NAO handleiding voor Zoeken, Sturen en Bewegen

Simon Pauw, Amsterdam, juni 2017

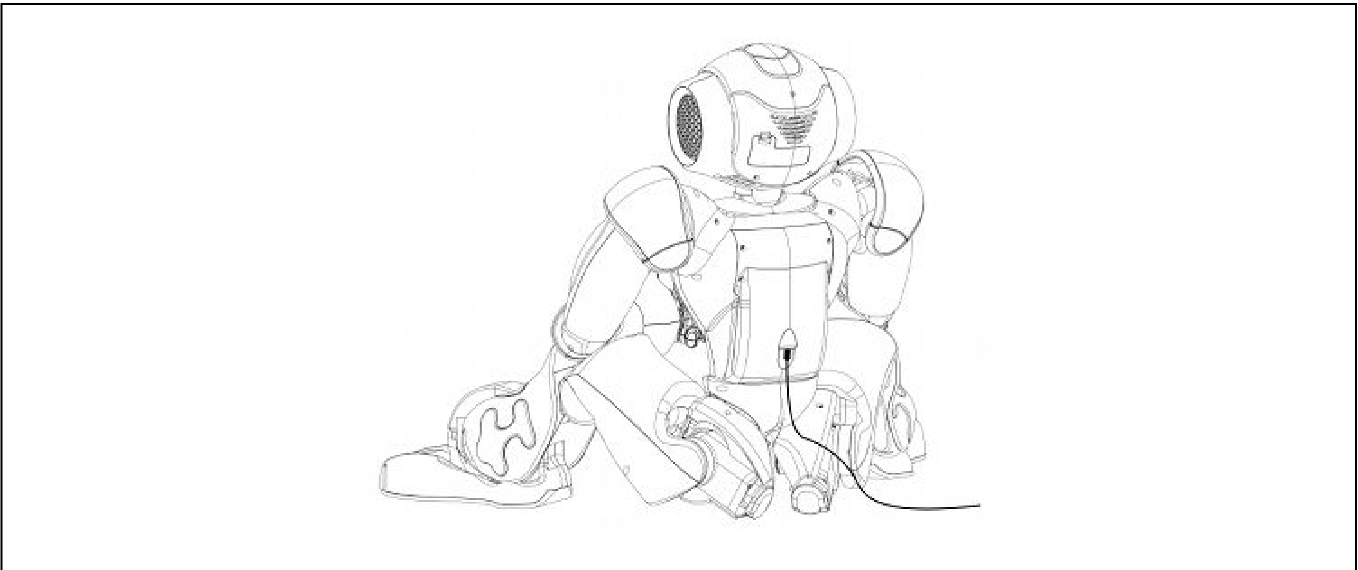
1. Voorbereiding

Download software:
Ga naar https://developer.softbankrobotics.com/us-en/downloads/nao-v5-v4 en download Choreographe 2.1.4 voor jouw OS.

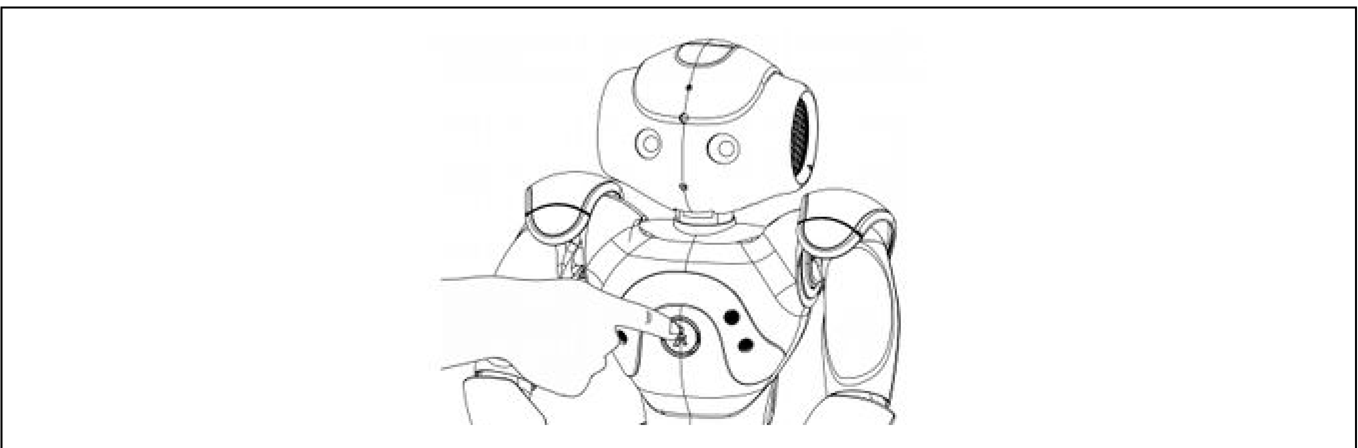
Robot klaarzetten:

Waarschuwing: De Nao robots zijn kwetsbaar. Behandel ze daarom met liefde.

Om de robot op te laden, hoef je alleen de lader in de rug te steken. Het led-lampje van de lader kleurt dan rood.



