Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3186

Selbstorganisierte Strukturbildung in verteilten Robotersystemen

Martin Weber

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. habil. Paul Levi
Betreuer:	Dr. Viktor Avrutin Dr. Michael Schanz Prof. Dr. Jens Starke
begonnen am:	01. April 2011
beendet am:	20. Juli 2011
CR-Klassifikation:	I.2.9

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Roboterschwärme und andere verteilte Systeme bieten in vielen Bereichen ein vielversprechendes Konzept. Insbesondere sind in der Robustheit, der aus vielen simplen Teilen bestehenden, aber komplex organisierten Strukturen große Vorteile zu finden. Ein Bereich dieses Arbeitsgebietes befasst sich damit, dass die verteilten Robotersysteme Aufgaben im Team lösen, zu denen ein einzelner Roboter nicht in der Lage wäre. Ein Beispiel wäre ein verteiltes Wartungs- und Reinigungssystem für Tankanlagen, deren Öffnungen für einzelne große Roboter zu klein sind. Hier könnten kleine verteilte Roboter einzeln vordringen und sich im Inneren entsprechend formieren.
Bei der Bildung von Strukturen aus einzelnen Komponenten muss unter anderem auch immer die Verteilung der Aufgaben berücksichtigt werden. In dieser Arbeit soll hierzu auf gekoppelte Selektionsgleichungen zurückgegriffen werden.
In dieser Arbeit werden zunächst die mathematischen Grundlagen des Konzepts dieser gekoppelten Selektionsgleichungen vorgestellt. Im wesentlichen Teil der Arbeit werden verschiedene Möglichkeiten aufgezeigt, wie mit Hilfe der Selektionsgleichungen zunächst einfache geometrische Strukturen gebildet werden können. Simulationen verdeutlichen die Funktionsweise und die Robustheit der gewählten Lösungswege.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Mathematische Grundlagen befasst sich mit den mathematischen Grundlagen. Es werden Definitionen eingeführt und die Grundzüge der gekoppelten Selektionsgleichungen vorgestellt.

Kapitel 3 – Software stellt die entstandene Software vor und erläutert deren Aufbau und Funktion.

Kapitel 4 – Strukturbildung an vorgegebenen Raumpositionen beschreibt einen ersten Ansatz zur Strukturbildung. Dieser basiert auf vorgegebenen Zielen, die die entstehende Struktur beinhalten.

Kapitel 5 – Sequentielle Strukturbildung stellt ein Verfahren vor, bei dem die Strukturen auf sequentieller Weise entstehen. Dabei werden die Roboter nacheinander der entstehenden Struktur hinzugefügt.

Kapitel 6 – Parallele Strukturbildung erweitert den Ansatz aus Kapitel 5, sodass die Roboter gleichzeitig in die entstehende Struktur eingebunden werden können.

Kapitel 7 – Robustheit befasst sich mit der Robustheit der gefundenen Lösung gegenüber Ausfällen oder Deadlocks.

Kapitel 8 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2. Mathematische Grundlagen

2.1. Allgemeine Zuordnungsprobleme

Bei der Aufgabe, aus verschiedenen Robotern Strukturen zu bilden, geht es auch darum, dass diskrete Ziele von Robotern bedient werden sollen. Hierbei ist zu bemerken, dass jedem Roboter immer nur ein Ziel, und jedem Ziel immer nur ein Roboter zugeordnet werden soll. Selbstverständlich kann es bei unterschiedlicher Anzahl von Robotern und Zielen vorkommen, dass einige Roboter ohne Ziel oder einige Ziele unbedient bleiben.
Bei der Bearbeitung eines Ziels durch einen Roboter entstehen immer Kosten. Im Fall der Strukturbildung sind dies im Wesentlichen Fahrtkosten in Form der Zeit, die der jeweilige Roboter benötigt, um das Ziel zu erreichen. Es ist klar, dass diese Kosten von den Robotern und den Zielen abhängen (hier hauptsächlich durch den Abstand von Roboter zu diesem Ziel). Daher sind die entstehenden Gesamtkosten immer von der Zuordnung der Roboter zu den Zielen abhängig. Ziel einer jeden Zuordnungsaufgabe ist es, eine Verteilung zu finden, die möglichst geringe Gesamtkosten besitzt. Hierzu sind verschiedenste Ansätze zu finden, aber im Folgenden soll der Ansatz mittels gekoppelter Selektionsgleichungen verfolgt werden.
Alternativ zur Beschreibung der Güte einer Zuordnung über ihre Kosten kann auch in umgekehrter Weise von Gewinnen gesprochen werden. Dann besteht das Ziel, die Gesamtgewinne zu maximieren. Im Fall der Abstände von Robotern zu Zielen müssen diese so in Gewinne umgerechnet werden, dass ein kurzer Abstand zu hohen Gewinnen und ein großer Abstand zu kleinen Gewinnen führt.

2.2. Selektionsgleichungen

Grundsätzlich geht es bei Selektionsprozessen immer darum, unter verschiedenen Alternativen die geeigneteste auszuwählen. Hierbei gilt die als am besten geeignet, die die geringsten Kosten oder alternativ den höchsten Gewinn zur Folge hat, also die begrenzt vorhandenen Ressourcen am besten nutzt. Bei diesen Selektionsprozessen wird eine „winner takes it all“-Strategie verfolgt. Das heißt, der Gewinner des Selektionsprozesses verfolgt das Ziel ganz und alleine. Jeder der Unterlegenen hat in der Folge keinerlei Anteil an diesem Ziel [M.S04]. Dieses Verhalten ist übrigens auch in der Natur an vielfachen Stellen zu beobachten. Ein Beispiel hierfür könnte das Paarungsverhalten in Löwenrudeln sein. Ein weiteres Beispiel ist das unterdrückte Wachstum (bis hin zum Absterben) von Bäumen, wenn sie zu nahe bei einem wuchsfreudigen Baum stehen. All diesen Beispielen ist gemein, dass der Gewinner derjenige ist, dessen Anfangswerte die höchsten Gewinne versprechen.

Die Selektionsgleichungen, die dieses Verhalten beschreiben, sind Differentialgleichungen der Form [M.So4, J.S98]:

$$\frac{d}{dt}\xi_i \quad = \quad \xi_i \left(1 - \xi_i^2 - \beta \sum_{i' \neq i} \xi_{i'}^2\right)$$

Hierbei beschreibt $\xi_i \in \mathbb{N}$ den Betrag der Mode i, und die Anfangswerte entsprechen den Einzelgewinnen. Der Parameter $\beta > 1$ gibt die Kopplungsstärke an. Dieser Parameter ist für die Stärke verantwortlich, mit der das stärkste Individuum die Werte der anderen in Richtung o zwingt. Ein Beispiel für dieses Verhalten ist in Abb. 2.1 zu sehen.

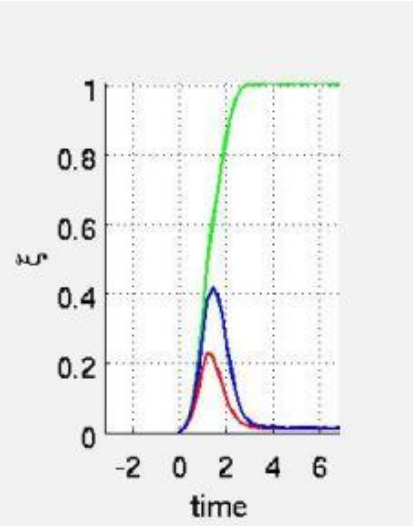

Abbildung 2.1.: Das Individuum mit dem größten Anfangswert (grün) gewinnt den Selektionsprozess, auch wenn alle Anfangswerte sehr klein sind.

Mit diesem Ansatz können Selektionsprobleme gelöst werden, bei denen ein Roboter von vielen ausgewählt werden soll, um ein Ziel zu bedienen. Dies wird zur Bildung von Strukturen aber noch nicht ausreichend sein.

2.3. Gekoppelte Selektionsgleichungen

Um eine möglichst günstige Zuordnung von verschiedenen Robotern zu verschiedenen Zielen erreichen zu können, wird das Konzept der gekoppelten Selektionsgleichungen eingeführt. Die Differentialgleichung aus 2.2 wird zu einem Differentialgleichungssystem erweitert [R.L, M.So4]:

$$\frac{d}{dt}\xi_{i,j} \quad = \quad \kappa\xi_{i,j} \left(1 - \xi_{i,j}^2 - \beta \sum_{i' \neq i} \xi_{i',j}^2 - \beta \sum_{j' \neq j} \xi_{i,j'}^2\right)$$

Hierbei ist κ ein Faktor zur Zeit-Skalierung und kann verwendet werden, um die Anzahl der Iterationsschritte bei einer nummerischen Integration zu senken. Bei den gekoppelten Selektionsgleichungen ist es ebenfalls so, dass jedem Ziel höchstens ein Roboter zugeordnet wird. Darüber hinaus wird jedem Roboter höchstens ein Ziel zugeordnet. Damit erzeugen die gekoppelten Selektionsgleichungen genau die gewünschte Zuordnung (siehe Abb. 2.2). Da sich bei den gegebenen Zuordnungsproblemen durchaus auch die Gegebenheiten ändern und Einfluss nehmen werden, müssen weitere Modifikationen an den gekoppelten Selektionsgleichungen vorgenommen werden. Eine Veränderung wird sein, dass durch die Bewegung von Robotern und Zielen die Kosten, die ein Roboter zum Erreichen eines Ziels aufbringen muss, von diesen Bewegungen beeinflusst werden. Außerdem kann es erforderlich sein, gewisse Ziele deaktivieren zu können. Hierzu ist es sinnvoll, einen Parameter einzuführen, der dies auf einfache Weise ermöglicht.

$$\frac{d}{dt}\xi_{i,j} = \kappa\xi_{i,j}\left(\alpha_{i,j}\left(\lambda_{i,j} - \xi_{i,j}^2\right) - \beta\sum_{i'\neq i}\xi_{i',j}^2 - \beta\sum_{j'\neq j}\xi_{i,j'}^2\right)$$

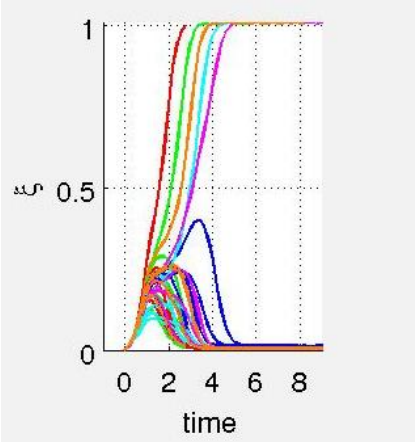

Abbildung 2.2.: Gekoppelte Selektionsgleichungen: Während 5 Roboter-Ziel-Zuordnungen sich durchsetzen können, unterliegen alle anderen Zuordnungen.

Der Parameter $\lambda_{i,j}$ ist der sogenannte Aufmerksamkeitsparameter und dient dazu, Ziele für Roboter freizuschalten oder zu sperren. Dabei bedeutet $\lambda_{i,j} = 1$, dass der Roboter i das Ziel j bedienen darf. Andernfalls muss $\lambda_{i,j} < 0$ gesetzt werden. Im Verlauf dieser Arbeit wird hier der Wert -1 gewählt werden.
Der zweite Parameter $\alpha_{i,j} \in [0,1]$ beschreibt die Attraktivität des Ziels j für den Roboter i. Da sich dieser Parameter zeitlich ändern kann, kann es vorkommen, dass Roboter den Selektionsprozess gewinnen, obwohl ihre Anfangswerte nicht die höchsten waren [J.S02]. Ein solches Beispiel ist in Abb. 2.3 zu sehen.

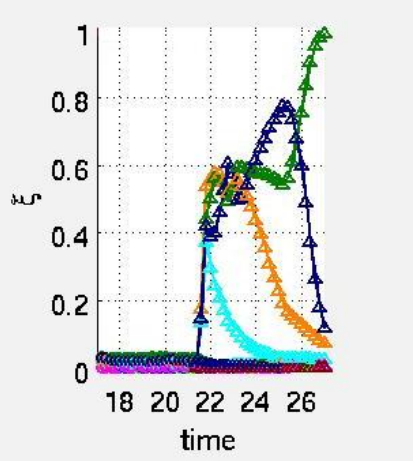

Abbildung 2.3.: Der Einfluss von α auf die gekoppelten Selektionsgleichungen: Deutlich zu sehen ist das Ansteigen des ξ Wertes des blauen Roboters. Aufgrund von Veränderungen der Positionen und damit der Werte α gewinnt trotzdem der grüne Roboter den Selektionsprozess.

2.4. Bewegungsmodell

2.4.1. Roboterbewegungen

Es wurde gezeigt, wie der Selektionsprozess zum Verteilen der Roboter auf die einzelnen Ziele stattfinden soll. Die Matrix ξ kann nach ausreichend langer Berechnung als eine Permutationsmatrix verstanden werden, die diese Verteilung eindeutig sicherstellt. Eine Möglichkeit, die Ziele der Roboter aus dieser Matrix zu erhalten, besteht darin, das Maximum einer jeden Zeile zu suchen. Dieses Vorgehen hat allerdings den Nachteil, dass bei noch nicht zu 1 angewachsenen Werten Zielzuordnungen als eindeutig angenommen werden, obwohl sie dies noch nicht sind. Ein Beispiel soll diese Problematik verdeutlichen: Es werden die beiden Werte $\xi_{i,a} = 0,4$ und $\xi_{i,b} = 0,45$ als die einzigen ξ-Werte des Roboters i angenommen, die größer sind als 0. Dann würde die Maximumsbildung suggerieren, dass Roboter i das Ziel b hat und dieses ansteuert. Eine andere Interpretation führt zu einer besseren Lösung bei Zielen die in Form von Raumkoordinaten vorliegen: Wenn man eine Linearkombination der ξ-Werte mit den Koordinaten der entsprechenden Ziele berechnet, können die Roboter sich entsprechend dieser Gewichtung auf die Ziele zubewegen. Dies hat den Vorteil, dass sich die Roboter noch nicht für ein Ziel endgültig entscheiden, bevor dies durch die gekoppelten Selektionsgleichungen eindeutig geschehen ist. Der gewichtete Schwerpunkt der Ziele ist allerdings eine sehr gute strategische Position. Entsprechend [J.S11, J.S02] wurden folgende Gleichungen zur Beschreibung der Roboterbewegungen gewählt:

$$\frac{d}{dt} r_i \quad = \quad v_i^0 \, \mathrm{e}_i^0 \left(r_i, \xi \right) + \mathrm{f}_i \left(r_i \right)$$

Hier handelt es sich bei r_i um die Position und bei v_i^0 um die Höchstgeschwindigkeit des Roboters i. Der Richtungsvektor (Linearkombination) wird mit e_i^0 bezeichnet. Der Term f_i steht für einen Korrekturterm zur Vermeidung von Kollisionen. Dieser wird später genauer beschrieben. Die Berechnung der Richtungsvektoren erfolgt auf folgende Weise [J.S11]:

$$\begin{aligned} \mathrm{e}_i^0\left(r_i, \xi\right) &= \mathrm{N}_{\gamma\delta}\left(\sum_j \xi_{i,j}(t)\,\mathrm{N}_{\gamma'\delta'}\left(g_j\left(t\right) - r_i\left(t\right)\right)\right) \\ \mathrm{N}_{\gamma\delta}\left(y\right) &= \frac{1}{\|y\| + \frac{1}{\gamma\|x\|+\delta}} \cdot y \end{aligned}$$

Hier soll gelten: $\gamma, \delta > 0$
$r_i(t)$ sollen die Koordinaten des Roboters i und $g_j(t)$ die Koordinaten des Ziels j zum Zeitpunkt t sein.

2.4.2. Kollissionsvermeidung

Auf dem Weg, den die Roboter beschreiben, können durchaus andere Roboter oder Gegenstände im Weg stehen. Es muss daher sichergestellt werden, dass es zu keinen Kollisionen kommen kann. Hierzu soll ein abstoßendes Potential aufgebaut werden, das der Bewegung der Roboter entgegenwirkt. Dieses Potential $\mathrm{f}_i(r_i)$ ist nach [J.S11, J.So2] wie folgt definiert:

$$\mathrm{f}_i\left(r_i\right) = \sum_{i' \neq i} \mathrm{f}_{ii'}^r\left(\mathrm{d}_{ii'}^r\right) + \sum_k \mathrm{f}_{ik}^o\left(\mathrm{d}_{ik}^o\right)$$

Hier resultiert die erste Summe aus den Abstoßungen der Roboter untereinander und die zweite aus den Abstoßungen der Roboter zu Hindernissen. Diese letzte Summe wird im Folgenden nicht weiter verwendet werden müssen, da es bei den später vorgestellten Simulationen keine Hindernisse geben wird. Der Vektor $\mathrm{d}_{ii'}^r \in \mathbb{R}^3$ ist der Abstandsvektor der Roboteroberflächen der Roboter i und i'. Um im Nahbereich der Roboter eine sehr starke und nach außen hin stark abfallende Abstoßung zu erreichen, ist die Abstoßungskraft $\mathrm{f}_{ii'}(\mathrm{d})$ für $\| \mathrm{d} \| \leq \sigma^r$ wie folgt definiert:

$$\begin{aligned} \mathrm{f}_{ii'}^r &= s^r\left(\tan\psi\left(\| \mathrm{d} \|\right) - \psi\left(\| \mathrm{d} \|\right)\right)\frac{\mathrm{d}}{\| \mathrm{d} \|} \\ \psi\left(\| \mathrm{d} \|\right) &= \frac{\pi}{2}\left(\frac{\| \mathrm{d} \|}{\sigma^r} - 1\right) \end{aligned}$$

Die Variable s^r soll hier die Stärke der Kollisionsvermeidung beschreiben.

3. Software

3.1. Vorbemerkung

Ziel einer Softwareentwicklung ist unter anderem, dass die Software von möglichst vielen Menschen optimal genutzt werden kann. Im Falle der hier entstandenen Software bedeutet das, dass sie von weiteren Wissenschaftlern genutzt werden soll, um weitere Experimente anzuschließen. Daraus folgt allerdings, dass die Software weiter entwickelt und ergänzt werden muss. Um diese Weiterentwicklung auch Wissenschaftlern aus dem mathematischen Bereich zu erleichtern, wurde entschieden, die Software in Matlab zu entwickeln, da diese Sprache im Fachbereich der Mathematik recht geläufig ist. Darüber hinaus bietet Matlab gute Möglichkeiten, Videos der Simulationen zu erzeugen, und auch das Darstellen von Funktionen ist leicht ausführbar.
Aus den genannten Gründen wurde bei der Erstellung der Software darauf geachtet, sie möglichst modular, in Funktionen gegliedert, aufzubauen. Um Vorwissen im Bereich der objektorientierten Programmierung nicht voraussetzen zu müssen, wurde auf die Verwendung dieses Konzeptes verzichtet.

3.2. Hauptfunktionen der Software

3.2.1. Funktion *scene_**

Die Funktionen *scene_** implementieren die Steuerung und den zeitlichen Ablauf der jeweiligen Szene. Im Allgemeinen beinhaltet dies:

- Setzen der Pfade für die Ausgabe
- Laden von Parametern aus externen Dateien
- Erzeugen bzw. Laden von Robotern und Zielen (gerade Nummern bedeuten, dass die Funktion die Roboterpositionen aus einer Datei lädt, und ungerade Nummern bedeuten, dass die Roboterpositionen zufällig erzeugt werden)
- Festlegung der Größe der Simulationswelt
- Initialisieren der Mengen von aktiven Zielen der verfügbaren Roboter und der bedienten Ziele
- Initialisierung der Matrizen α, λ und ξ

- Das Kernstück: Die zur Simulation benötigte Schleife (s.u.)
- Initiale und finale Behandlung der Ausgabe

Je nach gewünschtem Szenario müssen diese Teile angepasst werden. Der wesentliche Teil der Simulation spielt sich jedoch innerhalb einer while-Schleife ab. Hier werden folgende Punkte bearbeitet:

- Berechnung der neuen Matrizen α, λ und ξ (diese mit Hilfe der gekoppelten Selektionsgleichungen)
- Berechnung der neuen Positionen der Roboter
- ggf. Berechnung der neuen Positionen der Ziele, die sich aus den Positionen der Roboter ergeben (vgl. Docking-Stationen)
- Ausführen des Zeitschritts
- ggf. Ausgabe der aktuellen Situation
- Überprüfung, ob die Aufgabe erfüllt ist und damit die Simulation beendet ist sowie ggf. Berechnung der neuen Mengen von bedienten Zielen, von verfügbaren Robotern und aktiven Zielen

3.2.2. Funktion *cheackPosition*

Die Funktion *cheackPosition* dient zum Steuern der Schleife der Funktion *scene_**. In der Funktion *cheackPosition* wird überprüft, ob die gestellte Aufgabe vollständig ausgefürhrt worden ist. Der booleansche Rückgabewert der Funktion ist in diesem Fall *true*, andernfalls *false*. Darüber hinaus dient diese Funktion dazu, die Mengen der verfügbaren und bereits in der Struktur befindlichen Roboter und die Mengen der aktiven, der erreichten sowie der in Bearbeitung befindlichen Ziele anzupassen. Welche Mengen von der Funktion angepasst werden sollen bzw. müssen, hängt von der jeweiligen Aufgabe ab. Beispielsweise kann es Aufgaben geben, bei denen zu jedem Zeitpunkt alle Ziele aktiv sein sollen. Dementsprechend muss die Menge der aktiven Ziele nicht von der Funktion *cheackPosition* angepasst werden.

3.2.3. Funktion *eulerCoupSelecEq*

In der Funktion *eulerCoupSelecEq* wird eine Eulerintegration auf der Matrix ξ zur Lösung der Differentialgleichung

$$\frac{d}{dt}\xi_{ij} = \kappa\xi_{ij}\left(\alpha_{ij}\left(\lambda_{ij} - \xi_{ij}^2\right) - \beta_{ij}\sum_{i'\neq i}\xi_{i'j}^2 - \beta_{ij}\sum_{j'\neq j}\xi_{ij'}^2\right)$$

durchgeführt. Dabei wird auf Matrixoperationen von Matlab zurückgegriffen, um einen möglichst effizienten Code zu erhalten. So kann $\sum_{i'\neq i}\xi_{i'j}^2$ durch den Code :

```
- ones(rCount,rCount) * (Xi .* Xi) + ones(rCount,tCount) .* Xi .* Xi
```

berechnet werden. Dabei berechnet `(Xi .* Xi)` eine Matrix mit ξ_{ij}^2 als Einträgen. Durch die Multiplikation `ones(rCount,rCount) * (Xi .* Xi)` wird die eine Matrix erzeugt, deren j-te Spalte aus der Summe der Quadrate der Einträge der j-ten Spalte der Matrix ξ besteht. Durch die Korrektur mit `ones(rCount,tCount) .* Xi .* Xi` wird die gewünschte Summe berechnet. Analog hierzu wir die Summe $\sum_{j' \neq i} \xi_{ij'}^2$ berechnet.

3.2.4. Funktion *navigation*

Die Funktion *navigation* berechnet die neuen Positionen und Richtungen der Roboter. Hierzu müssen die Roboter selbst, die Ziele, die Matrix ξ für die Roboter-Ziel-Zuordnung sowie die Zeit und die Zeitschritte übergeben werden. In dieser Funktion findet auch die Kollisionsvermeidung mit Hilfe des Aufbaus eines abstoßenden Potentials der Roboter untereinander statt.
Zunächst werden in der Funktion benötigte Parameter eingelesen. Im zweiten Schritt wird für jeden Roboter sein in diesem Zeitschritt angestrebter Fahrweg berechnet. Hierzu wird eine Linearkombination in der Form gebildet, dass die zu dem Roboter i gehörende Zeile der Matrix ξ mit den Richtungen, in denen die entsprechenden Ziele liegen, multipliziert wird. Im dritten Schritt wird das anstoßende Potential der Roboter untereinander berechnet und anschließend der Geschwindigkeitsvektor für jeden Roboter bestimmt. Im letzten Schritt müssen die Ausrichtungen der Roboter im Raum ihren Geschwindigkeitsvektoren angepasst werden. Die Roboter werden also entsprechend ihren Geschwindigkeitsvektoren ausgerichtet. Befindet sich ein Roboter allerdings nah genug an seinem Ziel, so wird er entsprechend seines Ziels ausgerichtet.

3.2.5. Funktion *setLambda*

Um die Matrix λ in jedem Zeitschritt (auch bei der Integration) zu erhalten, wird die Funktion *setLambda* benötigt. Da λ in Abhängigkeit des gestellten Problems bestimmt werden muss, muss diese Funktion in der Regel der Problemstellung angepasst werden.

3.2.6. Funktion *setAlpha*

Die Funktion *setAlpha* berechnet die Matrix α. Da α die Attraktivität eines Ziels für einen Roboter beziffert, wird zur Berechnung der Abstand zwischen dem Roboter und diesem Ziel herangezogen.
Zunächst wird eine Matrix mit allen Abständen von Robotern und Zielen berechnet. Diese werden anschließend normiert. Da aber ein großer Wert für α eine hohe und ein kleiner Wert für α eine niedrige Attraktivität widerspiegeln soll, müssen die normierten Abstände

umgerechnet werden, damit ein nahes Ziel attraktiver dargestellt wird als ein weit entferntes [J.S11].

3.2.7. Funktion *createTargetsBetween*

Da es Problemstellungen geben kann, bei denen zwei Ziele vorgegeben sind und zwischen diesen beiden weitere Ziele in gleichmäßigem Abstand hinzugefügt werden sollen, wurde diese Funktion eingeführt. Zunächst wird der Abstand der beiden vorgegebenen Ziele berechnet. Dann wird die Anzahl der Ziele aus dem Abstand sowie der Roboterbreite und des minimalen Abstandes zwischen zwei Robotern bestimmt und damit auch der Abstand, den zwei aufeinander folgende Ziele haben sollen. Im dritten Schritt werden in einer Schleife die Ziele nacheinander in gleichem Abstand eingefügt.

3.2.8. Funktion *initBots*

Mit Hilfe der Funktion *initBots* kann eine Menge von „botCount“ vielen Robotern zufällig erzeugt werden. Die Roboter befinden sich an unterschiedlichen Positionen in einem durch zwei Eckpunkte definierten Quader. Um Überschneidungen der Roboter zu verhindern, muss die Breite der Roboter ebenfalls als Parameter übergeben werden.

3.2.9. *load** Funktionen

Die Funktionen *loadParameters*, *loadRobots* und *loadTargets* dienen dazu, die Programmparameter und die Positionen und Richtungen der Roboter und Ziele aus externen Dateien einlesen zu können.

3.3. Programmierung der Graphik

3.3.1. Funktion *plotWorld*

Diese Funktion erzeugt die Darstellung der Ausgabe. Hierzu werden erst Parameter eingelesen, die zur Konfiguration der Ausgabe dienen. Danach werden (in Abhängigkeit der eingelesenen Parameter) folgende weitere Funktionen aufgerufen:

- *plotRobotColors*: Zeichnet die Legende der Farben der Roboter.
- *plotTargetMarker*: Zeichnet die Legende der Symbole der Ziele.
- *plotMatrix*: Plottet die Matrizen ξ und λ.
- *informationOutput*: Plottet den zeitlichen Verlauf der ξ-Werte von allen Robotern und den im Bau befindlichen Zielen.
- *plotScene*: Zeichnet die räumliche Darstellung der Roboter und Ziele.

3.3.2. Funktion *informationOutput*

Die Funktion *informationOutput* zeichnet zum einen den zeitlichen Verlauf der ξ-Werte der Roboter und der im Bau befindlichen Ziele. Dazu muss die Menge der im Bau befindlichen Ziele von einer Ausgabe zur nächsten gespeichert werden, da geprüft werden muss, ob sich dieses Ziel dann immer noch im Bau befindet. Außerdem müssen die ξ-Werte für den nächsten Schritt gespeichert werden, um stetige Graphen zeichnen zu können.
Darüber hinaus kann die Funktion *informationOutput* - in Abhängigkeit von eingelesenen Konfigurationsparametern - Informationen darüber ausgeben, welche Roboter bereits in Strukturen eingebunden wurden und welche noch verfügbar sind sowie welche Ziele bereits bedient wurden und welche sich noch im Bau befinden. Bei dieser Ausgabe wurde besonderes Augenmerk auf eine sinnvolle Darstellung mit Zeilenumbrüchen gelegt.

3.3.3. Funktion *plotScene*

Da es sinnvoll ist, neben einer ansprechenden und aufwändigen Graphik auch eine vereinfachte und damit weitaus weniger rechenintensive Darstellung als Ausgabe zu produzieren, kann beides mit der Funktion *plotScene* realisiert werden. Nach der Positionierung des Subplots und dem Setzen von einigen Variablen wird die Unterscheidung in die simple und die komplexere Darstellung vorgenommen. Beide Fälle unterscheiden sich allerdings nur im Zeichnen der Gitter, der Roboter und der Ziele. Im einfachen Fall werden die Roboter und die Ziele durch Kreise dargestellt (die in dieser Funktion gezeichnet werden). Im Gegensatz dazu werden in der komplexeren Darstellung zwei Funktionen zum Zeichnen der Roboter und der Ziele aufgerufen. Der Ablauf ist in beiden Fällen jedoch gleich:

- Darstellung (und Festlegung) des Zeichenbereiches durch Zeichnen von zwei Eckpunkten
- Setzen der Darstellungsoptionen (Gitter) von Matlab
- Zeichnen der Ziele
- Zeichnen der aktiven Ziele
- Zeichnen der Roboter

3.3.4. Funktion *robot*

Für die komplexe Darstellung der Roboter wurde ein ameisenähnliches Aussehen gewählt. Es wurde bewusst auf eine Darstellung verzichtet, die eine Verwandtschaft zu bestehenden Robotern suggerieren könnte. Es soll hierdurch deutlich gemacht werden, dass es sich bei dieser Arbeit um die Simulation von Robotern und nicht um real existierende Systeme handelt. Die Roboter bestehen aus folgenden Körperteilen:

- Körper (Zylinder mit 8-eckiger Grundfäche)
- Kopf (mit Vorder- und Hinterseite: Kugelsegment und Kegelstumpf)
- Hinterteil (mit Vorder- und Hinterseite: Kegelstumpf und Kugelsegment)
- vier Beine (bestehend aus jeweils einem Ober- und einem Unterschenkel aus jeweils einem Zylinder)
- zwei Fühler (bestehend aus jeweils einem Stiel und einer Fühlerspitze: Zylinder und Kugel)

Bei der Gestaltung der Roboter wurde darauf geachtet, dass die Körperform (ohne Beine und Fühler) nicht aus einer Kugelhülle herausragt. Die Länge der Roboter entspricht dem Durchmesser der einhüllenden Kugel.

Um ein Körperteil zu zeichnen, wird zunächst ein geometrisches Objekt mit der Matlab-Funktion *cylinder* erzeugt. Mit dieser Funktion müssen nicht notwendigerweise ausschließlich Zylinder erzeugt werden, sondern es können auch andere rotationssymmetrische Körper erzeugt werden, wenn die den Radius beschreibende Koordinate in entsprechender Darstellung gewählt wurde. Das erzeugte geometrische Objekt wird anschließend auf die gewünschte Größe des entsprechenden Körperteils gestreckt und an den gewünschten Ort verschoben. Im dritten Schritt wird das Körperteil in die entsprechende Richtung gedreht. Hierzu können durchaus mehrere Drehungen erforderlich sein, da beispielsweise ein Oberschenkel zuerst nach seiner Erzeugung und Verschiebung um die Hochachse des Roboters nach vorne, und dann um die Längsachse nach oben gedreht wird. Danach folgen zwei weitere Drehungen, um die Körperteile (und damit den gesamten Roboter) entsprechend des Roboters Ausrichtung im Raum zu orientieren.

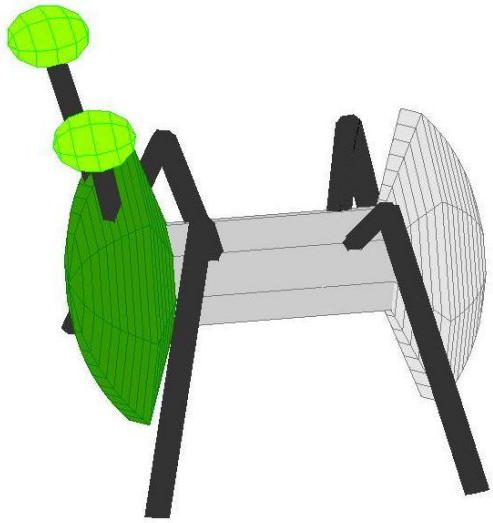

Abbildung 3.1.: Darstellung eines Roboters

3.3.5. Funktion *plotTarget*

Die Funktion *plotTarget* stellt ein Ziel als eine Kugel an der im Parameter „pos“ übergebenen Position dar. Der booleansche Parameter „active“ enthält die Information, ob es sich bei dem Ziel um ein aktives oder inaktives Ziel handelt. Dementsprechend wird die Kugel grün oder grau gezeichnet.

3.4. Abschließende Beurteilung der Software

Zum Abschluss dieses Kapitels sollen noch einige Punkte erwähnt werden, die im Laufe der Entwicklung der Software aufgefallen waren, aber nicht mehr in der gegebenen Zeit berücksichtigt werden konnten. Da Softwareentwicklung ein evolutionärer Prozess ist, können sich einige Designentscheidungen im Laufe der Zeit als längerfristig ungünstig herausstellen, obwohl sie in der Entscheidungssituation sinnvoll erschienen.

Parameter: In allen Programmteilen, die Parameter aus externen Dateien laden, wäre es sinnvoller, die Variablen, in die diese Parameter geschrieben werden, persistent zu halten, um nicht bei jedem Funktionsaufruf die Parameter erneut laden zu müssen. Es ist dann zwar ein Vergleich notwendig, der darüber entscheidet, ob die Datei geladen werden muss (erster Aufruf der Funktion) oder nicht, aber insbesondere bei größerer Anzahl von Parametern rechtfertigt sich dieser geringe Aufwand durchaus.

Funktionsnamen: Der Name einzelner Funktion kann als irreführend angesehen werden. In ihrer ursprünglichen Aufgabe hatten diese Funktion eine etwas andere Aufgabe, haben allerdings im Laufe der Entwicklung weitere Aufgaben übernommen.

Programmorganisation durch Funktionszeiger: Da die Hauptprogramme sich durch die verwendeten Funktionen unterscheiden, kann es sinnvoll sein, diese mittels Funktionszeigern austauschbar zu halten. Damit wäre es möglich, ein Hauptprogramm zu erstellen, das mit Hilfe einer weiteren Konfigurationsdatei die Funktionen auswählt, die abgearbeitet werden sollen. Ein Vorteil von dieser Architektur wäre, dass es nicht so viele verschiedene Hauptfunktionen (*scene_**) geben würde, sondern nur eine und entsprechende Konfigurationsdateien. Nachteilig wäre an diesem Konzept allerdings, dass man für Weiterentwicklungen das Wissen über die Programmierung von Funktionszeigern voraussetzen müsste. Hier sollten Entwickler, die an dieser Software weiterarbeiten, abwägen, welche Argumente aus ihrer Sicht überwiegen.

4. Strukturbildung an vorgegebenen Raumpositionen

Im ersten Schritt war es das Ziel, Roboter an vorgegebenen Positionen zu platzieren, um auf diese Weise Strukturen zu erzeugen. Die vorgegebenen Positionen müssen nicht notwendigerweise ortsfest sein, sondern können sich durchaus auch im Raum bewegen, sind aber selbst von den Positionen der Roboter unabhängig.
Jedes Ziel wird hierbei durch seine Position und seine Ausrichtung im Raum beschrieben. Die Art der Struktur hängt von der Positionierung der Ziele ab.

4.1. Verfahren zur Strukturbildung an vorgegebenen Raumpositionen

Als zu bildende Struktur wurde eine Linie im Raum gewählt. Diese soll sich zwischen einem Start- und einem Endpunkt befinden und alle Roboter sollen auf den Startpunkt ausgerichtet sein. Der minimale Abstand der Roboter untereinander ist festgelegt und die Roboter sollen - so dicht wie möglich stehend - die Linie bilden.

4.1.1. Idee des Verfahrens

Zu Beginn sind die Positionen des Start- und des Endpunktes der Linie bekannt. Daraus kann der Abstand der Punkte und damit die Länge der Linie berechnet werden. Da auch die Länge der Roboter und ihr minimaler Abstand untereinander bekannt sind, kann die maximale Anzahl und damit auch der tatsächliche Abstand der Roboter untereinander berechnet werden. Die weiteren Ziele werden mit diesem Abstand vom Startpunkt aus in Richtung des Endpunktes verteilt und alle Ziele in Richtung des Startpunktes hin ausgerichtet.
Der Selektionsprozess führt zu einer Roboter-Ziel-Zuordnung und die Roboter fahren ihre Ziele an. Wenn alle Ziele besetzt sind oder alle Roboter ein Ziel besetzen, ist die Aufgabe erfüllt.

