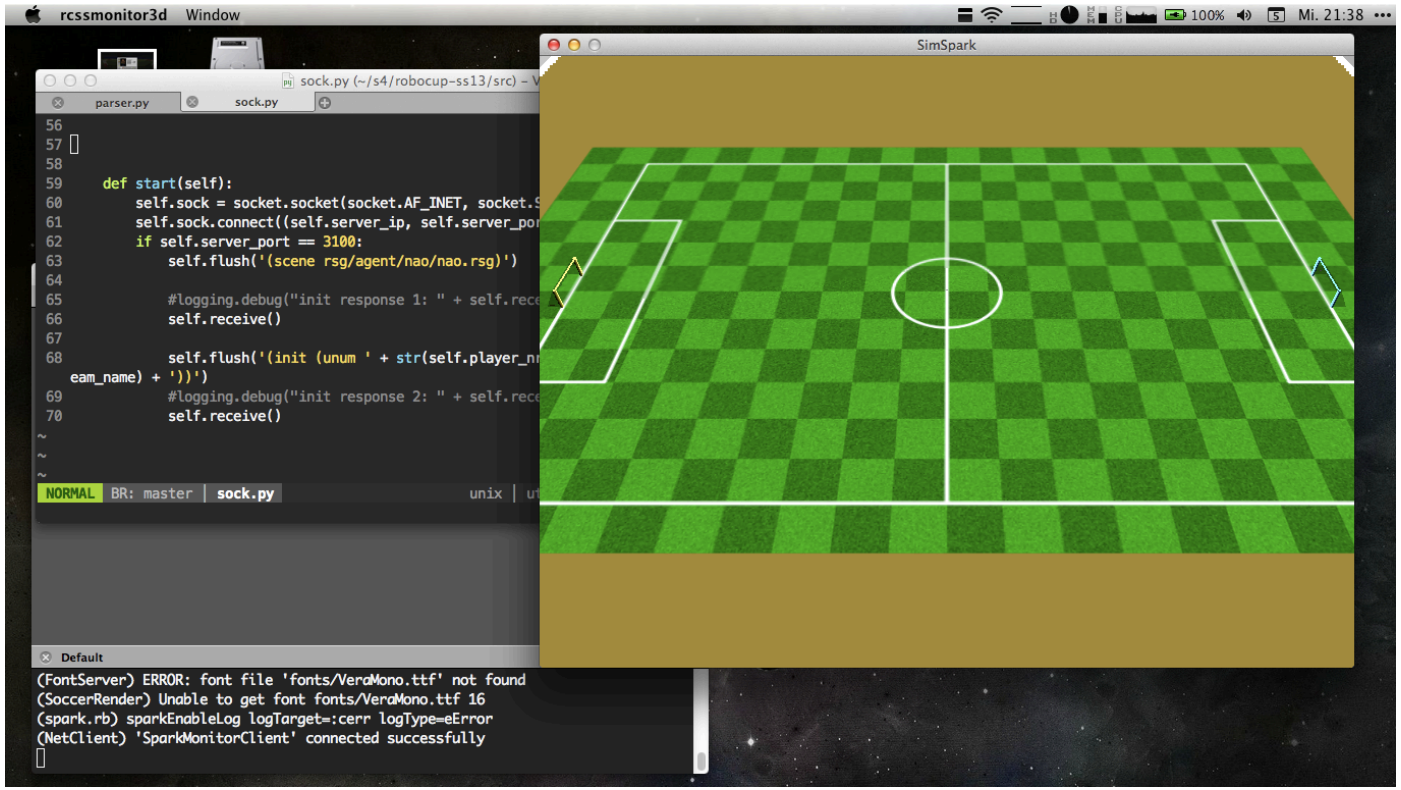
Präsentation für den zweiten Meilenstein

Robocup Team

9. Juni 2013

Einleitung

- NAO wird in einem Simulator ausgeführt



Einleitung

- Die Erkennung der Sensoren des NAO ist eingebaut.
- Wir verbinden uns via Socket zum NAO