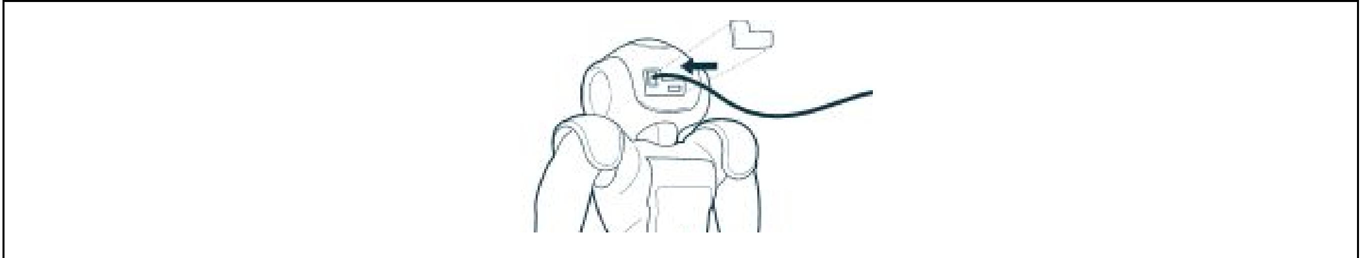
Zet de robot in de stabiele houding. De positie in het plaatje hierboven is een goede starthouding. Zet de robot niet op een tafel maar op de grond.



Zet de NAO aan met de knop op z'n borst.

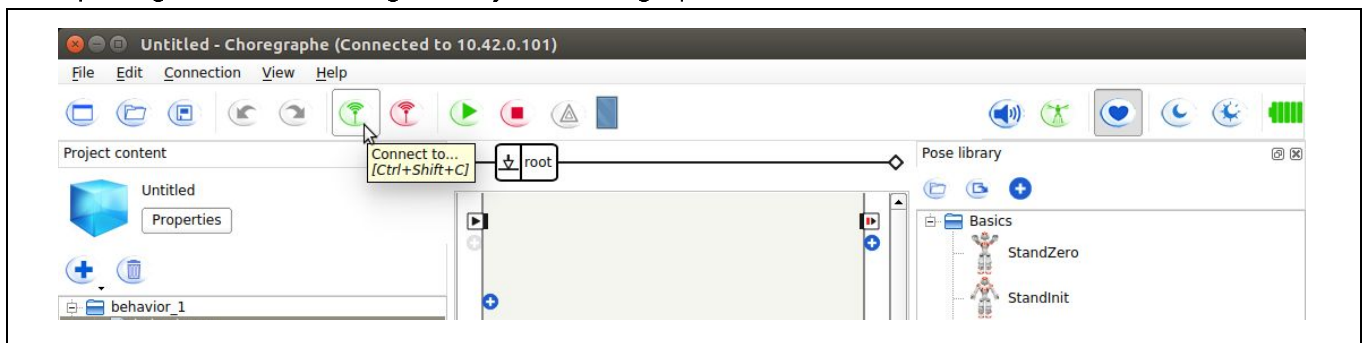
2. Verbinden met de NAO

Je kan verbinding maken met de NAO door hem direct met een UTP kabel op je laptop aan te sluiten.

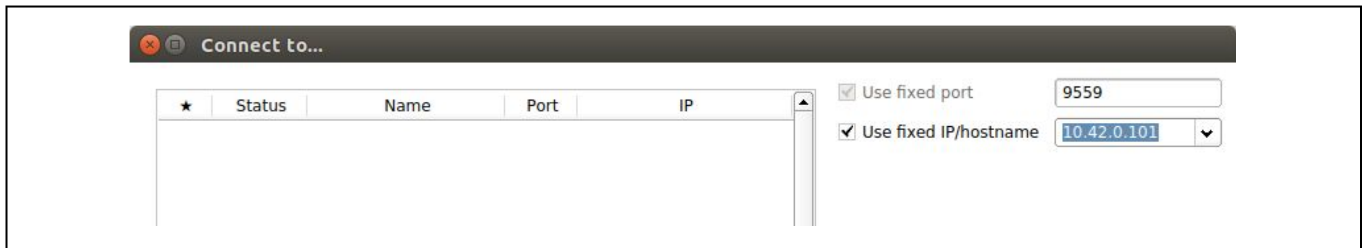


De NAO's in het robolab zijn voorzien van een vast IP adres in de range 10.42.0.*. Stel handmatig het ip-adres 10.42.0.1 en subnetmask 255.255.255.0 op je laptop in. Mocht je niet weten hoe dat moet, zie appendix A voor de instructies voor Ubuntu.

Verbinden met de NAO:
Klik op het groende verbindingsicoontje in Choregraphe**.**