4.1.2. Matlab Funktionen

cheackPosition

Die Funktion *cheackPosition* erkennt, ob alle Roboter sich in der Struktur befinden oder ob alle Ziele besetzt sind. Ist eines der Fall, so ist die Aufgabe abgeschlossen, und die Funktion liefert den Rückgabewert *true*. Darüber hinaus liefert die Funktion die Mengen der Roboter, die sich in der Struktur befinden, und die Menge der bedienten Ziele. Diese beiden Mengen müssen in der Funktion ohnehin erstellt werden, um entscheiden zu können, ob die Aufgabe erfüllt ist.
Als Parameter bekommt die Funktion die Roboter („bots“), die Ziele („targets“, die Menge der Roboter in der Struktur („robotsInStructure“) und die Menge der bedienten Ziele („servedTargets“) übergeben. Innerhalb der Funktion wird mit einer Schleife über die Roboter und einer Schleife über die Ziele für jede Roboter-Ziel-Kombination überprüft, ob sich der Roboter hinreichend nah an dem Ziel mit einer hinreichend genauen Ausrichtung befindet. Ist dies der Fall, so wird das Ziel als bedient und der Roboter als in der Struktur befindlich angesehen.

setLambda

Da alle Ziele zu jedem Zeitpunkt aktive Ziele sind, liefert die Funktion *setLambda* eine Matrix λ, die vollständig mit 1 besetzt ist. Bei der Implementierung wurde die Funktion so verwendet, wie sie auch für viele andere Aufgaben verwendet werden wird. Dies hat den Vorteil, dass so auch Ziele hinzu- oder ausgeschaltet werden können. Es wird also zuerst eine mit -1 besetzte Matrix erzeugt und anschließend die Einträge aller Spalten, die zu aktiven Zielen gehören, auf 1 gesetzt.

4.1.3. Beispiele

Beispiel: Bildung einer Linie mit vorgegebenen Zielen und fünf Robotern und sechs Zielen

Bei dem ersten Beispiel wurden als Start- und Endpunkt die Punkte (8, 3, 0) und (4, 10, 0) gewählt. Daraus ergibt sich bei einer Roboterlänge von 1 und einem minimalen Abstand zwischen den Robotern von 0,5 eine Anzahl von 6 Zielen. Die Simulation wurde mit 5 zufällig positionierten Robotern gestartet.
Wichtige weitere Parameter sind:

- $\kappa = 1.5$
- $\beta = 4$
- DT_XI_FACTOR von 8
- ein maximales Rauschen von 10^{-3}

- und eine Schrittweite $\delta t = 0.01$

In Abb. 4.1 ist der Start der Simulation zu sehen. Es ist deutlich, dass sich zwei Roboter sehr nahe an jeweils einem Ziel befinden. Nach zwei Simulationssekunden kann man in Abb. 4.2 sehen, dass sich die ξ-Werte dieser Roboter (und der entsprechenden Ziele) sehr schnell gesteigert haben. Obwohl Roboter 1 nicht so nahe am Ziel 6 steht, ist der ξ- Wert ebenfalls stark angewachsen. Dies kann dadurch erklärt werden, dass dieses Ziel für sonst keinen anderen Roboter günstig ist. In Abb. 4.3 ist die endgültige Zielzuordnung und der zeitliche Verlauf der ξ-Werte zu sehen.

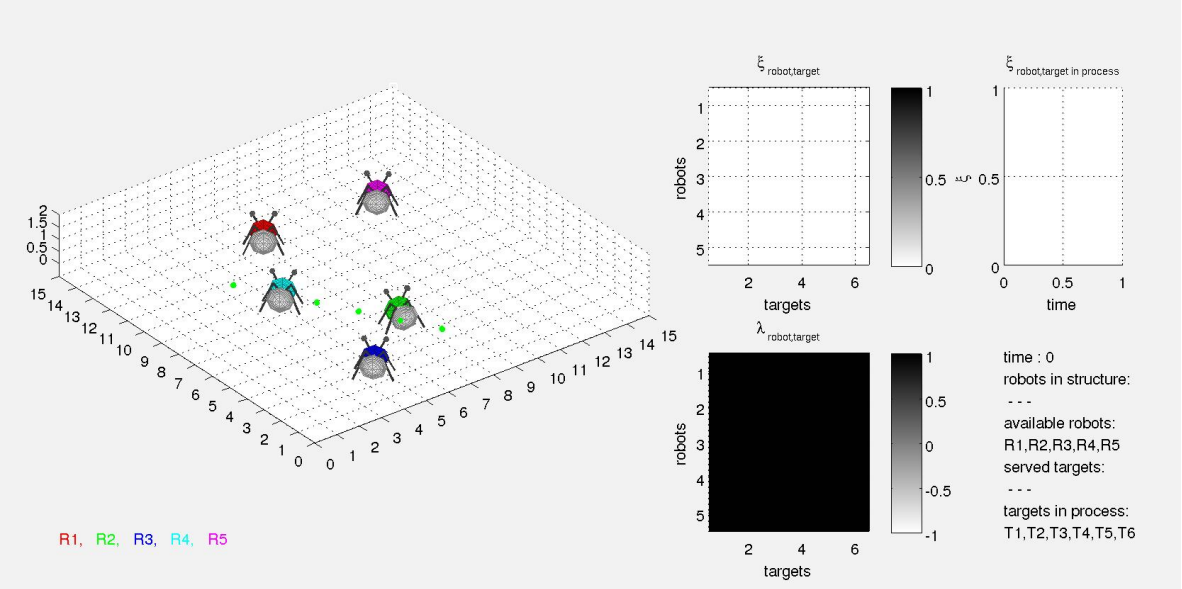

Abbildung 4.1.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 6 Zielen und 5 Robotern“ zu Beginn der Simulation

Beispiel: Bildung einer Linie mit vorgegebenen Zielen und sechs Robotern und fünf Zielen

Beim zweiten Beispiel sind die Koordinaten der Start- und Endpunkte (2, 5, 0) und (8.5, 5, 0). Es ergeben sich damit 5 Ziele. Die Simulation wurde mit den gleichen Parametern, aber mit sechs zufällig positionierten Robotern gestartet.

In Abb. 4.4 sind die Positionen der Roboter und der Ziele zu Beginn der Simulation zu sehen. Roboter 1 befindet sich sehr nahe am Ziel 5. Nach einer Simulationssekunde (zu sehen in Abb. 4.5) bewegt sich Roboter 1 jedoch nicht auf Ziel 5 zu, obwohl $\xi_{1,5}$ den höchsten Wert, also die größte Attraktivität für Roboter 1, aufweist. Stattdessen bewegt sich Roboter 1 auf einen Schwerpunkt der Ziele zu und damit weg von Ziel 5. Eine weitere Simulationssekunde später ist der Wert von $\xi_{1,5}$ so weit angestiegen, dass sich Roboter 1 (wie erwartet) auf das Ziel 5 zubewegt (siehe Abb. 4.6). Die übrigen Zielzuordnungen sind noch nicht entschieden,

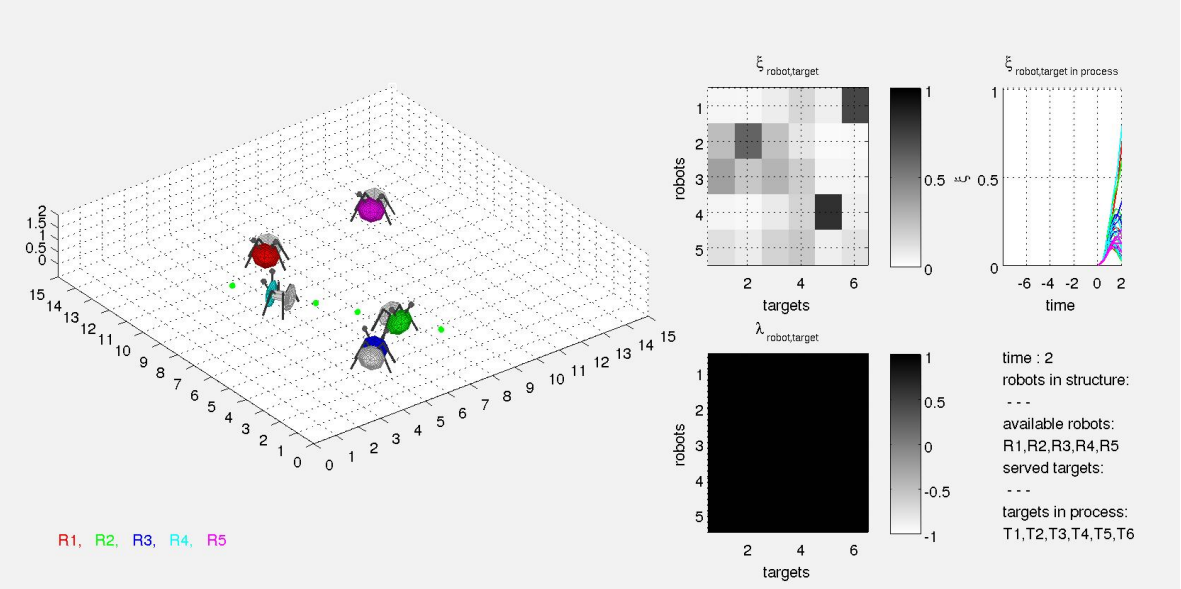

Abbildung 4.2.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 6 Zielen und 5 Robotern“ nach 2 Simulationssekunden.

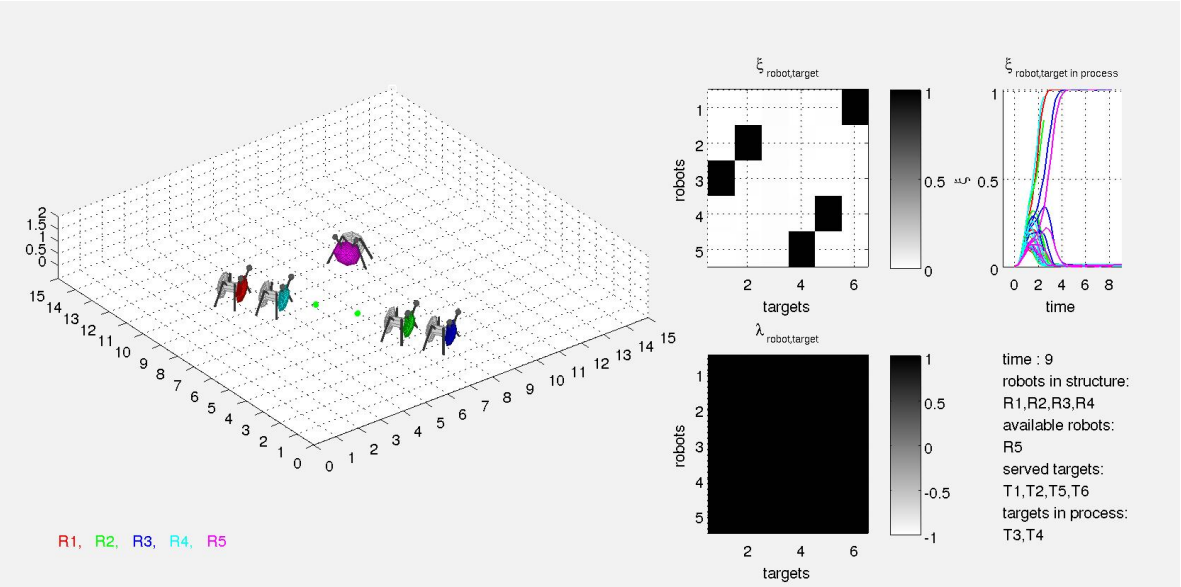

Abbildung 4.3.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 6 Zielen und 5 Robotern“ mit endgültiger Zielzuordnung

und insbesondere Roboter 3 und 5 befinden sich im Wettstreit um Ziel 3. In Abb. 4.7 ist die endgültige Zuordnung zu sehen. Interessant daran ist, dass Roboter 3 im Wettstreit mit

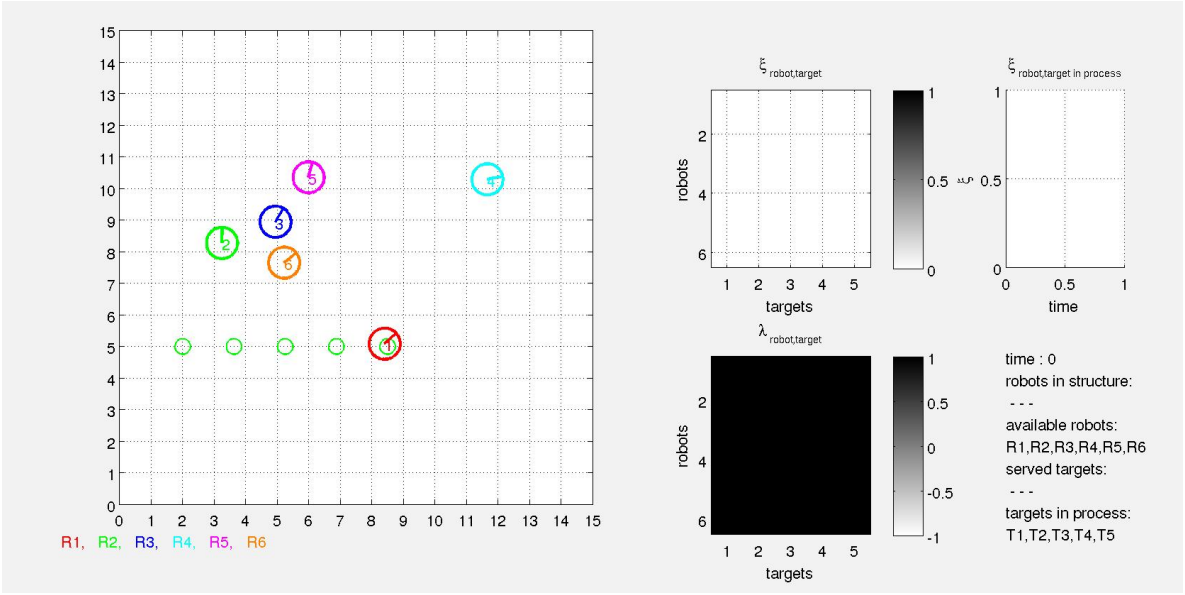

Abbildung 4.4.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 5 Zielen und 6 Robotern“ zu Beginn der Simulation

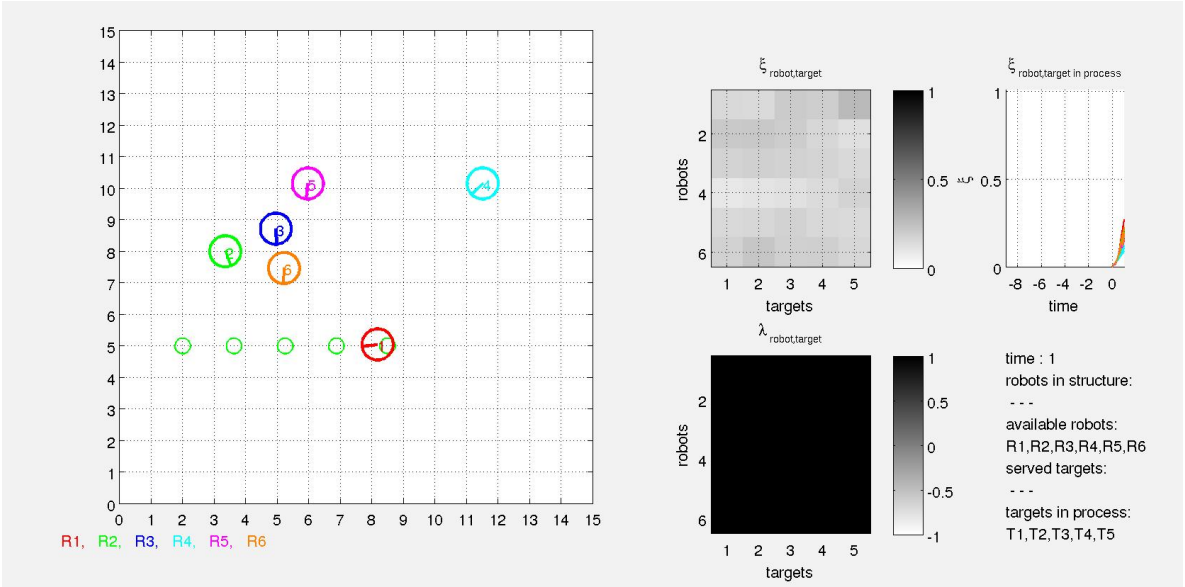

Abbildung 4.5.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 5 Zielen und 6 Robotern“ nach einer Simulationssekunde

Roboter 5 um Ziel 3 unterlegen ist, obwohl seine Ausgangsposition günstiger war. Das kann damit erklärt werden, dass Roboter 3 auch günstig zu den Zielen 2 und 4 stand und dadurch

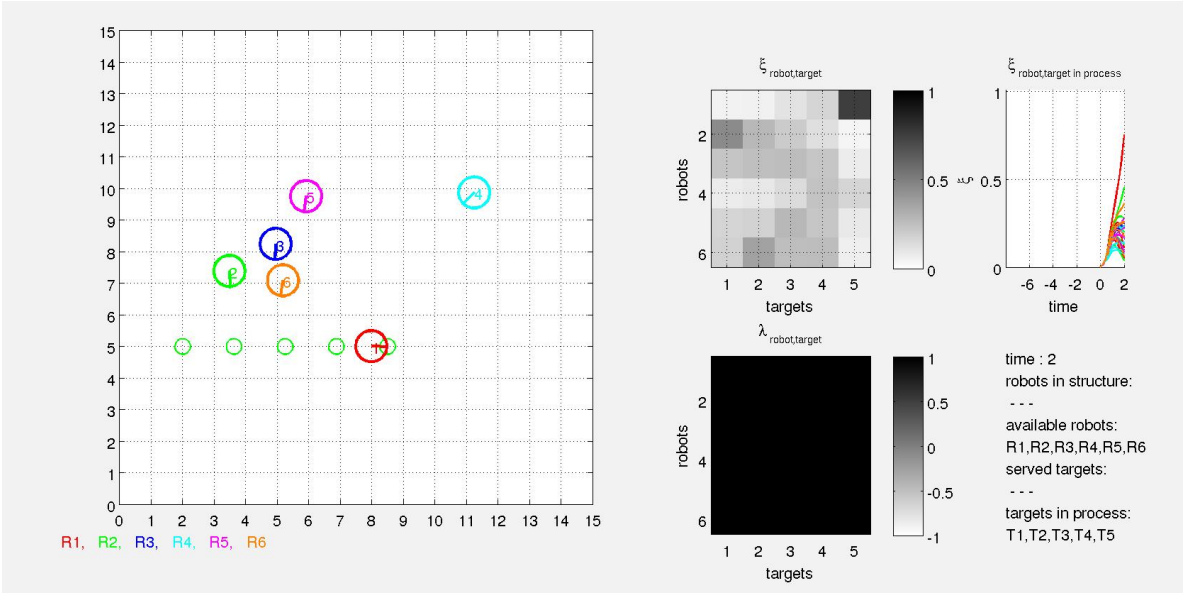

Abbildung 4.6.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 5 Zielen und 6 Robotern“ nach zwei Simulationssekunden

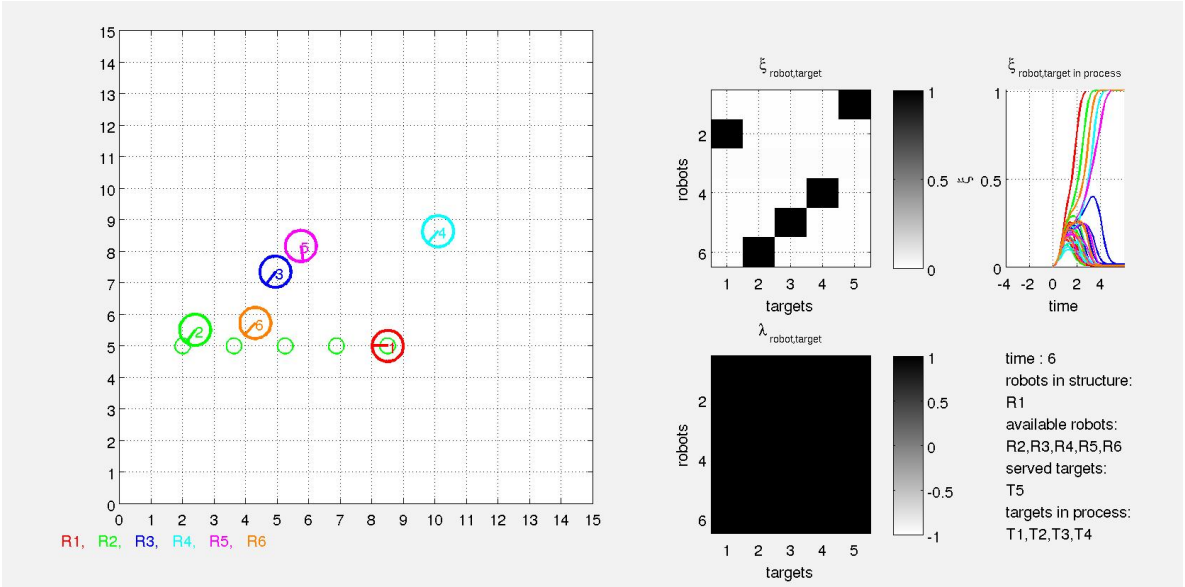

Abbildung 4.7.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 5 Zielen und 6 Robotern“ mit endgültiger Zielzuordnung

der Wert von $\xi_{3,3}$ nicht so schnell ansteigen konnte. Die Endpositionen sind in Abb. 4.8 zu sehen.

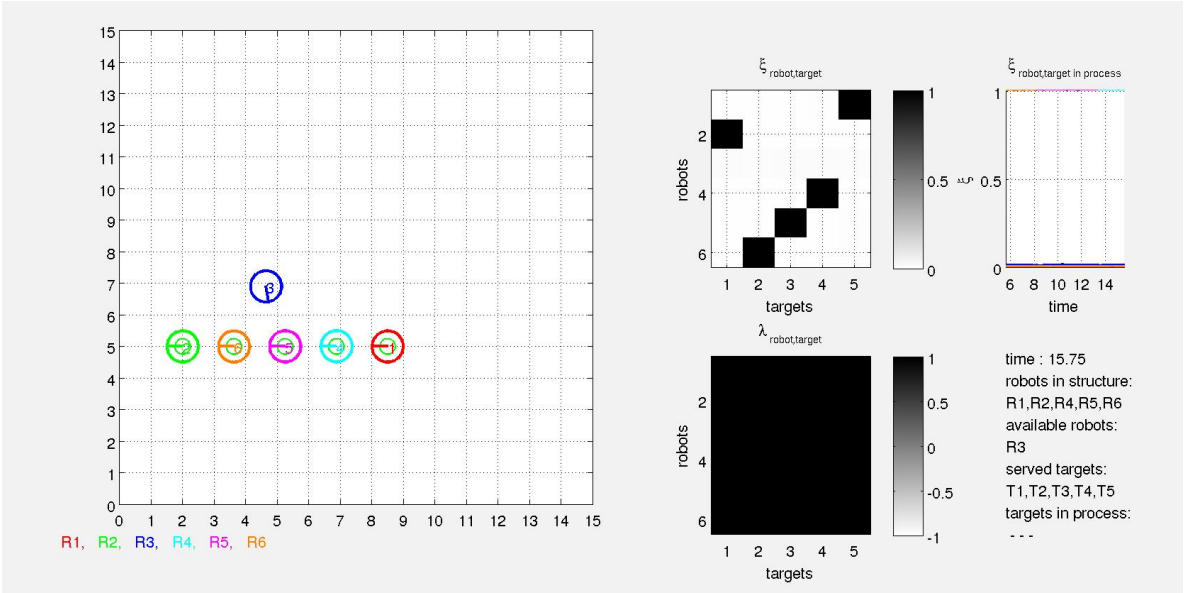

Abbildung 4.8.: Beispiel „Bildung einer Linie an vorgegebenen Positionen mit 5 Zielen und 6 Robotern“ am Ende der Simulation

4.2. Kritik an der Strukturbildung an vorgegebenen Raumpositionen

An der Strukturbildung an vorgegebenen Raumpositionen der Ziele kann man kritisieren, dass die Art der zu bildenden Struktur durch die Positionierung der Ziele vorgegeben wird. Sie ist damit kein Selbstorganisationsprozess der Roboter. Diese organisieren zwar die Aufgabenverteilung untereinander selbst, aber haben keinerlei Einfluss auf die Gestaltung der Struktur als solcher. Im Sinne einer selbstorganisierten Strukturbildung wäre es schöner, wenn die einzelnen Positionen der Ziele durch die Roboter während der Bildung der Struktur berechnet, oder besser gesagt, entstehen würden. Dieser Wunsch führt zur sequentiellen Strukturbildung.

5. Sequentielle Strukturbildung

Eine Möglichkeit, der Kritik an der Strukturbildung an vorgegebenen Raumpositionen zu begegnen, besteht darin, die möglichen Ziele an die Roboter zu binden. Das bedeutet, dass es zu den einzelnen Robotern sogenannte „Docking-Stationen“ gibt. Diese Docking-Stationen sind Ziele für alle anderen Roboter und bewegen sich mit dem Roboter, zu dem sie gehören. Sie sollen zunächst alle inaktiv geschaltet sein und erst dann aktiv werden, wenn sich der entsprechende Roboter in die zu bildende Struktur eingefunden hat. Durch dieses Vorgehen entsteht eine sequentielle Strukturbildung, bei der die einzelnen Roboter und damit auch die einzelnen Ziele nacheinander in die Struktur aufgenommen werden.
Wie viele solcher Docking-Stationen es bei den Robotern gibt und welche relativen Positionen und Ausrichtungen diese bezüglich der Roboter einnehmen, ist von der Aufgabe abhängig. Es ist selbstverständlich auch denkbar, dass Roboter ihre Docking-Stationen (relativ zu ihrer eigenen Position) bewegen und drehen können. Dies muss auch gewährleistet sein, um Strukturen aufbauen zu können, die nach ihrem Aufbau in der Lage sein sollen, sich in irgendeiner Form bewegen und zu verändern. Beispiel hierfür könnte ein aus einzelnen Robotern zusammengesetzter Roboterarm sein, der durch das Bewegen der Docking-Stationen seine Beweglichkeit erhalten könnte.

5.1. Sequentielles Bilden einer Linie

5.1.1. Idee des Verfahrens

Bei der sequentiellen Bildung einer Linie sollen die Roboter nacheinander der Linie angefügt werden. Dazu besitzt jeder Roboter genau eine Docking-Station an seinem hinteren Ende, an die jeder andere Roboter andocken kann. Diese Docking-Station ist so lange inaktiv, bis der Roboter selbst an einer anderen Docking-Station angedockt hat oder ein vorgegebenes Ziel erreicht hat. Dann wird die Docking-Station des Roboters aktiv geschaltet und so zu einem Ziel für jeden anderen Roboter. Nun soll der Selektionsprozess mittels der gekoppelten Selektionsgleichungen beginnen und der nächste Roboter an die bestehende Kette angefügt werden.
Damit auf diese Weise die entstehende Kette die Form einer Linie hat, müssen die Docking-Stationen die Richtung der andockenden Roboter bestimmen. Diese müssen genau die selbe Ausrichtung haben wie der vorhergehende Roboter.

5.1.2. Matlab Funktionen

cheackPosition

Die Funktion *cheackPosition* bekommt als Parameter die Vektoren der Roboter, der Ziele, die „activeTargets“, die „targtesInProcess“ und die „robotsInStructure“ übergeben, und sie liefert als Rückgabewerte die modifizierten Vektoren „activeTargets“, die „targtesInProcess“ und die „robotsInStructure“ sowie einen booleanschen Wert, der anzeigt, ob die gestellte Aufgabe erfüllt ist. Es soll immer nur ein einziges unbedientes Ziel geben. Dies ist entweder das im Raum vorgegebene Ziel oder die unbesetzte Docking-Station des letzten Roboters in der Kette. Da die Nummern der aktiven Ziele (also das im Raum vorgegebene Ziel und die Docking-Stationen der in der Kette befindlichen Roboter) in dieser Reihenfolge im Vektor „activeTargets“ abgelegt sind, ist das einzige unbediente Ziel das letzte Element dieses Vektors. Es braucht also innerhalb der Funktion *cheackPosition* nur überprüft werden, ob sich einer der Roboter auf der Position dieses Ziels (mit der korrekten Ausrichtung) befindet. Ist dies der Fall, so wird die Docking-Station dieses Roboters als neues letztes Element in den Vektor „activeTargets“ eingefügt werden. Darüber hinaus wird dieser Roboter in die Menge der „robotsInStructure“ aufgenommen und das neue aktive Ziel gesetzt. Wenn die Anzahl der Roboter, die sich in der Struktur befinden („robotsInStructure“), gleich der Anzahl aller Roboter ist, so wird *true* zurückgegeben, ansonsten *false*.

setLambda

Die Spalten der Matrix λ, die ein aktives Ziel oder eine aktive Docking-Station repräsentieren, müssen mit 1, diejenigen, die ein inaktives Ziel oder eine inaktive Docking-Station repräsentieren, mit -1 besetzt sein. Da keiner der Roboter die eigene Docking-Station als Ziel haben darf, muss die Hauptdiagonale der Matrix λ mit -1 besetzt sein.
Die Funktion *setLambda* realisiert dieses Verhalten und bekommt dazu die Roboter („bots“), die Ziele („targets“) und die aktiven Ziele („activeTargets“) übergeben.

5.1.3. Beispiele

Beispiel „Sequentielle Bildung einer Linie a“:

Bei dem Beispiel a gibt es 5 Roboter an zufällig gewählten Positionen. Man kann nach dem Einordnen eines jeden Roboters sehr deutlich sehen, dass bei der Roboter-Ziel-Zuordnung der zur Docking-Station am nächsten stehende Roboter dieses Ziel bedienen wird.

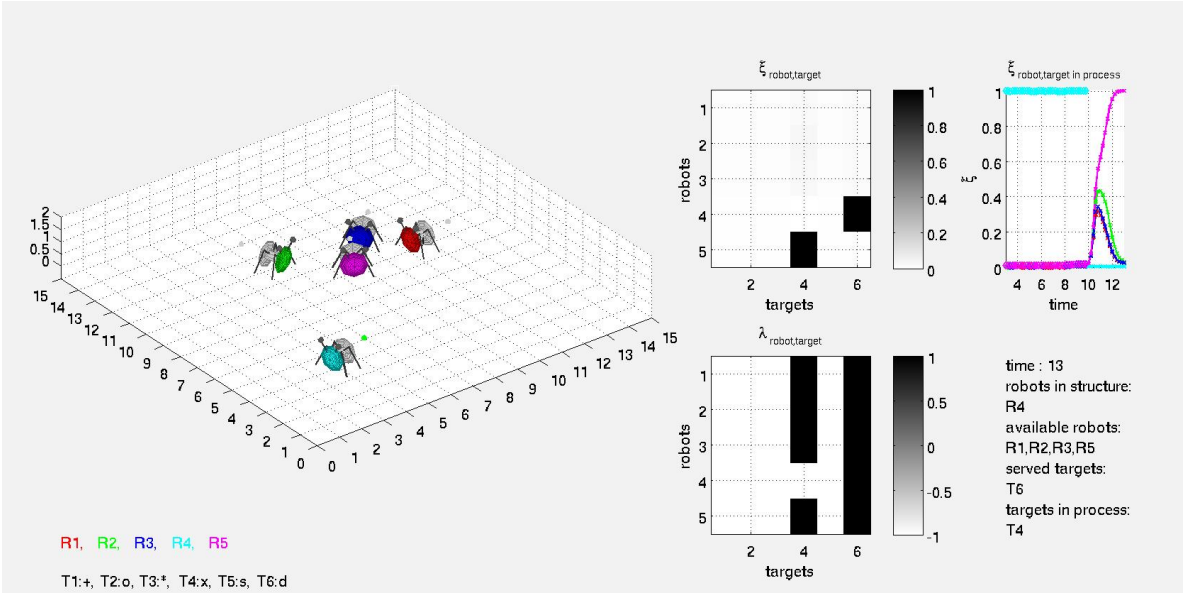

Abbildung 5.1.: Beispiel „Sequentielle Bildung einer Linie a“: Nach Einordnen des 1. Roboters

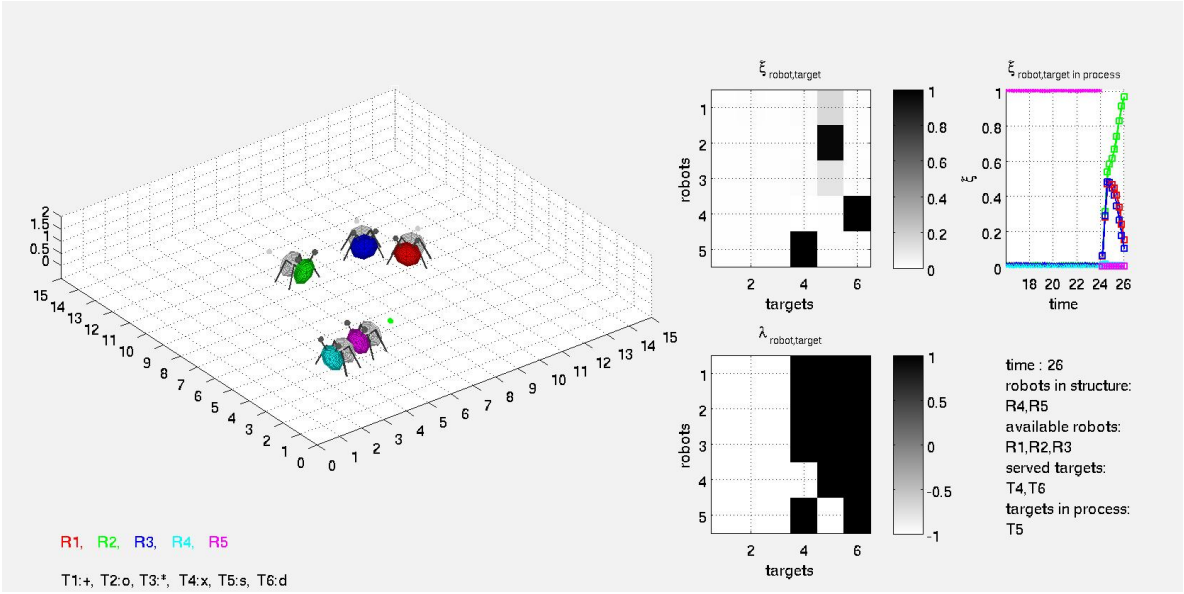

Abbildung 5.2.: Beispiel „Sequentielle Bildung einer Linie a“: Nach Einordnen des 2. Roboters

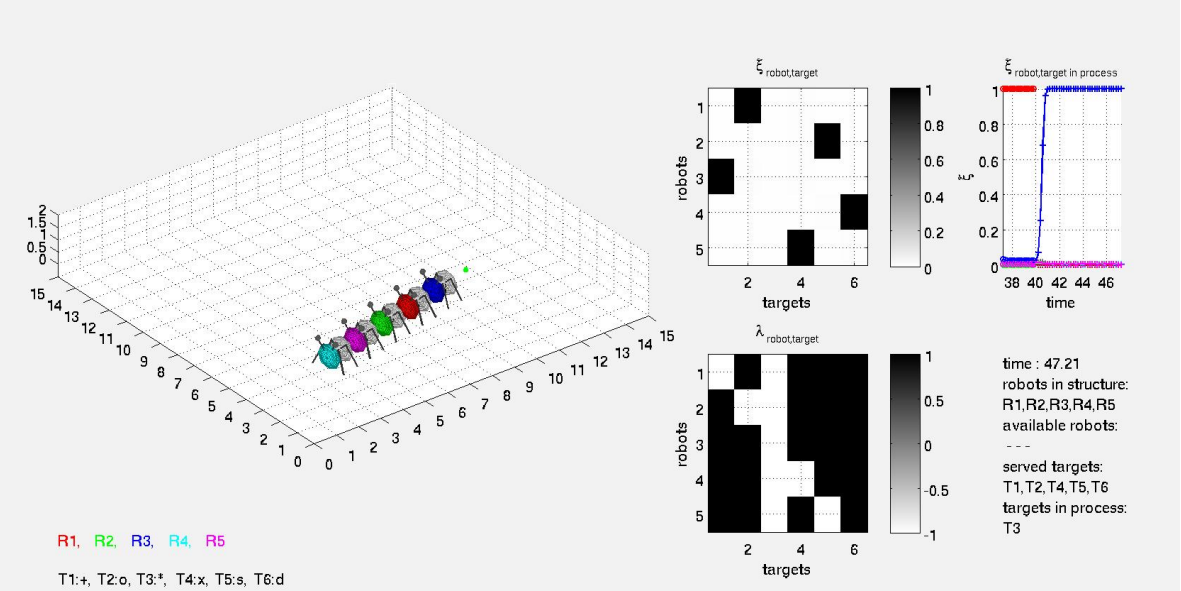

Abbildung 5.3.: Beispiel „Sequentielle Bildung einer Linie a“: Nach Einordnen aller Roboter

Beispiel „Sequentielle Bildung einer Linie b“:

Beispiel b zeigt die Situation mit 7 zufällig platzierten Robotern. In Abb. 5.4 kann man den Wettstreit der beiden Roboter Nr.1 und Nr.5 um das erste Ziel deutlich sehen, den Roboter Nr.5 durch die geringere Entfernung zu diesem Ziel für sich entscheiden kann. In Abb. 5.5 ist das schnelle Ansteigen des ξ-Wertes von Roboter Nr.1 in Bezug auf das Ziel Nr.5 (Docking-Station des Roboters Nr.5 und damit der $\xi_{1,5}$-Wert) zu sehen. Dies ist klar, da Roboter Nr.1 durch den vorangegangenen Wettstreit mit Roboter Nr.5 sehr nahe an dessen Docking-Station zum Stehen gekommen ist und somit die kürzeste Entfernung zu diesem Ziel aufweist. Darüber hinaus ist in Abb. 5.5 die Roboter-Ziel-Zuordnung für den dritten Roboter in der Linie zu sehen. In Abb. 5.6 ist die entstandene Linie nach rund 85 Simulationszeiteinheiten zu sehen.

