Robotik - Übung 02

Verwendung und Erstellung von Behaviours

Christoph Zinnen

WS17/18

LABORROBOTIK
Informatik - Computer Science

Informatik
Hauptcampus

HOCHSCHULE TRIER

Gliederung

Behaviour-Programmierung
- Direct Actions
- ArActionDesired
- Forward
- Swerve
- Parallele Ausführung

Aufgaben

Gliederung

Behaviour-Programmierung
- Direct Actions
- ArActionDesired
- Forward
- Swerve
- Parallele Ausführung

Aufgaben

Direct Actions

- **Direct Actions** sollten nicht parallel mit **Behaviours** verwendet werden. Denn sie verhindern, dass der Roboter wünsche von **Behaviours** entgegen nimmt.
- Wird eine **Direct Action** ausgeführt werden alle laufenden **Behaviours** handlungsunfähig.
- Um nach der Ausführung einer **Direct Action Behaviours** verwenden zu können kann die Funktion `ArAction::clearDirectMotion()` verwendet werden.

ArActionDesired

- Das erwünschte Roboter-Verhalten wird ARIA über ein Objekt der Klasse `ArActionDesired` mitgeteilt.
- Das `ArActionDesired` Objekt muss am Ende der fire Methode zurückgegeben werden.
- Der currentDesired Parameter der fire Methode ist vom Typ `ArActionDesired`.
  - Enthält Informationen aus vorherigem Durchlauf anderer Behaviour.
- Beispiel-Methoden der Klasse:

```
// Sets the velocity (mm/sec) and strength
void setVel(double vel, double strength = MAX_STRENGTH)

// Sets the delta heading (deg) and strength
void setDeltaHeading(double deltaHeading,
                     double strength = MAX_STRENGTH)
```

Forward

```
Forward(int speed=1000)
```

- **speed**
  Beschleunigt den Roboter auf die angegebene Geschwindigkeit[mm/sec].

Swerve

```
Swerve(Direction turnDir, int avoid=500, int
standoff=1000, int turnSpeed=20);
```

- **turnDir** bestimmt die Drehrichtung (*Left* = 1, *Right* = −1)
- **avoid** minimaler Abstand[*mm*]
- **standoff** maximaler Abstand[*mm*]
- **turnSpeed** Rotationsgeschwindigkeit[°/*sec*]