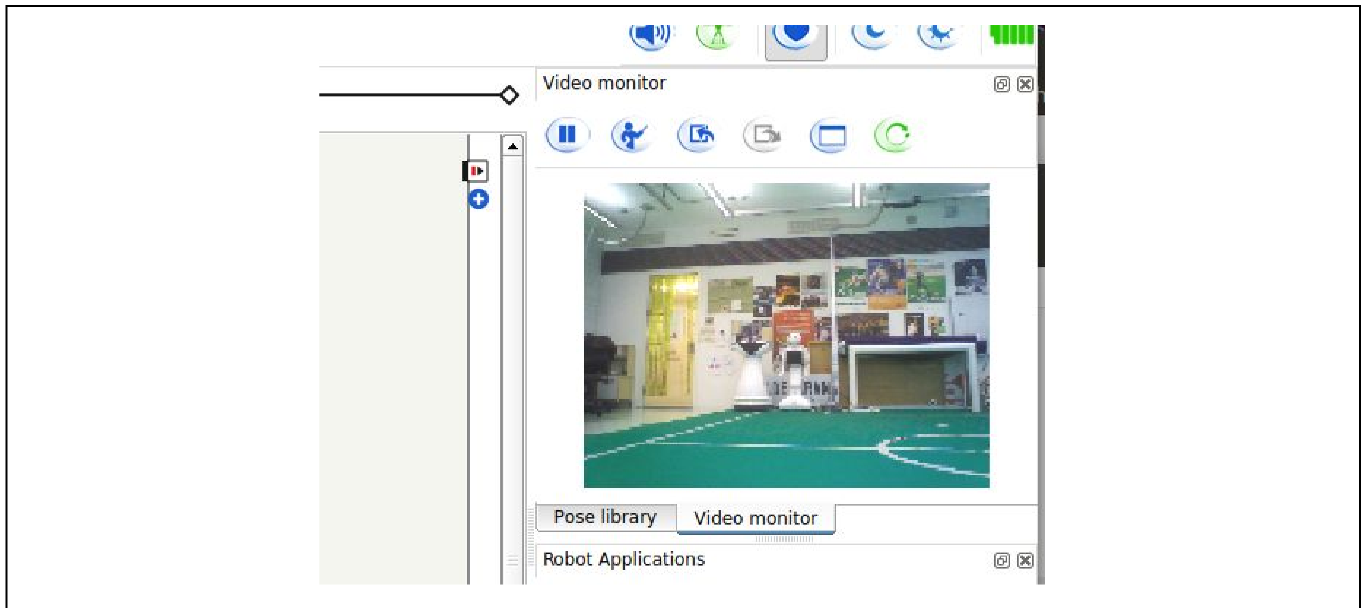
Voer handmatig het IP-adres van de NAO in.



NB: In sommige gevallen heeft de NAO een label met het IP-adres. Mocht dit niet zo zijn kun je met het volgende linux-commando het IP-adres achterhalen:

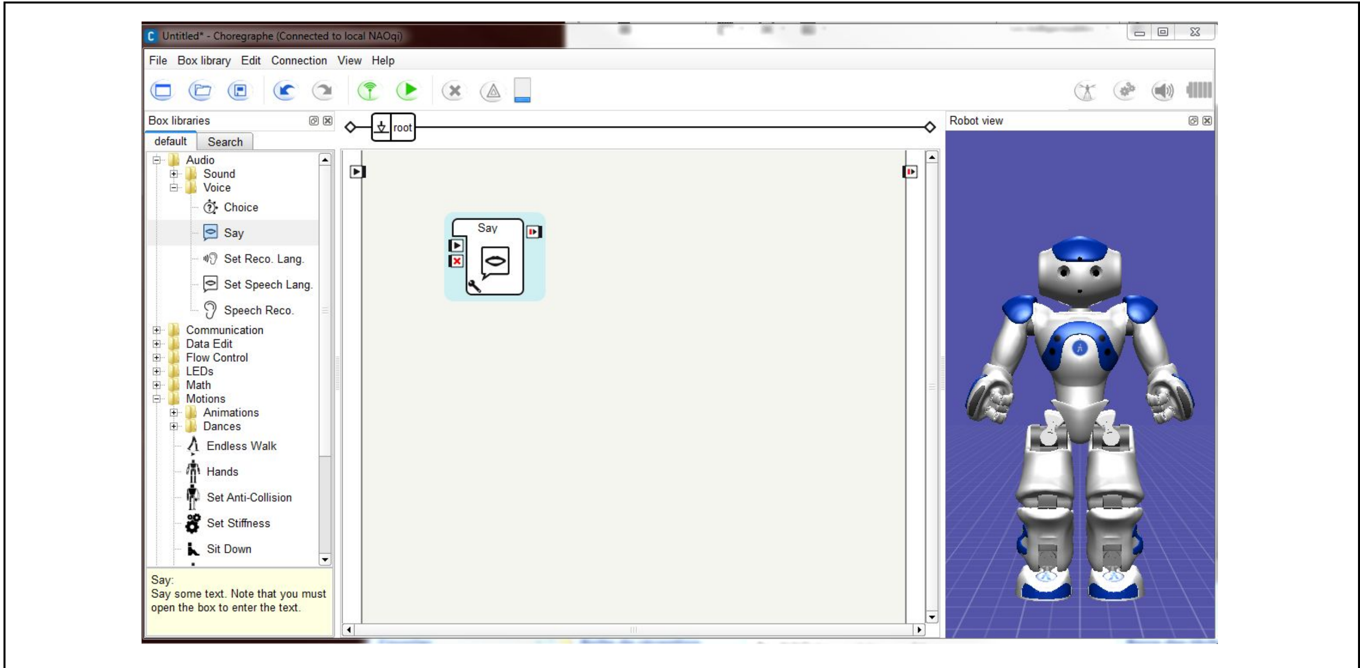
```
nmap -sn 10.42.0.*
```

Als alles goed is gegaan kun je nu in via video monitor met de NAO meekijken.