5.1.4. Bewertung der Lösung

Das Bilden einer Linie funktioniert mit dieser Methode zufriedenstellend. Allerdings wird es zu Problemen kommen, wenn zu Beginn nicht nur ein, sondern mehrere Ziele im Raum festgelegt sein werden. Das Problem besteht darin, dass das Ziel mit der größten Ordnungsnummer als das aktive Ziel angesehen wird. Alle weiteren Ziele werden inaktiv bleiben. Alle Ziele, deren Ordnungsnummer größer ist als die Anzahl der Roboter, werden als aktive Ziele anzusehen. Dies löst zwar dieses Problem, aber es tritt ein weiteres auf: In der Funktion *cheackPosition* wird immer nur überprüft, ob sich ein Roboter ausreichend nah genug an dem letzten der aktiven Ziele befindet. Das bedeutet, dass nur Roboter, die das letzte der aktiven

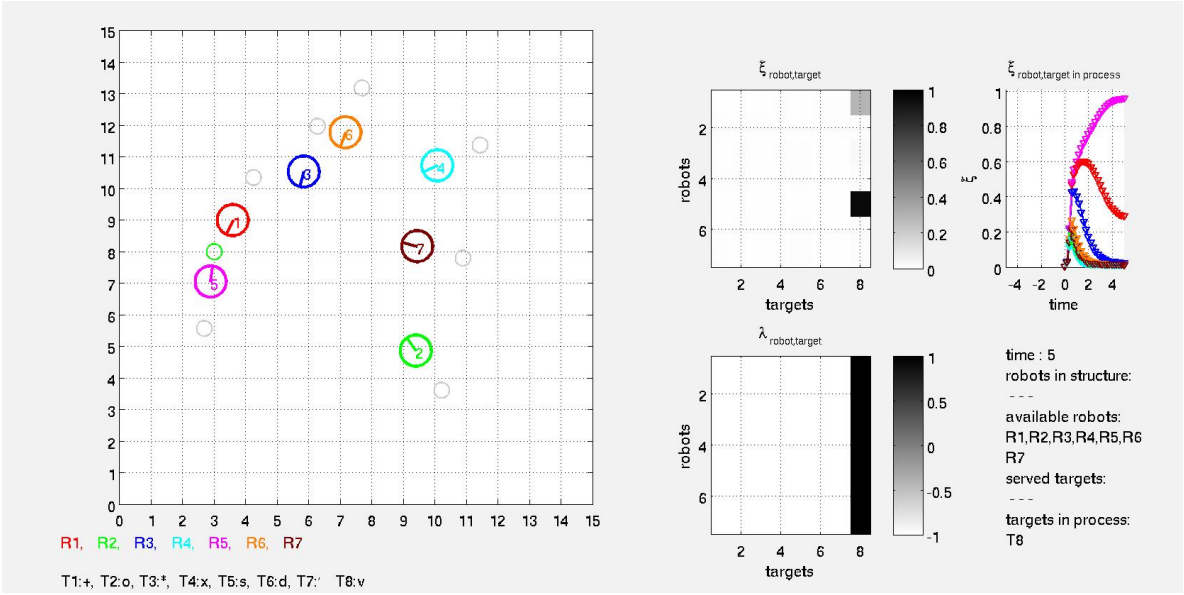

Abbildung 5.4.: Beispiel „Sequentielle Bildung einer Linie b“: Wettstreit zwischen Roboter Nr.1 und Roboter Nr.5 um das erste Ziel

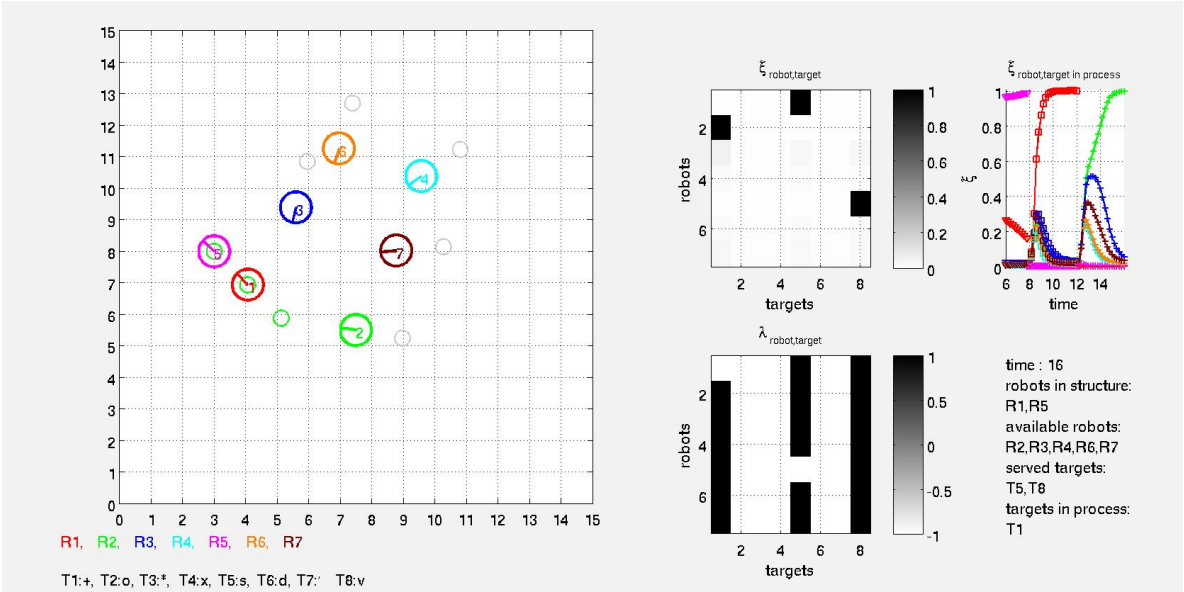

Abbildung 5.5.: Beispiel „Sequentielle Bildung einer Linie b“: Nach Einordnen des 2. Roboters

Ziele bedienen, als in der Struktur befindlich, und nur diese Ziele als bedient angesehen

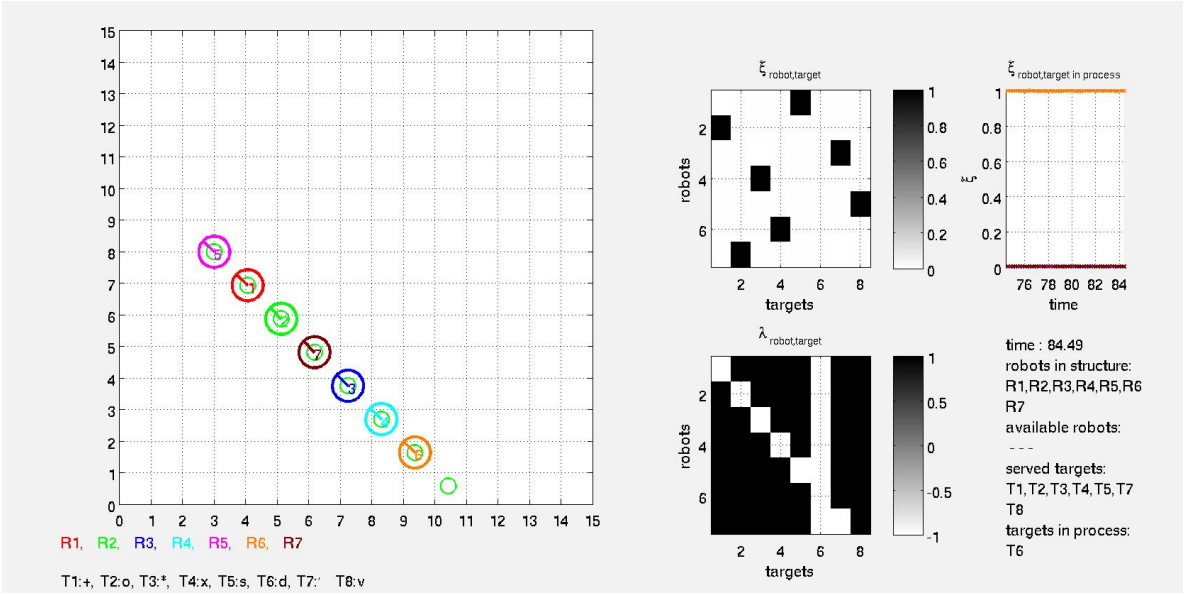

Abbildung 5.6.: Beispiel „Sequentielle Bildung einer Linie b“: Nach Einordnen aller Roboter

werden. Der Algorithmus muss also modifiziert werden, um mehrere Linien gleichzeitig in sequentieller Form aufbauen zu können.

5.2. Sequentielles Bilden beliebig vieler Linien

Wie bereits beschrieben, ermöglicht das bereits vorgestellte Verfahren nicht, mehrere Linien gleichzeitig sequentiell aufzubauen. Es soll nun hier eine Möglichkeit vorgestellt werden, wie dies erreicht werden kann. Damit wird auch die Konstruktion weiterer Strukturen z.B. von Sternen (mit drei Zacken) möglich sein, indem an drei entsprechend ausgerichteten Zielen gleichzeitig Linien gebildet werden sollen.

5.2.1. Idee des Verfahrens

Im Gegensatz zur sequentiellen Bildung genau einer Linie kann es nicht ausreichen, nur die jeweils letzte hinzugekommene Docking-Station daraufhin zu überprüfen, ob sich ein Roboter an ihr befindet oder nicht. Vielmehr müssen alle Docking-Stationen darauf überprüft werden. Dies macht aber nur bei den aktiven Docking-Stationen Sinn, da alle inaktiven ohnehin nicht zum Andocken freigeschaltet sind. Einen Roboter, der sich zufällig auf dieser Position befindet, als in der Struktur befindlich anzusehen, würde zu Fehlern, zumindest zu inkorrektem Verhalten führen.
Es ist also erforderlich, die Menge der aktiven Ziele in jedem Simulationsschritt sowohl zur Überprüfung, ob ein Roboter eines der aktiven Ziele erreicht hat, als auch zur Modifikation, wenn eine neue Docking-Station freigeschaltet wird, heranzuziehen.

5.2.2. Matlab Funktionen

cheackPosition

Die Funktion *cheackPosition* unterscheidet sich von der zur sequentiellen Bildung einer einzelnen Linie nicht in Bezug auf die Übergabeparameter oder die Rückgabewerte. Allerdings müssen zwei Schleifen ineinander geschachtelt arbeiten. Die eine Schleife iteriert über die aktiven Ziele, die zweite über die Roboter. Innerhalb der Schleifen wird für jede Kombination überprüft, ob sich der Roboter hinreichend genau an der Position des aktiven Ziels mit hinreichend genauer Ausrichtung befindet. Ist dies der Fall, werden die Mengen ebenfalls angepasst. Wichtig ist das Kopieren der Menge der aktiven Ziele in eine lokale Variable, da durch die Modifikation der Menge der aktiven Ziele ein Indizieren über dieser Menge zu Fehlern führen würde. Ein weiterer Unterschied besteht in der Modifikation der Mengen: Beim vorherigen Verfahren wurde das neue Element angefügt. Dies darf hier nicht gemacht werden, weil sonst Elemente (also Ziele oder Roboter) mehrfach in die entsprechenden Mengen aufgenommen würden. Daher muss der *union*-Befehl verwendet werden. Die Funktion liefert als Rückgabewert genau dann *true*, wenn die Anzahl der bedienten Ziele gleich der Anzahl der Roboter ist. Dies bedeutet nämlich, dass jeder Roboter ein Ziel bedient und somit kein Roboter mehr übrig ist.

setLambda

Um die Bildung beliebig vieler Linien zu ermöglichen, kann die Funktion setLambda unverändert übernommen werden.

5.2.3. Beispiele

Zur Simulation der Beispiele sind folgende Parameter gesetzt worden:

- $\kappa = 4$
- $\beta = 2$
- DT_XI_FACTOR von 2
- maximales Rauschen von 10^{-3}
- Schrittweite von 0.01

Beispiel „Bildung von zwei Linien mit zufällig gewählten Roboterpositionen“

Das erste Beispiel zeigt die Bildung von zwei Linien, also mit zwei im Raum vorgegebenen Zielen. An der Simulation sind insgesamt acht zufällig im Raum verteilte Roboter beteiligt. Die beiden Ziele haben die Koordinaten (2, 6, 0) und (11, 13, 0).

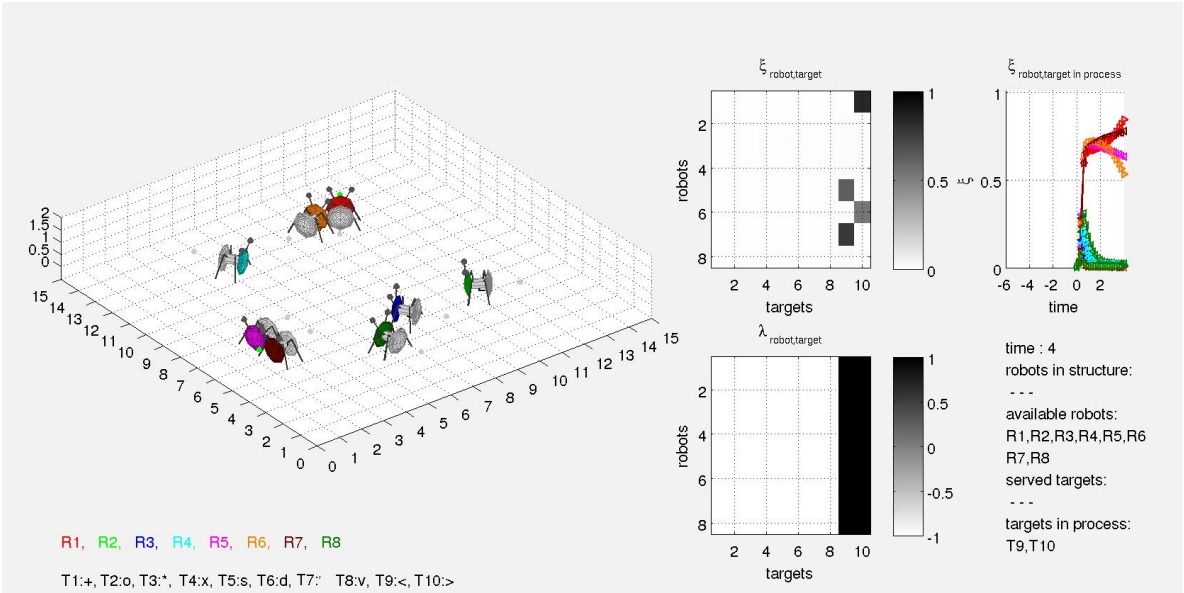

Abbildung 5.7.: Beispiel „Sequentielle Bildung zweier Linien“ nach 4 Simulationssekunden

In Abb. 5.7 ist der Wettstreit der Roboter 5 und 7 um das Ziel 9 und der Roboter 1 und 6 um das Ziel 10 zu sehen, den die Roboter 1 und 7 für sich entscheiden werden (siehe Abb. 5.8).

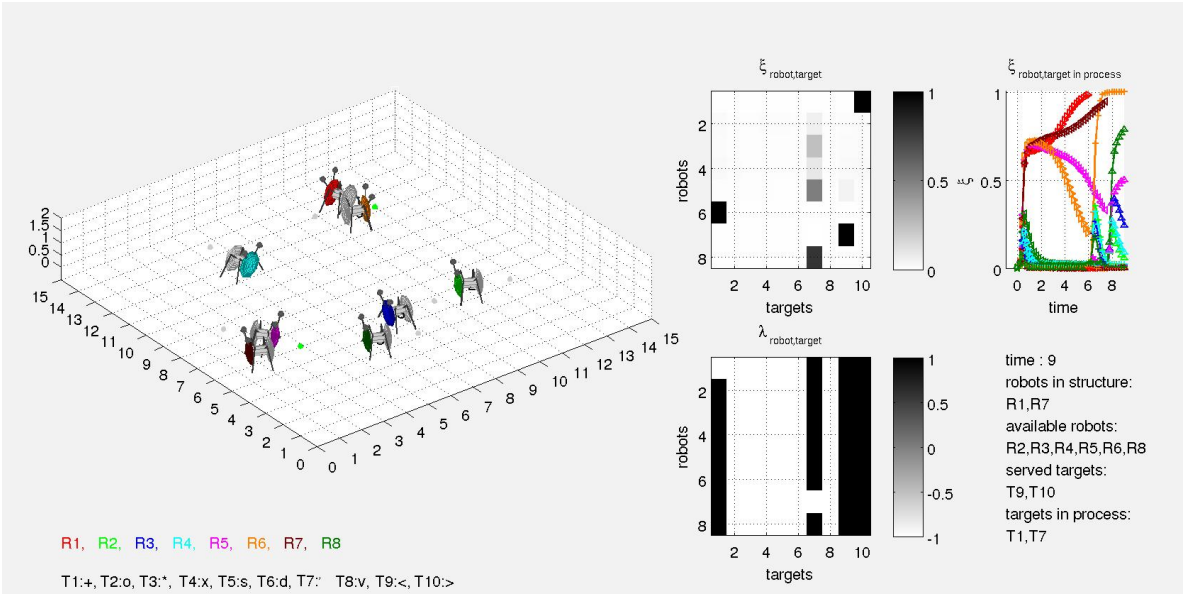

Abbildung 5.8.: Beispiel „Sequentielle Bildung zweier Linien“: Nach 9 Simulationssekunden

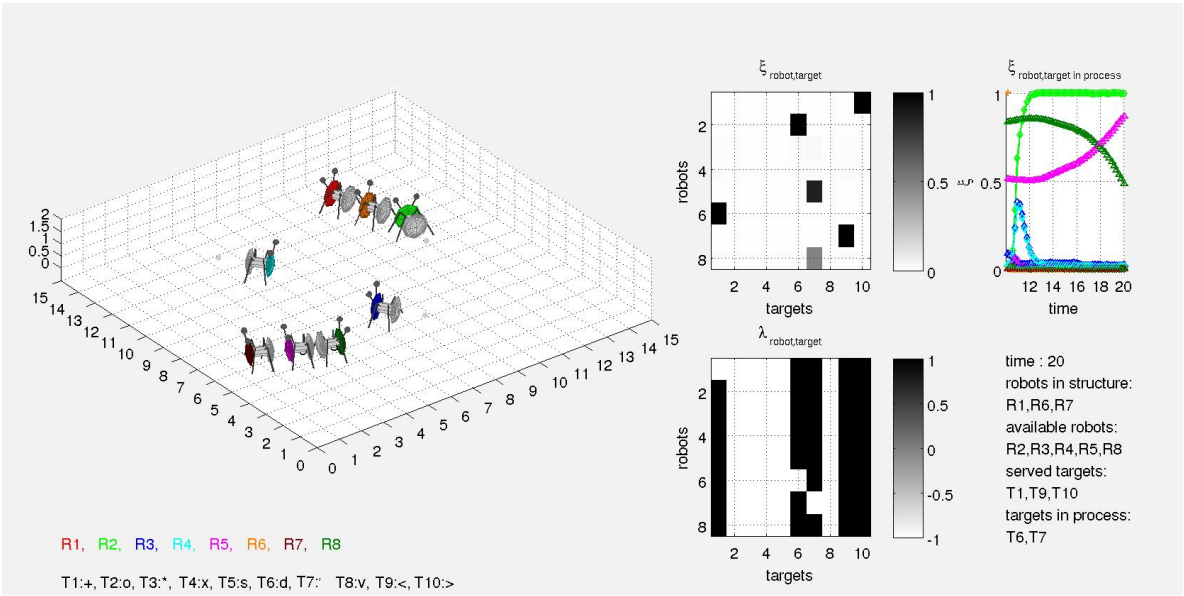

Abbildung 5.9.: Beispiel „Sequentielle Bildung zweier Linien“: Nach 20 Simulationssekunden

Hier ist zu erkennen, dass die unterlegenen Roboter günstig positioniert sind, um die neu freigeschaltete Docking-Station bedienen zu können. Interessant ist der beginnende Wettstreit der Roboter 5 und 8 um das Ziel 7. Obwohl Roboter 5 wesentlich günstiger positioniert ist als Roboter 8, kann der Wert $\xi_{5,7}$ anfangs nicht so stark anwachsen, da der Wert $\xi_{5,9}$ ebenfalls noch recht hoch ist. Dies führt zu einem Annähern beider Roboter an das Ziel 7, wobei sich Roboter 8 wesentlich schneller annähern kann. Roboter 5 hat das Ziel auf Grund seiner kurzen Entfernung jedoch schneller erreicht und der Wert $\xi_{5,7}$ übersteigt den Wert $\xi_{8,7}$. Roboter 8 wird abgedrängt und verliert letztendlich den Wettstreit um Ziel 7 (siehe Abb. 5.9).

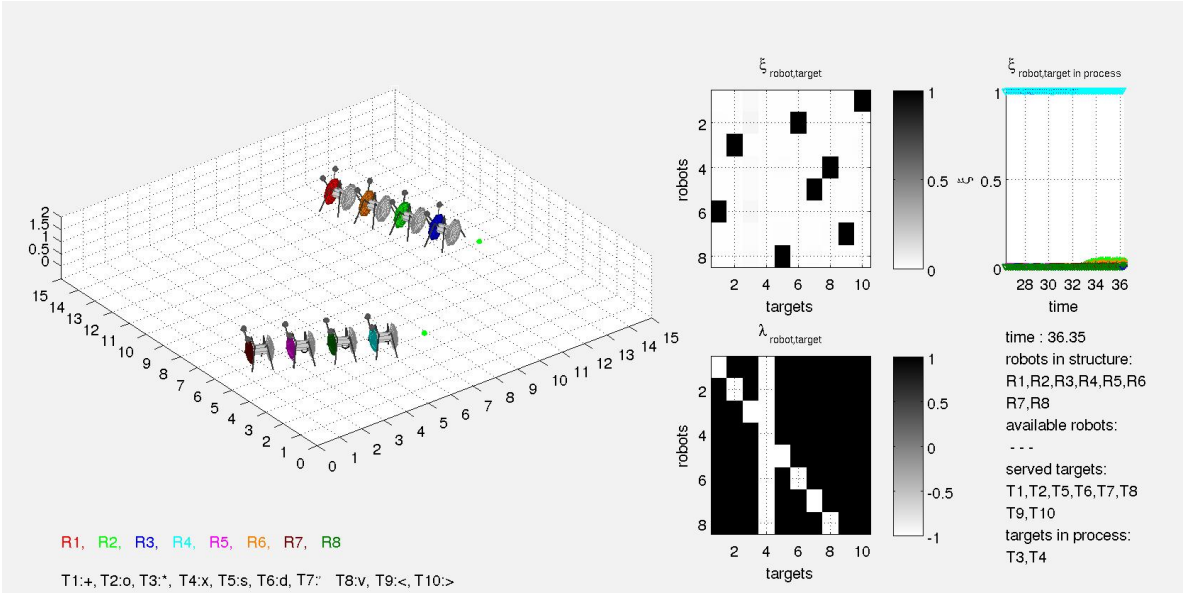

Abbildung 5.10.: Beispiel „Sequentielle Bildung zweier Linien“: Am Ende der Simulation

Die endgültigen Strukturen sind in Abb. 5.10 zu sehen. Zu bemerken ist, dass beide gebildete Linien gleiche Länge erreicht haben, ein Umstand, der an der relativ gleichmäßigen Verteilung der Roboter liegt.

Beispiel „Bildung eines Sterns mit zufällig gewählten Roboterpositionen“

Das zweite Beispiel ist die Bildung eines Sterns, der aus drei einzelnen, aber gleichzeitig sequentiell gebildeten Linien besteht. Auch hier sind die Anfangspositionen der nun zwölf Roboter zufällig im Raum gewählt, und es kommt auch hier durch die recht gleichmäßige Verteilung der Roboter zu einem gleichmäßigen Wachsen der drei Linien. Die Abbildungen 5.11 und 5.12 zeigen die Situationen am Anfang und am Ende der Simulation.

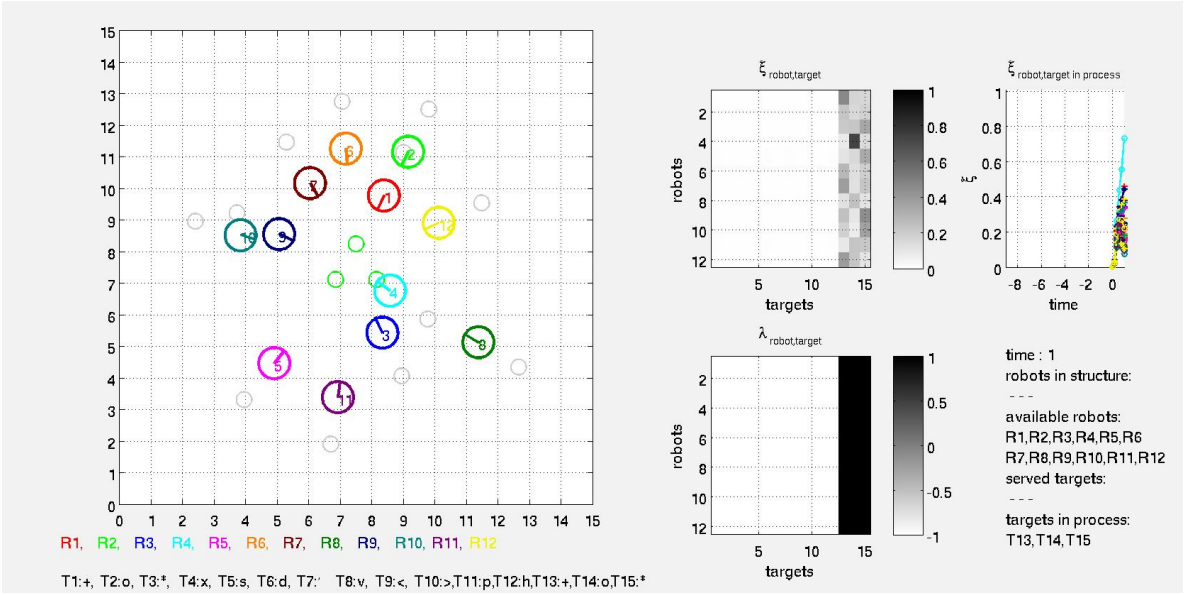

Abbildung 5.11.: Beispiel „Sequentielle Bildung eines Sterns“: Am Anfang der Simulation

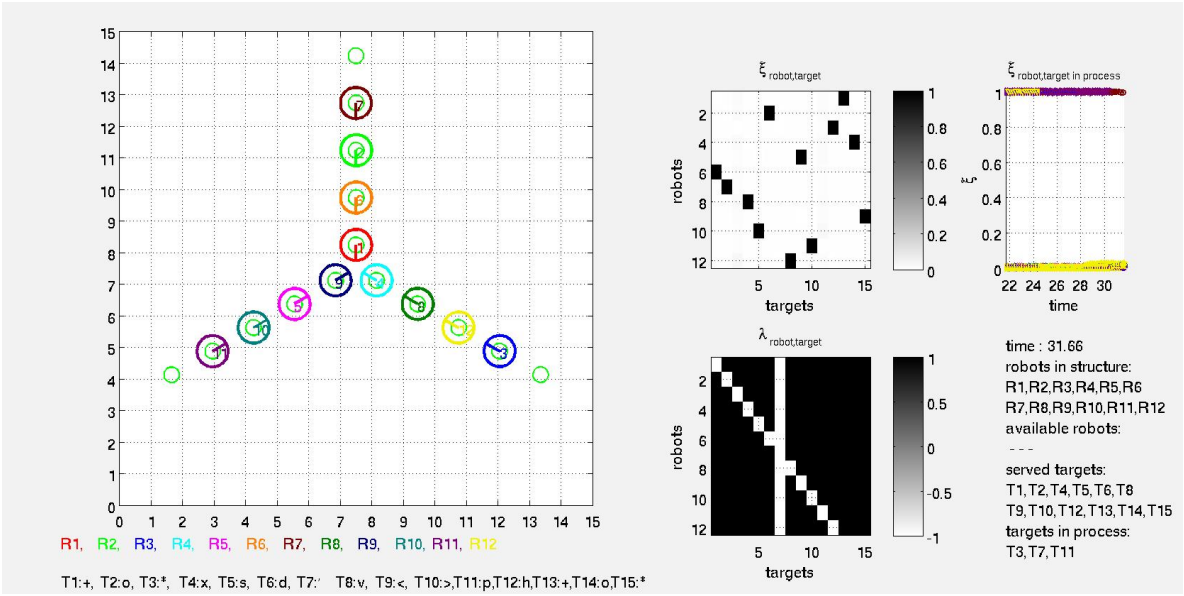

Abbildung 5.12.: Beispiel „Sequentielle Bildung eines Sterns“: Am Ende der Simulation

Beispiel „Bildung von mehreren Linien und Diskriminierung einiger Linien durch ungünstige Startpositionen der Roboter“

Durch sehr ungünstige Startpositionen der Roboter kann es dazu kommen, dass eine oder mehrere Linien in ihrer Bildung gegenüber anderen diskriminiert werden. Es kommt dazu, wenn die Wege der Roboter zu den freigeschalteten Docking-Stationen der einen Linie immer kürzer sind als die zu den anderen. Es ist klar, dass damit die eine Linie sehr viel schneller wachsen kann als die andere.

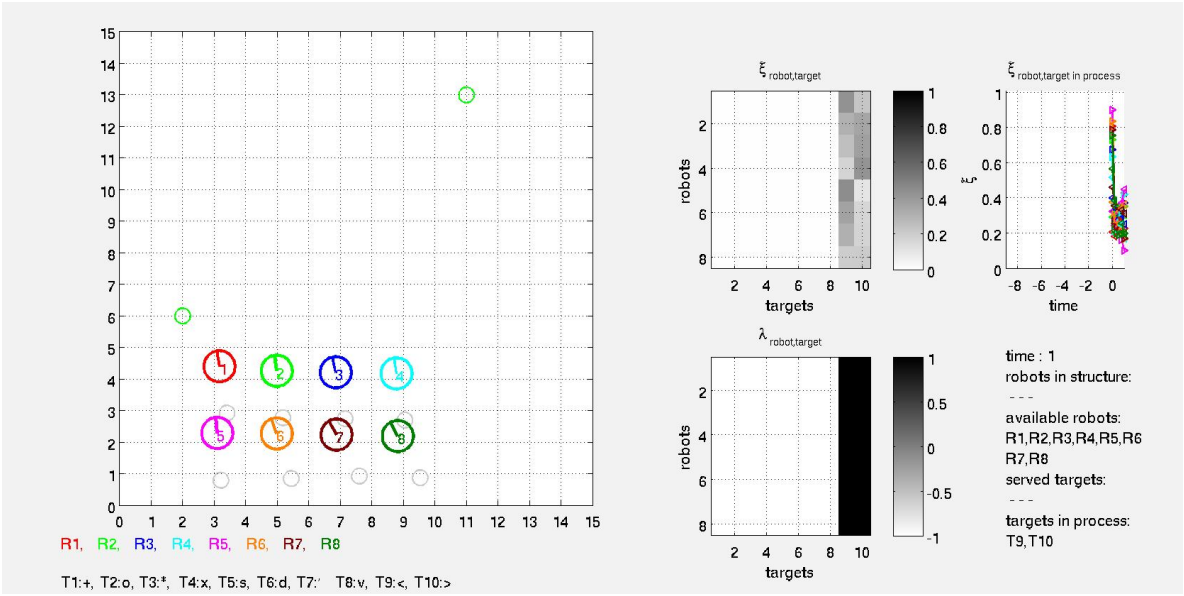

Abbildung 5.13.: Beispiel „Sequentielle Bildung zweier Linien mit Diskriminierung einer Linie“: Am Anfang der Simulation

In den Abbildungen 5.13 und 5.14 sind der Anfang und das Ende einer solchen Simulation zu sehen. Es sollen wie im ersten Beispiel zwei Linien sequentiell gebildet werden, nur befinden sich alle Roboter zu Beginn der Simulation sehr nahe an dem Ort, an dem die erste Linie liegen wird.
In einem zweiten Beispiel zur Diskriminierung soll ein Stern gebildet werden.

Die Abbildungen 5.15 und 5.16 zeigen hier den Anfang und das Ende der Simulation. Dieses Beispiel wirft die Frage auf, ob es möglich sein kann, mehrere Ketten gleicher Länge gleichzeitig in sequentieller Art zu bilden. Dies soll in einem späteren Teil der Arbeit diskutiert werden. Für weitergehende Abbildungen, die den Verlauf der Simulation genauer zeigen, soll hier auf die entstandenen Videos verwiesen werden.

