.call` en `session.subscribe` te plaatsen. **Let op:** in functies waar gebruik wordt gemaakt van `yield` en `await`, dien je voor Python boven de functie `@inlineCallback` te plaatsen en voor JavaScript `async` voor de function, bijvoorbeeld: `async function test () { ... }`.

Subscribe

Voordat je een stream op een modaliteit opent, moet je eerst hebben aangemeld (subscribe) op dit topic. Mocht je de volgorde omdraaien, dus eerst stream openen en daarna subscriben, dan bestaat de kans dat onze robot denkt dat er geen subscriptions zijn en sluit die automatisch weer de topic.

Wachten

Mocht je een bepaalde tijd willen wachten in de code, dan is het belangrijk in Python dat je **niet** gebruik maakt van de standaard `time.sleep()`! In plaats van de ingebouwde `time.sleep()`, dien je de `sleep` van `autobahn.twisted.util` te gebruiken. Een voorbeeld van hoe je dit zou moeten implementeren kan je hieronder vinden.

```
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep

@inlineCallbacks
def main(session, details):
    yield session.call("rie.dialogue.say",
        text="Ik wacht nu 20 seconden")

    # Wacht 20 seconden
    yield sleep(20)

    yield session.call("rie.dialogue.say",
        text="Ik ben klaar met wachten!")
```

Voor JavaScript kun je de standaard setTimeout() gebruiken of onze slaap functie die op dezelfde manier werkt als de `sleep` van Python, zie de voorbeeldcode hieronder.

```
function slaap(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}
async main(session) => {
    await session.call("rie.dialogue.say", [],
        {"text": "Ik wacht nu 20 seconden"})

    // Wacht 20 seconden
    await slaap(20_000)

    await session.call("rie.dialogue.say", [],
        {text: "Ik ben klaar met wachten!"})
}
```

ROM API

De ROM API is een stukje code die alle basisfunctionaliteit van een robot beschikbaar maakt voor ons. ROM staat voor Robot Operating Module en zorgt er dus voor dat de robot met ons platform kan verbinden zodat jij aan de slag kan gaan met het programmeren van blokjes of het schrijven van echte code. Een ROM bestaat globaal gezien uit twee onderdelen, namelijk:

- Sensoren: voor het ophalen van data zoals camerabeelden en aanrakingssensoren.
- Actuatoren: voor het aansturen van de robot. Denk hier bijvoorbeeld aan het direct aansturen van de motoren of het afspelen van geluid;

Frames

Als je gebruikt gaat maken van onze programmeer omgeving, zul je waarschijnlijk het woord ‘frame’ vaak voorbij zien komen. Een frame houd bij ons in een stukje informatie dat op een standaard manier gegevens beschrijft. Deze informatie kan je bijvoorbeeld vertellen of iemand één van de hoofdknoppen van Nao heeft ingedrukt of jij kan in een frame vertellen hoe de robot zijn motoren moet aansturen om ja te knikken. Een enkel frame bestaat uit een `time` en `data` key. De `time` key geeft de tijd aan in milliseconde wanneer iets gebeurt is of hoelang iets moet gaan duren. In de `data` key worden ubi’s opgeslagen met de daarbij behorende waardes. Deze waarde kan een sensor waarde van aanrakings sensoren zijn of de hoek die motor moet bereiken.

```
{
    time: time,
    data: {
        ubi1: value,
        ubi2: value,
        ubix: value
    }
}
```

Sensoren

Sensoren zijn slimme, kleine componenten op de robot die iets kunnen vertellen over de omgeving van de robot. Zo hebben alle robots van Robots in de Klas een camera waarmee ze rond kunnen kijken. Ook beschikken ze over een microfoon waarmee ze kunnen luisteren. In totaal ondersteunt ons platform vier verschillende sensoren:

1. Sight - Zicht: de camerabeelden van een robot;
2. Hearing - Gehoor: de audio van een microfoon;
3. Touch - Aanraking: de aanrakingsgevoelige sensoren op een robot;
4. Proprio - Proprioceptie: het uitlezen van alle hoeken die de motoren van een robot maken.

Elke sensor heeft dezelfde implementatie:

RPC	**Geeft terug**
rom.sensor.<modality>.info: vraag informatie op van een modaliteit	Een dictionary met daarin informatie over de specifieke modaliteit.
rom.sensor.<modality>.read: luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze modaliteit Argumenten: - ubi: (List<String>, default=[ubi]): Een lijst van UBI's waar je de data van wilt hebben voor deze read. - time: (Double, default=0.0): de tijd hoelang je de read wilt laten lopen in milliseconde, bij time=0.0 wordt een enkel frame teruggegeven.	De data die verzameld is van de specifieke modaliteit en ubi('s) over de vooraf ingestelde tijd (time).
rom.sensor.<modality>.stream: open een stream waarop data direct wordt geplaatst op de topic `rom.sensor.<modality>.stream` zodra het beschikbaar is. Belangrijk hierbij is dat je eerst subscribed voordat je de stream aanroept. Argument: - ubi: (List<String>, default=[ubi]): Een lijst van UBI's waar je naar wilt luisteren. Als je deze leeg laat dan stuurt de modaliteit alle beschikbare data op.	None
rom.sensor.<modality>.close: sluit een stream nadat je deze data niet meer nodig hebt.	None

Argument: - ubi: (List<String>, default=[ubi]): Een lijst van UBI's waar je niet meer naar wilt luisteren.	
rom.sensor.<modality>.sensitivity: de gevoeligheid geeft aan wanneer het verschil tussen twee datapunten groot genoeg is om door te sturen. Argument: - sensitivity: (Double, default=None): de sensitivity waar je de modaliteit op wilt zetten.	Een double van de gevoeligheid waar de robot nu op staat voor de geselecteerde modaliteit

Een voorbeeld van hoe je sensor data kunt verzamelen via de `stream` en `read` RPC in Python:

```
from twisted.internet.defer import inlineCallbacks

def proprio(frame):
  # Deze functie wordt elke keer aangeroepen wanneer
  # de robot beweegt
  print(frame)

@inlineCallbacks
def main(session, details):
  # Krijg de hoeken van alle motoren op dit moment
  frames = yield session.call("rom.sensor.proprio.read")
  print("Huidige motoren stand:")
  print(frames[0]["data"])
  print("")

  # Hier melden we ons aan voor de proprio data
  # en starten we een stream
  yield session.subscribe(proprio, "rom.sensor.proprio.stream")
  yield session.call("rom.sensor.proprio.stream")

  # Nu laten we de robot wat bewegen
  # zodat we proprio data krijgen
  # van onze rom.sensor.proprio.stream stream
  yield session.call("rom.optional.behavior.play",
    name="BlocklyStand")
  yield session.call("rom.optional.behavior.play",
    name="BlocklyWaveRightArm")
  yield session.call("rom.optional.behavior.play",
    name="BlocklyCrouch")
  session.leave() # Sluit de verbinding met de robot
```