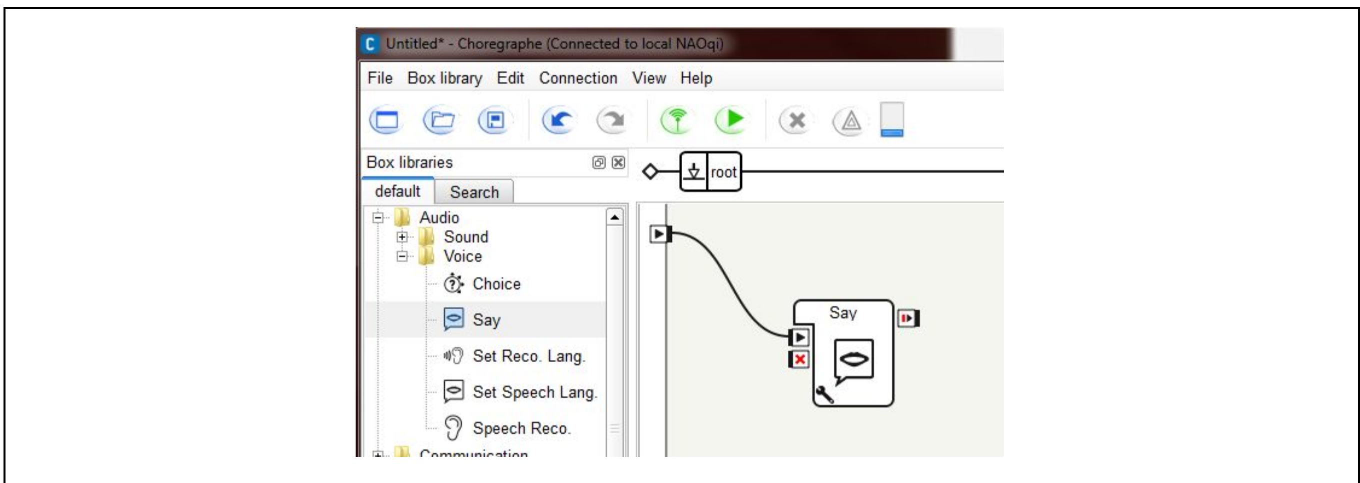


3. Choregraphe interface

De werking van het programma Choregraphe spreekt redelijk voor zich: Aan de linkerkant heb je een library met standaard commando's. Je kan deze commando's gebruiken door ze naar het middelste paneel te slepen.



Om ervoor te zorgen dat de robot ook daadwerkelijk het commando uitvoert moet je nog een *workflow* definiëren. Dit kun je doen door met je muis een lijn te slepen van ▸ aan de linkerkant van het paneel naar ▸ aan de linkerkant van het commando.

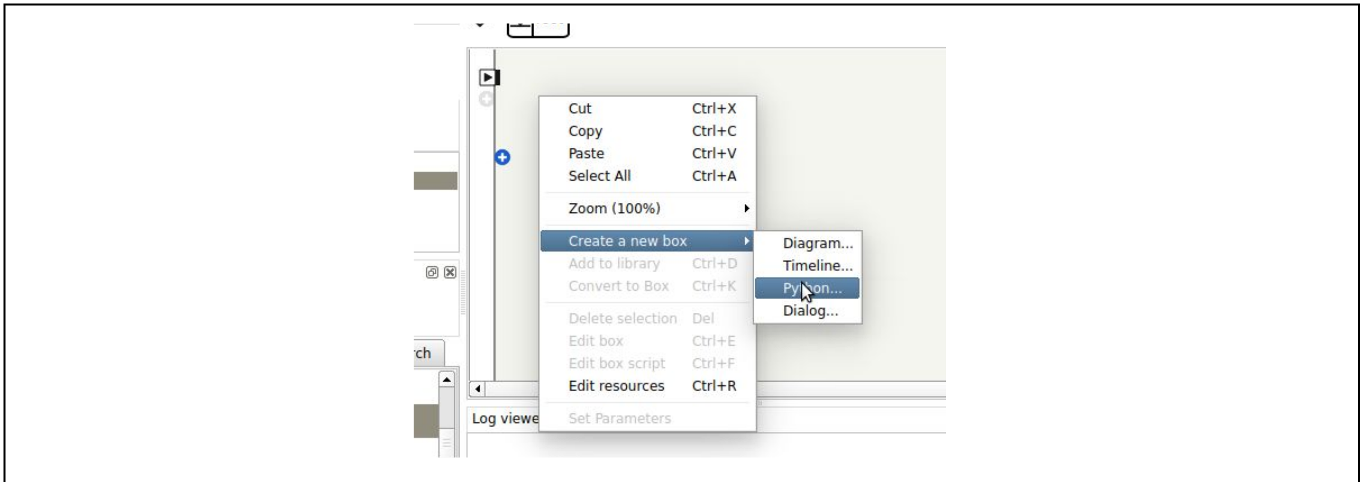


Door op de groene play-knop te drukken voert de NAO het programma uit.