Server: `localhost`
`Port: 3100`

Einleitung

- Der NAO sendet Sensordaten und empfängt Befehle als S-Expression:

```
(See (G2R (pol 17.55 -3.33 4.31)) (G1R (pol 17.52 3.27 4.07)) (F1R (pol 18.52 18.94
1.54)) (F2R (pol 18.52 -18.91 1.52)) (B (pol 8.51 -0.21 -0.17)) (P (team teamRed) (id 1)
(head (pol 16.98 -0.21 3.19)) (rlowerarm (pol 16.83 -0.06 2.80)) (rfoot (pol 17.00 0.29
1.68)) ) (L (pol 12.97 -37.56 -2.24) (pol 13.32 -32.98 -2.20)) )
```

- Wie nutzen wir diese Daten?

Parser

- Endlosschleife wartet auf Empfang neuer Daten

Parser

- Endlosschleife wartet auf Empfang neuer Daten
- Ein regulärer Ausdruck zergliedert rekursiv die S-Expression

```
term_regex = r'''(?mx)
    \s*(?:
        (?P<brackl>\()|
        (?P<brackr>\))|
        (?P<num>\-?\d+\.\d+e\-?\d+|\-?\d+\.\d+|\-?\d+)\|
        (?P<sq>"[^"]*")|
        (?P<s>\S+)
       )'''
```

Parser

- Endlosschleife wartet auf Empfang neuer Daten
- Ein regulärer Ausdruck zergliedert rekursiv die S-Expression

```
term_regex = r'''(?mx)
    \s*(?:
        (?P<brackl>\()|
        (?P<brackr>\))|
        (?P<num>\-?\d+\.\d+e\-?\d+|\-?\d+\.\d+|\-?\d+)\|
        (?P<sq>"[^"]*")|
        (?P<s>\S+)
       )'''
```

- Die entstandene Liste wird anschließend ins Weltmodell des NAO eingepflegt

Probleme

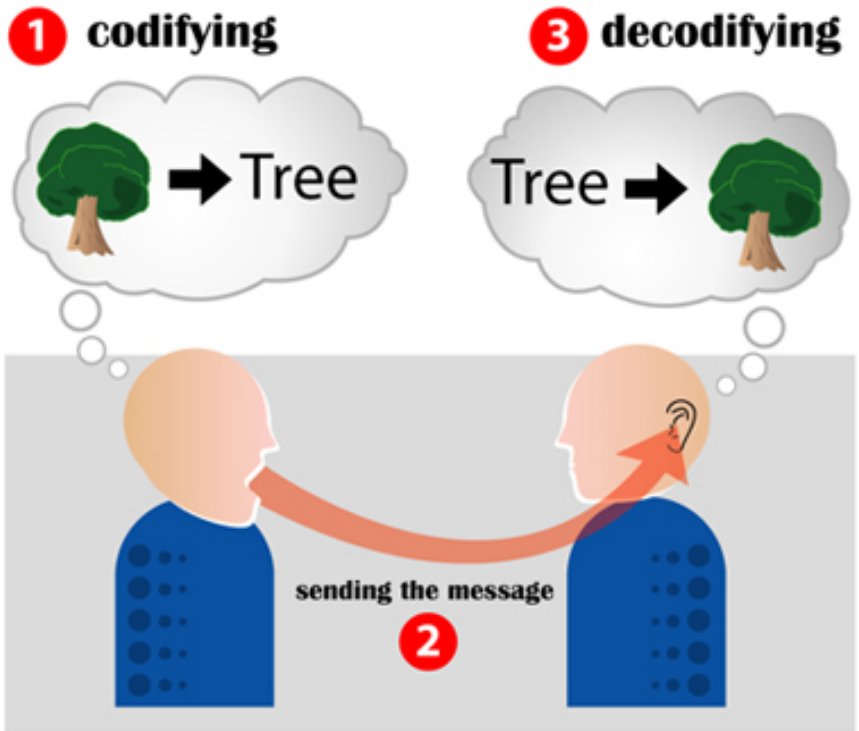
- Fehler im Regulären Ausdruck, Parameter der Sensoren wurden nicht erkannt

Einleitung

- Zur Koordination haben die NAOs die Möglichkeit, miteinander zu kommunizieren.