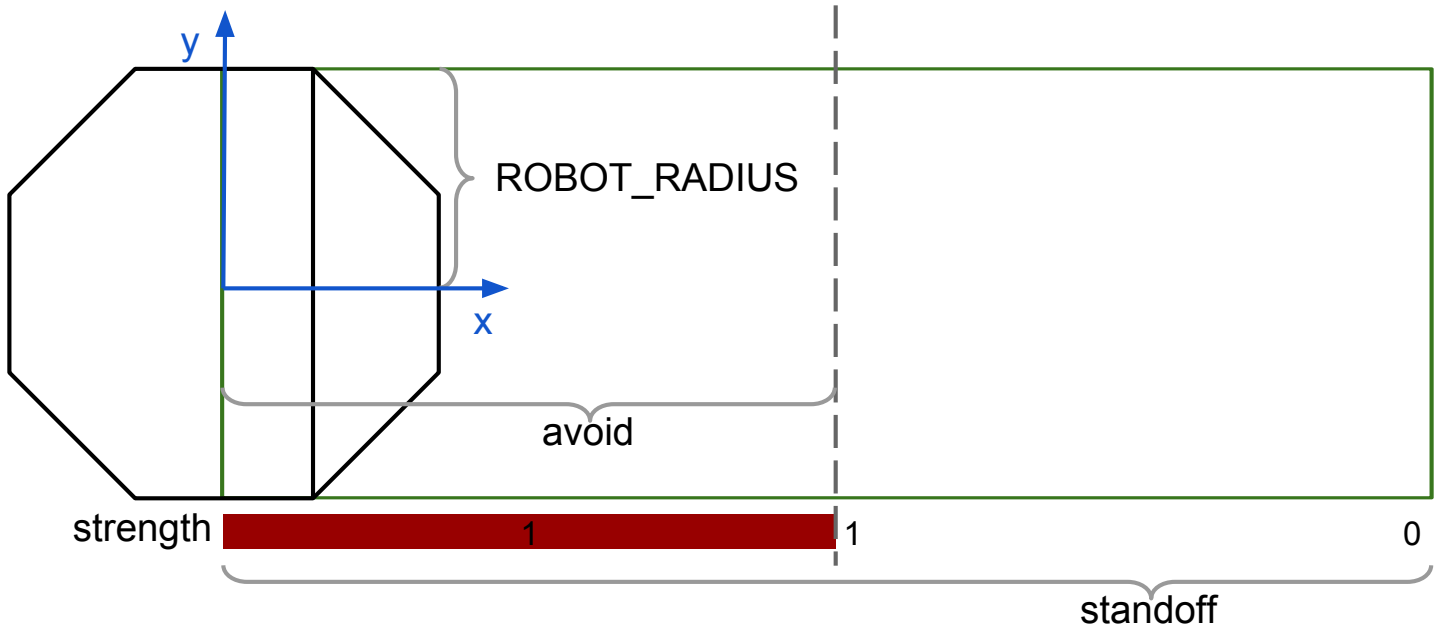
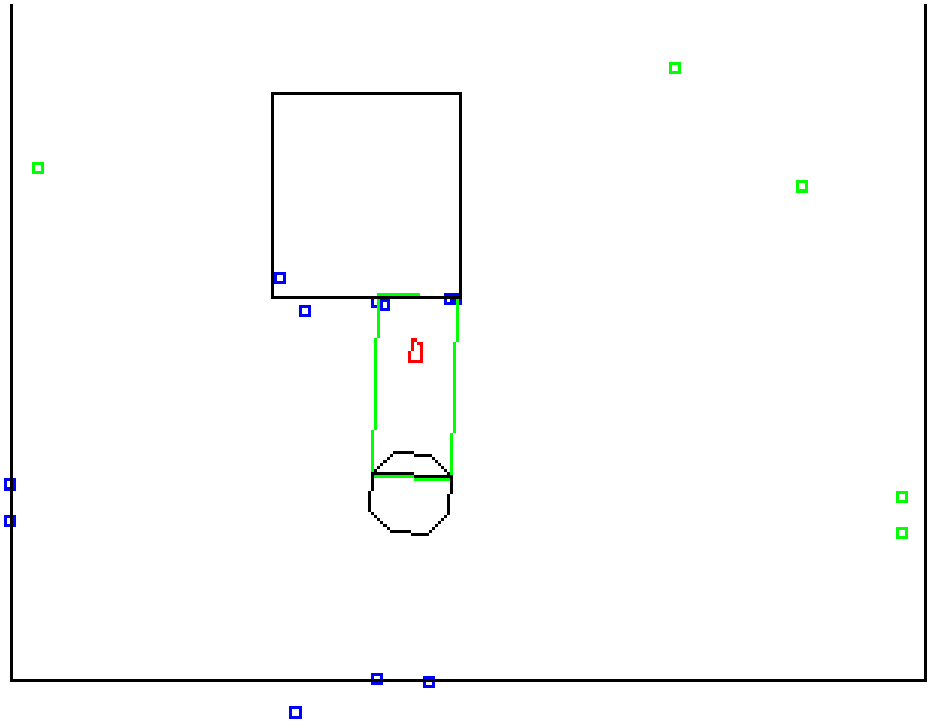


Abbildung 1: ReadingBox des Behaviour Swerve

Priority & Strength

- Die Priorität wird festgelegt, wenn eine Action zum Server Hinzugefügt wird.
  `addAction(ArAction &action, int priority=50)`
- Gültige Prioritäten befinden sich im Intervall $[0, 100]$.
- Behaviours werden in absteigender Reihenfolge ihrer Prioritäten ausgewertet.
- Die Wünsche werden in separaten Kanälen aufsummiert, bis die Stärke im jeweiligen Kanal `MAX_STRENGTH` erreicht.
  - velocity
  - heading
  - max forward velocity
  - max backwards velocity
  - max rotational velocity
- Wünsche von Behaviours gleicher Priorität werden entsprechend ihrer Stärke gemittelt.