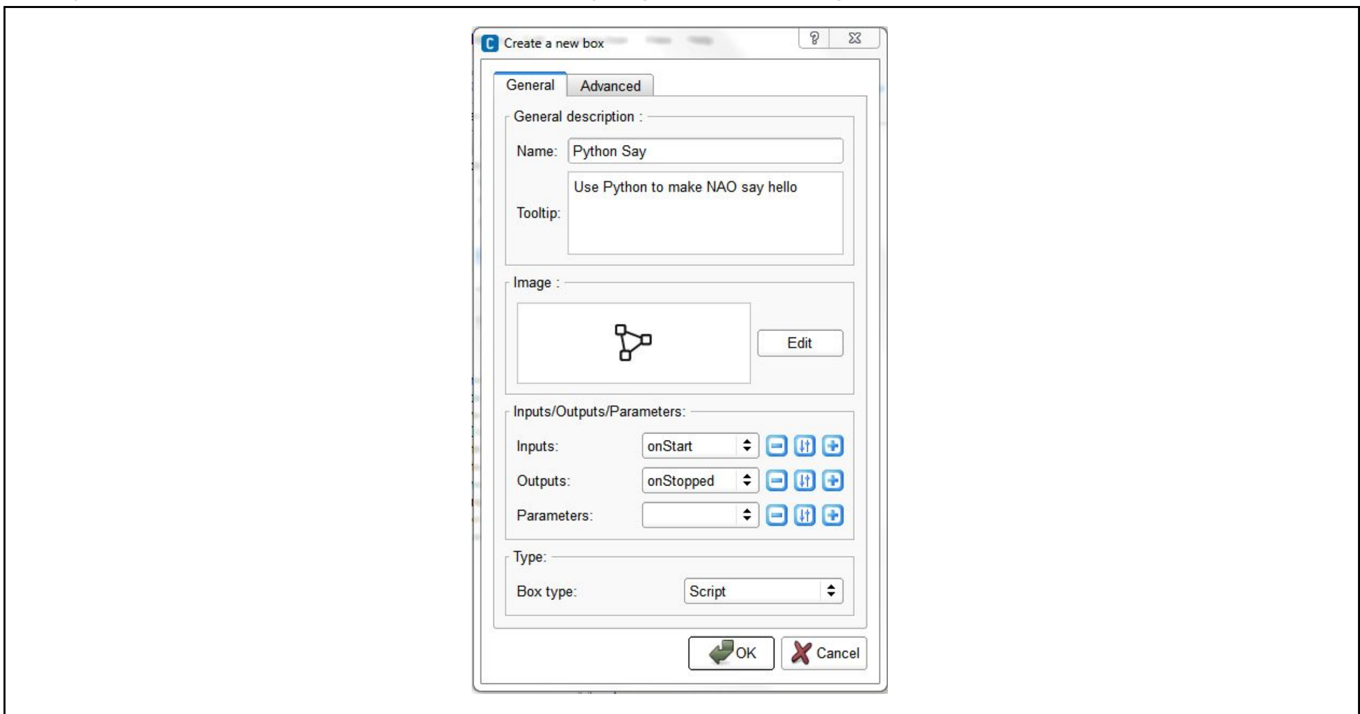
4. Een eigen choregraphe *box* maken

Alleen maar standaard gedragingen aan elkaar knopen blijft natuurlijk niet lang interessant. Gelukkig kunnen we ook zelf commando's maken in Choregraphe.

Klik met je rechtermuisknop op het werkveld en kies, *create new box* -> *python*



Geef je nieuwe box een naam en een beschrijving en klik vervolgen *OK*.



Nu zie je jouw *box* in het werkveld. Dubbelklik op de *box* om het bijbehorende script te editen.

Als het goed is zie je de volgende code:

```
class MyClass(GeneratedClass):
	def __init__(self):
		GeneratedClass.__init__(self)
		pass

	def onLoad(self):
		#~puts code for box initialization here
		pass

	def onUnload(self):
		#~puts code for box cleanup here
		pass

	def onInput_onStart(self):
		#~self.onStopped() #~ activate output of the box
		pass

	def onInput_onStop(self):
		self.onUnload() #~it is recommended to call on....
		pass
```

5. Forward kinematics (joint control)

Met de *join control*-API kun je de hoeken van de elke joint in de NAO besturen (*forward kinematics*). (Zie http://doc.aldebaran.com/1-14/naoqi/motion/control-joint.html voor de voleldige API.)

Kinematic chain:

Namen van de kinematische ketens en individuele joints:

Body	Head	LArm	LLeg	RLeg	RArm
	HeadYaw	LShoulderPitch	LHipYawPitch	RHipYawPitch1	RShoulderPitch
	HeadPitch	LShoulderRoll	LHipRoll	RHipRoll	RShoulderRoll
		LElbowYaw	LHipPitch	RHipPitch	RElbowYaw
		LElbowRoll	LKneePitch	RKneePitch	RElbowRoll
		LWristYaw2	LAnklePitch	RAnklePitch	RWristYaw2
		LHand2	RAnkleRoll	LAnkleRoll	RHand2

Proxy:

De NAO-API (Naoqi) werkt met proxies voor het besturen van de robot. Voor *joint control* gebruik je de *ALMotion*-proxy:

```
motionProxy = ALProxy("ALMotion")
```

Stiffness:

Als de motoren van de NAO aanstaan raken ze snel oververhit. Zet de motoren dus alleen aan als het nodig is, en vergeet ze erna niet uit te zetten!