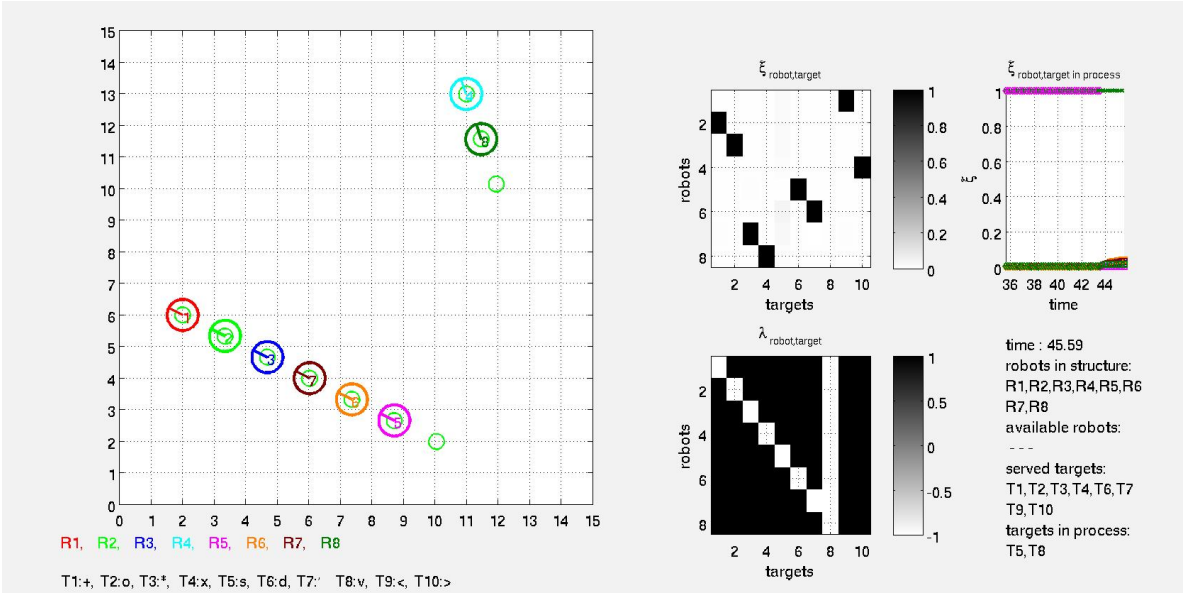

Abbildung 5.14.: Beispiel „Sequentielle Bildung zweier Linien mit Diskriminierung einer Linie“: Am Ende der Simulation

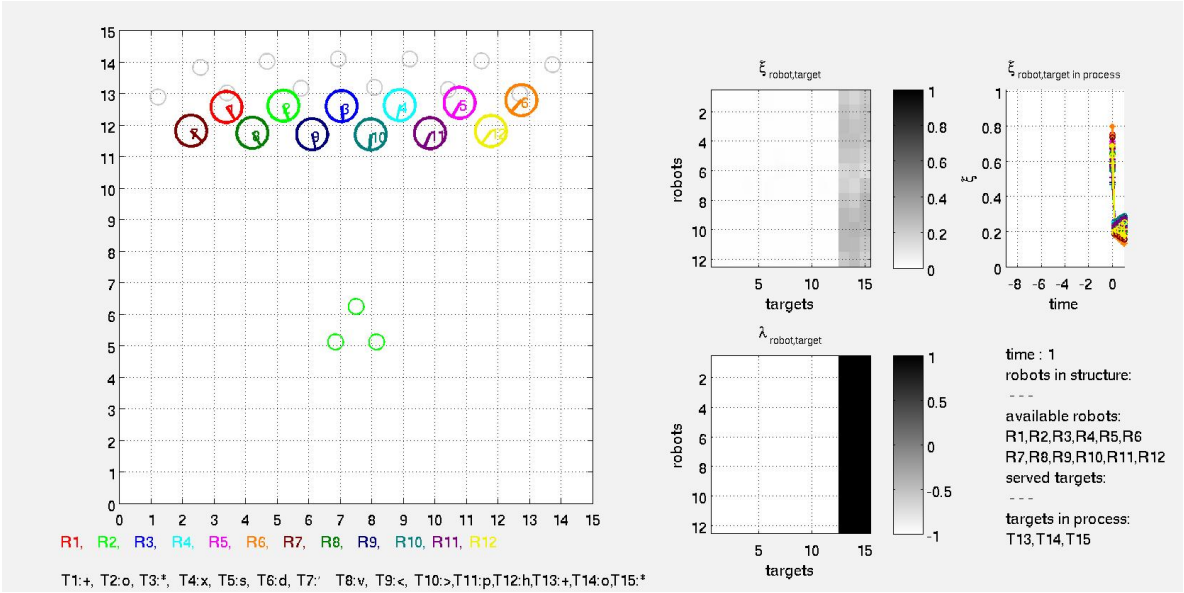

Abbildung 5.15.: Beispiel „Sequentielle Bildung eines Sterns mit Diskriminierung zweier Linien“: Am Anfang der Simulation

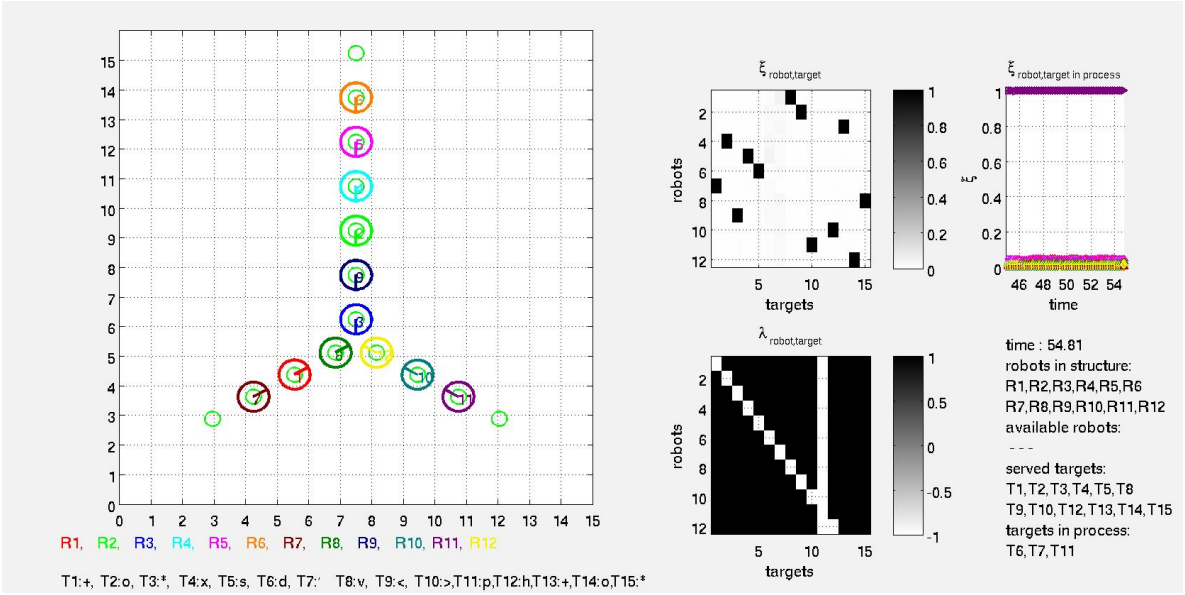

Abbildung 5.16.: Beispiel „Sequentielle Bildung eines Sterns mit Diskriminierung zweier Linien“: Am Ende der Simulation

5.2.4. Bewertung der Lösung

Es konnte gezeigt werden, dass durch eine relativ einfache Adaption der Funktion *cheackPosition* das Verfahren auch zum Bilden beliebig vieler Linien geeignet ist. Auch wurde eine Eigenschaft der Selbstorganisation gezeigt: Es ist zunächst nur sicher, dass sich Strukturen einer bestimmten Art bilden, aber ohne globaleres Wissen über das Vorhandensein weiterer solcher Strukturen kann nicht sichergestellt werden, dass alle sich bildende Strukturen vollständig gleiches Aussehen (also im Beispiel hier gleiche Länge der Linien) haben.
Von besonderem Interesse sollte in der Zukunft sein, auch Sterne zu bilden, ohne auf mehrere Startpositionen für mehrere Linien angewiesen zu sein. Dazu muss es für jeden Stern einen Roboter geben, der als Zentrum der Struktur dient und mehrere Docking-Stationen - für jede Zacke des Sterns eine - freischaltet. Da jeder der Roboter in der Lage sein sollte, diese Funktion zu übernehmen, muss jeder einzelne über weitere Docking-Stationen verfügen (im Winkel von 120°). Die weiteren Roboter werden sich dann in der gewohnten Form als Linien andocken und unverändert nur ihre hintere Docking-Station freischalten.

5.3. Sequentielle Bildung eines Sterns durch drei gleich lange Linien

Wie in 5.2 bereits beschrieben, können zwar Sterne durch geeignete Positionierung von mehreren Linien gebildet werden, aber es kann nicht sicher gestellt werden, dass deren Zacken gleiche Länge erreichen. Wie dies erreicht werden kann, soll in diesem Teil der Arbeit beschrieben werden.
Es werden zwei Ansätze beschrieben, von denen einer zum Erfolg führte. Denkbar sind selbstverständlich noch weitere Lösungsmöglichkeiten, die aber nicht verfolgt wurden.

5.3.1. Verfahren der Modifikation der Matrix α

Bisher wurden Ziele immer mit Hilfe der Matrix λ gesperrt oder freigegeben (über das Definieren von aktiven und inaktiven Zielen). Die Matrix α diente bisher ausschließlich dazu, nahe Ziele als attraktiv und ferne Ziele als unattraktiv zu bewerten. An diesen beiden Eingriffsmöglichkeiten des Selektionsprozesses soll sich auch nach wie vor nichts ändern, nur ist es durchaus denkbar, die Attraktivität eines Ziels nicht nur durch seine jeweilige Entfernung zu einem Roboter zu definieren.
Die Idee des Verfahrens sieht vor, eine Docking-Station, die sich am Ende einer Linie befindet, aber weiter vom Zentrum des Sterns entfernt ist als die anderen Docking-Stationen am Ende der anderen Linien, durch eine Korrektur der α-Werte im Selektionsprozess unattraktiver zu machen. Dies ist sinnvoll, da die Linie, deren Ende sich weiter vom Zentrum entfernt befindet als die Enden der übrigen Linien, offensichtlich bereits stärker angewachsen ist als die anderen. Wird die Docking-Station am Ende nun unattraktiver, so sollte diese Linie nicht so schnell weiter wachsen, als dies im herkömmlichen Verfahren der Fall wäre.
In Simulationen konnte auch der Effekt beobachtet werden, dass der Selektionsprozess durch diese Modifikation für eine solche Docking-Station verlangsamt abläuft. Allerdings muss beachtet werden, dass der Selektionsprozess im Verhältnis zur anschließenden Bewegung des Roboters (des Siegers) zu der jeweiligen Docking-Station eine relativ kurze Zeitdauer einnimmt. Das bedeutet, dass auch eine Verlangsamung des Selektionsprozesses dem Effekt der durch ungünstige Wege zu den Zielen bedingten ungleichmäßigen Bildung von Linien nicht entgegengewirkt werden kann. Außerdem kommt noch ein weiteres Hindernis hinzu: Bei der Bildung eines Sterns kann davon ausgegangen werden, dass fast immer nur um ein Ziel während des Selektionsprozesses gekämpft wird. Die übrigen Ziele sind entweder gar nicht freigeschaltet, oder der Selektionsprozess ist bereits abgeschlossen. Das bedeutet aber, dass alle verfügbaren Roboter um dieses eine Ziel kämpfen werden, unabhängig davon, wie klein der entsprechende Wert für $\alpha > 0$ auch gewählt werden mag. Hieraus folgt, dass es in jedem Fall einen Roboter geben wird, der dieses Ziel bedienen wird, und da es auch keine alternativen Ziele gibt, wird der Selektionsprozess auch nur sehr unwesentlich verlangsamt werden können. Aus diesen Gründen ist es erforderlich, nach anderen Alternativen zu suchen, um die gewünschte Aufgabe zu erfüllen.

5.3.2. Verfahren des gleichzeitigen Aktivierens der Linienenden

Eine Möglichkeit, die Aufgabe gleich langer Zacken eines Sternes zu erreichen, besteht darin, die Docking-Stationen der neu hinzugekommenen Roboter immer genau dann freizuschalten, wenn sie alle gleich weit vom Zentrum des Sterns entfernt sind.
Zunächst muss die Menge der Docking-Stationen bestimmt werden, die das Ende der Linien bilden. Dies sind aber genau die „targetsInProcess“. Diese Menge wird ohnehin bestimmt. Im zweiten Schritt wird die Position des Zentrums bestimmt und die Abstände der unbesetzten Docking-Stationen zu diesem Zentrum. Im dritten Schritt muss die Menge der Docking-Stationen gebildet werden, die ausgeschaltet werden sollen, weil sie sich in einer bereits längeren Kette befinden, also einen größeren Abstand zum Zentrum aufweisen. Im letzten Schritt wird diese Menge von der Menge der „targetsInProcess“ entfernt.

5.3.3. Matlab Funktionen

Alle Funktionen können unverändert übernommen werden - bis auf die Funktion *cheackPosition*.

cheackPosition

Entsprechend der Beschreibung der Arbeitsweise des Verfahrens muss die Implementierung der Funktion *cheackPosition* angepasst werden. Der erste Teil der Funktion bleibt dabei unverändert. Da die Positionen der vorgegebenen Ziele bekannt ist, kann die Position des Zentrums leicht berechnet werden. Ebenso ist die Berechnung der Abstände der unbesetzten Ziele zu der Position des Zentrums einfach. Im Anschluss daran muss das Minimum der Abstände bestimmt werden, da alle Docking-Stationen, die signifikant größere Abstände aufweisen, ausgeschaltet werden müssen. Danach wird die Menge der auszuschaltenden Docking-Stationen bestimmt und diese von der Menge der aktiven Ziele sowie der Menge „targetsInProcess“ entfernt.

5.3.4. Beispiele

Es soll ein Beispiel vorgestellt werden, das mit den selben Parametern erstellt wurde wie die Beispiele in 5.2.

Beispiel „Bildung eines gleichmäßigen Sterns mit ungünstigen Anfangspositionen der Roboter“

In diesem Beispiel wurden die Anfangspositionen der Roboter ebenso ungünstig gewählt wie in 5.2. Das Beispiel zeigt, dass mit dem vorgestellten Verfahren das Freischalten der Docking-Stationen so lange verzögert wird, bis alle drei Zacken des Sterns gleiche Länge

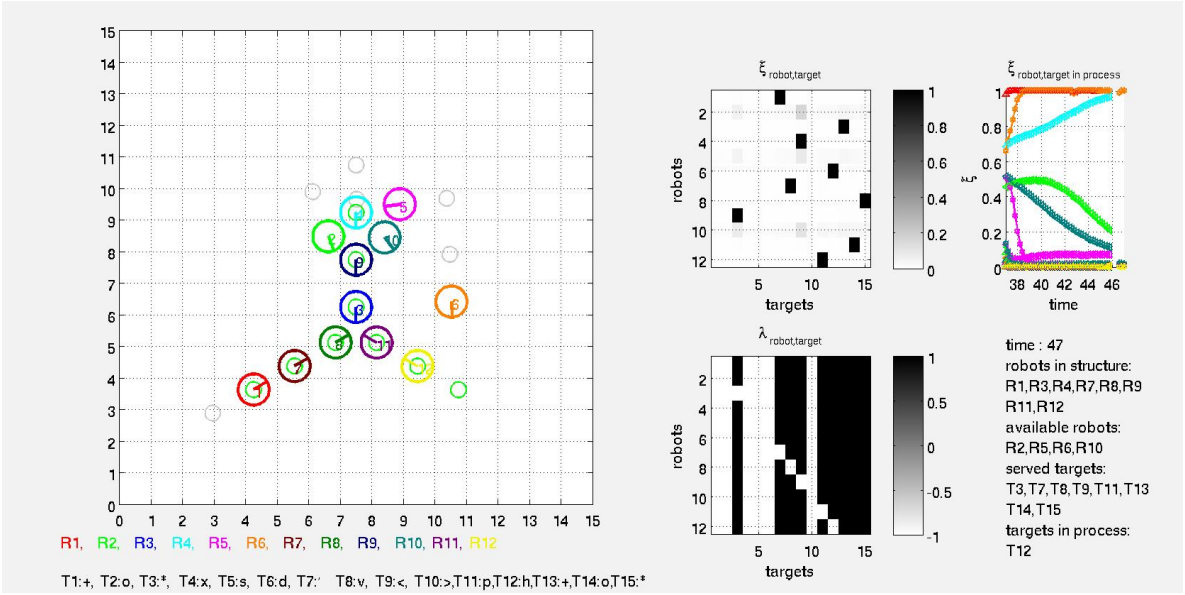

Abbildung 5.17.: Beispiel „Gleichmäßige Bildung eines Sterns“: Verzögern des Freischaltens

erreicht haben. Dies ist in Abb. 5.17 zu sehen. Obwohl Roboter 1 und 4 ihre Ziele erreicht haben, werden ihre Docking-Stationen nicht aktiv geschaltet, da sie weiter vom Zentrum des Sterns entfernt sind als die letzte Docking-Station der dritten Zacke.

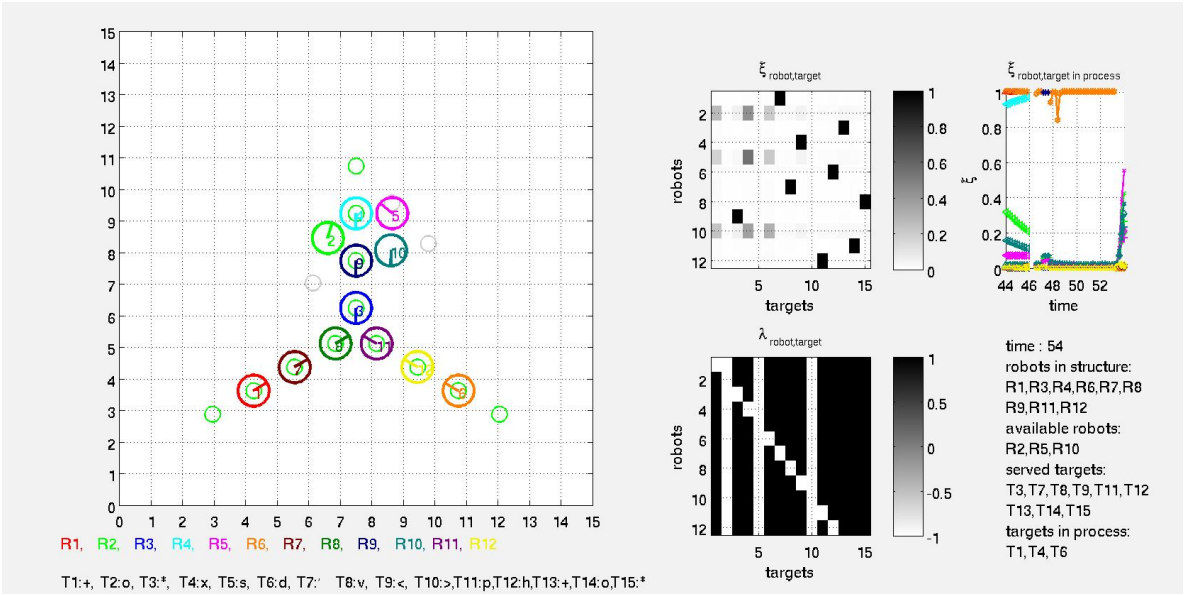

Abbildung 5.18.: Beispiel „Gleichmäßige Bildung eines Sterns“: Gleichzeitiges Freischalten

Erst mit dem Erreichen dieses Ziels durch Roboter 6 werden die drei Ziele 1, 4 und 6 gleichzeitig aktiv (siehe Abb. 5.18). Als Resultat entsteht ein Stern mit drei gleich langen Zacken.

5.3.5. Bewertung der Lösung

Neben der vorgestellten Lösung zur Bildung von gleichmäßigen Sternen sind durchaus noch viele weitere Lösungsansätze denkbar. Beispielsweise wäre es auch möglich, für jeden Roboter seine Ordnungsnummer innerhalb der Kette zu bestimmen. Es dürften nur dann die Docking-Stationen zu aktiven Zielen werden, wenn die höchsten Ordnungsnummern gleich sind. Alternativ könnte auch genau dann freigeschaltet werden, wenn die Anzahl der Ziele mit dieser Ordnungsnummer gleich drei ist. Als nachteilig kann an einer solchen Lösung angesehen werden, dass eine zusätzliche Information (nämlich die Ordnungsnummer) benötigt wird.
Vorteilhaft im Gegensatz zur vorgestellten Lösung wäre, dass das Verfahren nicht nur bei Sternen funktionieren würde, sondern bei beliebigen Konstrukten, bei denen gleich lange Linien gewünscht wären. Das vorgestellte Verfahren versagt bei allen Aufgaben, bei denen kein Zentrum zu erkennen ist, da dieses benutzt wird, um die Länge der einzelnen Linien zu bestimmen.

5.4. Verschieben einer sequentiell gebildeten Linie

Die bisherigen Strukturen wurden an ihrem Bestimmungsort gebildet und sollten dort verbleiben. Praxisnahe Beispiele zeigen jedoch schnell, dass es durchaus sinnvoll und erforderlich sein kann, eine Struktur an einem Ort aufzubauen und anschließend zu ihrem Einsatzort zu bewegen. Beipiele könnten hier Strukturen sein, zu deren Aufbau mehr Platz benötigt wird, als an ihrem Einsatzort zur Verfügung steht.
Aus diesem Grund soll in diesem Abschnitt der Bau von Linien vorgestellt werden, die nach dem Hinzufügen aller Roboter an eine andere Position verschoben werden.

5.4.1. Idee des Verfahrens

Eine Linie ist durch ein im Raum vorgegebenes Ziel und den darauf befindlichen ersten Roboter an einem Ort verankert. Wird dieses Ziel inaktiv, und es existiert ein anderes, aktives Ziel, so soll der erste Roboter dieses andere Ziel als neues Ziel annehmen und sich darauf zu bewegen. Der Roboter, der an dessen Docking-Station angedockt hat, wird dem ersten Roboter folgen, da er sein Ziel verfolgen wird. Ebenso werden alle weiteren Roboter folgen. Die übrigen Roboter werden das neue Ziel nicht als Ziel verfolgen, da sie eine bestehende starke Zielzuordnung zu der Docking-Station haben, an der sie angedockt haben.
Es muss allerdings sichergestellt werden, dass die Menge der aktiven Ziele immer nur wachsen kann, bis auf das erste vorgegebene Ziel, das als einziges Ziel aus dieser Menge entfernt werden darf. Der Grund für diese Einschränkung liegt im sequentiellen Bilden einer Linie: Würde die Menge der aktiven Ziele in jedem Simulationsschritt neu aufgebaut werden, so könnte sie auch kleiner werden. Dies hätte zur Folge, dass in dem Moment, in dem das erste vorgegebene Ziel aus der Menge der aktiven Ziele entfernt wird, der erste Roboter sein Ziel verliert, also nicht mehr an der Position eines Ziels steht und damit seine eigene Docking-Station inaktiv geschaltet würde. Ebenso würden alle weiteren Docking-Stationen inaktiv geschaltet werden und die Linie würde zusammenbrechen. Im Folgenden würde sich an dem neuen Raumziel eine neue Linie aufbauen.

5.4.2. Matlab Funktionen

Alle Funktionen außer der Funktion *cheackPosition* müssen nicht angepasst werden.

cheackPosition

Das Freischalten der Docking-Stationen funktioniert in der gleichen Art und Weise, wie dies auch bereits in 5.1 gezeigt wurde. Haben alle Roboter ein Ziel erreicht, so gibt es zwei Möglichkeiten:
1.: Die Linie ist gebildet und soll nun an das neue Ziel verschoben werden.
2.: Die Linie befindet sich bereits am neuen Ort und alle Roboter sind wieder zum Stehen gekommen.

Im ersten Fall muss das erste vorgegebene Ziel inaktiv und das zweite aktiv geschaltet werden. Im zweiten Fall kann die Simulation beendet werden. Um herauszufinden, ob es sich um den ersten oder den zweiten Fall handelt, kann die Menge der aktiven Ziele betrachtet werden: Befindet sich das erste vorgegebene Ziel in dieser Menge, handelt es sich um den ersten Fall, andernfalls um den zweiten.

5.4.3. Beispiele

Es sollen zwei Beispiele vorgestellt werden, die das Verschieben einer sequentiell gebildeten Linie veranschaulichen sollen. Dabei wurden folgende Parameter gewählt:

- $\kappa = 8$
- $\beta = 2$
- DT_XI_FACTOR von 4
- maximales Rauschen von 10^{-3}
- Schrittweite von 0.01

Beispiel a „Verschieben einer sequentiell gebildeten Linie“

Bei diesem Beispiel wird eine Linie von sechs Robotern an der Position (7.5, 7.5, 0) mit der Richtung (-1, 1, 0) gebildet und soll an das Ziel (1, 1, 0) mit der Richtung (-1, -1, 0) verschoben werden.

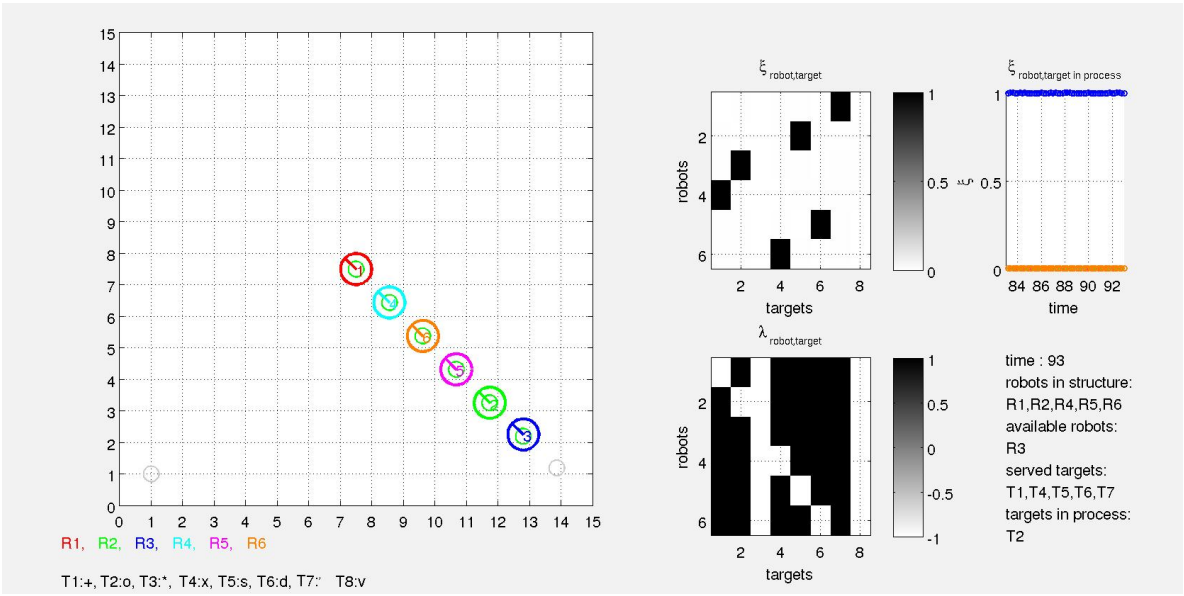

Abbildung 5.19.: Beispiel a „Verschieben einer sequentiellen Linie“: Fertig gestellter Linie

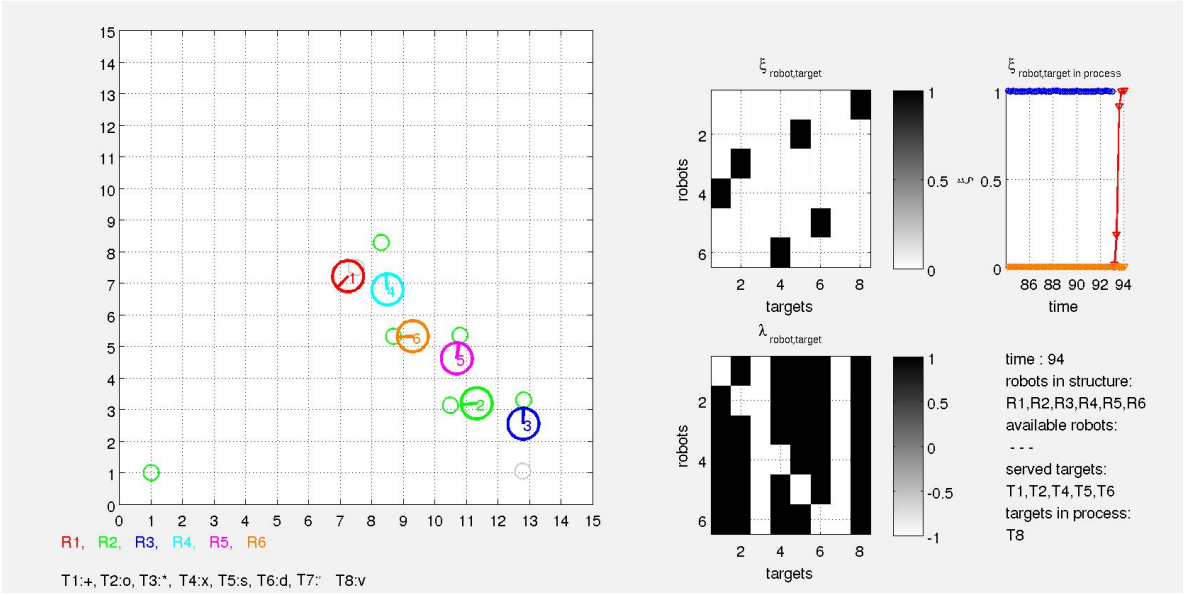

Abbildung 5.20.: Beispiel a „Verschieben einer sequentiellen Linie“: Beginn der Verschiebung

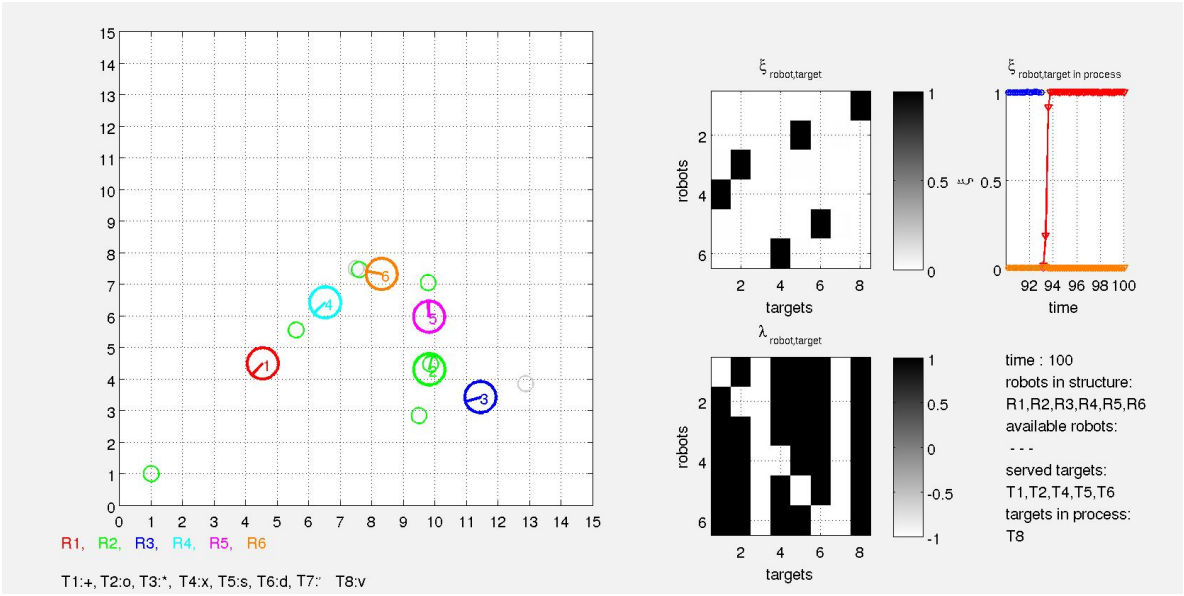

Abbildung 5.21.: Beispiel a „Verschieben einer sequentiellen Linie“: Situation während der Verschiebung

In Abb. 5.19 ist die fertig gestellte Linie am Punkt (7.5, 7.5, 0) zu sehen. Wie in Abb. 5.20 zu sehen ist, ist das ursprünglich vorgegebene Ziel inaktiv und das neue Ziel aktiv geschaltet worden. Darüber hinaus ist sehr deutlich eine beginnende Schwingung der Kette

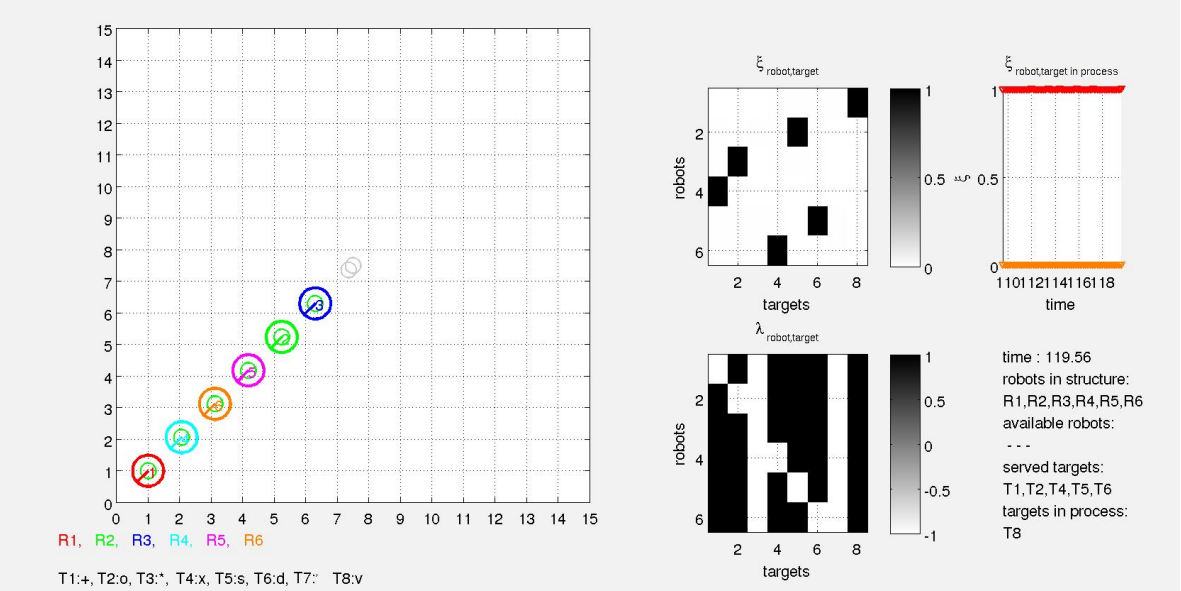

Abbildung 5.22.: Beispiel a „Verschieben einer sequentiellen Linie“: Nach der Verschiebung

festzustellen. Diese rührt daher, dass der erste Roboter auf das neue Ziel zudreht. Dadurch schwenkt seine Docking-Station aus der Linie der Kette aus und der zweite Roboter dreht sich in entgegengesetzter Richtung zum ersten. Entsprechend verhalten sich alle weiteren Roboter, und es kommt zu einer Schwingung. In Abb. 5.21 kann man sehen, dass sich der vordere Teil der Kette als Linie auf das neue Ziel zubewegt und der hintere Teil noch recht stark am Schwingen ist. Die endgültigen Positionen der Roboter sind in Abb. 5.22 zu sehen. Die Kette von Robotern ist am neuen Ziel angekommen und bildet dort wieder eine Linie.

Beispiel b „Verschieben einer sequentiell gebildeten Linie“

Im zweiten Beispiel soll eine Linie von vier Robotern von der Position (4, 10, 0) mit der Ausrichtung (-1, 0, 0) an die Position (10, 4, 0) mit entgegengesetzter Ausrichtung verschoben werden. Hierbei ist nicht nur die Schwingung am Anfang des Verschiebens durch das Eindrehen des ersten Roboters zu sehen, sondern auch noch eine Schwingung am Ende des Verschiebens durch erneutes Eindrehen des ersten Roboters auf die Richtung des neuen Ziels.

In Abb. 5.23 ist das anfängliche Schwingen der Kette zu erkennen. Roboter 4 ist sehr stark eingedreht, um das neue Ziel ansteuern zu können. Der schwingungslose Marsch der Roboterlinie auf das neue Ziel ist in Abb. 5.24 zu sehen. Mit Erreichen des neuen Ziels durch den ersten Roboter dreht dieser auf die Ausrichtung des Ziels ein, und die übrige Kette beginnt abermals zu schwingen (siehe Abb. 5.25).