Forward und Swerve I



	priority	strength
Forward	**50**	1
Swerve	**50**	0

Abbildung 2: Swerve noch nicht aktiv

Forward und Swerve II

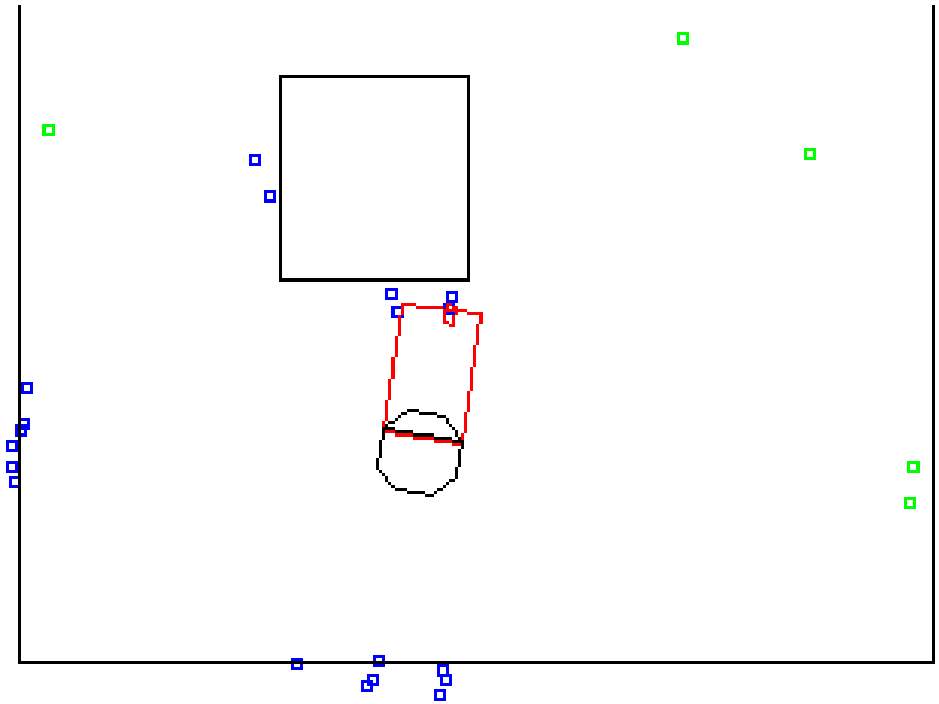
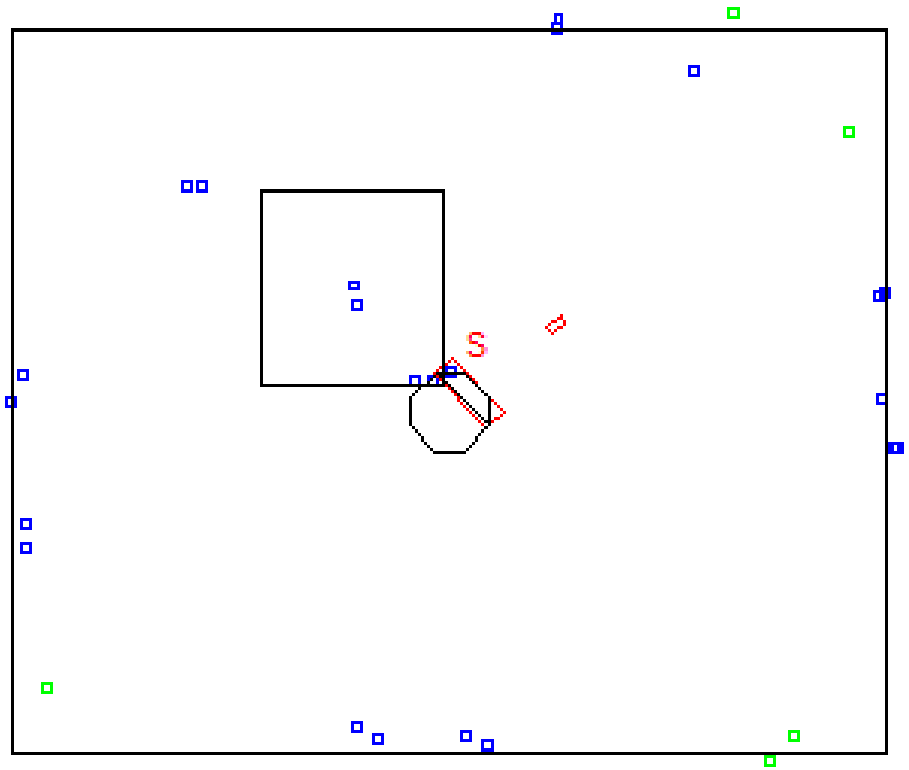