Met de volgende opdracht zet je de de motoren van de *Head chain* (*HeadYaw, HeadPitch*) aan:

```
motionProxy.setStiffnesses("Head", 1.0)
```

En zo zet je ze weer uit:

```
motionProxy.setStiffnesses("Head", 0.0)
```

Je kunt de motoren ook geleidelijk aan-/uitzetten:

```
names          = ["HeadYaw", "HeadPitch"]   # identical to "Head"
stiffness      = 0.0
time           = 1.0
motionProxy.stiffnessInterpolation(names, stiffness, time)
```

Met het volgende commando zorg je ervoor dat de NAO naar een veilige positie gaat en vervolgens alle motoren uitzet:

```
motionProxy.rest()
```

Set Angles (non blocking):

Met setAngles() kan je de individuele motoren van een chain in een specifieke stand zetten.

```
names                   = ["HeadYaw", "HeadPitch"]
angles                  = [0.3, 0]
fractionMaxSpeed        = 0.5
motionProxy.setAngles(names, angles, fractionMaxSpeed)
```

Angle Interpolation (blocking):

Met angleInterpolation() kan je de motoren geleidelijk tussen verschillende standen laten bewegen.

```
names           = ["HeadYaw", "HeadPitch"]
angles          = [0.3, 0]
times           = [0.5, 0.5]
isAbsolute      = True
motionProxy.angleInterpolation(names,angles,times,isAbsolute)
```

Hands:

De aansturing van de handen werkt anders dan de andere *joints*. Naoqi heeft hiervoor de functies openHand() en closeHand().

```
motionProxy.openHand("LHand")
motionProxy.closeHand("LHand")
```

Voorbeeld 1, setAngles():

```
class MyClass(GeneratedClass):
	def __init__(self):
		GeneratedClass.__init__(self)
		pass

	def onLoad(self):
		self.motionProxy = ALProxy("ALMotion")

	def onUnload(self):
		pass

	def onInput_onStart(self):
		names			= ["HeadYaw", "HeadPitch"]
		angles			= [[0.3, 0],[-0.3, 0]]
		fractionMaxSpeed	= 0.5

		self.motionProxy.setStiffnesses("Head", 1.0)
		for i in range(5):
			self.motionProxy.setAngles(names,angles[0],fractionMaxSpeed)
			time.sleep(0.5)
			self.motionProxy.setAngles(names,angles[1],fractionMaxSpeed)
			time.sleep(0.5)

		self.motionProxy.setStiffnesses("Head", 0.0)

	def onInput_onStop(self):
		self.onUnload()
```

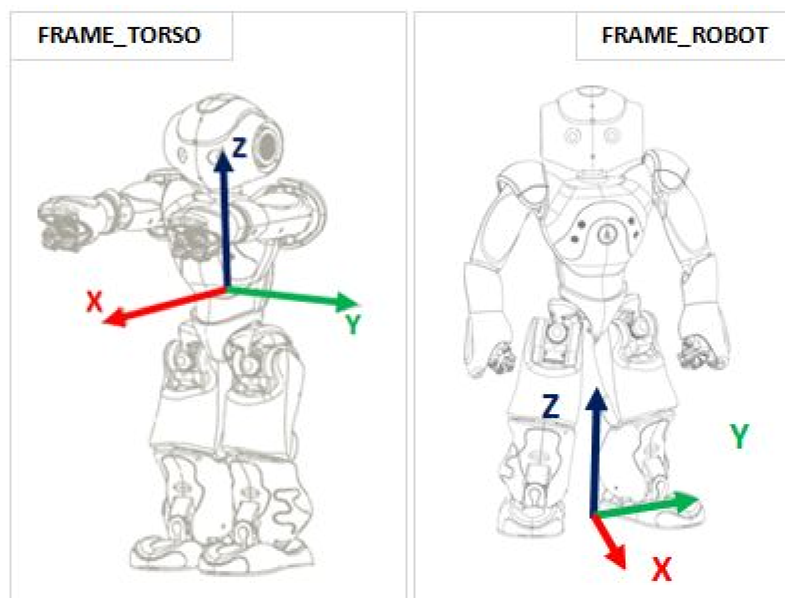
6. Inverse kinematics (cartesian control)

De NAO API heeft twee ingebouwde inverse kinematics solvers: De klassieke *effector chain solver* (voor het besturen van een kinematische keten zoals de rechterarm) en een gegeneraliseerde *whole body solver*. Hier volgt informatie over de klassieke *effector chain solver*. (Zie http://doc.aldebaran.com/1-14/naoqi/motion/control-cartesian.html voor de volledige API.)

Frames:
De API maakt gebruik van drie verschillende frames voor het beschrijven van transformaties:

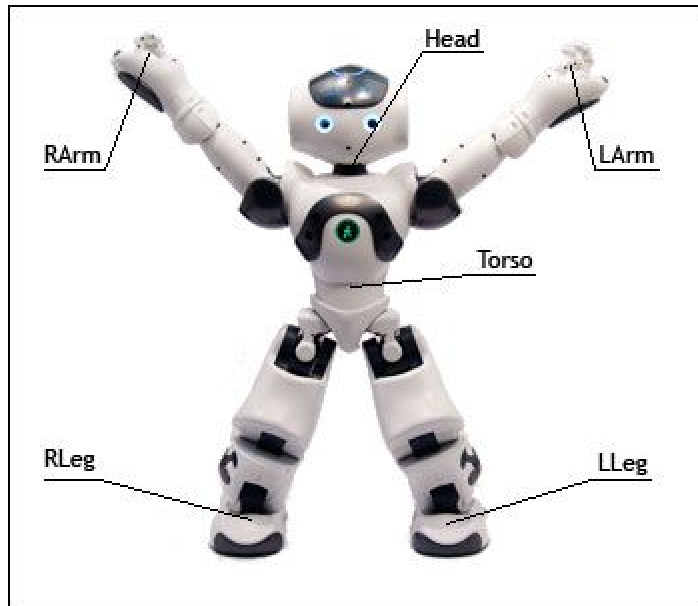
```
FRAME_TORSO
FRAME_ROBOT
FRAME_WORLD
```

De frames zijn als volgt gedefiniëerd:



FRAME_WORLD is bij het starten van het programma gelijk aan FRAME_ROBOT, maar als de robot beweegt blijft de FRAME_WORLD (zo goed als mogelijk) op de beginpositie staan.