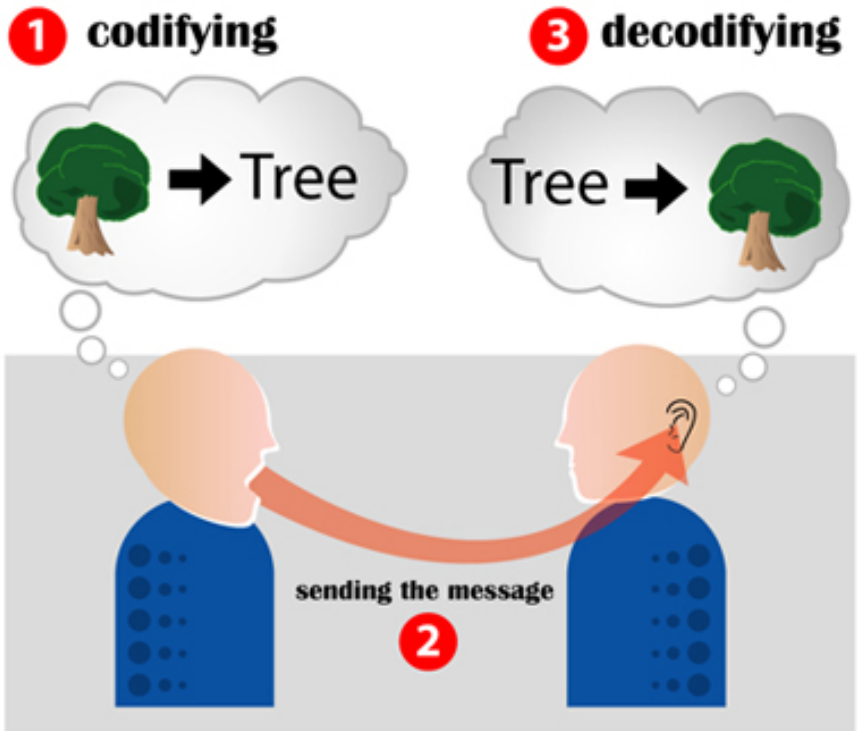
Einleitung

- Das funktioniert, im Prinzip, wie bei Menschen auch:



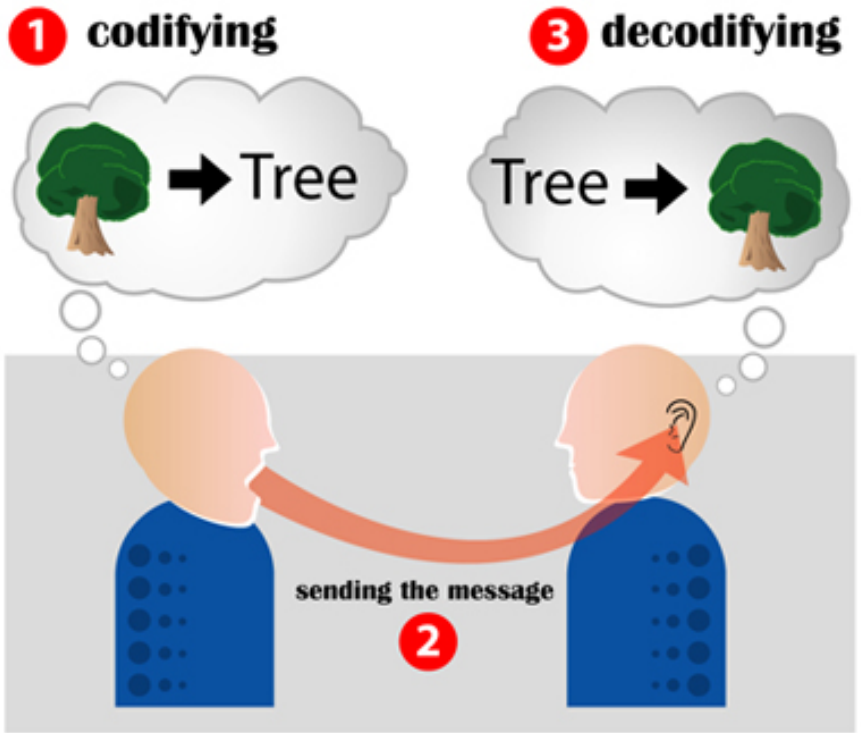
Einleitung

- Das funktioniert, im Prinzip, wie bei Menschen auch:
- Zunächst wird die Nachricht erstellt und codiert.



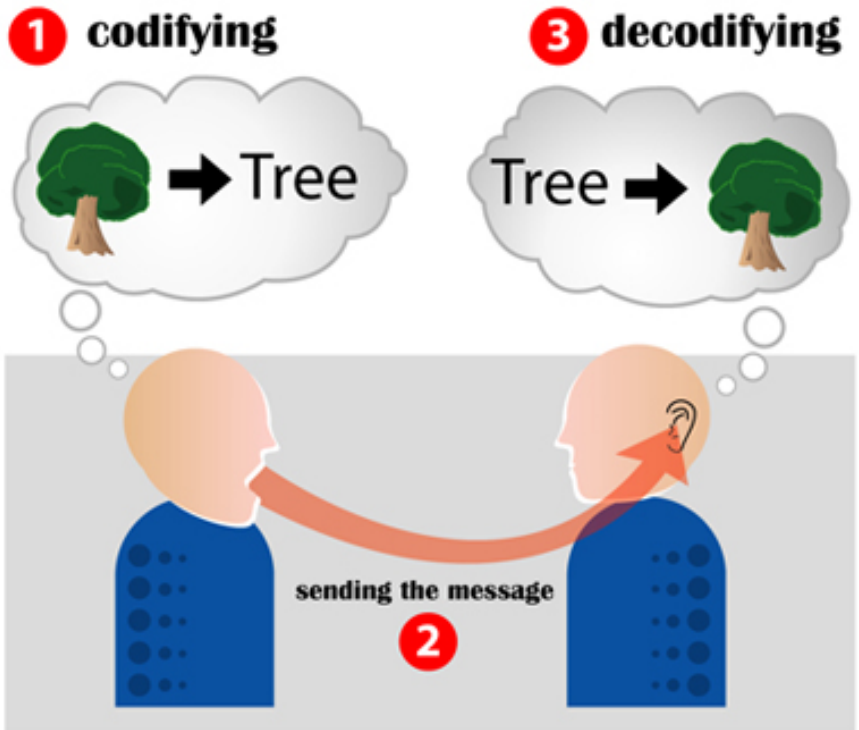
Einleitung

- Das funktioniert, im Prinzip, wie bei Menschen auch:
- Zunächst wird die Nachricht erstellt und codiert.
- Über den Server wird sie nun an die NAOs weitergeleitet.



Einleitung

- Das funktioniert, im Prinzip, wie bei Menschen auch:
- Zunächst wird die Nachricht erstellt und codiert.
- Über den Server wird sie nun an die NAOs weitergeleitet.
- Diese müssen die Nachricht entschlüsseln und interpretieren.



Die Rahmenbedingungen

- Die Kommunikation unterliegt einigen Einschränkungen:

Die Rahmenbedingungen

- Die Kommunikation unterliegt einigen Einschränkungen:
- Maximal 20 Zeichen pro Nachricht.

Die Rahmenbedingungen

- Die Kommunikation unterliegt einigen Einschränkungen:
- Maximal 20 Zeichen pro Nachricht.
- Es steht nur ein eingeschränkter ASCII-Zeichensatz zur Verfügung. (90 Zeichen)

Codierung

- Über say-Methoden kann eine Nachricht codiert werden.