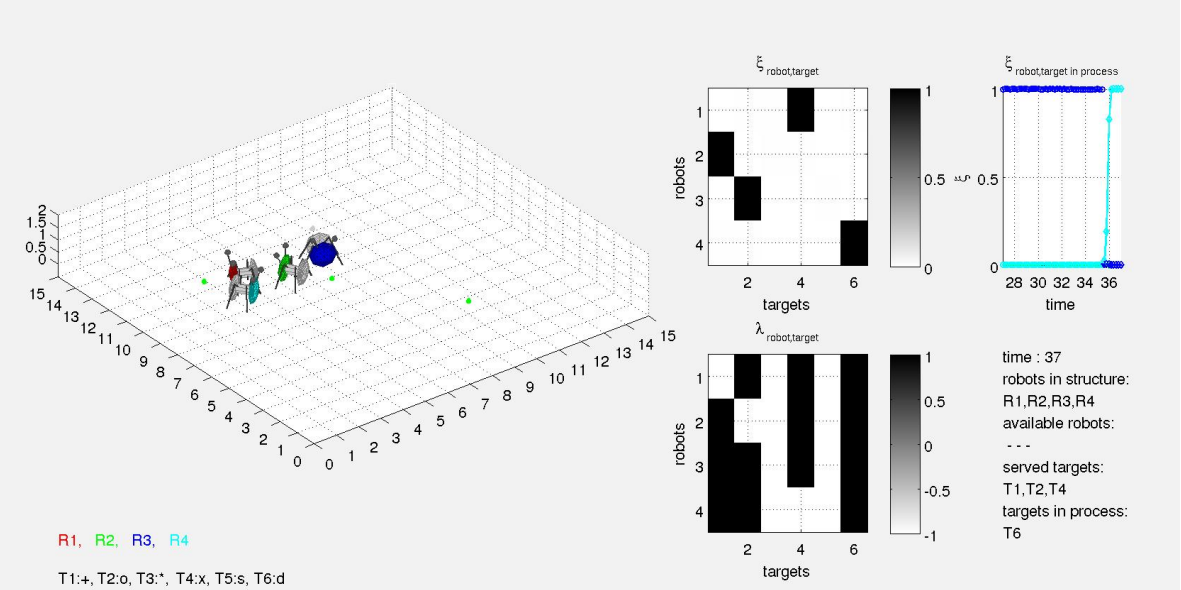

Abbildung 5.23.: Beispiel b „Verschieben einer sequentiellen Linie“: Eindrehen auf das neue Ziel

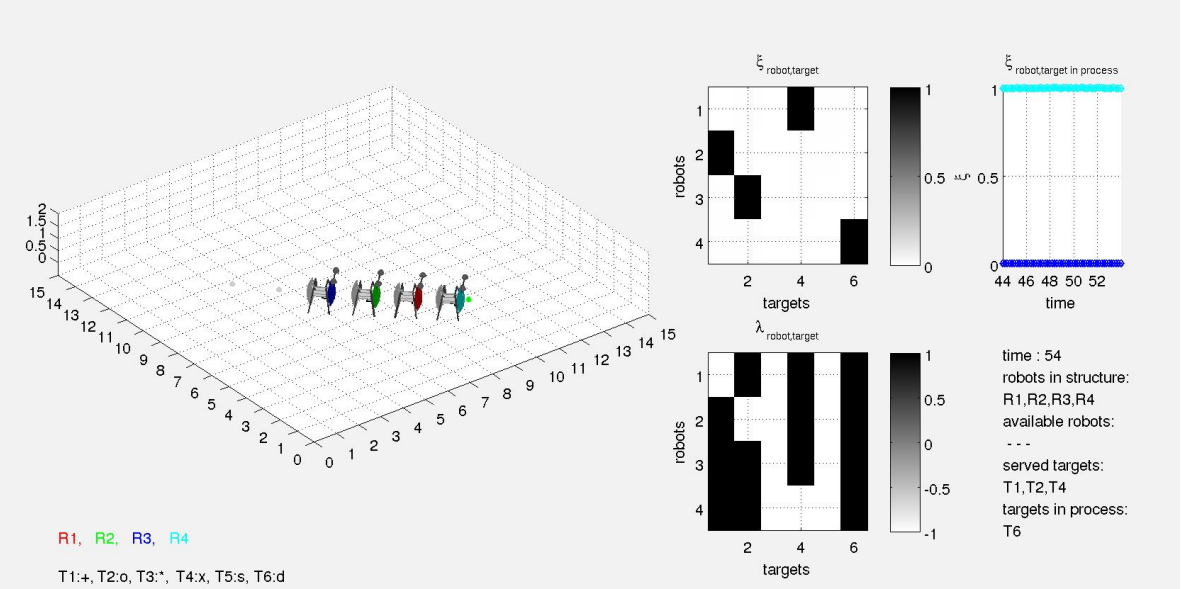

Abbildung 5.24.: Beispiel b „Verschieben einer sequentiellen Linie“: Situation während der Verschiebung

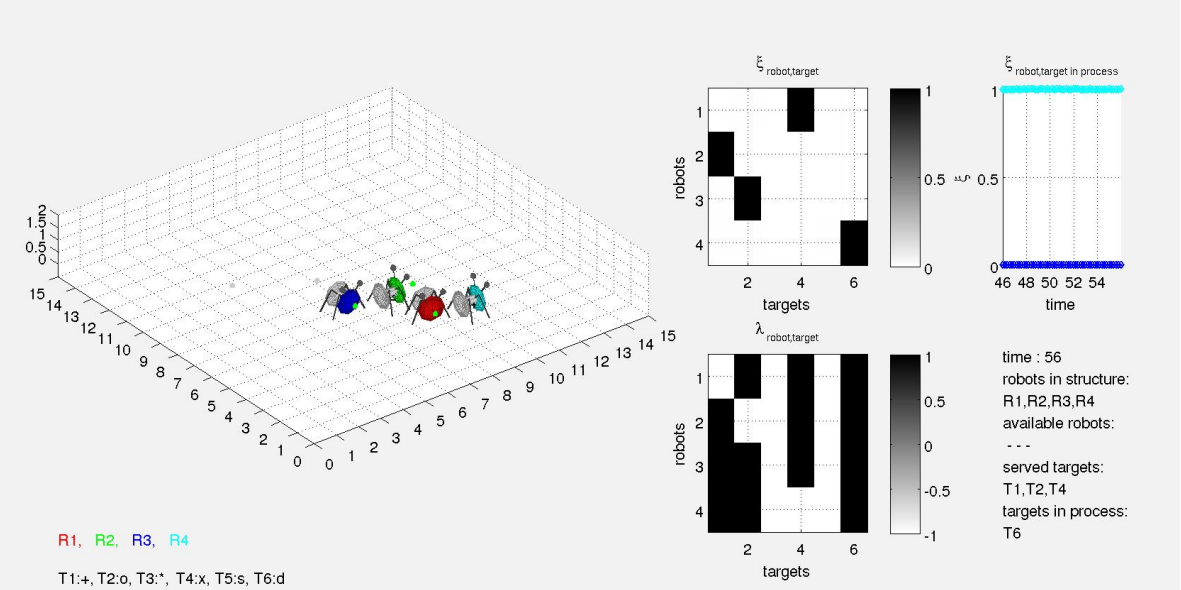

Abbildung 5.25.: Beispiel b „Verschieben einer sequentiellen Linie“: Erreichen des neuen Ziels

5.4.4. Bewertung der Lösung

Dadurch, dass die Menge der aktiven Ziele nicht immer neu gebildet wird, ist die Zielzuordnung sehr stabil. Das hat andererseits aber auch zur Folge, dass eine Störung von außen diese Zuordnung nicht verändern kann bzw. die Zuordnung sehr statisch bleibt und sich auf Störungen nicht anpassen kann. Die aktive Docking-Station des ersten Roboters beim Verschieben der Linie ist eine Ausnahme von der Regel, dass die Docking-Stationen nur freigeschaltet werden sollen, wenn sich der Roboter an einem Ziel befindet. Eine Lösung des Verschiebens einer Linie ohne eine solche Ausnahme in Kauf nehmen zu müssen, wird später im Abschnitt 6.4 vorgestellt.

5.5. Sequentielles Bilden von Brücken zwischen Punktpaaren

Alle vorgestellten sequentielle Strukturen wurden an einem Punkt im Raum gebildet und wuchsen in einer Richtung. Interessant können aber auch Strukturen sein, deren Bau an mehreren Stellen gleichzeitig beginnt, und die dann aufeinander zuwachsen. In der Natur sind solche Strukturbildungen (Heilungsprozesse bei Verletzungen, Bildung von Säulen in Tropfsteinhöhlen) ebenso anzutreffen wie in der Technik (Brückenbau von zwei Ufern, Tunnelbau von beiden Seiten). In diesem Abschnitt soll der Brückenbau von zwei Seiten aus betrachtet werden. Dazu soll jede Brücke, die als eine Linie gebildet werden soll, von zwei Zielen im Raum aus aufeinander zuwachsen. Wenn in der verbleibenden Lücke nur noch Platz für einen letzten Roboter besteht, soll nur eine Docking-Station genau in die Mitte zwischen die beiden Roboter verschoben werden, damit die Abstände gleichmäßiger ausfallen.

5.5.1. Idee des Verfahrens

Das hier vorgestellte Verfahren zur Bildung von Brücken basiert auf zwei wesentlichen Punkten: Erstens muss eine der beiden Docking-Stationen, die am Ende, kurz vor dem Schließen der Brücke, zu nah beieinander liegen, inaktiv geschaltet werden. Dabei wird die Docking-Station des Roboters mit der kleineren Id ausgewählt, um so sicherzustellen, dass niemals ein im Raum vorgegebenes Ziel inaktiv geschaltet werden kann. Der zweite Punkt ist die Verschiebung der verbliebenen Docking-Station in die Mitte zwischen die beiden Roboter. Dazu muss für jeden Roboter eine sog. „dockingStationScale“ eingeführt werden. Diese ist ein Faktor, mit dem jede Docking-Station nach hinten verschoben werden kann. Im Normalfall ist dieser Faktor gleich 1.

5.5.2. Matlab Funktionen

cheackPosition

Die Funktion *cheackPosition* ist stark auf die neue Aufgabe angepasst worden. Ausgehend von der Funktion *cheackPosition*, die bereits von der gleichzeitigen sequentiellen Bildung beliebig vieler Linien bekannt ist, wurden folgende Punkte modifiziert:

- Einführen der Variable „dockingStationScale“ auch als Ein- und Ausgabeparameter.
- Bei der Überprüfung, ob ein aktives Ziel bedient wurde oder nicht, muss unterschieden werden, ob es sich bei diesem Ziel um ein vorgegebenes Ziel handelt oder um eine Docking-Station. Handelt es sich um eine Docking-Station muss überprüft werden, ob der zugehörige „dockingStationScale“= 1 ist. Ist dies der Fall, so darf die Docking-Station des Roboters, der diese Docking-Station bedient, freigeschaltet werden. Wenn nicht, ist diese Docking-Station bereits die letzte, die in die Brücke hineinpasste und somit darf keine weitere mehr freigeschaltet werden.

- Betrachten aller Paare von aktiven Zielen und testen, ob diese zu nah beieinander stehen. Ist dies der Fall, so gibt es verschiedene Fälle:
  1. Fall: Die Ziele liegen so dicht beieinander, dass sie als ein einzelnes Ziel betrachtet werden können. In diesem Fall muss ein Ziel inaktiv geschaltet werden.
  2. Fall: Die Ziele liegen zu dicht beisammen, um beide zu bedienen; sie liegen in einer Linie und sind aufeinander ausgerichtet. Das bedeutet, dass eines ausgeschaltet und das andere verschoben werden muss.

scene_207 / scene_208

In den Funktionen *scene_207* und *scene_208* muss der Parameter „dockingStationScale“ an die Funktion *cheackPosition* übergeben und wieder entgegengenommen werden. Mit diesem Parameter müssen die Positionen der Docking-Stationen entsprechend berechnet werden, und er muss selbstverständlich auch zu Beginn der Funktion initialisiert werden (mit 1). Die beiden Funktionen *scene_207* und *scene_208* unterscheiden sich nur dadurch, dass die erste zufällige Startpositionen den Robotern zuweist und die zweite die Startpositionen aus einer externen Datei einliest.

5.5.3. Beispiele

Es soll hier nur eines der berechneten Beispiele vorgestellt werden. Für weitere Beispiele sei auf den Anhang bzw. die der Arbeit angefügte CD verwiesen. Bei allen Simulationen wurden folgende Parameter verwendet:

- $\kappa = 4$
- $\beta = 2$
- DT_XI_FACTOR von 2
- eine Schrittweite von 0.01

Beispiel „gleichzeitiger sequentieller Bau von zwei Brücken“

In diesem Beispiel sollen gleichzeitig zwei Brücken gebaut werden. Jede der beiden Brücken soll folgende zwei Brückenköpfe haben (Koordinaten und Ausrichtung):

- 1. Brückenkopf der 1. Brücke (Ziel 13): Position (3, 7, 0), Ausrichtung (-1, 1, 0)
- 2. Brückenkopf der 1. Brücke (Ziel 14): Position (7.5, 2.5, 0), Ausrichtung (1, -1, 0)
- 1. Brückenkopf der 2. Brücke (Ziel 15): Position (5, 10, 0), Ausrichtung (-1, 0, 0)
- 2. Brückenkopf der 2. Brücke (Ziel 16): Position (12, 10, 0), Ausrichtung (1, 0, 0)

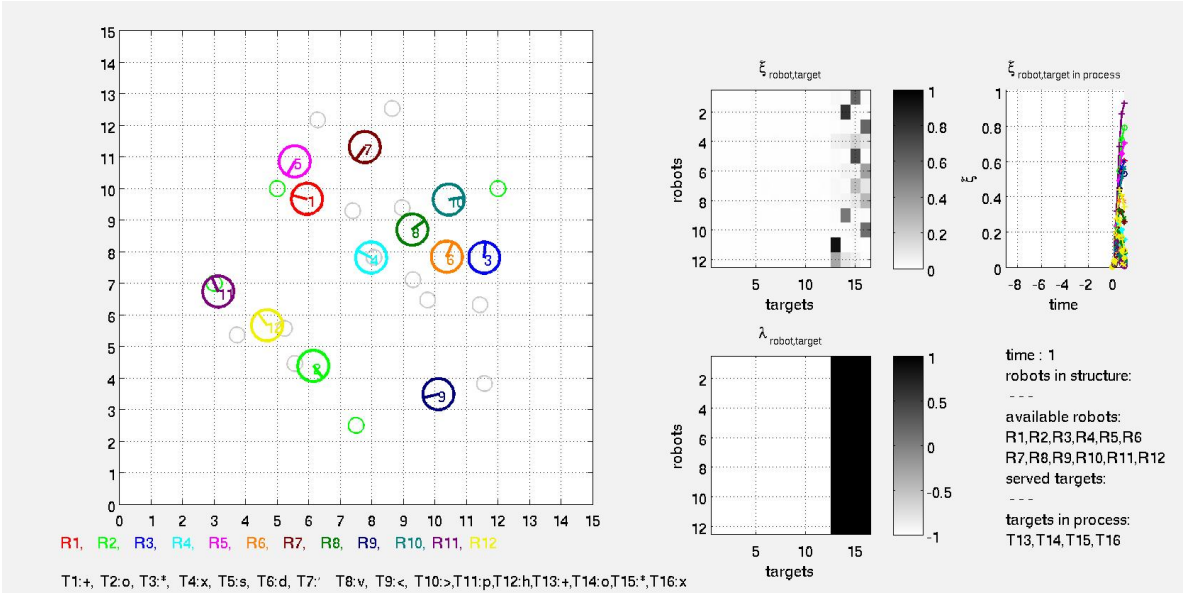

Abbildung 5.26.: Beispiel „gleichzeitiger sequentieller Bau von zwei Brücken“ zu Beginn der Simulation

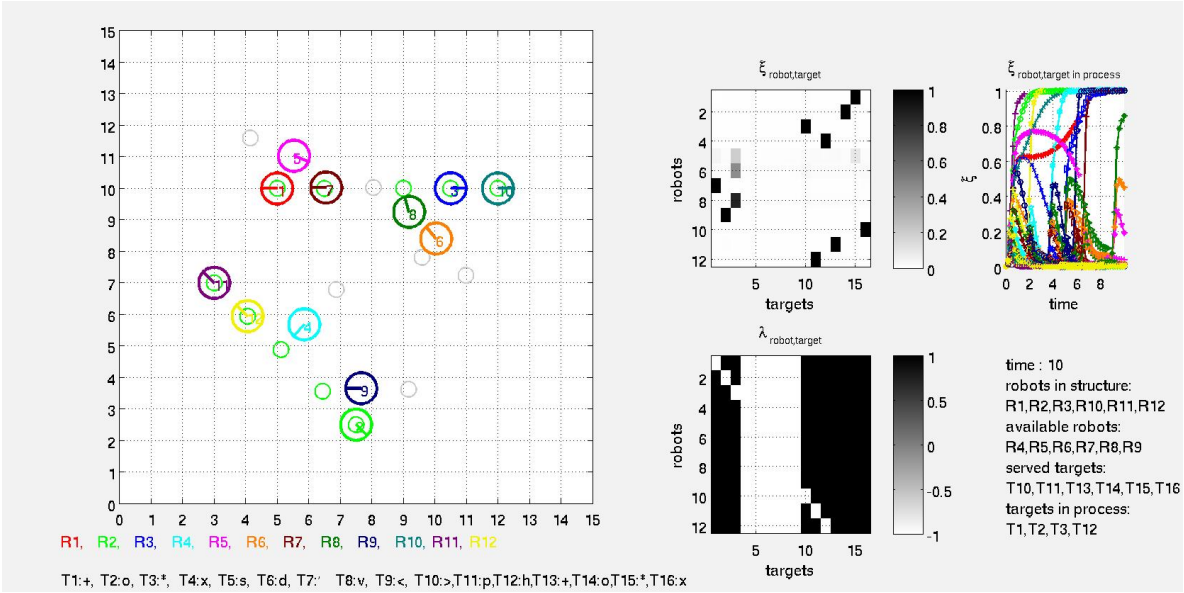

Abbildung 5.27.: Beispiel „gleichzeitiger sequentieller Bau von zwei Brücken“ vor der Verschiebung einer Docking-Station

Die Anzahl der zu der Simulation verwendeten Roboter beträgt 12. Deren Startpositionen sind zufällig gewählt.

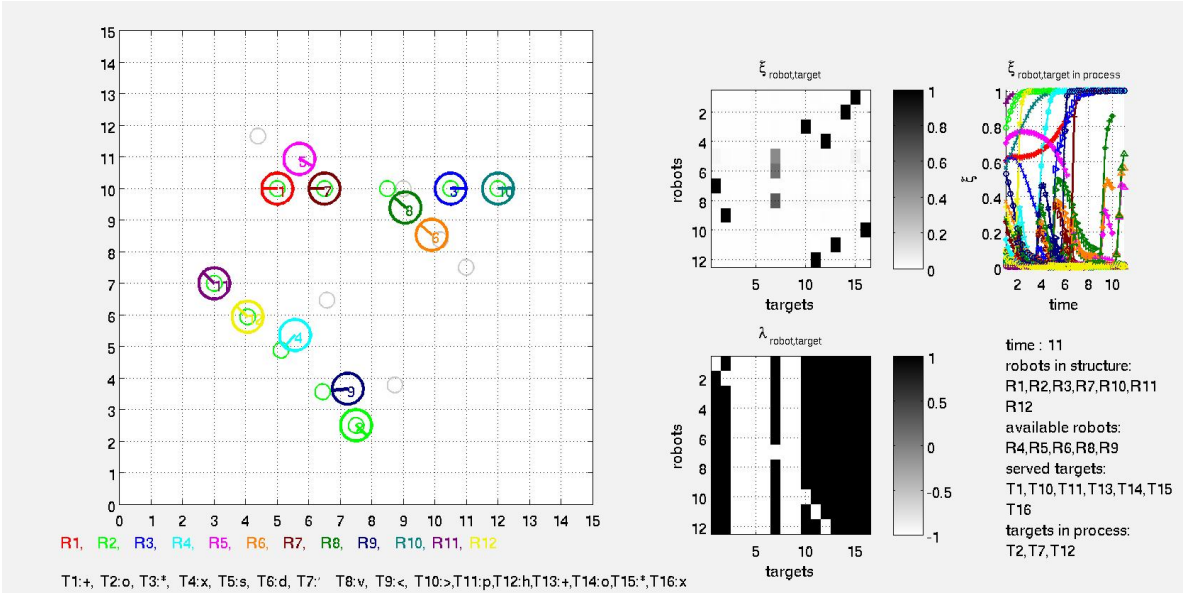

Abbildung 5.28.: Beispiel „gleichzeitiger sequentieller Bau von zwei Brücken“ nach der Verschiebung einer Docking-Station

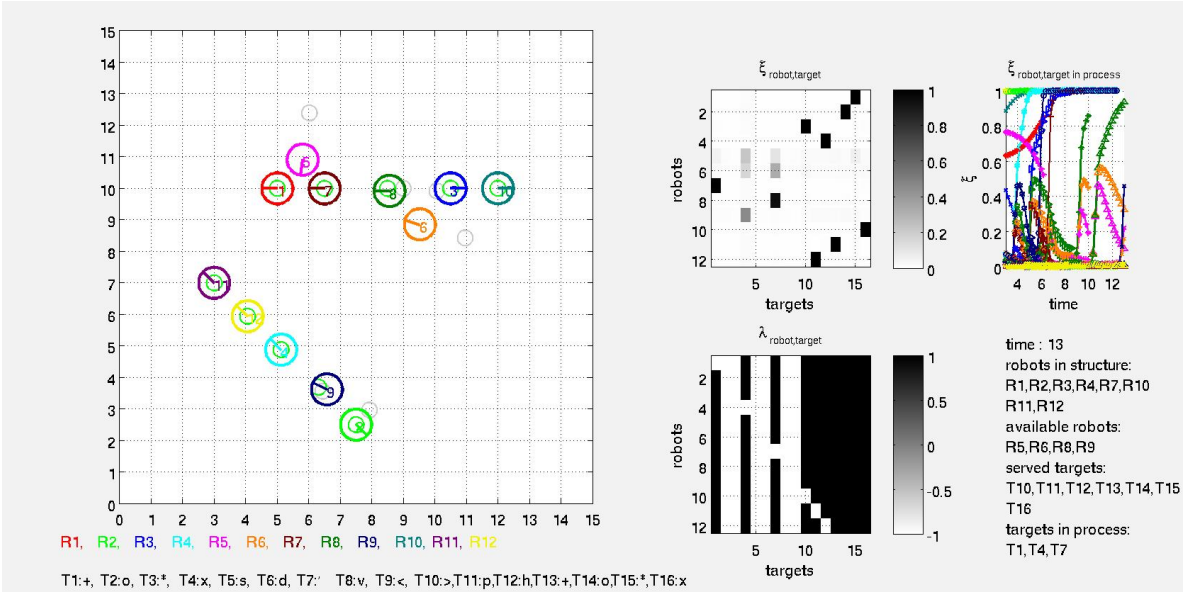

Abbildung 5.29.: Beispiel „gleichzeitiger sequentieller Bau von zwei Brücken“ endgültige Zielverteilung

In Abb. 5.26 ist die Situation kurz nach Beginn der Simulation zu sehen. Im Folgenden werden an allen vier Brückenköpfen sequentiell Linien aufgebaut. Abb. 5.27 zeigt die

Situation nach dem Hinzufügen einiger Roboter. In der 1. Brücke können noch zwei Roboter eingefügt werden. Dort streben die Roboter 4 und 9 Docking-Stationen in der Brücke an. In der 2. Brücke strebt Roboter 8 die Docking-Station von Roboter 2 an und Roboter 7 hat die von Roboter 1 fast erreicht. In Abb. 5.28 ist die Situation zu sehen, in der Roboter 7 sein Ziel erreicht hat. Es ist auch zu sehen, dass in der verbliebenen Lücke zu wenig Platz für zwei weitere Roboter ist. Das hat dazu geführt, dass die Docking-Station von Roboter 2 inaktiv geschaltet wurde und die von Roboter 7 in die Mitte zwischen Roboter 2 und 7 verschoben wurde. Damit hat Roboter 8 sein bisheriges Ziel verloren, und es beginnt ein Wettstreit der drei Roboter 5, 6 und 8 um die Docking-Station 7. In Abb. 5.29 ist die Situation kurz vor dem Schließen beider Brücken zu sehen.

5.5.4. Bewertung der Lösung

Es konnte gezeigt werden, dass der Bau von Brücken, die zunächst von beiden Seiten her als sequentielle Linien gebaut werden, möglich ist. Vorteilhaft an der gefundenen Lösung ist, dass niemals ausgewiesen werden muss, welche der Brückenköpfe zur selben Brücke gehören.
Als nachteilig kann angesehen werden, dass die Brückenköpfe immer relativ genau aufeinander ausgerichtet sein müssen. Von dieser Problematik sind allerdings auch reale Bauvorhaben betroffen (siehe Tunnelbau von zwei Seiten). Ebenfalls ist es klar, dass es zu Problemen führen kann, wenn sich zwei zu bauende Brücken kreuzen. Aber auch diese Problematik ist in allen realen Brückenbauten zu finden.
Ein wesentlicher Kritikpunkt könnte sein, dass die Abstände der Roboter innerhalb einer fertigen Brücke nicht immer gleich sind. Sie sind nur dann gleich, wenn die beiden letzten Ziele zum Schließen der Brücke genau aufeinander fallen. Andernfalls wird eine Docking-Station verschoben, und es kommt somit zu anderen Abständen an dieser Stelle im Vergleich zu den übrigen Abständen innerhalb der Brücke. Eine Alternative wäre, nicht nur eine, sondern alle Docking-Stationen zu verschieben. Alle Abstände wären gleich, wenn alle Docking-Stationen gleich stark verschoben sind.
Nachteilig an einem solchen Vorgehen wäre, dass Informationen innerhalb der Brücke propagiert werden müssten und die Entscheidungen, die ein Roboter treffen muss, nicht mehr auf dem lokalen Wissen des Roboters beruhen.
Eine weitere Möglichkeit Brücken zu bilden sieht folgendermaßen aus:
Jeder Roboter könnte an beiden Enden eine Docking-Station besitzen. Mit dieser koppeln sich die Roboter an ein Ziel (oder eine andere Docking-Station) an. An den Docking-Stationen soll der „KOpplungswinkel“ minimal (bzgl. der Mittellage) sein. Dadurch kann eine gestreckte Brücke erreicht werden. Dennoch hätte diese Brücke die Eigenschaft, sich etwas zu verbiegen, sollten die verwendeten Roboter in ihrer Gesamtlänge die zu überbrückende Strecke überschreiten. Dieser Lösungsansatz setzt allerdings ein vollkommen anderes Roboterdesign vorraus.

5.6. Sequentielles Bilden von Polygonen fester Größe

In den vorangegangenen Untersuchungen ging es immer um das Bilden von geradlinigen Strukturen. In diesem Abschnitt soll das Bilden von Kreisen, oder besser gesagt, von Polygonen untersucht werden. Im ersten Schritt soll sich dabei nahe an die Bildung von Linien angelehnt werden, um möglichst viel von den bisherigen Ergebnissen nutzen zu können.

5.6.1. Idee des Verfahrens

Bei der sequentiellen Bildung von Linien war es für das gestreckte Aussehen wesentlich, dass jede Docking-Station in der gleichen Richtung ausgerichtet ist wie der Roboter, zu der sie gehört, und dass jeder Roboter sich beim Erreichen einer Docking-Station genau so ausrichtet, wie diese es vorgibt. Das Verfahren zum sequentiellen Bilden von Polygonen sieht nun vor, dass eine Docking-Station hinter einem Roboter nicht mehr in dessen Richtung ausgerichtet ist, sondern um einen gewissen Winkel (ein ganzzahliger Teiler von 360°) verdreht ist.
Ein Roboter, der diese Docking-Station bedient, wird sich ihr entsprechend - um ihren Winkel - zu seinem Vorgänger verdreht positionieren. Auf diese Weise wird ein gleichförmiges Polygon mit festgelegten Kantenwinkeln, und damit festgelegter Eckenanzahl, entstehen. Dies kann auch bei mehreren Polygonen gleichzeitig geschehen. Beim Einfügen des letzten Roboters in das Polygon wird dessen Docking-Station auf exakt der selben Position zu liegen kommen, die auch das im Raum vorgegebene Ziel - und damit auch der erste Roboter - hat. Da auch der letzte Roboter eine Docking-Station erreicht hat, würde seine eigene freigeschaltet werden. Dies muss allerdings verhindert werden, da sonst weitere Roboter diese Position anstreben würden, obwohl sie bereits (durch den ersten Roboter) besetzt ist. An dieser Stelle soll nochmals auf den Unterschied zwischen einer Position und einem Ziel (Docking-Station) hingewiesen werden. Selbstverständlich ist die Docking-Station des letzten Roboters nicht besetzt, sondern nur deren Position.

5.6.2. Matlab Funktionen

Da weitgehend auf die sequentielle Bildung mehrerer Linien zurückgegriffen wurde, mussten nur zwei Funktionen angepasst werden.

cheackPosition

Die Funktion *cheackPosition* arbeitet zunächst genauso, wie die von der sequentiellen Bildung mehrerer Linien bekannte Funktion. Allerdings wird eine Matrix aufgebaut, in der die Information gehalten wird, welcher Roboter auf der Position welcher Ziele steht. Danach wird überprüft, welcher Roboter auf mehr als einer Zielposition steht, indem die Zeilensummen gebildet werden. Ist eine Zeilensumme größer als 1, so steht der entsprechende Roboter auf mehr als einer Zielposition. Dies kann nur dann der Fall sein, wenn das Polygon geschlossen ist und der erste Roboter sowohl auf der Position des vorgegebenen Ziels als auch auf der der

Docking-Station des letzten Roboters, die nicht sein Ziel ist, steht. In diesem Fall muss die Docking-Station des letzten Roboters aus der Menge der aktiven Ziele und aus der Menge der in Bearbeitung befindlichen Ziele wieder entfernt werden.

scene_215

Die Unterschiede der Funktion *scene_215* zu der bereits bekannten Funktion *scene_205* zur Bildung mehrerer Linien sind sehr gering. Die Funktion *scene_215* besitzt den zusätzlichen Übergabeparameter „dockingStationAngle“, mit dem der Winkel festgelegt wird, um den die Docking-Stationen der Roboter gegenüber der jeweiligen Längsachse der Roboter verdreht sein sollen. Der zweite Unterschied besteht in der geänderten Berechnung der Ausrichtung der Docking-Stationen.

5.6.3. Beispiele

Die Abb. 5.30 zeigt die Endsituation bei zwölf Robotern, zwei vorgegebenen Zielen, einem Winkel der Docking-Stationen von 60° gegenüber der Mittellinie des zugehörigen Roboters und den bisher bekannten sonstigen Parametern. Es ist klar, dass sich bei diesen Parametern Sechsecke bilden. Für weitere Abbildungen, Videos und Beispiele sei auf die CD verwiesen.

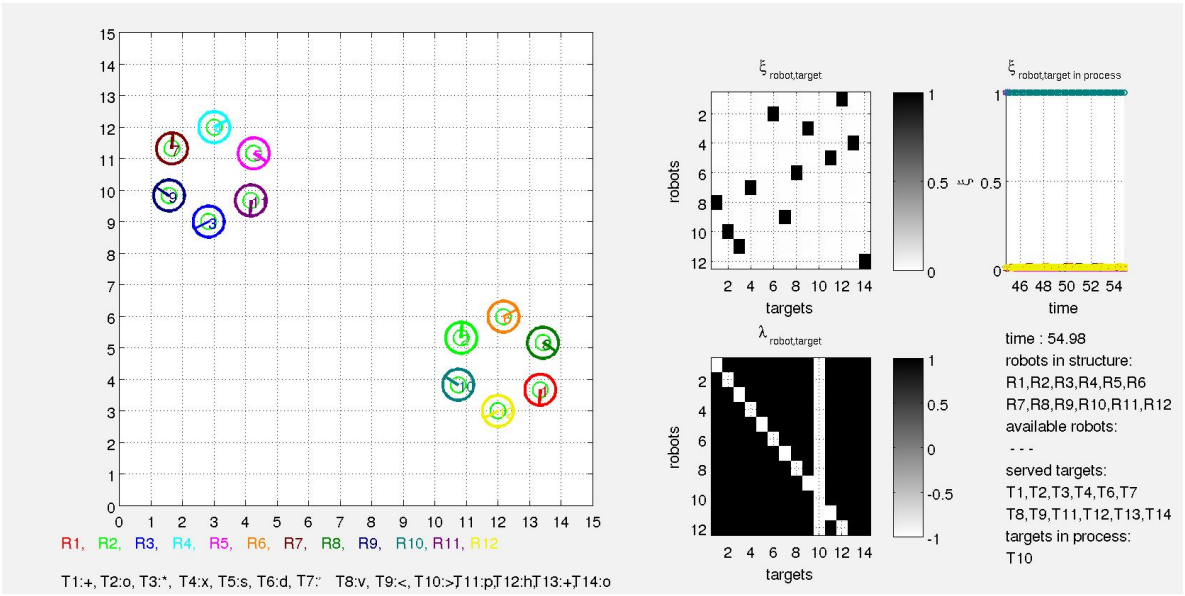

Abbildung 5.30.: Beispiel „sequentielle Bildung von Sechsecken“ am Ende der Simulation

5.7. Sequentielles Bilden von Polygonen variabler Größe

Es wurde gezeigt, dass Polygone fester Größe gebildet werden können. Im nächsten Schritt soll dieses Verfahren in der Art erweitert werden, dass ein bereits gebildetes Polygon um einen weiteren Roboter erweitert werden kann, wenn ein solcher zur Verfügung steht.

5.7.1. Idee des Verfahrens

Wenn ein Polygon gebildet ist und die Docking-Station des letzten Roboters nicht ausgeschaltet wird, dann wird ein verfügbarer Roboter diese Docking-Station als Ziel verfolgen. Da die Position dieser Docking-Station mit der Position eines vorgegebenen Ziels und so mit der Position des ersten Roboters übereinstimmt, wird der Roboter, der diese Docking-Station zu besetzen versucht, mit dem ersten Roboter des Polygons kollidieren. Die Idee des Verfahrens sieht in diesem Fall vor, den Winkel der Docking-Stationen, die sich in diesem Polygon befinden, in der Form zu verändern, dass Platz für den zusätzlichen Roboter geschaffen wird.

5.7.2. Matlab Funktionen

Um die gewünschte Modifikation des Simulationsprogramms durchführen zu können, müssen einige, bisher auch unveränderte Funktionen angepasst werden. Zunächst muss ein Vektor eingeführt werden, der die jeweiligen Winkel der Docking-Stationen beinhaltet. Die Werte dieser Winkel müssen bei der Aufweitung eines Polygons entlang der Kette von beteiligten Robotern hindurchpropagiert werden.
Zusätzlich zur bereits bekannten Matrix, die beinhaltet, welcher Roboter auf den Positionen welcher Ziele steht, wird eine weitere Matrix benötigt, die die Information beinhaltet, zwischen welchen Robotern es zu Kollisionen gekommen ist. Genauer gesagt, beinhaltet sie, zwischen welchen Robotern es zu Kollisionen gekommen wäre, wenn die Kollisionsvermeidung der Funktion *navigation* dies nicht verhindert hätte. Daher ist es notwendig, dass auch die Funktion *navigation* modifiziert wird.

cheackPosition

Die Funktion *cheackPosition* aus Abschnitt 5.6 wurde in folgenden Punkten erweitert:

- Wenn festgestellt wird, dass ein Roboter auf einer Docking-Station steht, wird der Winkel der besetzten Docking-Station auf die Docking-Station des Roboters weiter propagiert.
- Wenn festgestellt wird, dass ein Roboter auf der Position zweier Ziele steht, so wird überprüft, ob dieser Roboter an Kollisionen mit anderen Robotern beteiligt ist (diese Information ist in der übergebenen Variable „collisionMatrix“ zu finden). Ist dies der

Fall, so wird der Winkel seiner Docking-Station so berechnet, dass im Polygon Platz für einen weiteren Roboter entsteht.

navigation

Da die Information benötigt wird, ob es zu Kollisionen zwischen Robotern gekommen ist (bzw. gekommen wäre, hätte die Kollisionsvermeidung dies nicht verhindert), muss die Funktion *navigation* diese Information in der Matrix „collisionMatrix“ zur Verfügung stellen. Wenn die Kollisionsvermeidung bei einem Paar von Robotern feststellt, dass der Abstand so klein ist, dass eingegriffen werden muss, so wird in der Matrix „collisionMatrix“ für beide Roboter der entsprechende Wert gesetzt.

scene_217

Um den Datenaustausch zwischen den Funktionen *cheackPosition* und *navigation* zu gewährleisten, musste die Funktion *scene_217* entsprechend angepasst werden. Darüber hinaus gibt es im Vergleich zur Funktion *scene_215* aus 5.6 keine Veränderungen.

5.7.3. Beispiele

Es wurden verschiedene Beispiele mit unterschiedlichen Startbedingungen simuliert. Die Parameter des Selektionsprozesses wurden von vorherigen Beispielen unverändert übernommen. An dieser Stelle soll eines der Beispiele etwas genauer erläutert werden:

Beispiel „sequentieller Bau von vergrößerbaren Polygonen“

Diese Simulation arbeitet mit 15 Robotern und zwei im Raum vorgegebenen Zielen. Zunächst sind die Winkel der Docking-Stationen mit 90° vorgegeben, es werden also zunächst Quadrate entstehen. Die Startpositionen der Roboter sind zufällig gewählt.

In Abb. 5.31 ist die Situation zu sehen, dass das eine Quadrat bereits entstanden ist, während in das andere der 3. Roboter hinzugefügt wird. Da Roboter 10 mit Roboter 14 (dem ersten Roboter des Quadrates) kollidiert ist, werden die Winkel der Docking-Stationen verändert und Platz für einen weiteren Roboter geschaffen. Dies ist in Abb. 5.32 zu sehen. Roboter 10 hat die Docking-Station 1 bedient und durch die Nähe von Roboter 7 zu Roboter 14 wird abermals das Polygon erweitert. Abb. 5.33 zeigt die Situation, in der das eine Polygon zu einem Sechseck und das andere zu einem Achteck gewachsen ist. Roboter 13 strebt die Docking-Station von Roboter 5 an und löst eine weitere Erweiterung des Polygons durch seine Nähe zu Roboter 14 aus. Die endgültige Struktur ist in Abb. 5.34 zu erkennen. Es haben sich am Ende ein Sechseck und ein Neuneck gebildet.