Kinematic effectors:
Er zijn een aantal *effectors* gedefinieerd die je direct kan besturen met de *cartesian control* API:



Zie http://doc.aldebaran.com/2-1/family/robots/bodyparts.html voor meer info.

Axis mask:
De effectors worden beschreven aan de hand van 6 parameters ten opzichte van het gebruikte frame: de translatie (x,y,z) en de rotatie (wx, wy, wz). Je kan met een *axis mask* de gebruikte dimensies voor de inverse kinematics beperken. Voor, bijvoorbeeld, translaties gebruik je de volgende mask:

```
import almath
axisMask = almath.AXIS_MASK_X + almath.AXIS_MASK_Y + almath.AXIS_MASK_Z
```

Set position (non blocking):
Met de setPosition()-functie kun je een effector naar een bepaalde positie bewegen.

```
Effector             = "LArm"             # move LArm chain
Space                = motion.FRAME_TORSO # relative to TORSO
fractionMaxSpeed     = 0.5                # 50% speed
isAbsolute           = True               # positions are absolute (not deltas)

# [x,y,z,wx,wy,wz]
targetPositions      = [0.15, 0, 0.1, 0, 0, 0]

# move arm forward and up
self.motionProxy.setPosition(effector, space, targetPositions, fractionMaxSpeed, axisMask)
time.sleep(3)
```

Position interpolations (blocking):

Met positionInterpolations() kun je meerdere sequenties van bewegingen tegelijk animeren.

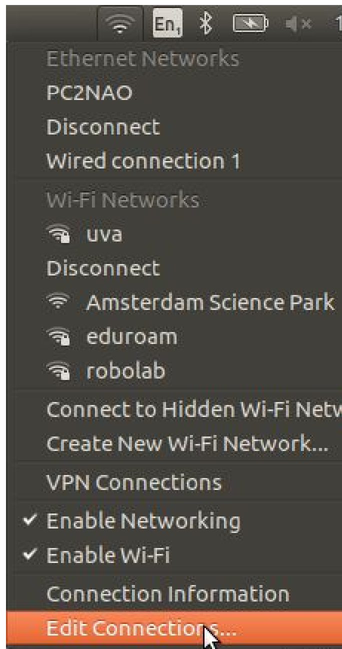
```
axisMask              = almath.AXIS_MASK_X + almath.AXIS_MASK_Y + almath.AXIS_MASK_Z
axisMasks             = [axisMask, axisMask]
Effectors             = ["LArm", "RArm"]            # move LArm chain
Space                 = motion.FRAME_TORSO          # relative to TORSO
times                 = [[1.0, 2.0],                # LArm times
                         [1.0, 2.0]]                # RArm times
isAbsolute            = True                        # postions are absolute (not deltas)

# [x,y,z,wx,wy,wz]
targetPositions       = [[[0.15,  0.06,   0, 0, 0, 0], [0.15,  0.06, 0.1, 0, 0, 0]],  # LArm
                         [[0.15, -0.06, 0.1, 0, 0, 0], [0.15, -0.06,   0, 0, 0, 0]]]  # RArm

motionProxy.positionInterpolations(effectors, space, targetPositions, axisMasks, times, isAbsolute)
```

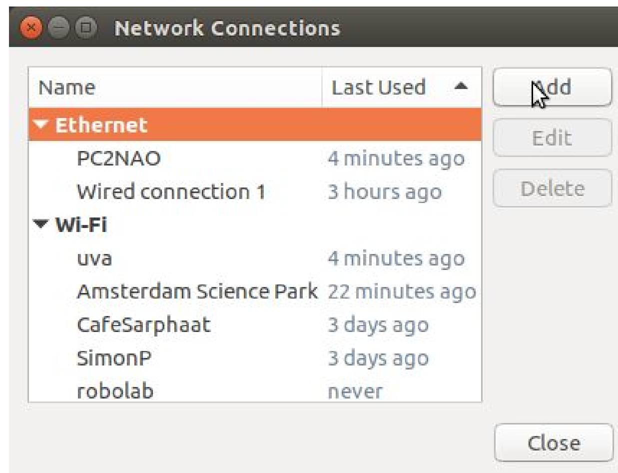
Voorbeeld 2, inverse kinematics:

```
import almath

class MyClass(GeneratedClass):
	def __init__(self):
		GeneratedClass.__init__(self)

	def onLoad(self):
		self.motionProxy = ALProxy("ALMotion")

	def onUnload(self):
		pass

	def onInput_onStart(self):
		# define parameters for inversed kinematics
		axisMask       = almath.AXIS_MASK_X + almath.AXIS_MASK_Y + almath.AXIS_MASK_Z
		axisMasks      = [axisMask, axisMask]       # only do translations (no rotations)
		effectors      = ["LArm", "RArm"]           # effector chain (for inverse kinem.)
		chain          = ["RArm", "LArm"]           # kinematic chain (for stiffness)
		space          = motion.FRAME_TORSO         # relative to TORSO
		times          = [[1.0, 2.0],               # LArm times
		                  [1.0, 2.0]]               # RArm times
		isAbsolute     = True                       # absolute positions

		#[x,y,z,wx,wy,wz]
		targetPositions = [[[0.15, 0.06, 0, 0, 0, 0], [0.15, 0.06, 0.1, 0, 0, 0]],
		                   [[0.15, -0.06, 0.1, 0, 0, 0], [0.15, -0.06, 0, 0, 0, 0]]]

		# turn motors on
		self.motionProxy.setStiffnesses(chain, 1.0)

		# move arms
		for i in range(2):
			self.motionProxy.positionInterpolations(effectors, space, targetPositions,
		axisMasks, times, isAbsolute)

		# release motors
		stiffnessLists        = 0.0
		timeLists             = 3.0
		self.motionProxy.stiffnessInterpolation(chain, stiffnessLists, timeLists)

	def onInput_onStop(self):
		self.onUnload()
		self.onStopped()
```