En in JavaScript:

```
proprio = (event) => {
    // Deze functie wordt elke keer aangeroepen wanneer
    // de robot beweegt
    frame = event[0]
    console.log(frame)
}

async function main(session) {
    // Krijg de hoeken van alle motoren op dit moment
    frames = await session.call("rom.sensor.proprio.read")
    console.log("Huidige motoren stand:")
    console.log(frames[0]["data"])
    console.log("\n") // Print een lege regel

    // Hier melden we ons aan voor de proprio data
    // en starten we een stream
    await session.subscribe("rom.sensor.proprio.stream", proprio)
    await session.call("rom.sensor.proprio.stream")

    // Nu laten we de robot wat bewegen
    // zodat we proprio data krijgen
    // van onze rom.sensor.proprio.stream stream
    await session.call("rom.optional.behavior.play", [],
                       {"name": "BlocklyStand"})
    await session.call("rom.optional.behavior.play", [],
                       {"name": "BlocklyWaveRightArm"})
    await session.call("rom.optional.behavior.play", [],
                       {"name": "BlocklyCrouch"})
    session.leave() // Sluit de verbinding met de robot
}
```

Actuatoren

Actuatoren zijn onderdelen van de robot die de robot in staat stellen om iets te doen. Denk hier bijvoorbeeld aan het afspelen van een muziekje of het zwaaien van de armen. Ons platform ondersteund vier standaard actuatoren die elke robot geïmplementeerd heeft, namelijk:

1. Motor
2. Audio
3. Licht
4. Gedrag

Een overzicht van alle RPC's in de actuator modaliteit:

Motor

```
rom.actuator.motor.info
rom.actuator.motor.write
rom.actuator.motor.stop
```

Audio

```
rom.actuator.audio.info
rom.actuator.audio.volume
rom.actuator.audio.play
rom.actuator.audio.stream
rom.actuator.audio.stop
```

Licht

```
rom.actuator.light.info
rom.actuator.light.write
rom.actuator.light.stop
```

Gedrag (Behavior)

```
rom.optional.behavior.info
Rom.optional.behavior.play
rom.optional.behavior.stop
```

Motor

Met de Motor modaliteit kun je direct de motoren aansturen en zo je eigen bewegingen creëren. Voor het aansturen van de motoren maken wij gebruik van een assenstelsel om te bepalen dat bijvoorbeeld een beweging tegen de klok in positief is en met de klok mee negatief. Meer informatie over hoe dit assenstelsel eruit ziet en hoe je ermee aan de slag kan gaan, kun je hier vinden. Een goed voorbeeld van hoe je je eigen bewegingen kunt creëren is te zien in onze Voorbeeldcode hoofdstuk waar we als voorbeeld de robot ja laten knikken. Let overigens hierbij op dat alle waardes voor de motoren in **radialen** zijn.

RPC	**Geeft terug**
rom.actuator.motor.info: vraag de informatie op van de motor modaliteit	Een dictionary met daarin informatie over de Motor modaliteit.
rom.actuator.motor.write: stuurt een motor of meerdere motoren aan naar de gewenste positie Argumenten: - frames: (List<Frame>): een lijst met daarin frames die vertellen naar welke oriëntaties de motoren toe moeten gaan en hoeveel tijd ze daarvoor hebben; - mode: (String, default='linear'): Met de mode kun je vertellen hoe de robot tussen de verschillende frames naar de volgende positie toe moet gaan. De modaliteit ondersteunt de volgende drie modes: 'linear', 'last' en 'none'. In de bijlage kun je een uitgebreide uitleg vinden wat deze modes doen tussen de frames. In de meeste gevallen is de default waarde 'linear' voldoende; - sync: (Bool, default=True): indien True, wacht totdat de robot alle motor commando's heeft uitgevoerd; - force: (Bool, default=False): indien True, probeer je zo dicht mogelijk bij de gewenste motor positie te komen ook al kan de motor niet de exacte positie bereiken door hardware limitaties. Indien False, dan ga je er van uit dat de robot naar de gewenste positie kan gaan.	None, maar indien sync=True, dan kan je wachten op de RPC totdat de robot klaar is met het uitvoeren van de motor commando's. Deze RPC kan een Error teruggeven indien je force op False staat en je probeert naar een positie te gaan die de robot niet kan bereiken. Als je bijvoorbeeld het hoofd 360 graden wilt laten draaien en het hoofd kan maar 180 graden draaien, dan geeft deze RPC een error bij force=False en de robot gaat naar 180 graden toe bij force=True.
rom.actuator.motor.stop: stop de draaiende `rom.actuator.motor.write` RPC. Let hierbij wel op dat de robot mogelijk in een instabiele houding komt te staan waardoor deze kan omvallen.	None

Audio

Met de Audio modaliteit kun je geluidjes en muziekjes afspelen en streamen op de robot. Zo kun je op de robot een radiozender afspelen terwijl de robot staat te dansen. In onze Voorbeeldcode hoofdstuk hebben we een voorbeeld staan die laat zien hoe je een muziek stream kunt afspelen op de robot.

RPC	**Geeft terug**
rom.actuator.audio.info: vraag de informatie op van de Audio modaliteit	Een dictionary met daarin informatie over de Audio modaliteit.
rom.actuator.audio.volume: pas het volume aan en/of krijg de huidige volume terug Argument: - volume: (Int): de volume ligt tussen een waarde van 0 (geen geluid) en 100 (maximaal volume).	Het volume waar de robot nu op staat
rom.actuator.audio.play: speel direct een ruwe stereo wave audio bestand af. Argumenten: - data: (byte[]): de ruwe stereo wave audio data; - rate: (int, default=44100): de sample rate van de audio die wordt opgestuurd; - sync: (Bool, default=True): indien True, wacht totdat het geluid volledig is afgespeeld.	None, maar indien sync=True, dan kan je wachten op de RPC totdat de robot klaar is met het afspelen van het geluid
rom.actuator.audio.stream: speel geluid van een webstream af. Argumenten: - url: (String): een link naar een (web) locatie waar audio vanaf wordt gestreamt. Belangrijk hierbij is dat Nao en Pepper alleen http streams ondersteunen en de virtuele robot alleen https; - sync: (Bool, default=False): indient True, wacht totdat de stream klaar is met het afspelen van het geluid.	None, maar indien sync=True, dan kan je wachten op de RPC totdat de robot klaar is met het afspelen van het geluid
rom.actuator.audio.stop: stop alle audio die wordt afgespeeld	None

Licht

Met de licht modaliteit kunnen we de kleur van lichtjes op een robot aanpassen. Bij bijvoorbeeld de Nao kunnen we de oogkleuren aanpassen via deze module. Zo kunnen we de oogkleur naar groen veranderen als het antwoord op een vraag correct is. De achterliggende techniek voor het veranderen van de kleurtjes lijkt sterk op hoe je motoren aanstuurt, alleen stuur je nu kleurinformatie op in plaats van hoeken.

RPC	**Geeft terug**
rom.actuator.light.info: vraag de informatie op van de licht modaliteit.	Een dictionary met daarin informatie over de licht modaliteit.
rom.actuator.light.write: zet de kleur van de lichten van de robot, denk hierbij bijvoorbeeld aan de oogkleur van de Nao robot. Argumenten: - frames: (List<Frame>, default=[ubi]): een lijst met daarin frames die vertellen naar welke kleuren de lichten moeten veranderen en hoeveel tijd ze daarvoor hebben. Een kleur is gedefinieerd als [Rood, Groen, Blauw] en kan een waarde tussen 0 en 255 hebben; - mode: (String, default='linear'): Met de mode kun je vertellen hoe de robot tussen de frames naar de volgende kleur toe moet gaan. De ROM ondersteunt de volgende drie modes: 'linear', 'last' en 'none'. Zie de bijlage voor een verdere uitleg over mode; - sync: (Bool, default=True): indien True, wacht totdat de robot alle licht commando's heeft uitgevoerd; - force: (Bool, default=False): indien True, dan worden je waardes boven en onder de limieten van 0 en 255 omgezet naar de min en max waarde. Indien False, dan krijg je een error terug als je een waarde opstuurt onder de 0 en boven de 255.	None, maar indien sync=True, dan kan je wachten op de RPC totdat de robot klaar is met het uitvoeren van het veranderen van de lichten. Deze RPC kan een Error teruggeven indien je force op False staat en je een kleur probeert te zetten die niet bestaan zoals [500,0,0] of [125,0]. In het eerste geval zit je over de limiet van 255 heen en in het tweede geval weet de licht modaliteit niet wat voor kleur dat is, er mist namelijk nog één waarde.
rom.actuator.light.stop: stop de `rom.actuator.light.write`	None