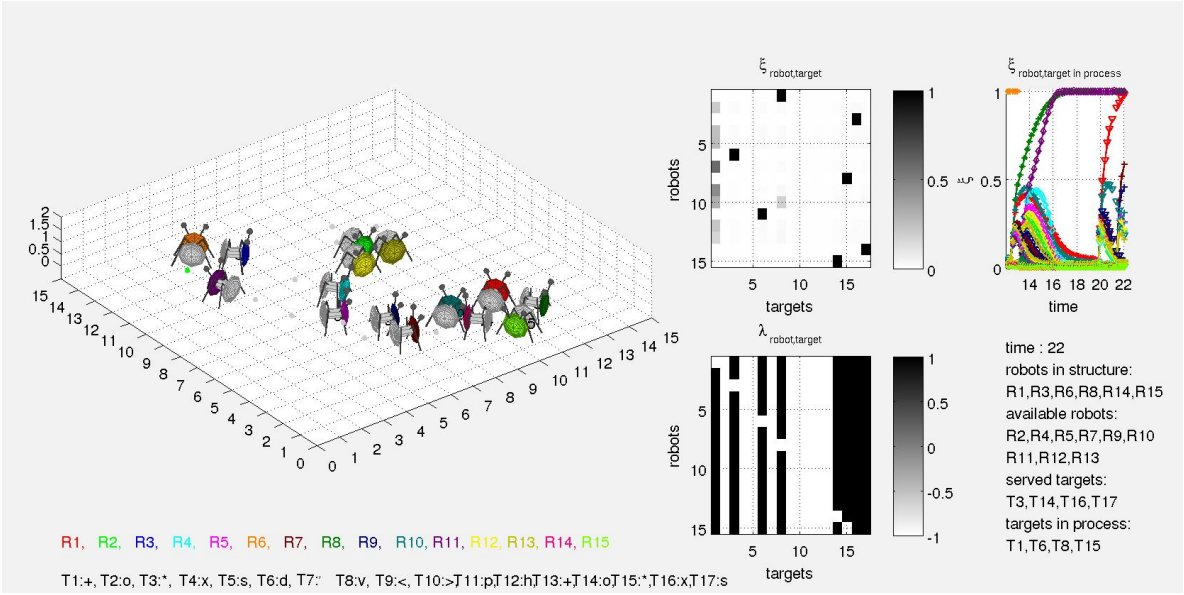

Abbildung 5.31.: Beispiel „sequentieller Bau von vergrößerbaren Polygonen“: Fertigstellung des 1. Quadrates

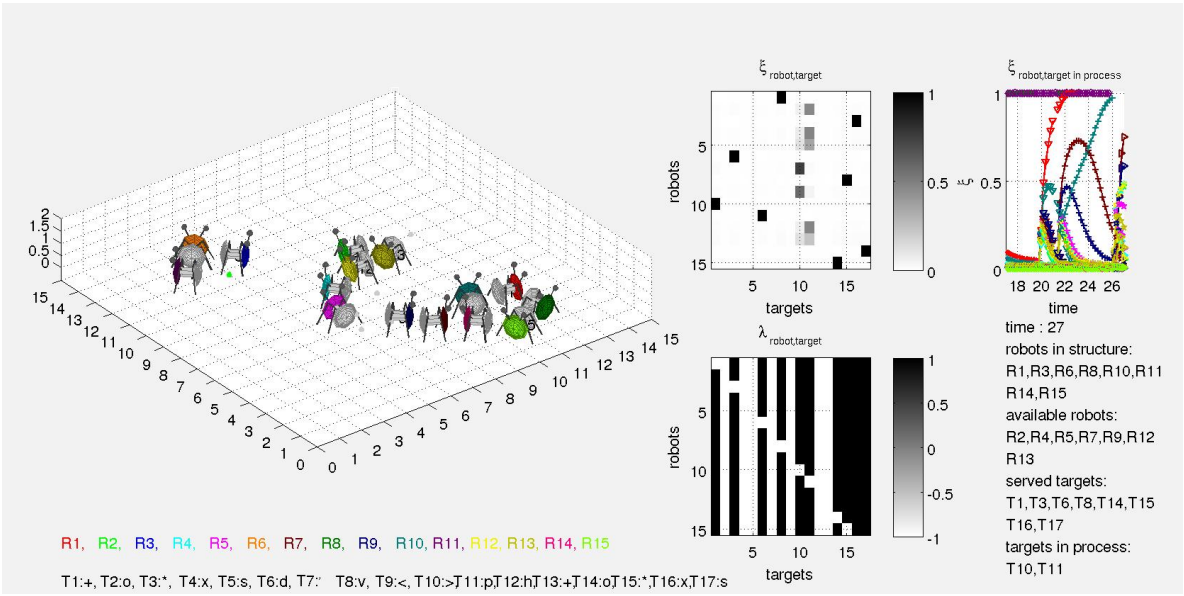

Abbildung 5.32.: Beispiel „sequentieller Bau von vergrößerbaren Polygonen“: Aufweitung des 1. Quadrates

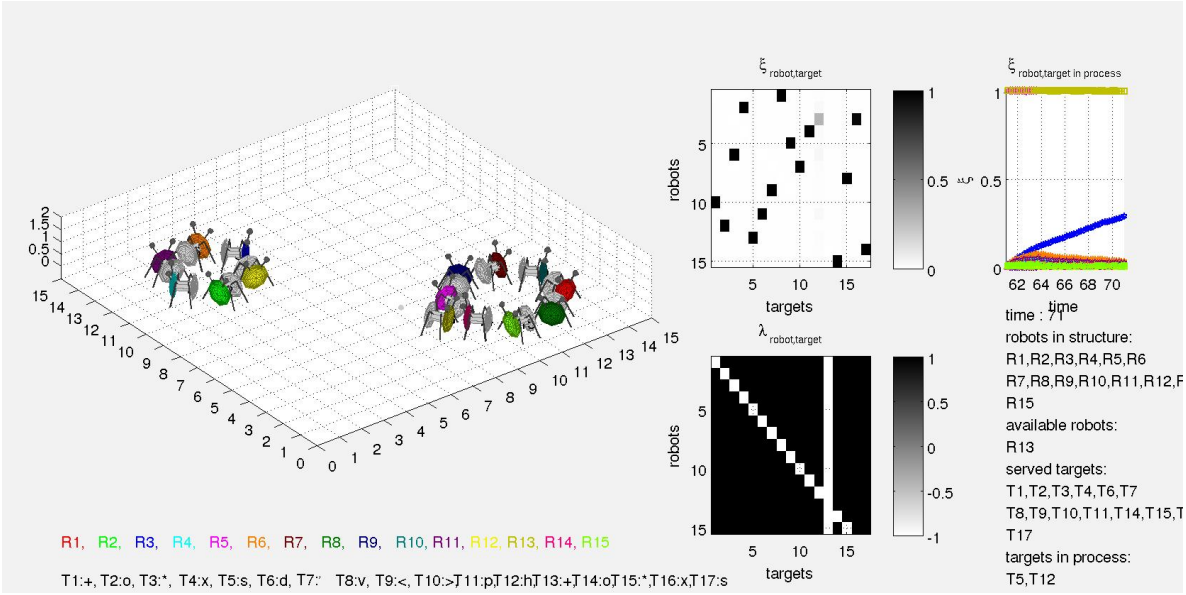

Abbildung 5.33.: Beispiel „sequentieller Bau von vergrößerbaren Polygonen“: Einfügen des letzten Roboters

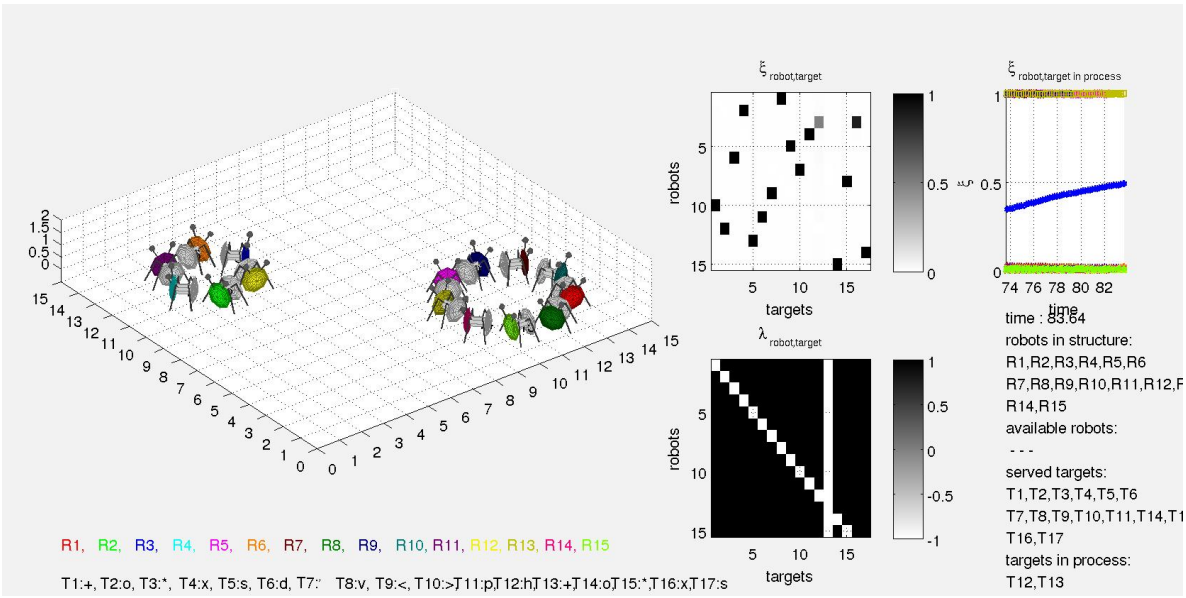

Abbildung 5.34.: Beispiel „sequentieller Bau von vergrößerbaren Polygonen“: Ende der Simulation

5.7.4. Bewertung der Lösung

Es konnte gezeigt werden, dass es möglich ist, erweiterbare und unterschiedlich große gleichmäßige Polygone zu bilden. Hierzu war es erforderlich, Informationen aus der Funktion

navigation zu ziehen.
Es kann an dieser Lösung kritisiert werden, dass die Struktur der Polygone durch die Vorgabe eines Winkels hart vorgegeben und weniger selbstorganisiert ist. Ein weiterer Ansatz zur Bildung von zyklischen Strukturen soll im Abschnitt 5.8 vorgestellt werden. Dieser kommt ohne die hier notwendigen Vorgaben aus.

5.8. Sequentielles Bilden von Ketten, die sich zu zyklischen Strukturen schließen

Die vorgestellten Verfahren zur Bildung von zyklischen Strukturen beruhten darauf, dass die Winkel der nachfolgenden Roboter durch die Docking-Stationen vorgegeben wurden. Nun soll ein Verfahren vorgestellt werden, das auf diese Vorgabe verzichtet.

5.8.1. Idee des Verfahrens

Im Abschnitt 5.4 wurde gezeigt, dass, wenn Linien an einem vorgegebenen Startpunkt gebildet worden sind, diese durch das Ausschalten des Ziels, an dem sie gebildet wurden, und durch das Einschalten eines neuen Ziels verschoben werden können. Wenn aber kein neues Ziel im Raum besteht, soll der erste Roboter einer Kette eines der bisher unbedienten Ziele, und zwar die Docking-Station eines am Ende einer Linie befindlichen Roboters, als Ziel suchen.
Die Zyklenbildung soll also in folgenden Schritten ablaufen:

- Bilden von einer oder mehreren Ketten an vorgegebenen Zielen im Raum
- Deaktivieren der vorgegebenen Ziele
- Selbstorganisiert schließen sich die Ketten zu zyklischen Strukturen zusammen

5.8.2. Matlab Funktionen

cheackPosition

Die Funktion *cheackPosition* ist in der Art angepasst worden, dass die vorgegebenen Ziele nach dem Einfügen aller Roboter in Ketten inaktiv geschaltet werden, damit die ersten Roboter in den Ketten neue Ziele finden können.

navigation

Bisher sollten Strukturen gebildet werden, bei denen sich die Roboter entsprechend ihren Zielen ausrichten sollten. Da dies bei diesem Ansatz anders ist, muss der Teil, der das Ausrichten ausführt, abgeändert werden. Es wurde eine Lösung gewählt, bei der die Rotationsgeschwindigkeit bei Erreichen eines Ziels begrenzt ist. Damit kann sichergestellt werden, dass in der ersten Phase, Linien gebildet werden können.

5.8.3. Beispiele

Es wurden verschiedene Simulationen durchgeführt. Dabei wurden zunächst eine, zwei oder auch vier Linien gebildet, die dann zu zyklischen Strukturen umgebildet wurden. Beispielhaft soll eine Simulation mit vier Linien gezeigt werden und für weitere Beispiele auf die Dateien und Videos auf der CD verwiesen werden.

Beispiel „zyklischer Zusammenschluss sequentieller Ketten“

In diesem Beispiel werden an vier Zielen im Raum Linien aus insgesamt zwölf - zufällig verteilten - Robotern gebildet.

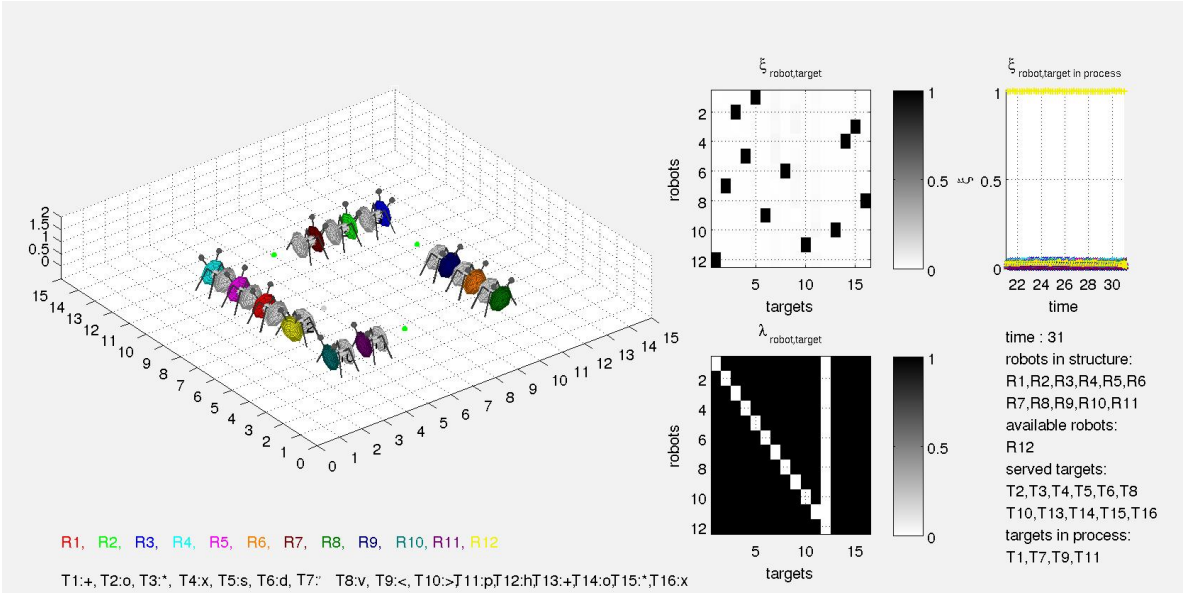

Abbildung 5.35.: Beispiel „zyklischer Zusammenschluss sequentieller Ketten“: Aufbau der Ketten

In Abb. 5.35 ist zu sehen, dass sich eine Kette mit zwei, zwei Ketten mit drei und eine vierte mit vier Robotern gebildet hat. Werden die Ziele der ersten Roboter der Ketten ausgeschaltet, suchen sich diese neue Ziele. Dies ist durch das Rauschen der Werte für ξ kein deterministischer Prozess und so kann es zu unterschiedlichen zyklischen Strukturen kommen. In diesem Beispiel entstehen zwei Zyklen: Der kleinere, weil Roboter 10 seinen einzigen Nachfolger, Roboter 11, als Ziel nimmt, und einen großen aus den restlichen 10 Robotern. In Abb. 5.36 ist diese Situation zu erkennen. Eine besondere Eigenschaft ist, dass die Zyklen in Bewegung sind und sich in der Richtung ihres Zusammenschlusses weiter drehen. Dabei rücken die Roboter immer näher zusammen, und ihre Bewegungen werden immer langsamer. Durch Simulationen bis zu 2000 Simulationssekunden, konnte gezeigt werden, dass sich dieser Effekt mit der Zeit verlangsamt. Es kann vermutet werden, dass

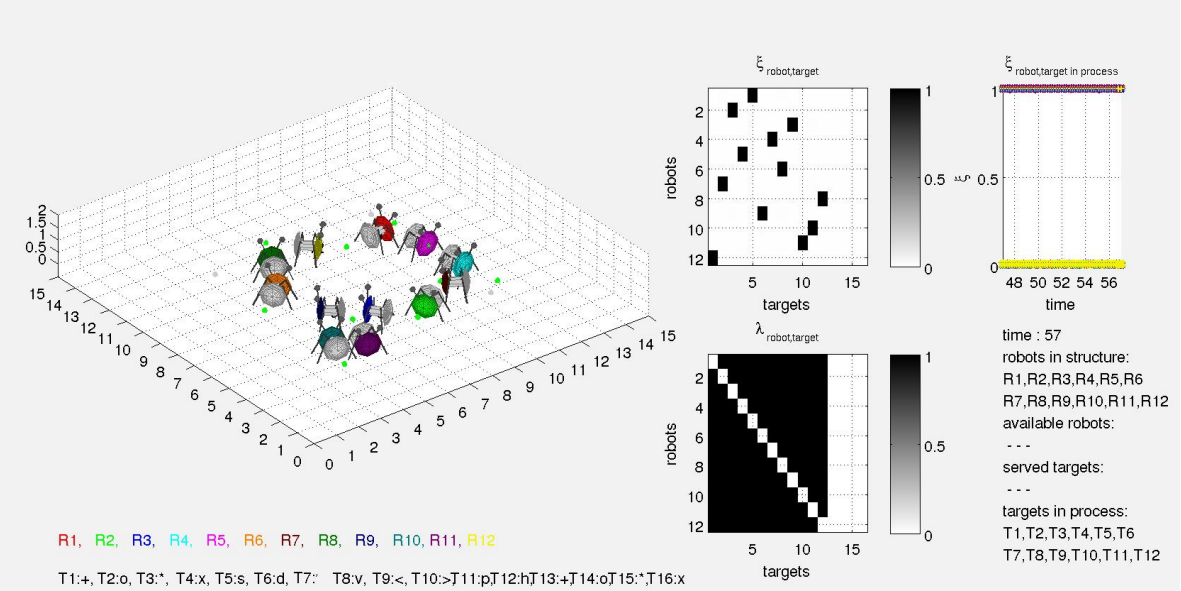

Abbildung 5.36.: Beispiel „zyklischer Zusammenschluss sequentieller Ketten“: Aufbau der Zyklen

die Roboter asymptotisch zum Stillstand gelangen werden. Um dies genauer zu beleuchten, müssten aufwendige und langwierige Messungen durchgeführt werden.

5.8.4. Bewertung der Lösung

Es konnte gezeigt werden, dass die Bildung von zyklischen Strukturen durch Deaktivieren der vorgegebenen Ziele erreicht werden kann. Die Anzahl der verwendeten Ketten spielt hierbei keinerlei Rolle. Jedoch ist es klar, dass sich die Wahrscheinlichkeit zum Bilden mehrerer Zyklen erhöht, je mehr einzelne Ketten vorher gebildet wurden. Auch ist es klar, dass die Position der einzelnen Ketten zueinander diese Wahrscheinlichkeiten verändern kann.

6. Parallele Strukturbildung

Das vorherige Kapitel hat das Funktionieren der sequentiellen Strukturbildung gezeigt. Allerdings ist festzustellen, dass dieses Verfahren durchaus sehr langsam sein kann. Insbesondere ist es dann langsam, wenn alle Roboter nacheinander den gleichen langen Weg zu ihren Zielen zurücklegen müssen. Sinnvoller wäre es, wenn die übrigen Roboter in der Zeit, in der sich einer auf das nächste Ziel zubewegt, auch bereits ihr Ziel kennen und es verfolgen könnten.
Diese Überlegung führt zur *parallelen Strukturbildung*. Sie sieht vor, dass die Roboter alle gleichzeitig ihre Ziele finden und verfolgen. Die parallele und die sequentielle Strukturbildung haben gemein, dass jeder Roboter (mindestens) eine Docking-Station hat. Ein wesentlicher Unterschied ist, dass die Docking-Stationen der Roboter aktiv geschaltet sind und somit für jeden Roboter als Ziel zur Verfügung stehen.
Dieses Vorgehen birgt leider ein schwieriges Problem: Es können unerwünschte Zyklen entstehen. Diese Zyklen verhindern den Aufbau von Linien und ggf. auch komplexeren Strukturen. Ein Zyklus kann dadurch charakterisiert werden, dass ein Roboter einen seiner Nachfolger als Ziel hat (für einen Kreis gilt dies beispielsweise für jeden Roboter). Zum Bilden einer Kette muss sichergestellt werden, dass genau dies unterbunden wird. Hierzu wird die Matrix λ verwendet, um Ziele von Robotern auszuschließen. Die Matrix λ wird also in zwei Schritten gebildet:

- Besetzen der Spalten der aktiven Ziele mit 1 und der inaktiven Ziele mit -1.
- Ausschließen von Zyklen durch Besetzen der Nachfolger eines Roboters in dessen Zeile mit -1.

Es geht also im Wesentlichen bei der parallelen Bildung von Linien darum, alle Nachfolger eines jeden Roboters zu bestimmen. Im Folgenden sollen Verfahren vorgestellt werden, wie diese Mengen bestimmt, und so Zyklen vermieden und damit parallele Linien aufgebaut werden können.

6.1. Paralleles Bilden einer Linie mit Hilfe der ξ-Werte

Die Information, welcher Roboter welches Ziel bedient, ist in der Matrix ξ enthalten. Da die Bewegungen der Roboter als Linearkombination der Zeilenvektoren der Matrix ξ implementiert sind, braucht nicht das Maximum einer Zeile bestimmt zu werden, um die Zielzuordnung zu kennen. Diese Informationen der Matrix ξ sollen zur Zyklenvermeidung genutzt werden.

6.1.1. Idee des Verfahrens

Mit Hilfe der Matrix ξ kann festgestellt werden, welcher Roboter welches Ziel verfolgt. Hierzu wird das Maximum einer zu einem Roboter gehörenden Zeile gebildet. Dieses Maximum ist das Ziel, das der Roboter vorrangig[1] verfolgt. Da es aber auch bei einer schwach ausgeprägten Zielzuordnung auch ein Zeilenmaximum gibt, das aber nicht als ein Ziel eines Roboters verstanden werden sollte, wird ein Schwellwert eingeführt, bei dessen Überschreitung eine Zielzuordnung als gegeben angesehen wird.
Damit sind die Ziele der Roboter und umgekehrt die auf ein Ziel zielenden Roboter (also die Nachfolger eines Roboters, sofern es sich bei dem Ziel um eine Docking-Station handelt) bekannt. Was jetzt noch fehlt, ist die Information, welche Roboter zur Menge der Nachfolger eines Roboters gehören. Hierzu bietet es sich an, als Darstellung der Zielzuordnungen Zeiger zu verwenden, denn dann ist es möglich, diese zu verketten und so eine sehr effiziente Datenstruktur zu erhalten. Es braucht dann nur für jeden Roboter diese Liste durchlaufen zu werden und in der Matrix λ der entsprechende Wert auf -1 gesetzt zu werden.

6.1.2. Matlab Funktionen

cheackPosition

Da in dieser ersten Aufgabe ausschließlich Linien gebildet werden sollen, sind alle Ziele von Anfang an aktiv und sollen auch nie inaktiv werden. Dies vereinfacht die Funktion *cheackPosition*, da hier keine Logik zur Strukturgestaltung mehr enthalten sein muss. Sie dient dazu, festzustellen, welche Ziele bedient sind und welche Roboter sich bereits in der Struktur befinden. Dies zeigt, ob die Aufgabe erfüllt ist, also ob alle Roboter ein Ziel erreicht haben. Damit ist der erste Teil der Bildung der Matrix λ erledigt.

setLambda

In der Funktion *setLambda* wird der zweite Teil der Bildung der Matrix λ vollzogen. Da es in Matlab keine Pointer gibt, müssen die Listen der Nachfolger eines Roboters anders aufgebaut werden:
Zuerst wird die Matrix „headOfRobots“ bestimmt, die eine 1 in der i-ten Zeile und der j-ten Spalte enthält, wenn der Roboter i von dem Roboter j gefolgt wird. Alle übrigen Einträge sind 0.
Im zweiten Schritt wird für jeden Roboter i Folgendes gemacht:

- Bestimme das Maximum der Zeile i (der Matrix „headOfRobots“ und den Index den das Maximum in dieser Zeile hat. Speichere das Maximum in der Variable „isHead“ und den Index in der Variable „index“.

[1] Der Roboter bedient dieses Ziel nur vorrangig deshalb, da durch die zur Bewegung gebildete Linearkombination ein gewichteter Schwerpunkt der Ziele das Ziel des Roboters ist. Somit werden auch weitere Ziele in gewisser Form bedient

- Iteriere so lange „isHead“ gleich 1, und i ungleich „index“ ist:
  - Setze $\lambda_{i,index} = -1$
  - Bestimme das Maximum der Zeile „index“ und den Index, den das Maximum in dieser Zeile hat.

In der ersten Iteration wird der λ-Wert für den direkten Nachfolger gesetzt. In der zweiten Iteration wird der λ-Wert für dessen Nachfolger gesetzt. Das Verfahren geht so lange so weiter, bis es entweder keinen Nachfolger mehr für einen Roboter gibt, oder bis der erste Roboter wieder erreicht wurde. Damit ist ausgeschlossen, dass das Verfahren in eine Endlosschleife gerät. Es kann allerdings sein, falls die Matrix „headOfRobots“ bereits Zyklen enthält, dass alle im Zyklus beteiligten Roboter ihre Ziele verlieren. Dies kommt daher, dass innerhalb eines Zyklus jeder Roboter als der erste angesehen wird, und damit alle Roboter keinen ihrer Nachfolger mehr als Ziel haben dürfen. Sollte dies der Fall sein, werden durch die negativen λ-Werte die bestehenden Zielzuordnungen geschwächt und neue Zielzuordnungen müssen sich ausbilden. Dabei werden sich durchaus einige der vormals bestandenen Zuordnungen abermals neu bilden. Die Experimente haben allerdings gezeigt, dass dieser Fall sehr selten ist. Es ist sehr unwahrscheinlich, dass die Werte $\xi_{i,j}$ und $\xi_{j,i}$ zweier Roboter i und j gleichzeitig, also im selben Iterationsschritt, den Schwellwert überschreiten. Sobald einer der Werte den Schwellwert überschritten hat, wird der Roboter als Nachfolger des anderen angesehen und die gekoppelten Selektionsgleichungen führen durch den veränderten λ-Wert zum Abnehmen des anderen ξ-Wertes. Das beschriebene Verhalten tritt auch in weiteren Situationen auf und kann allgemein formuliert werden:
Seien zwei bestehende Ketten $a = (a_1, a_2, \cdots, a_m)$ und $b = (a_1, b_2, \cdots, b_n)$ von Robotern mit der folgenden Eigenschaft gegeben:

$$\begin{aligned} \forall a_i: \quad \max_{\forall j}(\xi_{a_{i+1},j}) &= \xi_{a_{i+1},a_i} \\ \wedge \quad \xi_{a_{i+1},a_i} &> s \end{aligned}$$

Wenn s der gewählte Schwellwert ist (in den Experimenten wurde ein Wert von 0,99 gewählt). Es ist nun sehr unwahrscheinlich, dass im Iterationsschritt $k+1$ gilt

$$\begin{aligned} \xi_{a_1,b_n} &> s \\ \wedge \quad \xi_{b_1,a_m} &> s \end{aligned}$$

und im Iterationsschritt k beide Werte noch kleiner waren als der Schwellwert s. Sei o.B.d.A. angenommen, dass der Wert $\xi_{b_1,a_m} > s$ sei, so heißt das, dass die Kette b, deren erster Roboter b_1 ist, an die Kette a und zwar an den letzten Roboter a_m angehängt wird. Hierdurch werden alle Roboter der Kette b zu Nachfolgern des Roboters a_1 und damit alle $\lambda_{a_1,b_j} = -1$ gesetzt. Dies führt durch die gekoppelten Selektionsgleichungen auch zum Sinken des Wertes ξ_{a_1,b_n} [2] und schließt damit die Zyklenbildung aus.

[2] Genaugenommen führt es zum Sinken aller Werte ξ_{a_1,b_j}. Die vorangehenden Roboter der kette b sind allerdings ohnehin nicht als Ziel so attraktiv, da sie bereits von einem Roboter als Ziel angesehen werden. Durch die gekoppelten Selektionsgleichungen ist ein Anwachsen dieser Werte ohnehin unmöglich.

scene_301 / scene_302

Die Funktionen *scene_301* und *scene_302* müssen bezüglich der Parameterübergabe angepasst werden. Sonst gibt es hier keinerlei Veränderungen gegenüber den bereits bekannten Funktionen.

6.1.3. Beispiele

Bei den erstellten Beispielen wurde der Parameter DT_XI_FACTOR gleich 1 gewählt und hat damit keinen Einfluss mehr auf die Simulation. Die übrigen Parameter wurden wie folgt gewählt.

- $\kappa = 2$
- $\beta = 2$
- maximales Rauschen von $5 \cdot 10^{-3}$
- Schrittweite von 0.01

Ein zu großes κ kann zu überstarkem Ansteigen der ξ-Werte führen und damit auch dazu, dass die Wahrscheinlichkeit von gleichzeitigem Überschreiten des Schwellwertes (s.o.) ansteigt.

Beispiel „Paralleles Bilden einer Linie (1.Methode)“

Im ersten Beispiel soll eine Linie von 9 Robotern an einem vorgegebenen Ziel gebildet werden. Die Roboter sind zunächst zufällig im Raum verteilt.

In Abb. 6.1 ist die Situation nach einer Simulationssekunde zu sehen. Interessant hierbei sind insbesondere die Roboter 1 und 3. Man kann sehen, dass sich Roboter 3 für Roboter 1 als Ziel entwickelt und umgekehrt. Ohne ein weiteres Eingreifen würden sich diese beiden Roboter zu einem Zyklus zusammenschließen. Es ist auch zu erkennen, dass der Wert $\xi_{1,3}$ größer ist als der Wert $\xi 3,1$ und daher recht wahrscheinlich den Schwellwert von 0.99 früher überschreiten wird. Dies ist auch in Abb. 6.2 verdeutlicht. Dadurch ist Roboter 1 zum Nachfolger von Roboter 3 geworden und der Wert $\lambda_{3,1}$ ist auf -1 gesetzt worden. Dies hat zum Sinken des Wertes $\xi_{3,1}$ geführt. Roboter 3 muss sich also ein neues Ziel suchen. Darüber hinaus ist auch zu sehen, dass Roboter 2 zum Vorgänger von Roboter 4 und Roboter 8 vom Vorgänger von Roboter 5 geworden ist. Die Werte $\xi_{8,6}$ und $\xi_{6,5}$ sind beide recht stark angewachsen. Sie sind daher von besonderem Interesse, weil nur eine dieser beiden Zielzuordnungen bestehen darf, da sich so entweder die Kette $(6, 8, 5)$ oder die Kette $(8, 5, 6)$ bilden wird. Eine Simulationssekunde (siehe Abb. 6.3) später hat der Wert $\xi_{8,6}$ den Schwellwert überschritten und damit darf Roboter 6 die beiden Roboter 8 und 5 nicht mehr als Ziel verfolgen. Dementsprechend sinkt der Wert $\xi_{6,5}$ und Roboter 6 muss sich ein neues Ziel suchen. In Abb. 6.4 ist die endgültige Zielverteilung zu erkennen (Kette von Robotern: $(2, 4, 9, 7, 3, 1, 6, 8, 5)$). Da Roboter 2 insgesamt acht Nachfolger hat, sind für ihn

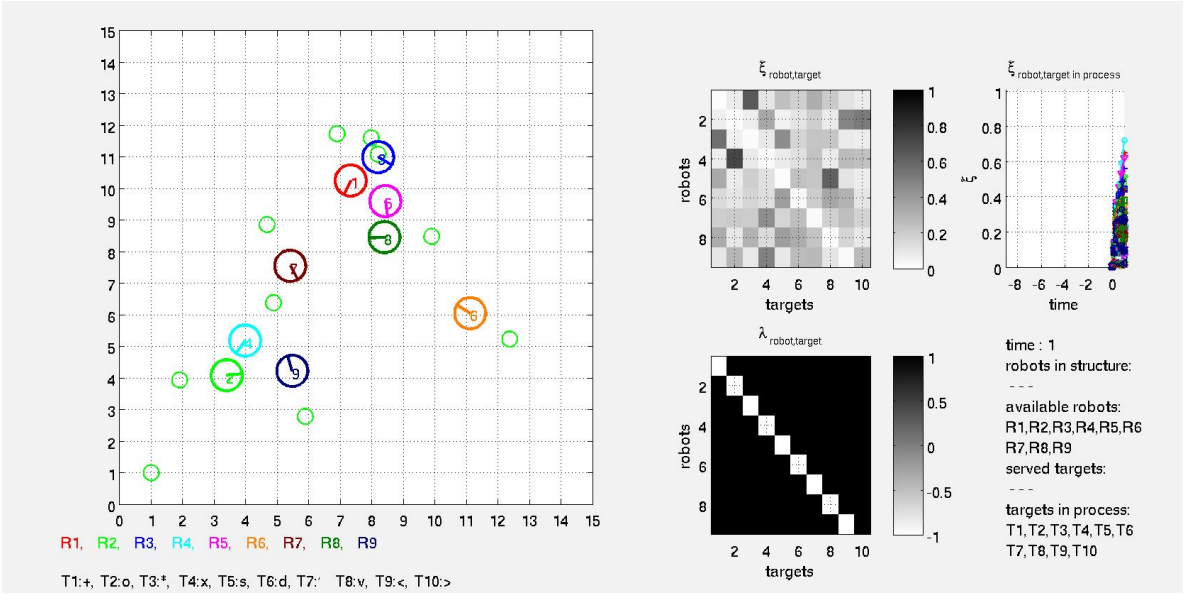

Abbildung 6.1.: Beispiel „Paralleles Bilden einer Linie (1.Methode)“: Nach einer Simulationssekunde

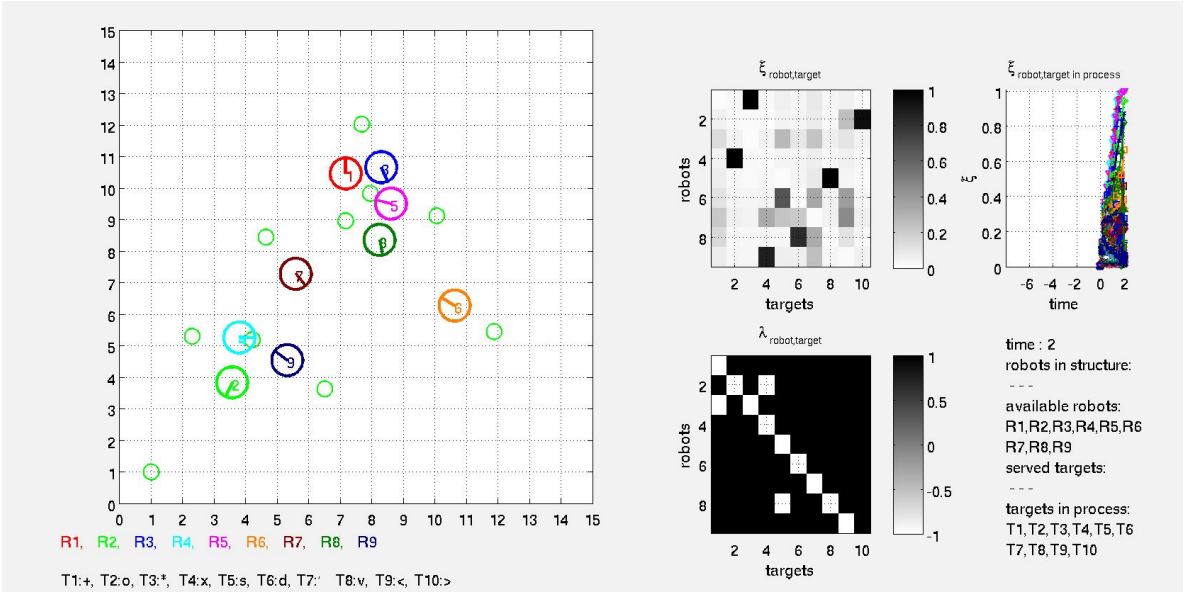

Abbildung 6.2.: Beispiel „Paralleles Bilden einer Linie (1.Methode)“: Erste Paarbildungen

neun Ziele (sich selbst eingeschlossen) unzulässig. Für Roboter 4 an zweiter Position sind es 8. Dies setzt sich ebenso fort, bis für den letzten Roboter (Nr.5) lediglich seine eigene Docking-Station ein unzulässiges Ziel ist. Ebenso ist zu sehen, dass die Docking-Station