Appendix A. IP-adres instellen

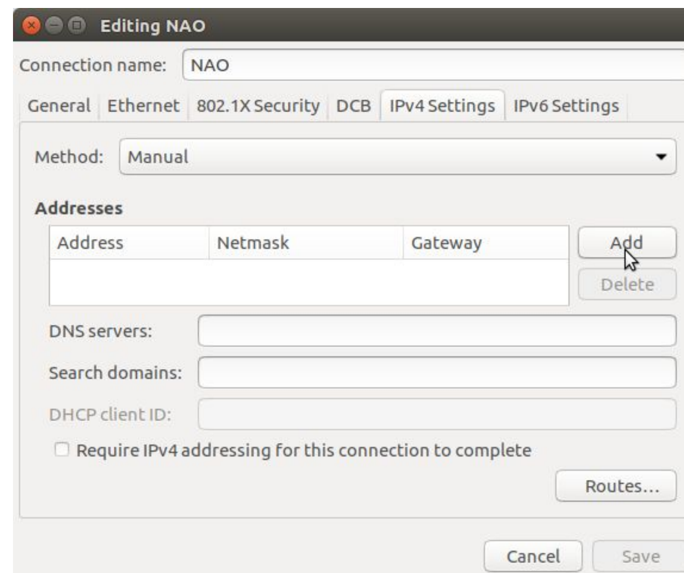
IP-adres instellen:



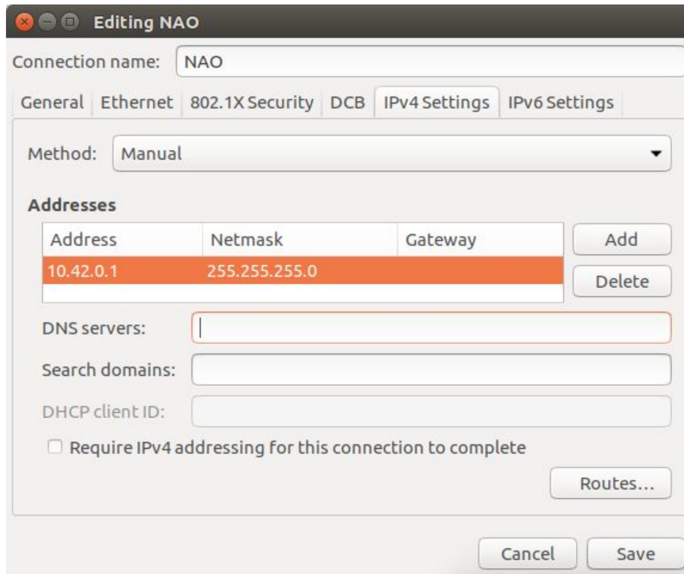


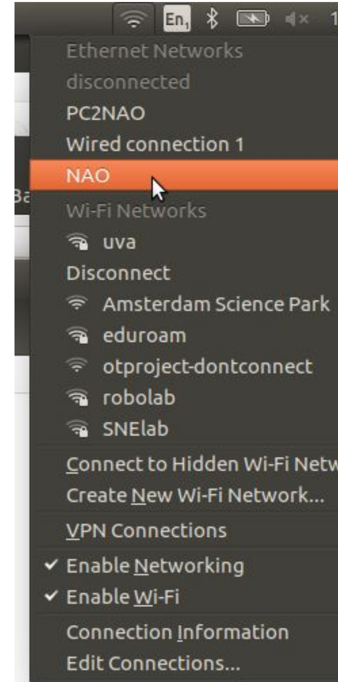
Klik op het netwerkicoontje en ga naar "Edit connections". Klik vervolgens op "Add".

Kies "Ethernet" en klik op "Create"



Kies een handige naam voor "Connection name", en ga vervolgens naar "IPv4 Settings". Kies bij *Method* voor *Manual*. Klik vervolgens op *Add*.





Voer de volgende gegevens in: *Address*: 10.42.0.1, *Netmask*: 255.255.255.0