Een voorbeeld van hoe je de oogkleuren van de robot kunt aanpassen in Python:

```
# Verander de oogkleuren van groen, naar rood, naar blauw
yield session.call("rom.actuator.light.write",
  mode="linear",
  frames=[{"time": 1000, "data": {"body.head.eyes": [0,255,0]}},
          {"time": 2000, "data": {"body.head.eyes": [255,0,0]}},
          {"time": 3000, "data": {"body.head.eyes": [0,0,255]}}])
```

En in JavaScript:

```
// Verander de oogkleuren van groen, naar rood, naar blauw
await session.call("rom.actuator.light.write", [],
  {"frames": [
    {"time": 1000, "data": {"body.head.eyes": [0,255,0]}},
    {"time": 2000, "data": {"body.head.eyes": [255,0,0]}},
    {"time": 3000, "data": {"body.head.eyes": [0,0,255]}}
  ],
  "mode": "linear"})
```

Gedrag (Behavior)

Met de Gedrags (Behavior) modaliteit, is het mogelijk om de robot vooraf ingeprogrammeerde bewegingen te laten uitvoeren. Zo is het mogelijk om op een simpele manier de robot te laten zwaaien met de behavior: BlocklyWaveRightArm.

RPC	**Geeft terug**
rom.optional.behavior.info: vraag de informatie op van de Behavior modaliteit.	Een dictionary met daarin een lijst van behaviors die afgespeeld kunnen worden.
rom.optional.behavior.play: speel een behavior af op de robot. Argumenten: - name: (String, default=''): de naam van de behavior, een lijst van behaviors kan je opvragen door eerst `rom.optional.behavior.info` aan te roepen; - sync: (Bool, default=True): indien True, wacht deze RPC totdat de robot klaar is met het uitvoeren van zijn beweging.	None, maar indien sync=True, dan kan je wachten op de RPC totdat de robot klaar is met het uitvoeren van de beweging.
rom.optional.behavior.stop: stop de `rom.optional.behavior.play`. Let hierbij wel op dat de robot mogelijk in een instabiele houding komt te staan waardoor deze kan omvallen.	None

In de bijlage kun je een lijst terugvinden met alle standaard geïnstalleerde gedragingen op de robot.

Een voorbeeld van hoe je de robot kunt laten opstaan. Vervolgens wat laten zeggen terwijl de robot daar bij zwaait. En als laatst stap, de robot weer laten zitten. In Python:

```
# Laat de robot opstaan
yield session.call("rom.optional.behavior.play",
    name="BlocklyStand")

# Laat de robot wat zeggen terwijl die zwaait
behavior = session.call("rom.optional.behavior.play",
    name="BlocklyWaveRightArm")
yield session.call("rie.dialogue.say", text="Hallo!")
yield behavior

# Laat de robot weer zitten
yield session.call("rom.optional.behavior.play",
    name='BlocklyCrouch')
```

In JavaScript:

```
// Laat de robot opstaan
await session.call("rom.optional.behavior.play", [],
    {"name": "BlocklyStand"})

// Laat de robot wat zeggen terwijl die zwaait
behavior = session.call("rom.optional.behavior.play", [],
    {"name": "BlocklyWaveRightArm"})
await session.call("rie.dialogue.say", [], {"text":"Hallo!"})
await behavior

//Laat de robot weer zitten
await session.call("rom.optional.behavior.play", [],
    {"name": "BlocklyCrouch"})
```

Data

Met de Data modaliteit is het mogelijk om in één opslag alle informatie terug te krijgen van alle geïmplementeerde modaliteiten.

De RPC is `rom.data.info` en geeft bijvoorbeeld het volgende terug:

```
{
    data:   {
        serial: "abcd0123456789",
        type:   "nao",
        version:"v5"
    },
    sensor: {
        sight:        {...},
        hearing:      {...},
        touch:        {...},
        proprio:      {...}
    },
    actuator:   {
        light: {...},
        audio: {...},
        motor: {...}
    },
    optional:   {...}
}
```

Naast de info van alle modaliteiten, kun je met de Data modaliteit ook de huidige status opvragen met de RPC: `rom.data.status`. Deze RPC geeft het volgende terug:

```
data:  {
    battery: 50,
    network: {
        ssid:       "naonet",
        strength:   60
    }
}
```

RIE API

Naast de basisfunctionaliteiten van de robot, kunnen we ook de robot geavanceerde dingen laten doen. Zo kunnen we met de geavanceerde functionaliteiten gezichten zoeken en de robot iets laten zeggen. Dit stukje geavanceerde functionaliteit noemen we RIE[2].

Een overzicht van alle RPC's:

Dialoog:

```
rie.dialogue.say
rie.dialogue.say_animated
rie.dialogue.config.native_voice
rie.dialogue.config.language
rie.dialogue.config.profanity
rie.dialogue.ask
rie.dialogue.stop
rie.dialogue.keyword.add
rie.dialogue.keyword.remove
rie.dialogue.keyword.clear
rie.dialogue.keyword.stream
rie.dialogue.keyword.close
rie.dialogue.keyword.read
rie.dialogue.stt.hints
rie.dialogue.stt.info
rie.dialogue.stt.stream
rie.dialogue.stt.read
rie.dialogue.stt.close
```

Vision:

```
rie.vision.card.info
rie.vision.card.stream
rie.vision.card.read
rie.vision.card.close
rie.vision.face.info
rie.vision.face.stream
rie.vision.face.read
rie.vision.face.close
rie.vision.face.find
rie.vision.face.track
rie.vision.face.stop
```

Cloud modules:

```
rie.cloud_modules.ready
```

[2] Robot Interaction Engine: een geavanceerd stukje code waarmee we de robot slimme dingen mee kunnen laten doen

Dialoog

De dialoog module is ontworpen voor het mogelijk maken van interactie met een gebruiker via spraak. Zo kunnen we via deze module de robots iets laten zeggen, maar kan de robot ook luisteren naar de gebruiker.

Tekst-naar-Spraak (TTS)

Met Tekst-naar-Spraak (TTS) kun je de robot iets laten zeggen. Zo kun je met TTS bijvoorbeeld een gebruiker begroeten als de robot iemand ziet of feedback geven op een vraag van de gebruiker. De TTS module kan gebruik maken van twee stemmen afhankelijk van of de robot zelf ook kan praten. De eerste stem is van de robot zelf en de tweede stem is afkomstig van ReadSpeaker. Sommige robots, zoals de Alpha-Mini, hebben zelf geen stem en ondersteunen alleen maar de stem afkomstig van ReadSpeaker.