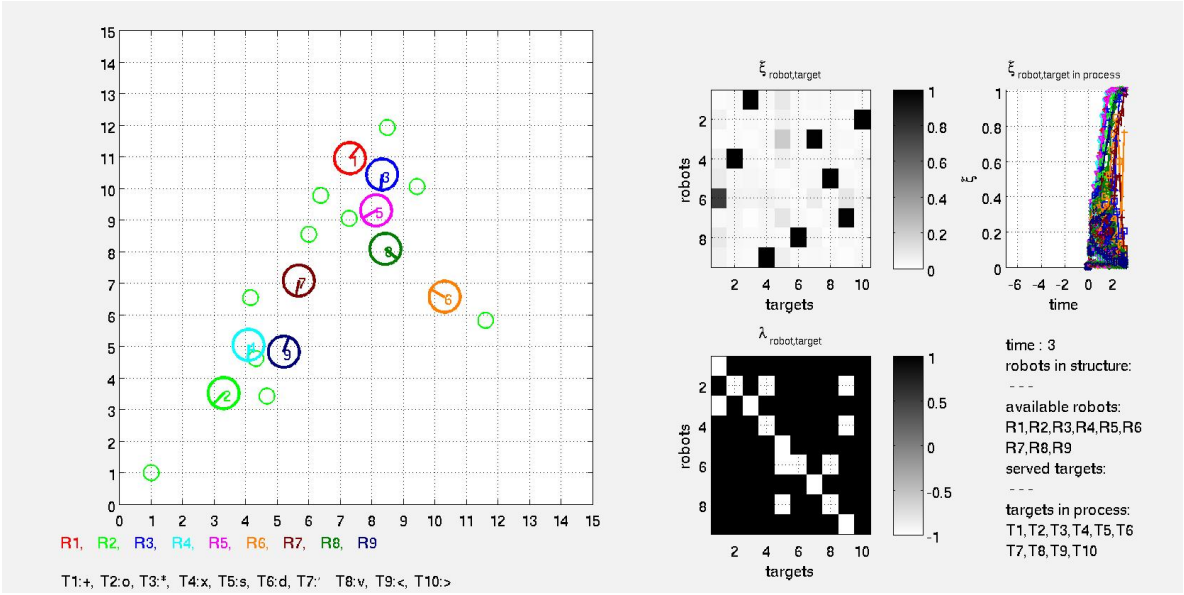

Abbildung 6.3.: Beispiel „Paralleles Bilden einer Linie (1.Methode)“: Anfügen von Robotern zu bestehenden Paaren

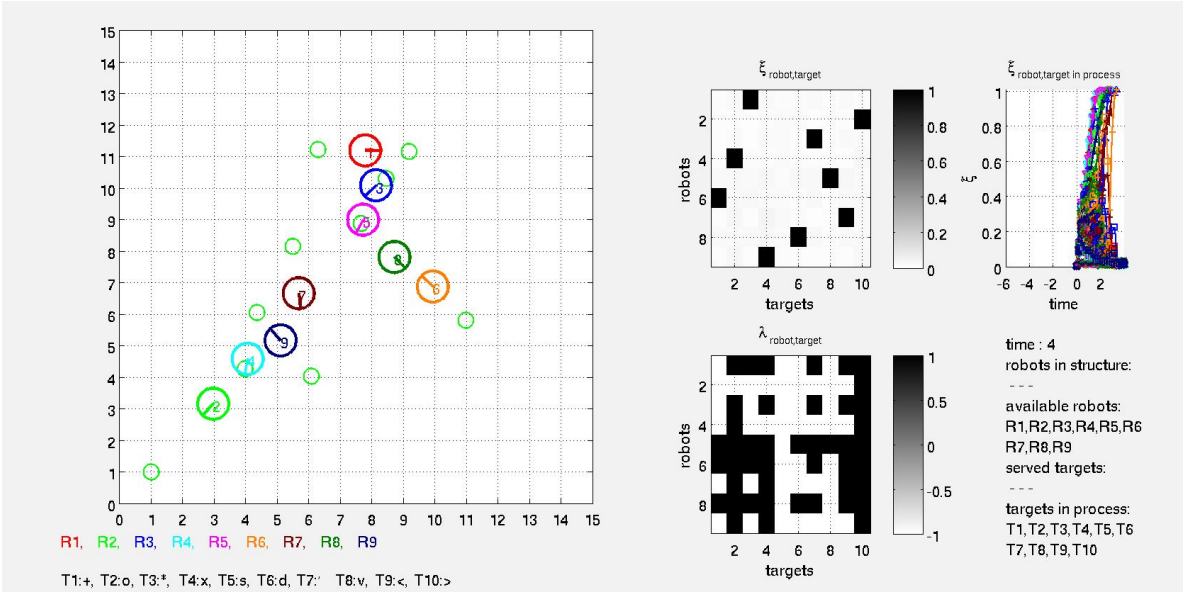

Abbildung 6.4.: Beispiel „Paralleles Bilden einer Linie (1.Methode)“: Endgültige Zielverteilung

Nr.2 für alle Roboter - außer Roboter Nr.2 - ein zulässiges Ziel ist (also für 8 Roboter ein erlaubtes Ziel). Docking-Station Nr. 4 ist für 7 Roboter ein erlaubtes Ziel. Auch dies setzt

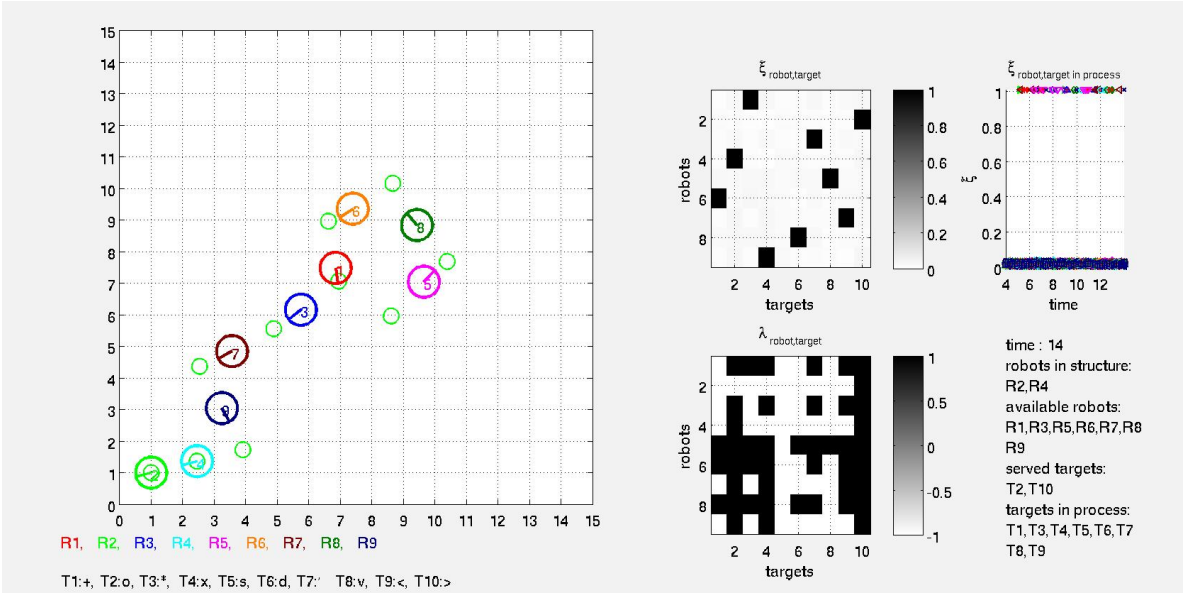

Abbildung 6.5.: Beispiel „Paralleles Bilden einer Linie (1.Methode)“: Bewegung zu den jeweiligen Zielen

sich weiter fort, bis die Docking-Station des letzten Roboters in der Kette (Docking-Station Nr.5) für keinen der Roboter mehr ein zulässiges Ziel darstellt. Da die Zielverteilung bereits jetzt abgeschlossen ist, werden die Roboter nun lediglich Bewegungen ausführen, um ihr jeweiliges Ziel zu erreichen (siehe Abb. 6.5).

Beispiel „Paralleles Bilden zweier Linien (1. Methode)“

Im zweiten Beispiel sollen zwei Linien aus insgesamt zwölf Robotern gebildet werden. Auch hier sind nach kurzer Simulationszeit Paare von Robotern zu erkennen, deren ξ-Werte so anwachsen, dass sie sich gegenseitig als Ziele auffassen werden (siehe Abb. 6.6). Beispiele hierzu sind die Paare $(1,4)$ und $(6,7)$. Dies kommt daher, dass die beiden Roboter jeweils dicht beieinander und etwas abseits der übrigen stehen. In Abb. 6.7 ist zu sehen, dass diese Paare keine zyklischen Abhängigkeiten ausgebildet haben. Außerdem sind bisher insgesamt vier Paare entstanden. Nach drei Simulationssekunden sind bereits drei Ketten von Robotern gebildet (siehe Abb. 6.8): Diese sind $(13,3,8,2,4,1)$, $(9,5,1,11)$ und $(6,7)$. Kurze Zeit später ist die endgültige Zielverteilung zu erkennen, bei der das letzte Paar an eine der Ketten angehängt ist und somit zwei (zufälligerweise) gleich lange Ketten ausgebildet sind (siehe Abb. 6.9). Deutlich ist der Unterschied in der Matrix λ zu sehen, da in den Zeilen 5, 9, 10 und 11 die Ziele 6 und 7 als unerlaubte Ziele hinzugekommen sind. Die Roboter bewegen sich nun auf ihre Ziele zu, bis sie diese alle erreicht haben und so zum Stillstand kommen. In Abb. 6.10 ist eine Situation während dieser Bewegung beobachten.

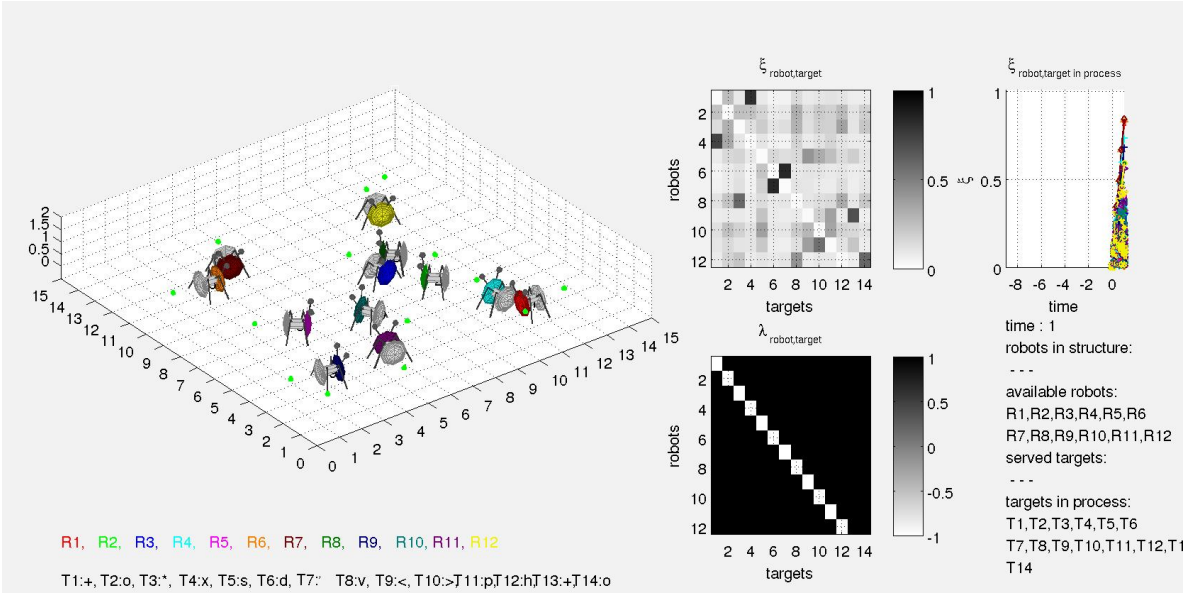

Abbildung 6.6.: Beispiel „Paralleles Bilden zweier Linien (1.Methode)“: Nach einer Simulationssekunde

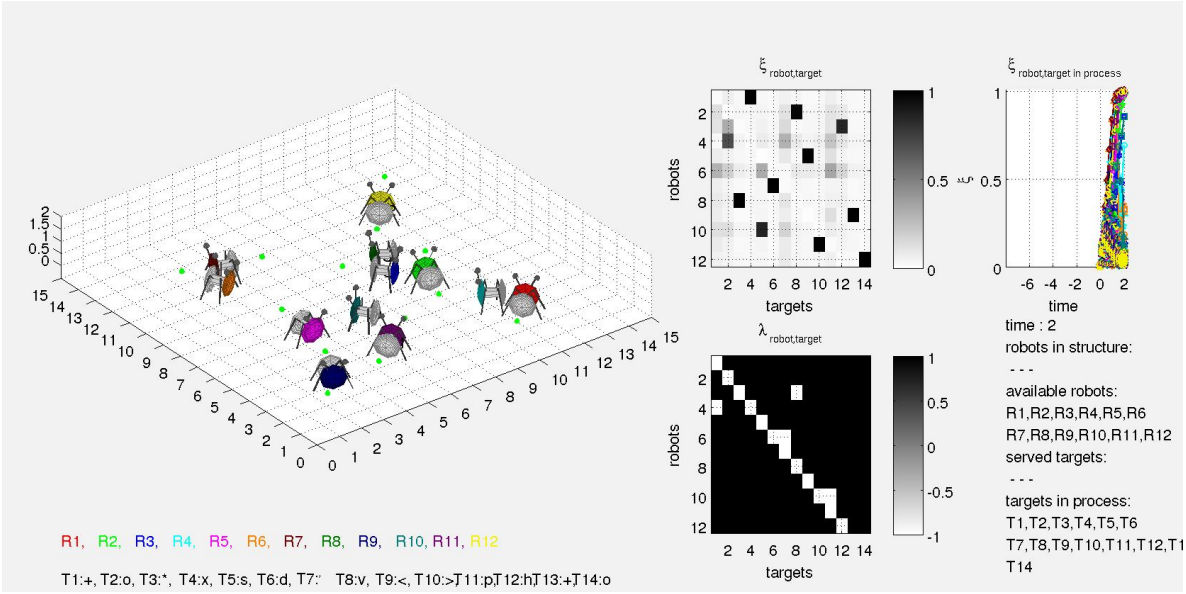

Abbildung 6.7.: Beispiel „Paralleles Bilden zweier Linien (1.Methode)“: Erste Paarbildungen

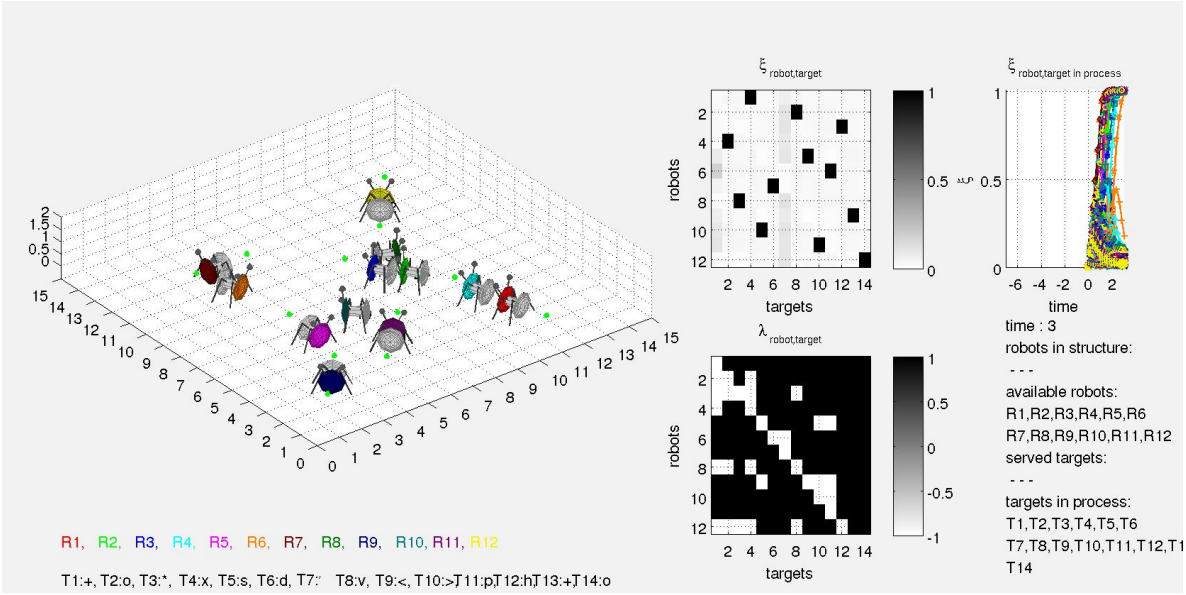

Abbildung 6.8.: Beispiel „Paralleles Bilden zweier Linien (1.Methode)“: Anfügen von Robotern zu bestehenden Paaren

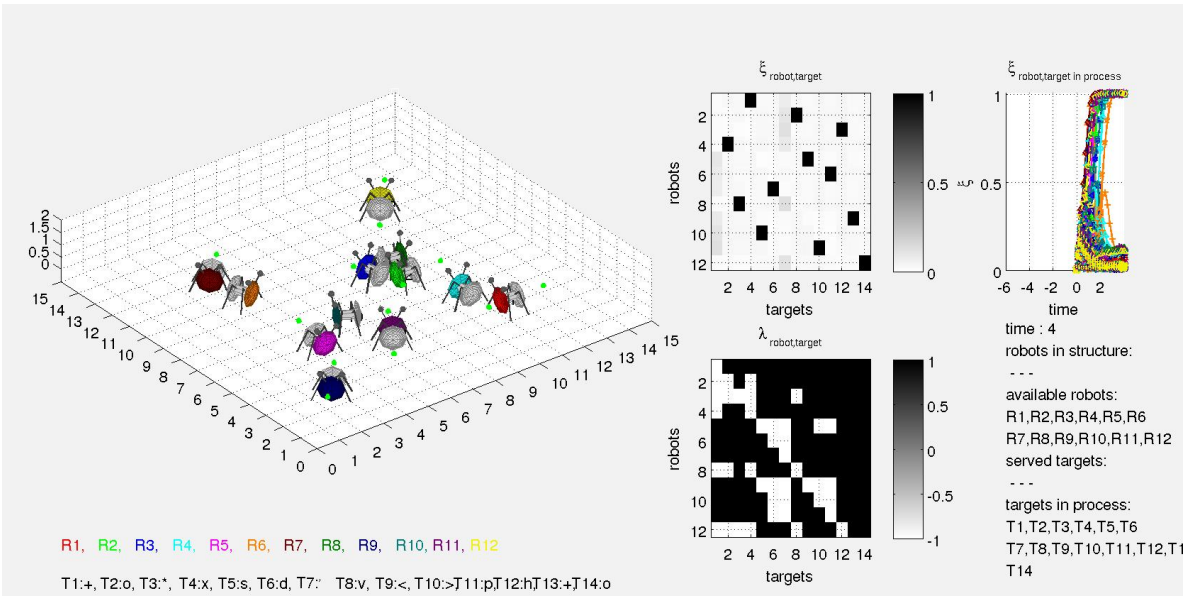

Abbildung 6.9.: Beispiel „Paralleles Bilden zweier Linien (1.Methode)“: Endgültige Zielverteilung

Beispiel „Paralleles Bilden eines Sterns (1. Methode)“

Analog zur sequentiellen Bildung eines Sterns (siehe 5.2.3), bei der für jede der Zacken ein Ziel im Raum vorgegeben sein muss, kann auch bei der parallelen Bildung eines Sterns

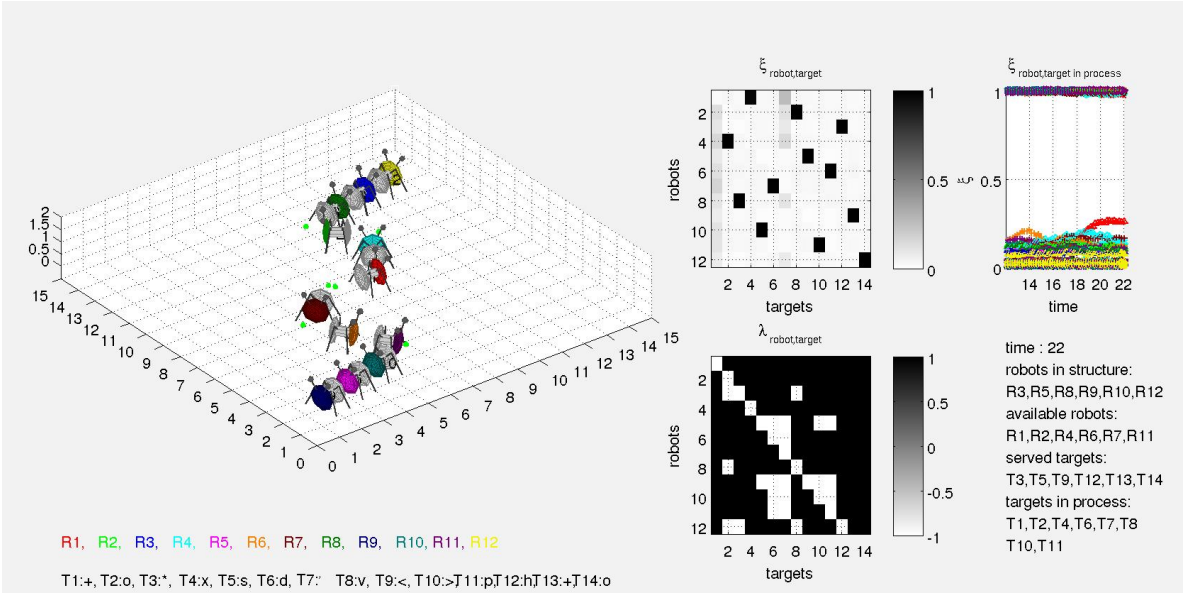

Abbildung 6.10.: Beispiel „Paralleles Bilden zweier Linien (1.Methode)“: Bewegung zu den jeweiligen Zielen

vorgegangen werden. Für Abbildungen soll an dieser Stelle auf den Anhang der Arbeit bzw. die Dateien auf der CD verwiesen werden. Eine vergleichbare Möglichkeit zur Erzeugung von gleichmäßig langen Zacken der Sterne, wie bei der sequentiellen Bildung von Sternen gesehen, gibt es hier nicht, da ohnehin alle Docking-Stationen anfangs aktiv geschaltet sind.

Beispiel „Paralleles Bilden einer Linie ohne Startpunkt (1. Methode)“

In diesem Beispiel wurde eine Simulation mit vier Robotern, aber ohne ein vorgegebenes Ziel durchgeführt. Es zeigte sich, dass hierbei eine kettenförmige Hintereinanderreihung der Roboter im Sinne von Zielzuordnungen möglich ist. Allerdings ist es auf diesem Weg nicht möglich, eine geometrische Linie zu bilden.

In Abb. 6.11 ist die Zielzuordnung zu sehen. Darüber hinaus ist zu erkennen, dass die Roboter 1, 3 und 4 auf dem Weg hinter Roboter 2 sind. Da aber die ξ-Werte für Roboter 2 nicht gleich 0 sind, gibt es immer leichte Bewegungen von Roboter 2 in Richtung dieser Ziele, die - sollte die Linie gebildet sein - alle hinter ihm liegen. Dies führt zu Drehungen (und langsamen Bewegungen) von Roboter 2. Durch die Drehungen bewegt sich die Docking-Station von Roboter 2 allerdings relativ schnell, und Roboter 1 kann diese nicht erreichen. Daher kann ohne ein vorgegebenes Ziel keine geometrisch stabile Linie aufgebaut werden.

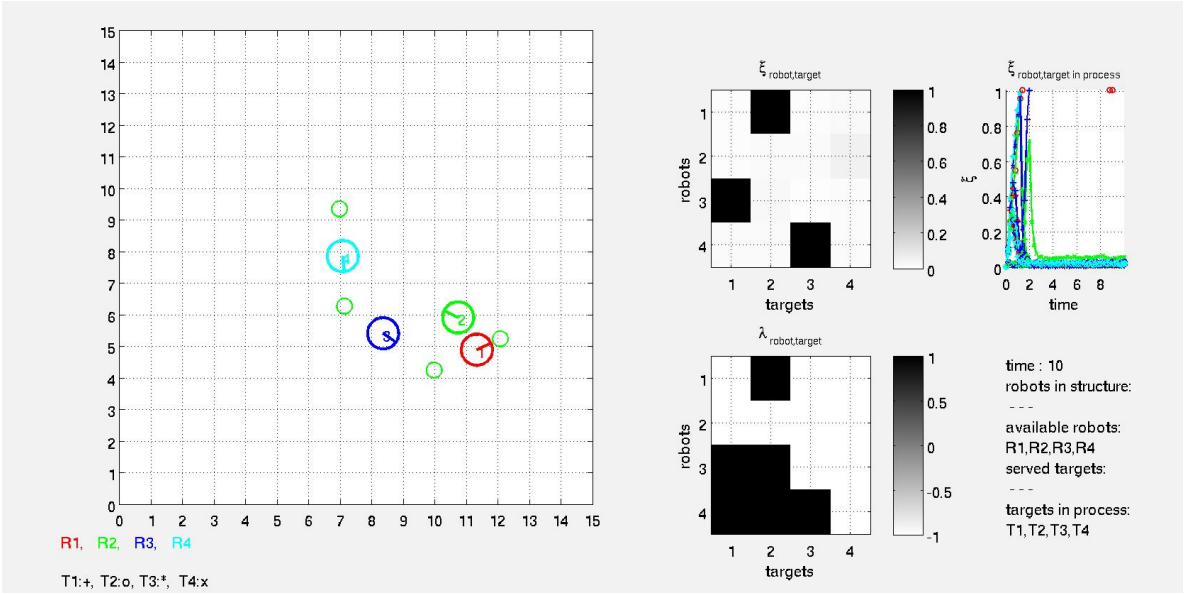

Abbildung 6.11.: Beispiel „Paralleles Bilden einer Linie ohne Startpunkt (1.Methode)“: Bewegung zu den jeweiligen Zielen

Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (1. Methode)“

Bei diesem Beispiel wurden die Startpositionen der vier Roboter vorgegeben. Dabei stehen sich jeweils zwei Roboter sehr nah gegenüber.

In Abb. 6.12 ist die Anfangssituation zu sehen. Das Beispiel wurde gewählt, um zu zeigen, dass die Zielverteilung bei dieser Methode relativ schnell vonstatten geht, und somit die beiden Roboterpaare sich nicht lange aufeinander zubewegen werden. Durch die Zyklenvermeidung müssen sich die Roboter 2 und 3 sehr früh (dies ist bereits nach einer Simulationssekunde passiert und in Abb. 6.13 zu sehen) neue Ziele suchen, und sie werden dadurch die Bewegung auf die anderen beiden Roboter abbrechen.

6.1.4. Bewertung der Lösung

Es konnte in diesem Abschnitt gezeigt werden, dass es möglich ist, Strukturen - genauer gesagt - Linien - zuverlässig parallel zu bilden. Damit ist ein wesentlich schneller arbeitendes Verfahren als die sequentielle Strukturbildung vorgestellt worden. Allerdings gibt es einige Punkte, die im Folgenden diskutiert werden sollen.

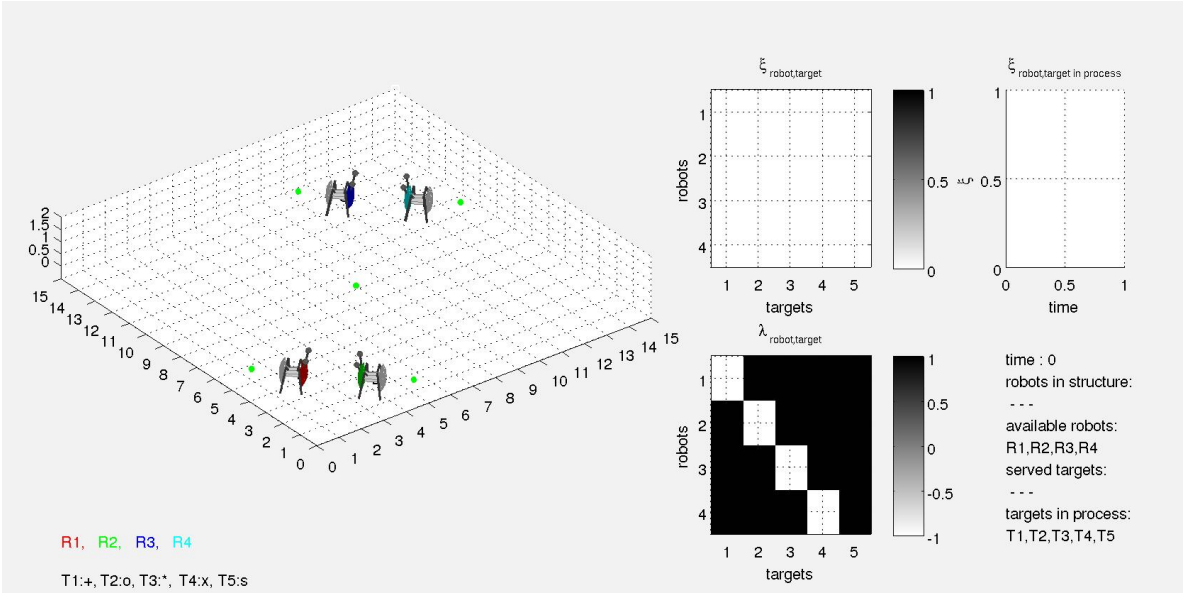

Abbildung 6.12.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (1.Methode)“: Zu Beginn der Simulation

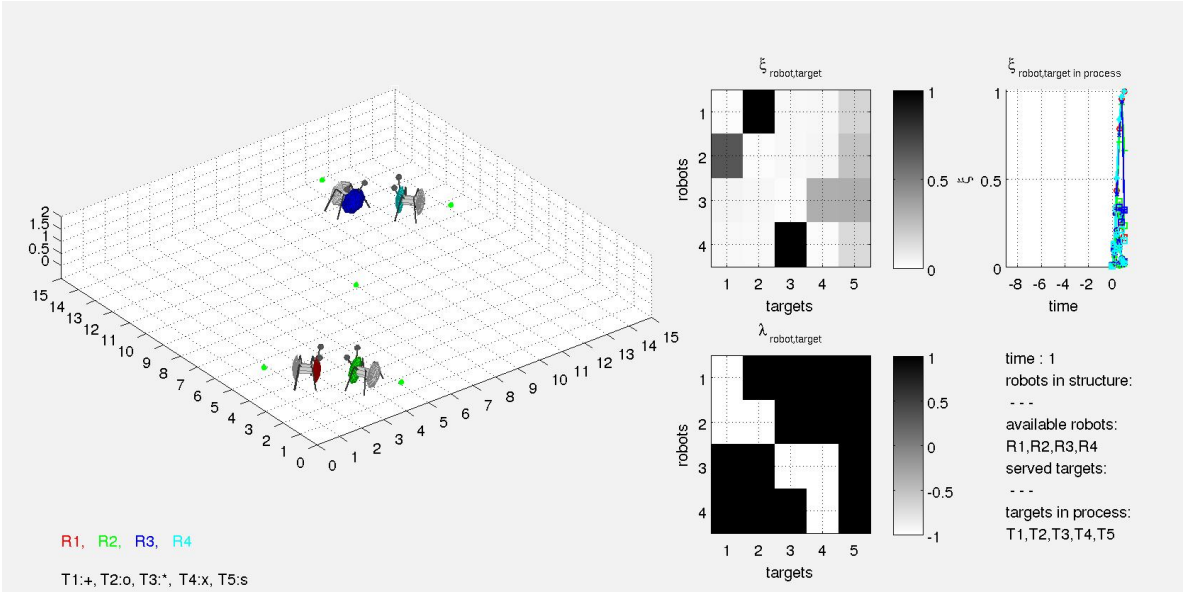

Abbildung 6.13.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (1.Methode)“: Zyklenvermeidung hat bei den Paaren eingegriffen

Die Bildung der Struktur liegt in der Logik der Matrix λ und nicht in den Selektionsgleichungen

An der hier vorgestellten Methode kann durchaus kritisiert werden, dass die Logik zur Linienbildung in der Matrix λ liegt, da bei der Erstellung dieser Matrix die Zyklenvermeidung zum Eingriff kommt. Man könnte sagen, dass die Selektionsgleichungen nur zur Roboter-Ziel-Verteilung dienen und nicht zur Ausgestaltung der eigentlichen Struktur. Hiergegen kann eingewendet werden, dass die Matrix λ genau den Zweck hat, zwischen erlaubten (aktiven) und unerlaubten (inaktiven) Zielen zu unterscheiden. Bei ihrer Erstellung kann durch logische Berechnungen auf if-Konstrukte verzichtet werden, und somit handelt es sich hierbei nicht um Sequenzen zum Abfragen von Einzelfällen.
Da die Matrix λ ein wesentlicher Bestandteil der gekoppelten Selektionsgleichungen ist, liegt die Logik zur Strukturbildung also doch auch in diesen. Insbesondere in dem (selteneren) Fall, dass ein vollständig geschlossener Zyklus, bei dem alle ξ-Werte den Schwellwert überschritten haben, entstanden ist. Hier wird durch die neuen, negativen Werte für λ die Zielzuordnung aufgebrochen und muss sich neu einstellen. In diesem Fall ist deutlich zu sehen, dass die Selektionsgleichungen einen wesentlichen Anteil an der Strukturgestaltung einnehmen.

Bildung komplexerer Strukturen

Unter 5.2 wurde beschrieben, dass komplexere Strukturen durch die sequentielle Strukturbildung erzeugt werden könnten, wenn man die Roboter mit mehreren Docking-Stationen ausstatten würde. Prinzipiell ist auch bei der parallelen Strukturbildung ein ähnlicher Weg möglich. Dieser soll am Beispiel der Sternbildung erläutert werden:
Jeder Roboter verfügt über drei Docking-Stationen im Winkel von 120°. Zunächst sollen wieder alle Ziele aktiv sein. Für die beiden zusätzlichen Docking-Stationen muss aber gelten, dass sie nur bei den Robotern aktiv bleiben dürfen, die vorgegebene Ziele im Raum als Zentrum eines Sterns bedienen. Eine Möglichkeit, dies zu erreichen, besteht darin, die zusätzlichen beiden Docking-Stationen aller übrigen Roboter inaktiv zu schalten, wenn alle vorgegebenen Ziele besetzt sind (alternativ wenn eindeutige Zielzuordnungen hierauf bestehen). Es ist sehr wahrscheinlich, dass sich bis zu diesem Zeitpunkt Zielzuordnungen zu diesen Docking-Stationen aufgebaut haben werden. Diese werden dann aufbrechen und die Roboter andere, nahe liegende Ziele suchen.
Eine andere Alternative besteht darin, die zusätzlichen Docking-Stationen eines Roboters inaktiv zu schalten, wenn er als Ziel eine Docking-Station verfolgt. Diese Variante hätte den Vorteil, dass die zusätzlichen Docking-Stationen, die nicht bedient werden dürfen, früher inaktiv geschaltet würden. Außerdem kann es hier als Vorteil angesehen werden, dass kein globales Wissen, also die Information, ob alle vorgegebenen Ziele bedient sind, benötigt wird.

Frühes Aufbrechen von Paaren

Im letzten der vorgestellten Beispiele wurde das Verhalten von Roboterpaaren gezeigt. Es ist sehr auffällig, dass die Paare sehr früh aufbrechen, da die ξ-Werte recht schnell anwachsen und damit die Zielverteilung schnell abgeschlossen ist. Dies führt ebenfalls schnell zum Deaktivieren eines der Ziele und so muss sich einer der Roboter sehr früh umorientieren. Er kann dann nicht weiter auf den anderen Roboter zufahren. Es kann durchaus aber erwünscht sein, dass die Roboter möglichst lange aufeinander zufahren und erst dann die Zyklenvermeidung eingreift. Dieses Verhalten hätte insofern Vorteile, dass die Paare schneller gebildet sein könnten, da der eine Roboter nicht vor dem anderen wegfährt. Eine Möglichkeit, dieses Verhalten zu erreichen, soll in Abschnitt 6.2 beschrieben werden.

6.2. Paralleles Bilden einer Linie mit Hilfe der ξ-Werte und des Abstandes der Roboter untereinander

Im vorangegangenen Abschnitt wurde gezeigt, dass das parallele Bilden von Ketten mit Hilfe der ξ-Werte zu einem frühen Aufbrechen von Paaren führt und diese sich nicht mehr aufeinander zubewegen, sondern dass ein Teil sich von dem anderen wegbewegt. Um dieses Verhalten zu vermeiden, soll hier ein weiteres Verfahren vorgestellt werden.