Codierung

- Über say-Methoden kann eine Nachricht codiert werden.
- Eine codierte Nachricht ist wie folgt aufgebaut:

Codierung

- Über say-Methoden kann eine Nachricht codiert werden.
- Eine codierte Nachricht ist wie folgt aufgebaut:

- $< \mathit{MessageCode} >< \mathit{Parameter} >< \mathit{Prüfsumme} >$

Codierung

- Über say-Methoden kann eine Nachricht codiert werden.
- Eine codierte Nachricht ist wie folgt aufgebaut:
- $< \mathit{MessageCode} >< \mathit{Parameter} >< \mathit{Prüfsumme} >$
- Ein Message-Code definiert hier die Art der Nachricht.

Codierung

- Über say-Methoden kann eine Nachricht codiert werden.
- Eine codierte Nachricht ist wie folgt aufgebaut:

- $< \textit{MessageCode} >< \textit{Parameter} >< \textit{Prüfsumme} >$

- Ein Message-Code definiert hier die Art der Nachricht.
- Sender, Empfänger und Parameter sind dahinter angehängt. Doubles als Parameter haben einen Wertebereich von ± 364.00.

Codierung

- Über say-Methoden kann eine Nachricht codiert werden.
- Eine codierte Nachricht ist wie folgt aufgebaut:

- $< \textit{MessageCode} >< \textit{Parameter} >< \textit{Prüfsumme} >$

- Ein Message-Code definiert hier die Art der Nachricht.
- Sender, Empfänger und Parameter sind dahinter angehängt. Doubles als Parameter haben einen Wertebereich von ± 364.00.
- Am Ende wird die Nachricht durch eine Prüfsumme validiert.

Senden der Nachricht

- Nachdem die Nachricht codiert ist, kann sie versendet werden.

Senden der Nachricht

- Nachdem die Nachricht codiert ist, kann sie versendet werden.
- Hierzu steht einem der Say-Effector des NAO zur Verfügung.

Senden der Nachricht

- Nachdem die Nachricht codiert ist, kann sie versendet werden.
- Hierzu steht einem der Say-Effector des NAO zur Verfügung.
- Über ihn gelangt die Nachricht an den Server.

Empfangen der Nachricht

- Nun muss die Nachricht natürlich auch empfangen werden.

Empfangen der Nachricht

- Nun muss die Nachricht natürlich auch empfangen werden.
- Dafür nutzen wir den Hear-Perceptor des NAO.

Empfangen der Nachricht

- Nun muss die Nachricht natürlich auch empfangen werden.
- Dafür nutzen wir den Hear-Perceptor des NAO.
- Wenn der NAO gerade zuhört, empfängt er über ihn die Nachricht.

Empfangen der Nachricht

- Nun muss die Nachricht natürlich auch empfangen werden.
- Dafür nutzen wir den Hear-Perceptor des NAO.
- Wenn der NAO gerade zuhört, empfängt er über ihn die Nachricht.
- Hierfür steht eine Methode hear(...) zur Verfügung.

Decodierung

- Zunächst wird die Nachricht von einem Translator in ihre Bestandteile aufgetrennt.

Decodierung

- Zunächst wird die Nachricht von einem Translator in ihre Bestandteile aufgetrennt.
- Daraus wird ein HearObject erstellt.

Decodierung

- Zunächst wird die Nachricht von einem Translator in ihre Bestandteile aufgetrennt.
- Daraus wird ein HearObject erstellt.
- Dieses unterscheidet sich je nach Nachricht, und alle relevanten Daten sind in ihm gespeichert.

Decodierung

- Zunächst wird die Nachricht von einem Translator in ihre Bestandteile aufgetrennt.
- Daraus wird ein HearObject erstellt.
- Dieses unterscheidet sich je nach Nachricht, und alle relevanten Daten sind in ihm gespeichert.
- Über eine eval()-Methode kann das Objekt nun ausgewertet werden. (Taktik)

Weitere Details

- Es gibt noch einige weitere Einschränkungen der Kommunikation, die zu beachten sind:

Weitere Details

- Es gibt noch einige weitere Einschränkungen der Kommunikation, die zu beachten sind:
- Ein NAO ist 50m weit zu hören.