RPC	**Geeft terug**
rie.dialogue.say: laat de robot iets zeggen zonder bewegingen daarbij uit te voeren. Argumenten: - text (String): de tekst die de robot moet zeggen; - lang (String, Default=None): de taal waarin de robot de tekst moet zeggen. Deze kan leeg gelaten worden en dan gebruikt de robot de laatste gezette taal. Een voorbeeld van lang is ‘nl’ en ‘en’.	None
rie.dialogue.say_animated: laat de robot iets zeggen terwijl de robot daarbij bewegingen uitvoert. Argumenten: - text (String): De tekst die de robot moet zeggen; - lang (String, Default=None): De taal waarin de robot de tekst moet zeggen. Deze kan leeg gelaten worden en dan gebruikt de robot de laatste gezette taal. Een voorbeeld van lang is ‘nl’ en ‘en’.	None
rie.dialogue.config.native_voice: zet welke spraak engine de voorkeur heeft. Argument: - use_native_voice (Boolean): True → de dialoog module probeert de al geïnstalleerde taal op de robot te gebruiken, maar valt terug op ReadSpeaker als de robot geen spraak engine heeft of de gewenste taal	None

niet ondersteund. False → de robot gebruikt alleen de stem van ReadSpeaker.	
rie.dialogue.config.language: zet de taal van de dialoog module. De taal kan succesvol worden gezet wanneer de spraak van de robot of ReadSpeaker de gewenste taal ondersteund. Als je de variabele lang leeg laat, dan geeft de RPC direct terug welke taal die nu gebruikt. Argument: - lang (String, Default=None): de taal code. Gesupporte formaten zijn: ‘en' en ‘en_uk’. Taal codes die eindigen met een liggend streep worden niet ondersteund.	String van de taal code die de dialoog module op dit moment gebruikt.
rie.dialogue.config.speed: zet de spreeksnelheid van de dialoog module. Argument: - Speed (Integer, Default=None): de spreeksnelheid kan een waarde zijn tussen 0 en 200 waar 0 heel langzaam is en 200 zeer snel.	De spreeksnelheid die nu gebruikt wordt
rie.dialogue.config.pitch: zet de toonhoogte van de stem. Argument: - pitch (Integer, Default=None): de toonhoogte kan een waarde zijn tussen 0 en 200 waar 0 heel laag is en 200 zeer hoog.	De toonhoogte die nu gebruikt wordt

Voorbeeld van hoe je de robot iets kan laten zeggen in Python:

```
# Laat de robot iets zeggen:
yield session.call("rie.dialogue.say",
    text="Hallo, goed je te zien!")

# Maak alleen gebruik van de ReadSpeaker TTS-engine:
yield session.call("rie.dialogue.config.native_voice",
    use_native_voice=False)

# Zeg wat met de ReadSpeaker stem:
yield session.call("rie.dialogue.say",
     text="Dit is de ReadSpeaker stem!")

# Verander de taal naar Engels:
yield session.call("rie.dialogue.config.language", lang="en")


# Zeg iets met de ReadSpeaker stem in het Engels:
yield session.call("rie.dialogue.say",
     text="I am now speaking English!")
```

In JavaScript:

```
// Laat de robot iets zeggen:
await session.call("rie.dialogue.say", [],
    {"text":"Hallo, goed je te zien!"})

// Maak alleen gebruik van de ReadSpeaker TTS-engine:
await session.call("rie.dialogue.config.native_voice", [],
    {"use_native_voice": false})

// Zeg wat met de ReadSpeaker stem:
await session.call("rie.dialogue.say", [],
    {"text":"Dit is de ReadSpeaker stem!"})

// Verander de taal naar Engels:
await session.call("rie.dialogue.config.language", [],
    {"lang": "en"})

// Zeg iets met de ReadSpeaker stem in het Engels:
await session.call("rie.dialogue.say", [],
    {"text": "I am now speaking English!"})

// En weer terug naar Nederlands
await session.call("rie.dialogue.config.language", [],
    {"lang": "nl"})
```

Slimme vraag

Een ander slim trucje dat we met RIE kunnen doen is op een slimme manier een vraag stellen aan iemand die een interactie aangaat met de gebruiker. Wat je vaak ziet gebeuren bij interacties is dat de robot een vraag stelt, vervolgens geeft de gebruiker een antwoord en de robot heeft de gebruiker niet goed verstaan. Wat er dan gebeurd is dat de robot je met een koude blik aan blijft kijken en je als gebruiker niet weet of de robot je niet verstaan heeft of dat de robot nog bezig is om de tekst die je net zei te verwerken. Dit zorgt voor een ongemakkelijke situatie bij een gebruiker. Wat deze module doet, is een vraag aan een gebruiker stellen en vervolgens slim met de antwoorden omgaan die de gebruiker zegt. Als de gebruiker bijvoorbeeld zegt: 'Wat zei je?', dan zegt de robot op zijn beurt: "Ik zal de vraag voor u herhalen. De vraag is: ...". Ook als de robot niet in staat was om in één keer het juiste antwoord te horen, dan zegt de robot bijvoorbeeld beleefd: "Sorry, kunt u het herhalen wat u net zei?". Als de slimme vraag module een antwoord match heeft gevonden, dan stuurt die je het antwoord terug en daarop kan jij dan een gepersonaliseerde response op geven. Zie het voorbeeld hieronder hoe je de slimme vraag module kan implementeren.

RPC	**Geeft terug**
Rie.dialogue.ask: vraag op een slimme manier een vraag. Argumenten: - question (String): de vraag die de robot moet stellen; - answers (Dict<List>, default=None): Een dictionary die bestaat uit het antwoord als keys en een lijst met woorden die klinken als het antwoord. Die lijst helpt onze algoritme om uit audio gesproken tekst te halen. Een belangrijke tip hierbij is dat je moet proberen niet twee antwoorden te hebben die hetzelfde klinken (een homofoon zoals 'wij' en 'wei') of hetzelfde zijn maar iets anders betekenen (een homoniem zoals 'bank'); - max_attempts (Int, default=4): de maximale hoeveelheid pogingen de robot waagt mocht die de gebruiker niet begrijpen; - language (String, Default=None): de taalcode. Op dit moment wordt alleen 'nl' en 'en' ondersteund.	Eén van de antwoorden van de parameter answers mocht `rie.dialogue.ask` een match hebben gevonden. In alle andere gevallen geeft die None terug als die geen match heeft kunnen vinden of de RPC was gestopt omdat iemand `rie.dialogue.stop` heeft aangeroepen.
rie.dialogue.stop: mocht `rie.dialogue.ask` actief zijn, dan kan je de RPC stoppen door `rie.dialogue.stop` aan te roepen.	None

Voorbeeld van hoe je de slimme vraag kan implementeren in Python:

```
question = "Welke kleur vind je mooier, rood of blauw?"
answers = {"rood": ["rood", "rot"], "blauw": ["blauw", "lauw"]}

answer = yield session.call("rie.dialogue.ask",
                            question=question,
                            answers=answers)

if answer == "rood":
    yield session.call("rie.dialogue.say",
        text="Rood is zeker een mooie kleur!")
elif answer == "blauw":
    yield session.call("rie.dialogue.say",
        text="Ik ben gek op blauw!")
else:
    yield session.call("rie.dialogue.say",
        text="Sorry, maar ik heb je niet goed verstaan.")
```

En in JavaScript:

```
let question = "Welke kleur vind je mooier, rood, of blauw?"
let answers = {
  "rood": ["rood", "rot"],
  "blauw": ["blauw", "lauw"]
}

let answer = await session.call("rie.dialogue.ask", [],
    {"question":question, "answers": answers})

if (answer == "rood") {
  await session.call("rie.dialogue.say", [],
      {"text":"Rood is zeker een mooie kleur!"})
} else if (answer == "blauw") {
  await session.call("rie.dialogue.say", [],
      {"text":"Ik ben gek op blauw!"})
} else {
  await session.call("rie.dialogue.say", [],
      {"text":"Sorry, maar ik heb je niet goed verstaan."})
}
```

Keyword

Keywords zijn specifieke woorden waar we naar willen luisteren. Denk hier bijvoorbeeld aan een situatie waar je een gesloten vraag stelt, een vraag waar je van te voren het antwoord op weet, en je wacht totdat je een antwoord hoort. In die situatie kun je gebruik maken van keywords omdat je van te voren al weet wat de antwoorden zijn.