Abbildung 3: Swerve aktiv

	priority	strength
Forward	**50**	1
Swerve	**50**	0.5

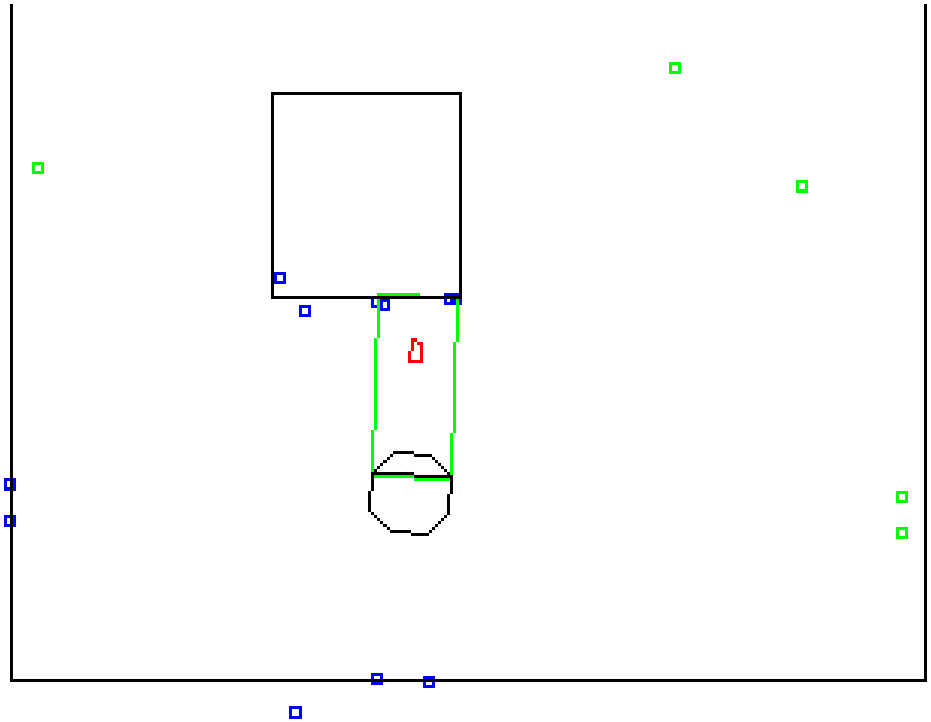
Forward und Swerve III



	priority	strength
Forward	**50**	1
Swerve	**50**	1

Abbildung 4: Forward ist zu stark

Forward und Swerve (priority) I



	priority	strength
Swerve	**51**	0
Forward	50	1

Abbildung 5: Swerve noch nicht aktiv

Forward und Swerve (priority) II

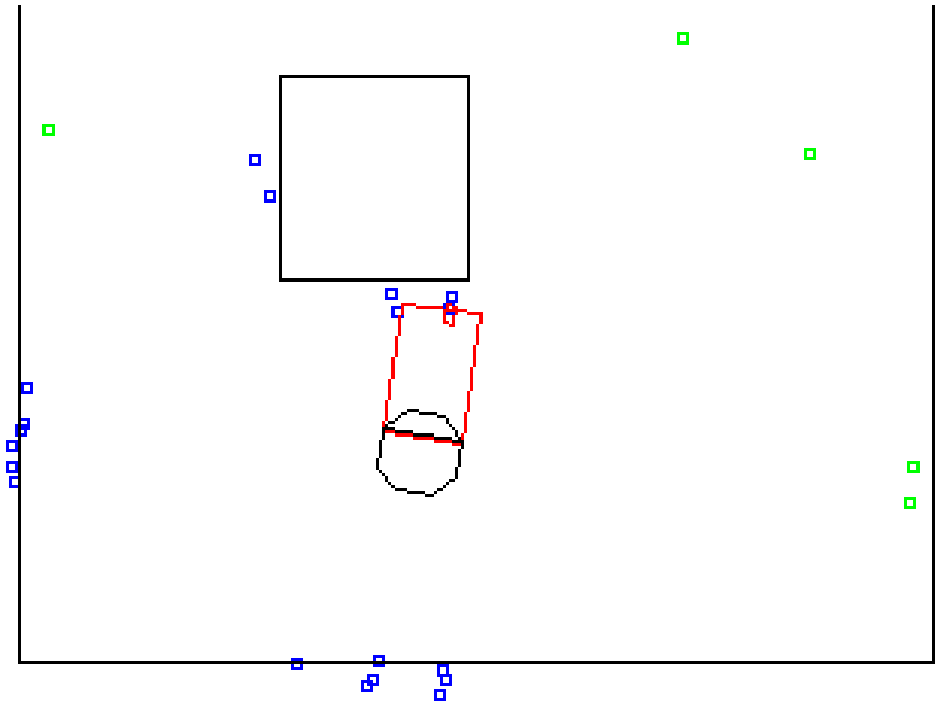
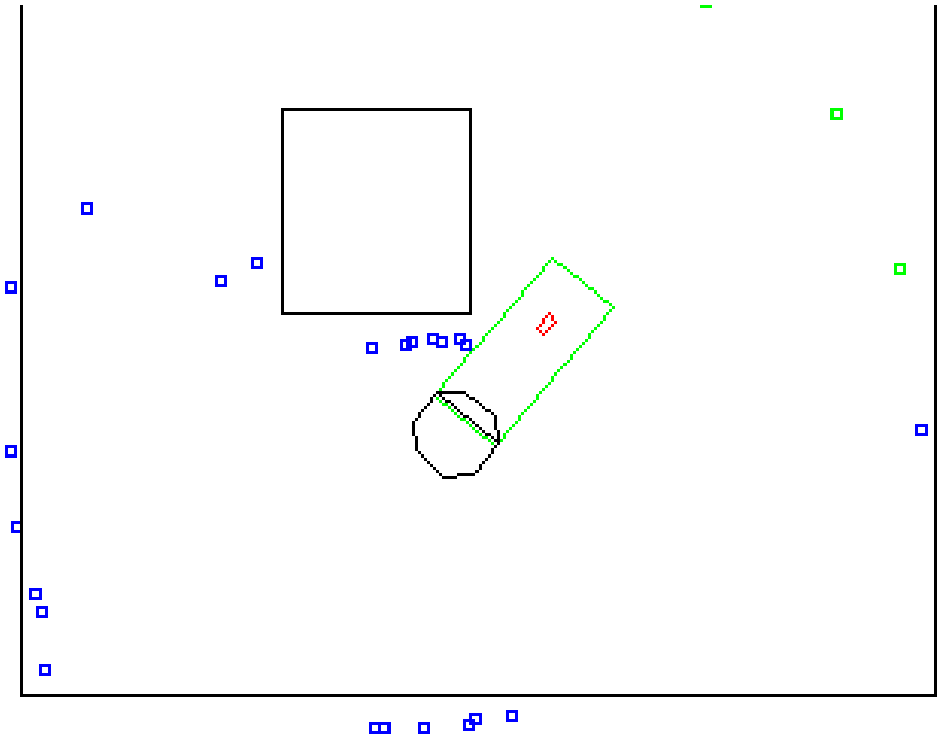


Abbildung 6: Swerve aktiv

	priority	strength
Swerve	**51**	0.5
Forward	50	1

Forward und Swerve (priority) III



	priority	strength
Swerve	**51**	0
Forward	50	1

Abbildung 7: Kurskorrektur vollzogen

Gliederung

Behaviour-Programmierung
- Direct Actions
- ArActionDesired
- Forward
- Swerve
- Parallele Ausführung

Aufgaben

Aufgaben

1. **GoTo**
   - Schreiben Sie ein Behaviour *GoTo*.
   - Das Behaviour soll eine Pose entgegen nehmen und auf direktem weg zu dieser Pose fahren.
   - Dabei soll es einstellbar sein ob der Winkel θ der Pose berücksichtigt wird.
2. **GoTo und Swerve**
   - Was ist zu beachten, wenn GoTo auch in Kooperation mit Swerve funktionieren soll?
   - Ändern Sie GoTo entsprechend ab.

Hinweis:

Als Pose empfiehlt sich die ArPose aus ARIA zu verwenden.