Weitere Details

- Es gibt noch einige weitere Einschränkungen der Kommunikation, die zu beachten sind:

- Ein NAO ist 50m weit zu hören.
- Er kann ausserdem nur alle 0.04s etwas hören. In der Zwischenzeit kann er keine Nachrichten empfangen.

Weitere Details

- Es gibt noch einige weitere Einschränkungen der Kommunikation, die zu beachten sind:

- Ein NAO ist 50m weit zu hören.
- Er kann ausserdem nur alle 0.04s etwas hören. In der Zwischenzeit kann er keine Nachrichten empfangen.
- Sollten 2 NAOs gleichzeitig sprechen, wird nur einer von ihnen gehört. Der Sprecher hört sich jedoch immer selbst.

Weitere Details

- Es gibt noch einige weitere Einschränkungen der Kommunikation, die zu beachten sind:

- Ein NAO ist 50m weit zu hören.
- Er kann ausserdem nur alle 0.04s etwas hören. In der Zwischenzeit kann er keine Nachrichten empfangen.
- Sollten 2 NAOs gleichzeitig sprechen, wird nur einer von ihnen gehört. Der Sprecher hört sich jedoch immer selbst.
- Die Teams sprechen versetzt, man kann also nicht die Kommunikation des anderen Teams blockieren.

Zusammenfassung

- Communication.py stellt Methoden zur Kommunikation zur Verfügung.

Zusammenfassung

- Communication.py stellt Methoden zur Kommunikation zur Verfügung.
- say-Methoden dienen dem senden von Nachrichten.

Zusammenfassung

- Communication.py stellt Methoden zur Kommunikation zur Verfügung.
- say-Methoden dienen dem senden von Nachrichten.
- Soll der NAO zuhören was gesagt wird, benutzt man hear().

Zusammenfassung

- Communication.py stellt Methoden zur Kommunikation zur Verfügung.
- say-Methoden dienen dem senden von Nachrichten.
- Soll der NAO zuhören was gesagt wird, benutzt man hear().
- Man erhält nun ein HearObject.

Zusammenfassung

- Communication.py stellt Methoden zur Kommunikation zur Verfügung.
- say-Methoden dienen dem senden von Nachrichten.
- Soll der NAO zuhören was gesagt wird, benutzt man hear().
- Man erhält nun ein HearObject.
- Über eval() wertet man es zu einem geeigneten Zeitpunkt aus.

Taktik

Konzept

- Schwarmverhalten als Grundkonzept
- Prinzipiell keine festen Rollen sondern dynamische Handlungen
- Handlungen hängen prinzipiell von der aktuellen Position ab
- Grundlegende Funktion ist $f(x) = \frac{1}{x}$, wobei x der Abstand zu einem anderen Objekt ist

Algorithmus zur Auswahl der aktuellen Aktion

- Funktionen, die Werte zwischen 0 und 1 zurückgeben.
- Funktionen hängen vom Abstand des Nao zu anderen Entitäten ab
- 0 ist niedrigste Priorität und 1 höchste
- Alle Funktionen ausgewertet, die mit der höchsten Zahl wird ausgeführt

Beispiele von Funktionen

- `enemy_owns_ball`: Gibt zurück ob der der Gegner den Ball hat
- `run_to_ball`: Repräsentiert die Priorität zum Ball zu laufen
- `run_to_enemy_goal`: Repräsentiert die Priorität zum gegnerischen Tor zu laufen. Das macht Sinn, wenn man den Ball hat
- `stay`: Einfach stehen bleiben. Das macht wenig Sinn und hat daher eine konstant niedrige Priorität

Beispielaufruf I

- `run_to_ball`: 0.83
- `run_to_own_goal`: 0.451
- `run_to_enemy_goal`: 0.034
- `stay`: 0.1

Beispielaufruf II

- `run_to_ball`: 0.83 $\rightsquigarrow$ wird ausgewählt, da der Wert am höchsten ist
- `run_to_own_goal`: 0.451
- `run_to_enemy_goal`: 0.034
- `stay`: 0.1