RPC	**Geeft terug**
rie.dialogue.keyword.info: vraag de informatie op van de Keyword module.	Een dictionary met daarin informatie welke talen worden ondersteund door de Keyword module
rie.dialogue.keyword.add: voeg een lijst met nieuwe keywords toe waar de robot naar moet luisteren. Argument: - keywords: (List<String>, default=None): Voeg een lijst met keywords toe waar de robot naar moet luisteren.	Een lijst met daarin de keywords waar de robot op dit moment naar luistert
rie.dialogue.keyword.remove: verwijder eerder toegevoegde keywords van de lijst van keywords waar de robot naar moet luisteren Argument: - keywords: (List<String>, default=None): Haal een paar keywords uit de lijst van waar de robot naar moet luisteren.	Een lijst met daarin de keywords waar de robot op dit moment naar luistert
rie.dialogue.keyword.clear: verwijder alle keywords van de lijst van keywords waar de robot naar moet luisteren.	None
rie.dialogue.keyword.language: verander de taal waarin we de keywords verwachten. Als de taal van de Keyword module in het Nederlands staat, dan zal de module grote moeite hebben om naar Engelse woorden te luisteren. Argument: - lang: (String, default=None): de taalcode van de taal die je wenst voor de Keyword module. Gesupporte formaten zijn: ‘en' en ‘en_uk’. Taalcodes die eindigen met een liggend streep worden niet ondersteund.	De taalcode van de taal die de robot nu gebruikt. Kan een ValueError geven als je een taal probeert te zetten die niet gesupport wordt door de robot
rie.dialogue.keyword.read: luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze module.	De data die verzameld is van de keyword module over de vooraf ingestelde

Argumenten: - ubi: (List<String>, default=[ubi]): luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze module; - time: (Double, default=0.0): de tijd hoelang je de read wilt laten lopen in milliseconde, bij time=0.0 wordt een enkel frame teruggegeven.	tijd (time).
rie.dialogue.keyword.stream: open een stream waarop data direct wordt geplaatst op de topic `rie.dialogue.keyword.stream` zodra het beschikbaar is. Belangrijk hierbij is dat je eerst subscribed voordat je de stream aanroept. Argument: - ubi: (List<String>, default=[ubi]): een lijst van ubi's waarvan je de data wilt hebben. Meestal is het voldoende om deze parameter weg te laten.	None
rie.dialogue.keyword.close: sluit een stream nadat je deze data niet meer nodig hebt. Argument: - ubi: (List<String>, default=[ubi]): een lijst met ubi's waar je niet meer subscribed op wilt zijn.	None

Voorbeeld van hoe je de keyword module kunt gebruiken in Python:

```
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep

@inlineCallbacks
def main(session, details):
    global sess
    sess = session

    def on_keyword(frame):
      global sess
      if ("certainty" in frame["data"]["body"] and
          frame["data"]["body"]["certainty"] > 0.45):
        sess.call("rie.dialogue.say", text= "Ja")

    yield session.call("rie.dialogue.say",
        text="Zeg rood, blauw of groen")
    yield session.call("rie.dialogue.keyword.add",
        keywords=["rood", "blauw", "groen"])
    yield session.subscribe(on_keyword,
        "rie.dialogue.keyword.stream")
    yield session.call("rie.dialogue.keyword.stream")

    # Wacht 20 seconden voordat we de keyword stream sluiten
    yield sleep(20)

    yield session.call("rie.dialogue.keyword.close")
    yield session.call("rie.dialogue.keyword.clear")
    session.leave() # Sluit de verbinding met de robot
```

En in JavaScript:

```
var sess = null
// Gebruik await slaap(s) om s seconden te wachten
let slaap = s=>new Promise(r=>setTimeout(r, s*1000))

async function on_keyword(event) {
  let frame = event[0]
  if (frame["data"]["body"]["certainty"] > 0.45) {
    await sess.call("rie.dialogue.say", [], {"text": "Ja"})
  }
}

async function main(session, event) {
  sess = session
  await session.call("rie.dialogue.say", [],
      {"text": "Zeg rood, blauw of groen"})

  await session.call("rie.dialogue.keyword.add", [],
      {"keywords": ["rood", "blauw", "groen"]})

  await session.subscribe("rie.dialogue.keyword.stream",
                          on_keyword)
  await session.call("rie.dialogue.keyword.stream")

  // Wacht 20 seconden voordat we de keyword stream sluiten
  await slaap(20)

  await session.call("rie.dialogue.keyword.close")
  await session.call("rie.dialogue.keyword.clear")
  wamp.close() // Sluit de verbinding met de robot
}
```

Spraak-naar-Tekst (STT)

Met Spraak-naar-Tekst (STT) kun je direct gesproken tekst halen uit wat de robot denkt te horen. Zo kunt je bijvoorbeeld eerst een open-vraag stellen aan een gebruiker en middels STT het antwoord van de gebruiker verkrijgen en verder verwerken in je code.

RPC	**Geeft terug**
rie.dialogue.stt.info: vraag de informatie op van de STT module.	Een dictionary met daarin informatie welke talen worden ondersteund door de STT module
rie.dialogue.stt.hints: voeg een lijst met woorden toe die waarschijnlijk gezegd gaan worden tijdens de conversatie. Door gebruik te maken van hintwoorden, kan de STT module beter inschatten wat er gezegd wordt. Argument: - words: (List<String>, default=None): Voeg een lijst met woorden toe waar de robot naar moet luisteren.	None
rie.dialogue.stt.read: luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze module. Argumenten: - ubi: (List<String>, default=[ubi]): luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze module; - time: (Double, default=0.0): de tijd hoelang je de read wilt laten lopen in milliseconde, bij time=0.0 wordt een enkel frame teruggegeven.	De data die verzameld is van de STT module over de vooraf ingestelde tijd (time).
rie.dialogue.stt.stream: open een stream waarop data direct wordt geplaatst op de topic `rie.dialogue.stt.stream` zodra het beschikbaar is. Belangrijk hierbij is dat je eerst subscribed voordat je de stream aanroept. Argument: - ubi: (List<String>, default=[ubi]): een lijst van ubi's waarvan je de data wilt hebben. Meestal is het voldoende om deze parameter weg te laten.	None
rie.dialogue.stt.close: sluit een stream nadat je deze data niet meer nodig hebt.	None

Argument: - ubi: (List<String>, default=[ubi]): een lijst met ubi's waar je niet meer subscribed op wilt zijn.	

Voorbeeld van hoe je de STT module kunt gebruiken in Python:

```
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep

@inlineCallbacks
def main(session, details):
    global sess
    sess = session

    def on_stt_result(frame):
      global sess
      print(frame["data"]["body"]["text"])

    yield session.call("rie.dialogue.say",
        text="Wat zou jouw antwoord zijn op een open vraag?")

    yield session.subscribe(on_stt_result,
        "rie.dialogue.stt.stream")
    yield session.call("rie.dialogue.stt.stream")

    # Wacht 20 seconden voordat we de stt stream sluiten
    yield sleep(20)

    yield session.call("rie.dialogue.stt.close")
    session.leave() # Sluit de verbinding met de robot
```

En in JavaScript:

```
var sess = null
// Gebruik await slaap(s) om s seconden te wachten
let slaap = s=>new Promise(r=>setTimeout(r, s*1000))

async function on_stt_result(event) {
  let frame = event[0]
  console.log(frame.data.body.text)
}

async function main(session, event) {
  sess = session
  await session.call("rie.dialogue.say", [],
      {"text": "Wat zou jouw antwoord zijn op een open vraag?"})

  await session.subscribe("rie.dialogue.stt.stream",
                          on_stt_result)
  await session.call("rie.dialogue.stt.stream")

  // Wacht 20 seconden voordat we de stt stream sluiten
  await slaap(20)

  await session.call("rie.dialogue.stt.close")
  wamp.close() // Sluit de verbinding met de robot
}
```

Vision

Met de vision module laten we slimme algoritmes los op de camerabeelden van de robot. Hiermee kunnen we bijvoorbeeld kaartjes herkennen uit camerabeelden, en kunnen we ook gezichten detecteren en volgen.

Kaart detectie

Met de camerabeelden van de robot kunnen we allerlei soorten kaartjes herkennen. Zo kan een Nao robot, Naomarks herkennen en daarop reageren. Andere soorten robots kunnen helaas niet de Naomarks herkennen. Om er toch voor te kunnen zorgen dat alle robots dezelfde soorten kaartjes kunnen herkennen, maken wij gebruik van markers genaamd Aruco. Deze markers zijn vergelijkbaar met zijn meer bekende broertje QR-code. Zo kunnen alle robots toch nog kaartjes met een marker herkennen en als bonus ondersteunt Nao ook nog steeds de Naomarks.

Wij maken op dit moment gebruik van 6x6 Aruco Markers en deze kunnen gegenereerd worden via de volgende website: http://chev.me/arucogen/.

RPC	**Geeft terug**
rie.vision.card.info: vraag de informatie op van de Kaart detectie module.	Een dictionary met daarin informatie over de Kaart detectie module.
rie.vision.card.read: luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze module. Argumenten: - ubi: (List<String>, default=['body']): een lijst van ubi's waarvan je de data wilt hebben. Meestal is het voldoende om deze parameter weg te laten; - time: (Int, default=0): de tijd dat deze RPC kaart detectie moet noteren in een frame. Deze tijd is in milliseconde. Als je time=0 opgeeft, dan wacht deze RPC totdat er een nieuw kaartje gedetecteerd is.	De data die verzameld is van de Kaart detectie module over de vooraf ingestelde tijd (time). Mocht time=0 zijn, dan krijg je één frame terug met daarin het gedetecteerde kaartnummer
rie.vision.card.stream: opent een stream op de topic `rie.vision.card.stream` waarop de gedetecteerde kaartnummers direct wordt geplaatst zodra een kaartnummer wordt gedetecteerd. Belangrijk hierbij is dat je eerst de subscribe doet voordat je de stream aanroept.	None

Argumenten: - ubi: (List<String>, default=[ubi]): een lijst van ubi's waarvan je de data wilt hebben. Meestal is het voldoende om deze parameter weg te laten. - time: (Double, default=0.0): de tijd hoelang je de read wilt laten lopen in milliseconde, bij time=0.0 wordt een enkel frame teruggegeven.	
rie.vision.card.close: sluit een stream nadat je deze data niet meer nodig hebt. Argument: - ubi: (List<String>, default=[ubi]): een lijst met ubi's waar je niet meer subscribed op wilt zijn.	None

Een voorbeeld van het gebruik van rie.vision.card.* in Python:

```
from twisted.internet.defer import inlineCallbacks

def on_card(frame):
  # Deze functie wordt elke keer aangeroepen wanneer
  # de robot een kaartje ziet
  print(frame)

@inlineCallbacks
def main(session, details):
  # Wacht totdat we een kaartje zien
  frames = yield session.call("rie.vision.card.read")
  print(frames[0])

  # Hier abonneren we ons op de card data en starten we een stream
  yield session.subscribe(on_card, "rie.vision.card.stream")
  yield session.call("rie.vision.card.stream")
```

En in JavaScript:

```
function on_card(event) {
    // Deze functie wordt elke keer aangeroepen wanneer
    // de robot een kaartje ziet
    let frame = event[0]
    console.log(frame)
}

async function main(session, event) {
    // Wacht totdat we een kaartje zien
    let frames = await session.call("rie.vision.card.read")
    console.log(frames[0])

    // Hier abonneren we ons op de card data en starten we een stream
    await session.subscribe("rie.vision.card.stream", on_card)
    await session.call("rie.vision.card.stream")
}
```

Gezichts detectie en volgen

Een andere geavanceerde functionaliteit wat we met camerabeelden kunnen doen is het detecteren en volgen van gezichten. Zo kunnen we op camerabeelden bepalen waar een gezicht zich bevindt in de ruimte en daar het hoofd van de robot naar toe bewegen. Let hier trouwens wel bij op dat het hier gaat om gezichts detectie en niet gezichtsherkenning. De robot weet dus niet wie er voor hem staat, hij weet alleen dat er iemand voor hem staat.

RPC	**Geeft terug**
rie.vision.face.info: vraag de informatie op van de Gezichts detectie module.	Een dictionary met daarin informatie over de Gezichts detectie module.
rie.vision.face.stream: opent een stream op de topic `rie.vision.face.stream` waarop de hoeveelheid gezichten die de robot 'ziet' gepubliceerd wordt. Belangrijk hierbij is dat je eerst de subscribe doet voordat je de stream aanroept. Argument: - ubi: (List<String>, default=[ubi]): een lijst van ubi's waarvan je de data wilt hebben. Meestal is het voldoende om deze parameter weg te laten.	None
rie.vision.face.read: luister voor een bepaalde tijd naar alle data die verzameld wordt voor deze module Argumenten: - ubi: (List<String>, default=[ubi]): een lijst van ubi's. Meestal is het voldoende om deze parameter weg te laten; - time: (Double, default=0.0): de tijd hoelang je de read wilt laten lopen in milliseconde, bij time=0.0 wordt een enkel frame teruggegeven.	De data die verzameld is van de Gezichts detectie module over de vooraf ingestelde tijd (time).
rie.vision.face.close: sluit een stream nadat je deze data niet meer nodig hebt. Argument: - ubi: (List<String>, default=[ubi]): een lijst met ubi's waar je niet meer subscribed op wilt zijn.	None
rie.vision.face.find: Laat de robot zijn hoofd beweging om te kijken of de robot een gezicht ziet.	None

Argument: - active: (Bool, default=True): indien True, dan gaat de robot actief op zoek naar een gezicht. Dit houdt in dat de robot zijn hoofd beweegt totdat die een gezicht heeft gevonden. Bij active=False, zal de robot alleen voor zich uit kijken totdat die een gezicht heeft gevonden.	
rie.vision.face.track: zodra we een gezicht hebben gevonden met rie.vision.find, kunnen we het gezicht volgen (=tracken) door deze RPC aan te roepen. Deze RPC blijft het gezicht volgen totdat die hem niet meer ziet. Argumenten: - track_time: (Int, default=0): de tijd in milliseconde dat je het gezicht wilt volgen. Als je de default waarde van 0 gebruikt, dan blijft de RPC het gezicht volgen totdat die hem niet meer ziet. - lost_time: (Int, default=4000): de tijd in milliseconde dat we een gezicht mogen missen voordat we bepalen dat we geen gezicht meer zien.	None
rie.vision.face.stop: stop de `rie.vision.face.find` en `rie.vision.face.track` RPC's	None

Een voorbeeld van hoe je rie.vision.face.* kan gebruiken in Python:

```
from twisted.internet.defer import inlineCallbacks

def on_face(frame):
  # Deze functie wordt elke keer aangeroepen wanneer het aantal
  # gezichten dat de robot ziet verandert
  print(frame)

@inlineCallbacks
def main(session, details):
  # Wacht totdat we op zijn minst 1 gezicht zien
  frames = yield session.call("rie.vision.face.read")
  print(frames[0])

  # Hier abonneren we ons op de face data en starten we een stream
  yield session.subscribe(on_face, "rie.vision.face.stream")
  yield session.call("rie.vision.face.stream")
```

En in JavaScript:

```
function on_face(event) {
  // Deze functie wordt elke keer aangeroepen wanneer het
  // aantal gezichten dat de robot ziet verandert
  frame = event[0]
  console.log(frame)
}

async function main(session, event) {
  // Wacht totdat we op zijn minst 1 gezicht zien
  let frames = await session.call("rie.vision.face.read")
  console.log(frames[0])

  // Hier abonneren we op de face data en starten we een stream
  await session.subscribe("rie.vision.face.stream", on_face)
  await session.call("rie.vision.face.stream")
}
```

Cloud Modules

De cloud modules is de verzamelnaam voor de hierboven genoemde geavanceerde functionaliteiten. Het zorgt ervoor dat zodra de robot wakker wordt, de dialoog module en vision module wordt opgestart. Mocht je zelf een programma maken die direct moet draaien wanneer de robot opstart, dan is het goed om te checken of de cloud modules klaar zijn voor gebruik. Dit doe je door te controleren of de volgende RPC al bestaat:

RPC	Geeft terug
rie.cloud_modules.ready: deze RPC wordt geregistreerd nadat alle andere RPC's van de cloud module geregistreerd zijn.	None

Een voorbeeld van hoe je in je programma kan wachten totdat de cloud modules volledig klaar zijn voor gebruik. In Python:

```
from autobahn.twisted.util import sleep
from twisted.internet.defer import inlineCallbacks
from autobahn.wamp.exception import ApplicationError

@inlineCallbacks
def main(session, details):
    print("Waiting for cloud modules")
    ready = False
    while not ready:
        try:
            yield session.call("rie.cloud_modules.ready")
            ready = True
        except ApplicationError:
            print("Cloud modules are not ready yet")
            yield sleep(0.25)

    print("Cloud modules are ready to go!")
```

En in JavaScript:

```
// Gebruik await slaap(s) om s seconden te wachten
let slaap = s=>new Promise(r=>setTimeout(r, s*1000))

async function main (session, details) {
    console.log("Waiting for cloud modules")
    let ready = false
    while (!ready) {
        try {
               await session.call("rie.cloud_modules.ready")
               ready = true
        } catch (error) {
               console.log("Cloud modules are not ready yet")
               await slaap(0.25)
        }
    }
    console.log("Cloud modules are ready to go!")
}
```

FAQ

Welke programmeertaal raden jullie aan?
Als dit de eerste keer is dat je aan de slag gaat met programmeren, dan raden wij je aan om te starten met Python of JavaScript. Beide talen zijn gemakkelijke instap talen die je snel onder knie kunt krijgen. De talen lijken in zekere zin ook best wel op elkaar, maar er zitten wel wat verschillen in hoe je de code draait en hoe je bijvoorbeeld functies definieert. Om je een beter beeld te geven van beide talen, hebben we hieronder een vergelijkings tabel gemaakt met de pluspunten van elke taal:

Python**:**

+ Python code staat er bekend om dat het heel goed leesbaar is;
+ Python is zeer geliefd onder de programmeurs;
+ De taal is bedacht door een Nederlander :-)

JavaScript**:**

+ JavaScript is sneller op te zetten dan Python;
+ JavaScript is beginner vriendelijk, al kan het soms lastig zijn om fouten (bugs) uit je code te halen;
+ Er zijn meer programmeurs wereldwijd die in JavaScript programmeren dan in Python.

Mocht je ervaring hebben in een andere taal, dan kan je waarschijnlijk ook in die taal aan de slag met programmeren. Waar je op moet letten is dat jouw taal ook een WAMP connectie ondersteund. Voor bijvoorbeeld C++ kun je gebruik maken van Autobahn-cpp.

Als ik mijn programma probeer te starten krijg ik steeds een No Such Realm error
Controleer of de realm overeenkomt met de realm van de robot. Dit doe je door naar de robots pagina te gaan en daar op de hamburger menu te klikken van de robot waar je mee aan de slag wilt gaan. Vervolgens klik je daar op het klembord en staat onderin het scherm dat nu verschijnt de realm van de robot. Controleer of je de volledige realm naam hebt gekopieerd. Een voorbeeld van een volledige realm naam:
`rie.5e1312363dbf49eed032e123`

Changelog

De software voor de robots wordt steeds weer verbeterd waardoor ze nog beter de interactie aan kunnen gaan. Om dit te kunnen doen, zullen er in de toekomst nieuwe RPC's worden toegevoegd aan het platform. Maar het kan ook voorkomen dat argumenten van bestaande RPC's worden veranderd of dat sommige RPC's niet meer ondersteund worden. Met dit hoofdstuk houden we je op de hoogte van de veranderingen die wij doorvoeren.

V1.0

- Eerste versie van documentatie ROM en RIE specificaties.

V1.1

- Code voorbeeld van Keywords is geüpdatet. De subscribe verwees naar een niet bestaande functie.

V1.2

- JavaScript voorbeelden zijn toegevoegd aan de ROM en RIE api hoofdstukken;
- In de bijlage is een lijst met standaard geïnstalleerde gedragingen toegevoegd.

V1.3

- Voorbeeldcode waarbij de Python in een loop terecht kwam is opgelost.

V1.4

- De RPC's rie.dialogue.config.speed en rie.dialogue.config.pitch zijn toegevoegd;
- Voorbeeldcode voor het aanpassen van de oogkleuren bevatte een bug, deze is verholpen.

V1.5

- Beschrijving voor de STT module is toegevoegd;

Bijlage

A. Session voorbeelden
B. Actuator modes
C. Standaard gedragingen

A. Session voorbeelden

Om het session object nog beter te begrijpen hoe het werkt en hoe jij het in je programma kunt verwerken, hebben we hieronder twee voorbeelden toegevoegd. Deze voorbeelden laten zien hoe je in Python en JavaScript de verschillende acties die je met het session object kunt doen kan uitvoeren. Je zult zien dat er kleine verschillen zijn tussen Python en JavaScript die belangrijk zijn om te weten mocht je met beide talen aan de slag gaan.

De voorbeelden hieronder starten met het aanmelden van de `ridk.voorbeeld.hallo` topic, zie stap `1)`. Vervolgens publiceren we in de volgende stap, stap 2), een bericht op de topic `ridk.voorbeeld.hallo`. Doordat we in stap 1 de functie `onevent` gebonden hebben aan het topic `ridk.voorbeeld.hallo`, wordt deze functie nu aangeroepen omdat we data publiceren op dit topic.

In de volgende stap, stap 3), registreren we een nieuwe RPC genaamd `ridk.voorbeeld.optellen` en elke keer als iemand deze RPC aanroept voeren wij de code in de functie `optellen` uit. Dit zul je ook zien als we naar stap 4) gaan, daar roepen we namelijk de RPC `ridk.voorbeeld.optellen` aan en vragen we aan de RPC om de getallen 2 en 3 bij elkaar op te tellen. We wachten op een antwoord door gebruik te maken van `yield` en `await` en we slaan het antwoord op in de variabele `antwoord`. Aan het einde van het programma printen we het antwoord om er zeker van te zijn dat we het juiste antwoord krijgen.

Python Code:

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks
from autobahn.wamp.types import PublishOptions

@inlineCallbacks
def main(session, details):

    # 1. Abonneer je op het topic ridk.voorbeeld.hallo
    def onevent(bericht):
        print("Ontvangen bericht: " + bericht)

    yield session.subscribe(onevent, "ridk.voorbeeld.hallo")

    # 2. Publiceer text op het topic ridk.voorbeeld.hallo
    yield session.publish("ridk.voorbeeld.hallo",
            "Dit is een publish bericht",
            options=PublishOptions(exclude_me=False))

    # 3. Registreer een optellen RPC
    def optellen(getallen):
        return getallen[0] + getallen[1]

    yield session.register(optellen, "ridk.voorbeeld.optellen")

    # 4. Roep de geregistreerde optellen RPC aan
    antwoord = yield session.call("ridk.voorbeeld.optellen",
                        getallen=[2,3])

    print("Het antwoord is: " + str(antwoord))

    session.leave()

# Create wamp conn
wamp = Component(
    transports=[{
        "url": "ws://wampdev.robotsindeklas.nl",
        "serializers": ["msgpack"],
        "max_retries": 0
    }],
    realm="rie.5e1312363dbf49eed032e123",
)
wamp.on_join(main)

if __name__ == "__main__":
    run([wamp])
```

JavaScript code:

```
let wamp = new autobahn.Connection({
    url: "wss://wamp.robotsindeklas.nl",
    realm: "rie.5e1312363dbf49eed032e123",
    protocols: ["wamp.2.msgpack"]
})

wamp.onopen = async function (session) {

   // 1. Abonneer je op het topic ridk.voorbeeld.hallo
   function onevent(berichten) {
     console.log("Ontvangen bericht: " + berichten[0])
   }
   session.subscribe("ridk.voorbeeld.hallo", onevent)

   // 2. Publiceer text op het topic ridk.voorbeeld.hallo
   session.publish("ridk.voorbeeld.hallo",
                   ["Dit is een publish bericht"],
                   {},
                   {exclude_me: false})

   // 3. Registreer een optellen RPC
   function optellen(argumenten) {
     return argumenten[0] + argumenten[1]
   }
   await session.register("ridk.voorbeeld.optellen", optellen)

   // 4. Roep de geregistreerde optellen RPC aan
   let antwoord = await session.call("ridk.voorbeeld.optellen",
                             [2, 3])

   console.log("Het antwoord is: " + antwoord)

   session.leave()
}

wamp.open()
```

B. Actuator modes

Bij het aansturen van actuator modalitieten mag je frames versturen die niet alle ubi's bevatten of zelfs lege frames. Dit is handig als je bijvoorbeeld het linker en rechteroog onafhankelijk en op verschillende momenten van kleur wilt veranderen. Of als je door de proprio modaliteit opgenomen data weer wilt afspelen.

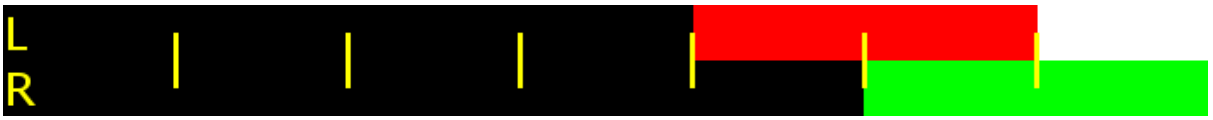
Om de actuator te vertellen wat deze met dit soort frames moet doen, kun je een mode mee geven. We geven hier het voorbeeld met de ogen en leggen dit uit met de drie modes. De data die we opsturen ziet er als volgt uit:

Python code:

```
yield session.call("rom.actuator.motor.write",
 frames=[{"time": 1000, "data": {"body.head.eyes.left":  [0,0,0]}}, //off
         {"time": 2000, "data": {"body.head.eyes.right": [0,0,0]}}, //off
         {"time": 3000, "data": {}},
         {"time": 4000, "data": {"body.head.eyes.left":  [255,0,0]}}, //red
         {"time": 5000, "data": {"body.head.eyes.right": [0,255,0]}}, //green
         {"time": 6000, "data": {"body.head.eyes.left":  [255,255,255]}}],//white
   mode="last",
   force=True)
```

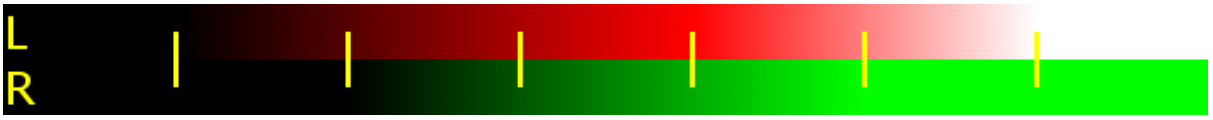
- **mode=none**:

Ga pas op het exacte moment zo snel mogelijk naar dit frame toe en vul lege frames niet in. Let hierbij op dat dit tot instabiel gedrag zal leiden bij de motor modaliteit.



- **mode=linear**:

Lege frames worden opgevuld met de waardes tussen de ingevulde frames. Tussen frames zal de overgang geleidelijk zijn



- **mode=last**:

Vul lege frames in met de laatst opgegeven waarde, met geleidelijke overgangen tussen frames. Deze mode werkt het beste voor data opgenomen met de proprio modaliteit.

C. Standaard gedragingen

Lijstje van standaard geïnstalleerde gedragingen:

BlocklyRightHandOpen	BlocklyStand	BlocklyBothHandsOpen
BlocklyLeftHandClosed	BlocklyRightHandClosed	BlocklyBothHandsClosed
BlocklyWaveRightArm	BlocklyMoveForward	BlocklyTurnLeft
BlocklyTurnRight	BlocklyTurnAround	BlocklyTouchHead
BlocklyTouchShoulders	BlocklyTouchKnees	BlocklyTouchToes
BlocklyDuck	BlocklyArmsForward	BlocklyLeftArmForward
BlocklyRightArmForward	BlocklyArmsUp	BlocklyLeftArmUp
BlocklyRightArmUp	BlocklyLeftArmSide	BlocklyRightArmSide
BlocklyArmsBackward	BlocklySprinkle	BlocklySitDown
BlocklyDab	BlocklyBow	BlocklyFreeWalk
BlocklyTaiChiChuan	BlocklyDabLong	BlocklySneeze
BlocklyHappyBirthday	BlocklyGangnamStyle	BlocklyMacarena
BlocklyHappy	BlocklyLeftHandOpen	BlocklySafeStand
BlocklySaxophone	BlocklyCrouch	BlocklyShrug
BlocklyYouAndMe	BlocklyInviteRight	BlocklyLookAtChild
BlocklyLookingUp	BlocklyFearUp	BlocklyApplause
BlocklyDiscoDance	BlocklyVacuum	BlocklyStarWars
BlocklyPride	BlocklyFollowMe	