6.2.1. Idee des Verfahrens

Im Prinzip soll das Verfahren aus 6.1 nur adaptiert werden, um der Forderung nachkommen zu können, dass Paare von Robotern (oder Roboterketten) sich länger aufeinander zubewegen, bevor einer der beiden Teile durch die Zyklenvermeidung gestoppt wird und sich ein neues Ziel suchen muss. Es wird also eine Distanz definiert, ab der die Zyklenvermeidung ansprechen darf. Sind die Roboter weiter entfernt, so werden zyklische Strukturen nicht erkannt und sie bewegen sich weiterhin aufeinander zu. Wichtig ist, dass diese Distanz nicht zu klein definiert wird, da sonst die Zyklenvermeidung nicht früh genug ansprechen kann, und die Roboter tatsächlich in rotierende zyklische Strukturen fallen können.

6.2.2. Matlab Funktionen

Die Funktionen konnten alle - bis auf die Funktion *setLambda* - unverändert übernommen werden.

setLambda

Die Funktion *setLambda* arbeitet vollkommen analog zur Funktion *setLambda* aus 6.1, außer, dass bei der Erstellung der Matrix „headOfRobots“ nicht nur die ξ-Werte, sondern auch der Abstand der betroffenen Roboter berücksichtigt wird. Das Propagieren der weiteren Nachfolger bleibt hiervon vollkommen unverändert.

6.2.3. Beispiele

Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (2. Methode)“

Um die zweite Methode mit der ersten vergleichen zu können, wurde das gleiche Beispiel mit zwei paarweise aufgestellten Robotern gewählt.

In Abb. 6.14 ist die identische Startaufstellung zu sehen. Während bei der ersten Methode das Aufbrechen der Zyklen (und damit auch der Paare) bereits nach einer Simulationssekunde zu beobachten war, kann dies durch die zweite Methode verzögert werden (siehe Abb. 6.15).

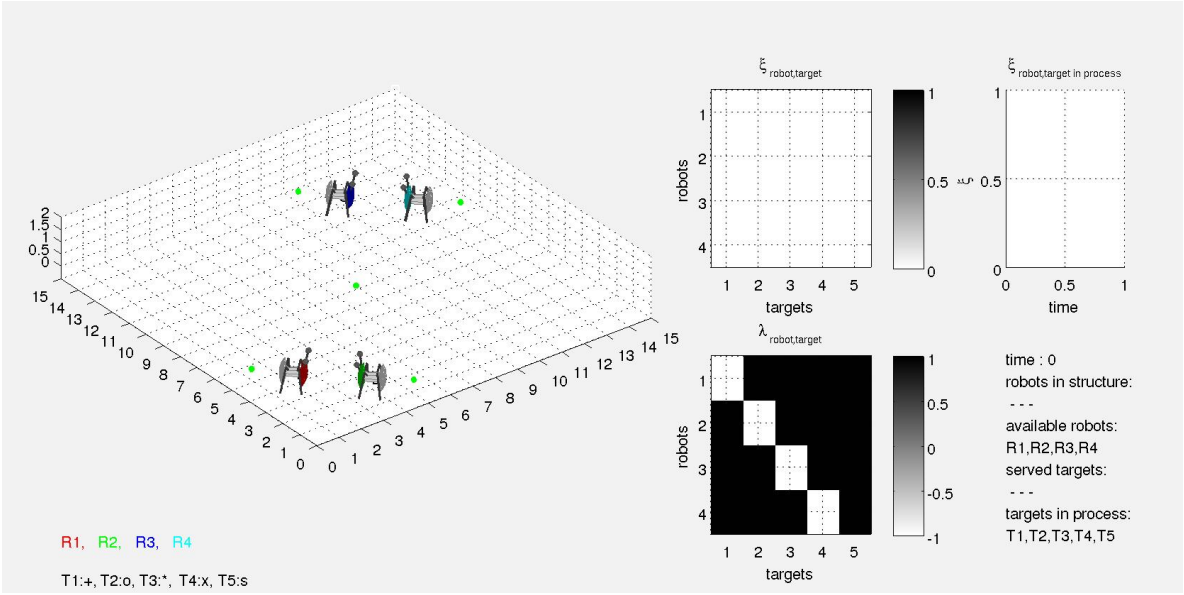

Abbildung 6.14.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (2.Methode)“: Zu Beginn der Simulation

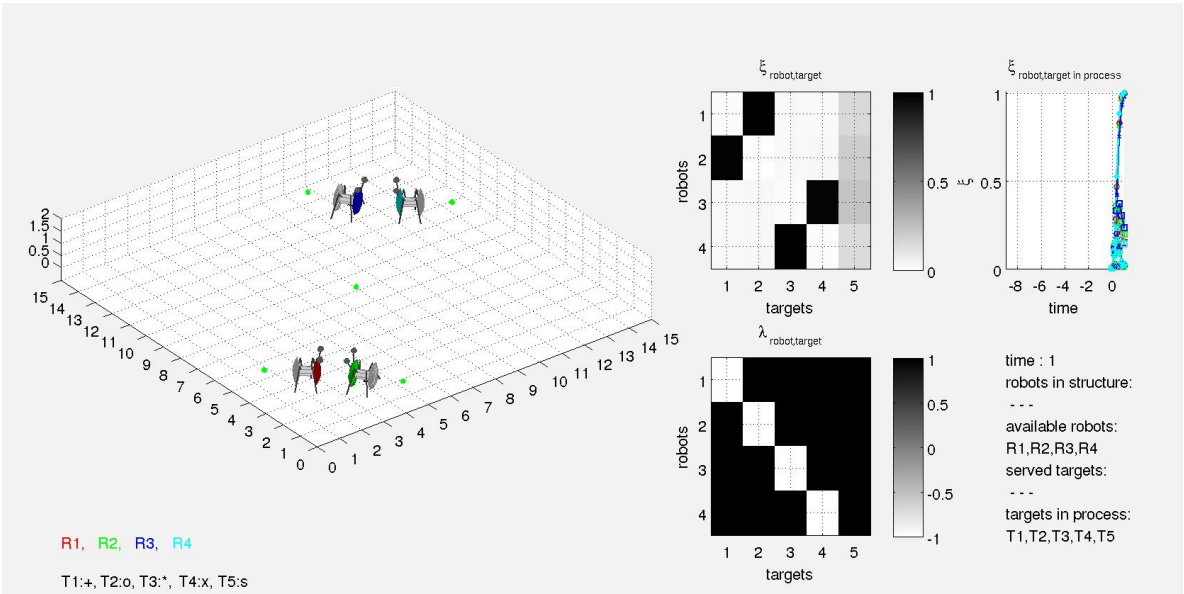

Abbildung 6.15.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (2.Methode)“: Zyklenvermeidung wird verzögert

Die Roboterpaare bewegen sich unvermindert schnell aufeinander zu. Erst nachdem ein definierter Abstand unterschritten ist, setzt die Zyklenvermeidung ein. In Abb. 6.16 ist zu

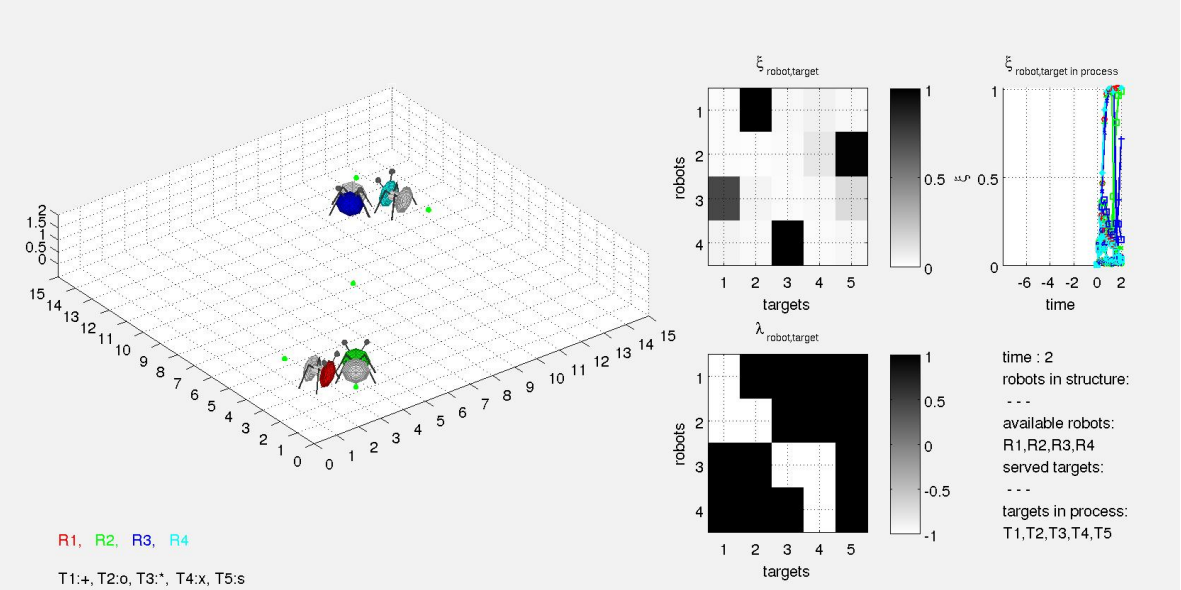

Abbildung 6.16.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (2.Methode)“: Zyklenvermeidung hat bei den Paaren eingegriffen

sehen, wie dicht sich die Roboter jeweils annähern konnten. Es ist auch zu erkennen, dass nur zwei der ξ-Werte ($\xi_{1,2}$ und $\xi_{4,3}$) den Schwellwert von 0.99 überschritten haben. Dies ist daran festzustellen, dass andernfalls Zyklen in der Matrix „headOfRobots“ bestanden hätten, was zu Zielverlusten aller Roboter innerhalb des Zyklus geführt hätte. In diesem Fall würden alle Werte für ξ sinken und sich nach Unterschreitung des Schwellwertes neu einstellen. Es wäre interessant, diesen Vorgang in einer eigenen Simulation genau zu zeigen. Hierzu müssten die Roboterpaare weiter entfernt stehen, damit die Zeit zum Anstieg der ξ-Werte größer wird und somit alle Werte über den Schwellwert steigen, bevor die Distanzen der Roboter unter den vorgegebenen Wert sinken.

6.2.4. Bewertung der Lösung

Es konnte gezeigt werden, dass das vorgestellte Verfahren die Erwartungen in den Beispielen erfüllt. Allerdings ist anzumerken, dass bei diesem Verfahren die Wahrscheinlichkeit von Zyklen in der Matrix „headOfRobots“ und damit erzwungenen Zielverlusten des gesamten Zyklus erheblich steigt. Dies kommt daher, dass durch das längere Aufeinanderzufahren es wahrscheinlicher ist, dass beide ξ-Werte den Schwellwert überschritten haben. Da die zweite Bedingung, der Abstand zwischen den beiden Robotern, keine Ordnung der Roboter vornimmt, werden beide Roboter jeweils als Vorgänger des anderen angesehen und verlieren ihn als Ziel.
Eine Alternative, dieses Problem größtenteils zu umgehen, besteht darin, nicht den Abstand zwischen zwei Robotern als entscheidendes Maß zu wählen, sondern den Abstand zwischen

einem Roboter und der entsprechenden Docking-Station. Es kann als ausgesprochen unwahrscheinlich angesehen werden, dass bei einem Roboterpaar die Abstände von Robotern zu Docking-Stationen der anderen Roboter identisch sind. Es muss dann allerdings zusätzlich überprüft werden, welcher der Abstände eines Paares der kürzere ist, um die Matrix „headOfRobots“ anzupassen.
Bei beiden Verfahren kann die Implementierung mit Hilfe von if-Konstrukten erfolgen. Es kann aber auch weitestgehend auf solche Konstrukte verzichtet und stattdessen mit logischen Termen gearbeitet werden.

6.3. Paralleles Bilden einer Linie mit Hilfe des Abstandes der Roboter untereinander

In 6.2 wurden sowohl die Werte von ξ als auch die Abstände der Roboter zur Kollisionsvermeidung eingesetzt. Es erscheint, als würden zwei unterschiedliche Technologien vermischt eingesetzt. Dies wirft die Frage auf, ob es nicht möglich sein könnte, die Kollisionsvermeidung auch ausschließlich mit Hilfe der Abstände der Roboter zu implementieren.

6.3.1. Idee des Verfahrens

Soll die Kollisionsvermeidung durch die Abstände der Roboter die Werte der Matrix λ steuern, muss zunächst definiert werden, wann davon ausgegangen werden soll, dass ein Roboter ein Ziel verfolgt und damit der Nachfolger dieses Ziels ist.
In Anbetracht der Tatsache, dass ein Roboter, der eine Docking-Station oder ein vorgegebenes Ziel erreicht hat, die selben Koordinaten wie diese hat, erscheint der Abstand als ein gutes Maß für den Zielzuordnungsbegriff. Es soll also davon ausgegangen werden, dass der Roboter, der am nächsten zu einer Docking-Station steht, auch Nachfolger des Roboters der Docking-Station ist. Es ist klar, dass es Situationen geben kann, in denen diese Annahme falsch ist. Ein Beispiel hierzu wäre folgende Situation: Eine Linie soll gebildet werden. Wenn nun der hintere Teil der im Entstehen befindlichen Linie sich am vorderen Teil vorbeibewegen muss und hierbei die Abstände der Roboter relativ groß sind, kann es vorkommen, dass ein Roboter aus dem hinteren Teil der Kette einer Docking-Station des vorderen Teils näher kommt als der eigentliche Nachfolger dieses Roboters. Die Zyklenerkennung würde fälschlicherweise den Roboter des hinteren Teils der Kette als den Nachfolger dieses Roboters ansehen. In Beispielen wird allerdings gezeigt werden, dass das Verfahren der gekoppelten Selektionsgleichungen robust gegenüber solchen fehlerhaften Annahmen ist.
Auch bei diesem Verfahren sollen nur die direkten Nachfolger der Roboter bestimmt werden. Die weiteren Nachfolger in der Kette sollen analog zu den bereits vorgestellten Verfahren durch Propagierung ermittelt werden. Eine wichtige Voraussetzung, um diese Methode zu verwenden, ist, dass die Vorgänger-Nachfolger-Relation eindeutig ist. Dies war bei den vorangegangenen Methoden durch die Nutzung der Informationen der Matrix ξ oberhalb des definierten Schwellwertes immer eindeutig. Bei der Nutzung des Abstandes zur Bestimmung des Nachfolgers eines Roboters ist hierauf besonders zu achten.

6.3.2. Matlab Funktionen

Da sich bei der Entwicklung dieses Verfahrens die Änderungen nur auf die Bestimmung der Matrix λ beschränken, konnten alle anderen Funktionen unverändert übernommen werden.

setLambda

Zur Bildung der Matrix λ werden folgende Schritte durchgeführt:

- Erzeugen einer Matrix als Einheitsmatrix, bei der die Hauptdiagonale mit -1 besetzt ist.
- Bestimmung der Matrix „headOfRobots“.
- Propagieren der Nachfolger-Relation und gleichzeitiges Setzen von Werten in der Matrix λ.

Der erste Punkt ist von den bisherigen Funktionen bekannt.
Die Bestimmung der Matrix „headOfRobots“ ist hier aufwändiger und läuft in vier Schritten ab:
Im ersten Schritt werden die Abstände zwischen Robotern und Zielen dahingehend überprüft, ob diese kleiner als ein definierter Abstand sind. Nur wenn dies der Fall ist, kann der Roboter als ein Nachfolger des Ziels angesehen werden. In diesem Fall soll die Matrix „headOfRobots“ an dieser Stelle eine 1, andernfalls eine 0 tragen. Es ist sinnvoll, bei dieser Betrachtung nur Ziele zu berücksichtigen, die Docking-Stationen sind. Es ist klar, dass diese Betrachtung nicht eindeutig ist und daher nicht als ausreichende Nachfolger-Relation angesehen werden kann.
Im zweiten Schritt sollen alle Zeilen eindeutig gemacht werden. Es darf also in jeder Zeile der Matrix „headOfRobots“ nur eine 1 von den vorhandenen übrig bleiben. Diese soll es nur in der Spalte geben, die für die Docking-Station steht, zu der der Roboter den geringsten Abstand hat. Alle übrigen Einträge einer Zeile werden 0 gesetzt.
Es kann allerdings immer noch sein, dass nach diesem Schritt einzelne Spalten nicht eindeutig sind. Dies bedeutet, dass diese Docking-Station für mehrere Roboter das nächste Ziel darstellt und die Abstände auch kleiner sind als der definierte Abstand. Um die Spalten eindeutig zu machen, wird analog zu den Zeilen verfahren.
Um den sonst häufigen Verlust ganzer Zielzuordnungen zu vermeiden, werden alle Paare von Robotern i und j betrachtet, und im Fall einer zyklischen Abhängigkeit der Paare wird derjenige Roboter als Nachfolger ausgewählt, der eine kürzere Distanz zur Docking-Station des anderen aufweist. In aller Regel ist dies eindeutig.
Nach diesen Maßnahmen ist die Matrix „headOfRobots“ garantiert eine eindeutige Nachfolger-Relation, und sie kann zur Propagierung dieser Information und zum Setzen der unerlaubten Ziele in der Matrix λ in der bekannten Art und Weise verwendet werden.

6.3.3. Beispiele

Alle Beispiele wurden mit den gleichen Parametern erstellt, wie alle anderen Beispiele für die parallele Strukturbildung auch. Es soll hier insbesondere auf die spezifischen Eigenschaften der neuen Methode eingegangen werden, da hier fehlerhafte Annahmen über die Zielzuordnungen getroffen werden können (s.o.).

Beispiel:„Paralleles Bilden einer Linie (3.Methode)“

Bei diesem Beispiel handelt es sich um das Bilden einer Linie aus insgesamt neun Robotern mit zufällig gewählten Anfangspositionen.

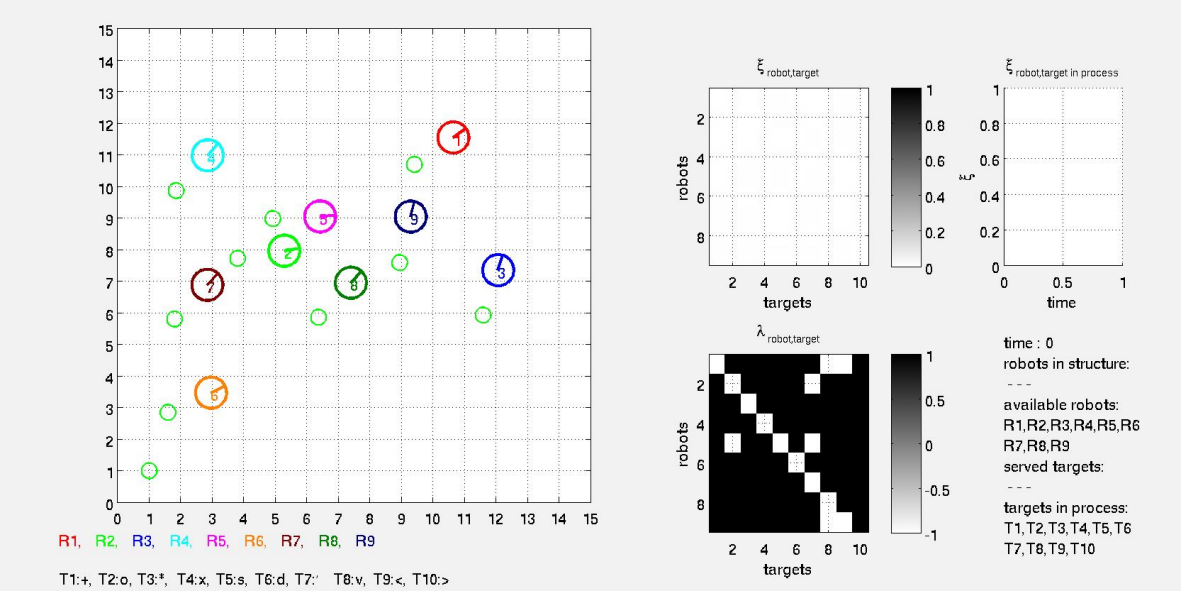

Abbildung 6.17.: Beispiel „Paralleles Bilden einer Linie (3.Methode)“: Zu Beginn der Simulation

Bereits zu Beginn der Simulation (siehe Abb. 6.17) werden von der Zyklenvermeidung Annahmen auf Grund der Positionen getroffen, die so nicht richtig sein können: Es wird angenommen, dass bereits zwei Ketten von Robotern bestehen. Zum einen ist dies die Kette $(1,9,8)$ und zum anderen die Kette $(5,2,7)$. Diese Annahmen, resultierend aus den nahe gelegenen Positionen der Roboter, müssen allerdings falsch sein, da zu Beginn die Werte für ξ nahe Null sein müssen. Nach zwei Simulationssekunden (siehe Abb. 6.18) haben sich folgende Zuordnungen gebildet: Die Kette $(6,7,4,2,5,8)$ und der Zyklus $(1,9,3)$. Korrekt als Nachfolger eines Roboters wird in dieser Situation noch kein Roboter erkannt. Fälschlicher Weise wird Roboter 8 als Nachfolger von Roboter 2 und Roboter 4 als Nachfolger von Roboter 5 angesehen. Als besonders interessant ist hier zu bemerken, dass Roboter 5 tatsächlich Roboter 2 als Ziel hat, dieses Ziel aber im Folgenden (siehe Abb. 6.19) verliert, weil die Roboter $(5,2,8)$ als eine Kette angesehen werden. Daher darf Roboter 5 unter dieser Annahme weder Roboter 2 noch Roboter 8 weiter als Ziel behalten. Es ist in Abb. 6.19 aber auch zu sehen, dass die Roboter $(6,7,4)$ korrekt als Kette angesehen werden, da sie zur tatsächlichen Kette $(6,7,4,2)$ gehören. Der bestehende Zyklus aus den Robotern $(1,9,3)$ hat sich bisher nicht verändert und konnte nicht erkannt werden, weil das Dreieck, das diese drei Roboter bilden, zu groß ist und damit von der Zyklenerkennung nicht als drei aufeinander folgende Roboter erkannt werden kann. Es ist allerdings sehr deutlich zu sehen, dass dieses Dreieck bei der Rotation, die es vollzieht, immer kleiner wird, und so die

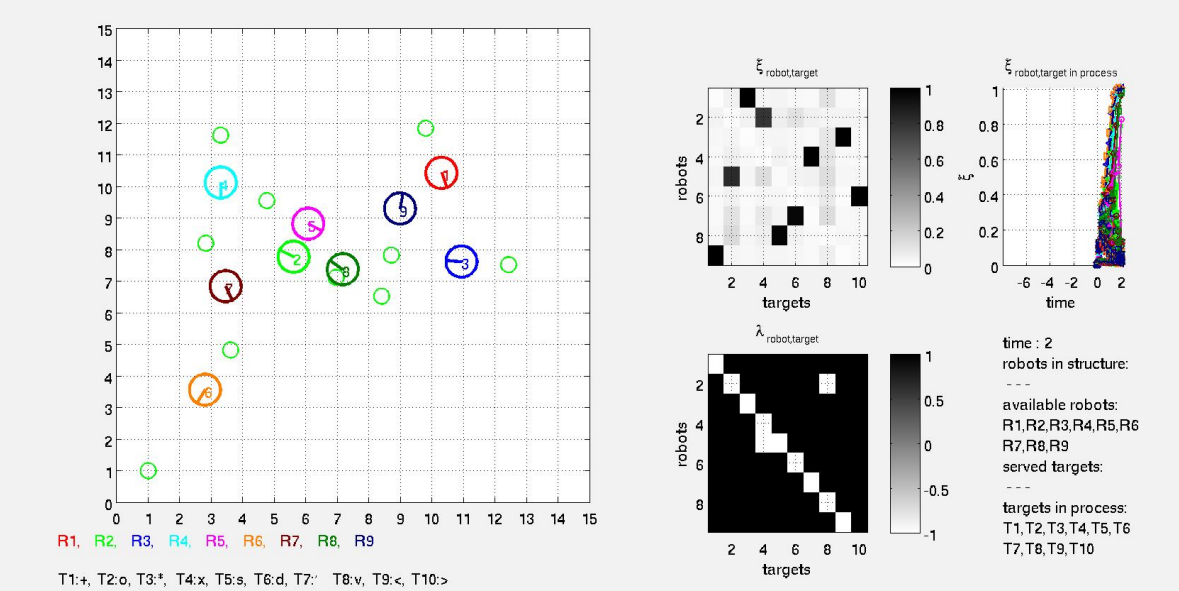

Abbildung 6.18.: Beispiel „Paralleles Bilden einer Linie (3.Methode)“: Erste Zielzuordnungen sind ausgebildet, werden aber bisher noch nicht korrekt erkannt.

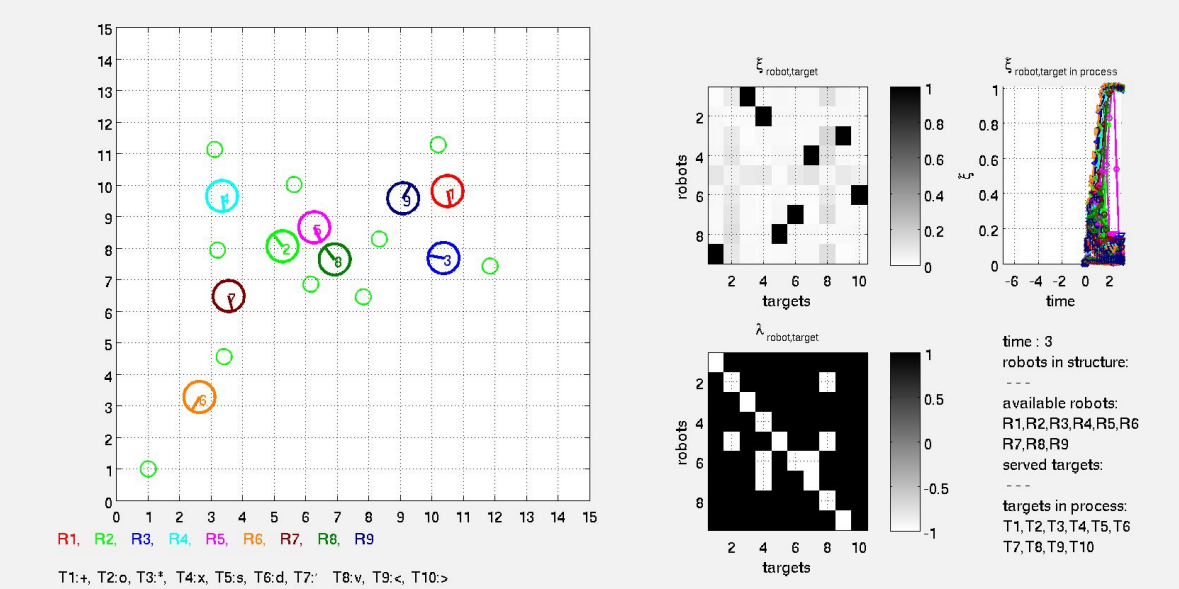

Abbildung 6.19.: Beispiel „Paralleles Bilden einer Linie (3.Methode)“: Roboter 5 verliert sein Ziel 2, auf Grund einer falschen Annahme

Abstände der Roboter verkürzt werden. Dies geschieht so lange, bis Roboter als Vorgänger und Nachfolger erkannt werden und die Vorgänger alle ihre Nachfolger als Ziele verlieren.

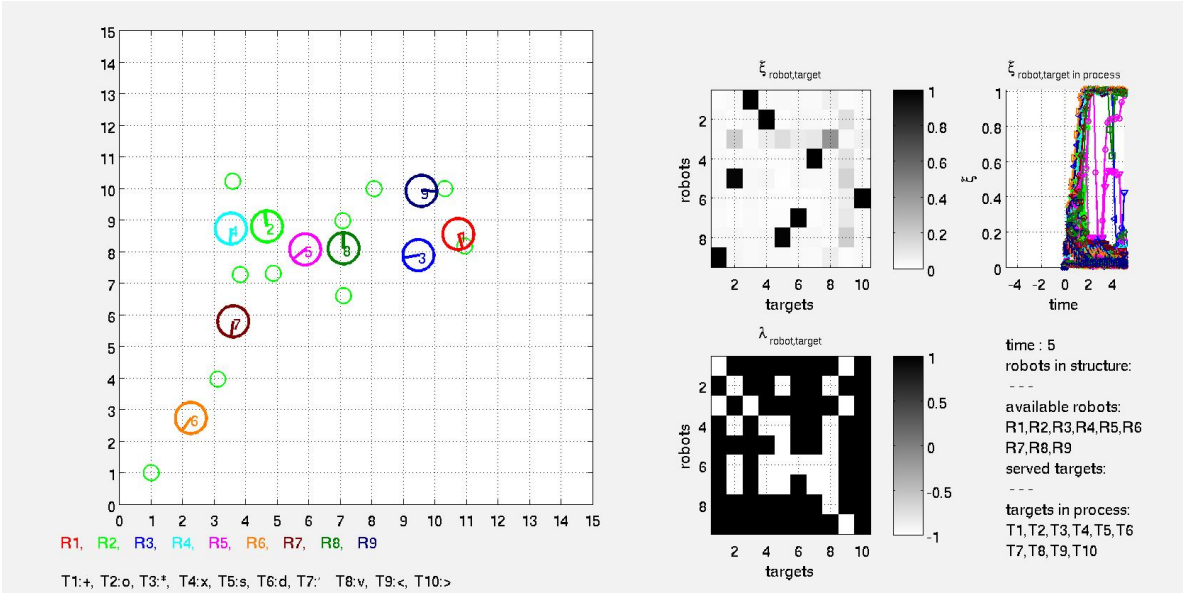

Abbildung 6.20.: Beispiel „Paralleles Bilden einer Linie (3.Methode)“: Aufbrechen des Zyklus (1,9,3)

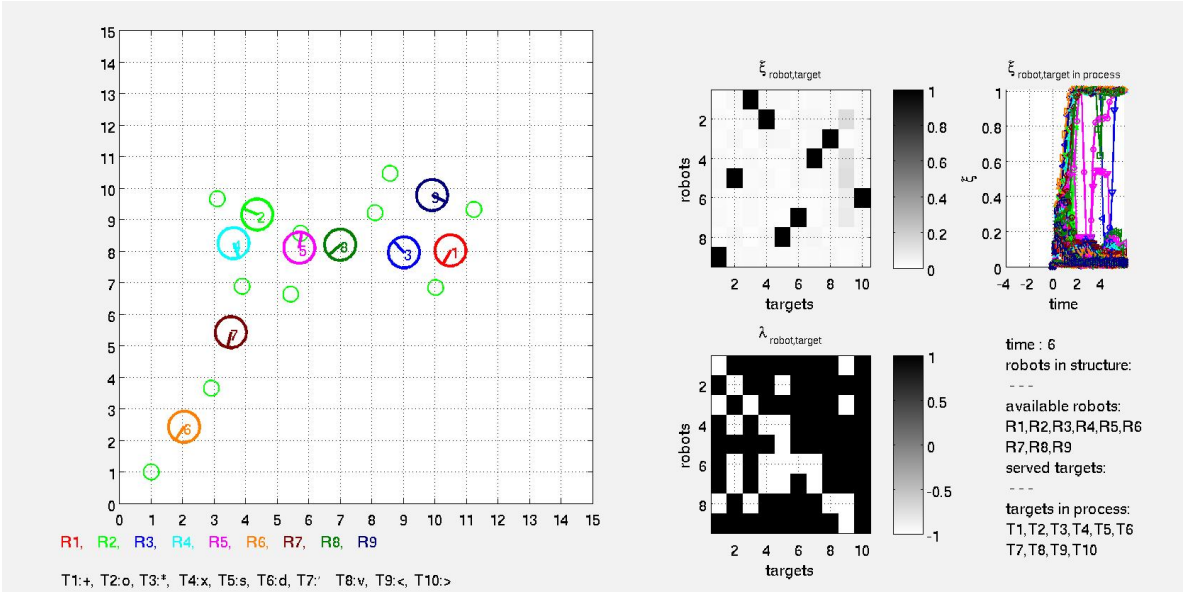

Abbildung 6.21.: Beispiel „Paralleles Bilden einer Linie (3.Methode)“: Endgültige Zielverteilung und deren teilweise Erkennung

Zu sehen ist dieser Vorgang in Abb. 6.20. Hier wurden die Roboter 1 und 9 als Nachfolger des Roboters 3 erkannt und so der Zyklus aufgebrochen. Roboter 3 wird sich ein neues

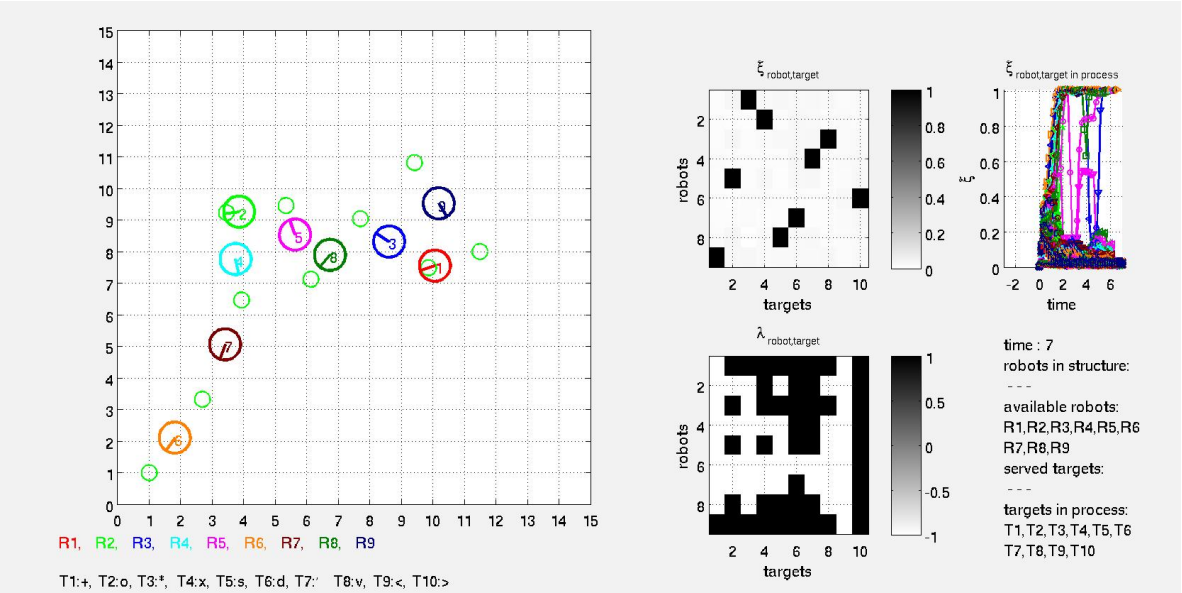

Abbildung 6.22.: Beispiel „Paralleles Bilden einer Linie (3.Methode)“: Endgültige Zielzuordnung und deren korrekte Erkennung

Ziel suchen (und dies in Roboter 8 finden). Sehr deutlich ist die korrekte Erkennung der beiden bestehenden Ketten $(6,7,4,2,5,8)$ und $(3,1,9)$ zu sehen. Da Roboter 3 ein neues Ziel sucht, aber das einzige für Roboter 3 erlaubte Ziel, das sonst kein anderer Roboter verfolgt, Roboter 8 ist, ist das Ansteigen des Wertes $\xi_{3,8}$ eine erwartete Folge. Damit ist die endgültige Zielverteilung abgeschlossen (siehe Abb. 6.21). Allerdings wird diese Kette bisher noch nicht als eine Kette, sondern als zwei Ketten erkannt $((6,7,4,2,5)$ und $(8,3,1,9))$. Dies liegt an dem zu großen Abstand von Roboter 8 zu Roboter 5. In Abb. 6.22 ist die Kette von Robotern auf ihrem Weg zum vorgegebenen Ziel zu sehen. Die Zielzuordnungen werden so erkannt, wie sie auch tatsächlich durch die Matrix ξ repräsentiert werden. Da es bei der Bewegung der Kette auch zu Schwingungen kommt (vgl. 5.4 und 6.4), kann es passieren, dass die vormals wahrgenommene Kette nur noch als mehrere Teilketten erkannt werden kann, wenn der Abstand zwischen zwei aufeinander folgenden Robotern zu groß geworden ist (hauptsächlich durch die Drehung eines Roboters verursacht).

Beispiel:„Paralleles Bilden zweier Linien (3.Methode)“

Bei diesem Beispiel sollen zwei Linien mit insgesamt zwölf Robotern gebildet werden. Die Startpositionen der Roboter sind wieder zufällig gewählt und die beiden Ziele sind von vorangegangenen Beispielen bekannt.

Es würde an dieser Stelle zu weit führen, die gesamte Simulation in jedem Detail zu kommentieren. Daher soll hier auf die Bilder und Videos auf der CD verwiesen werden. Beispielhaft soll eine Situation beschrieben werden:

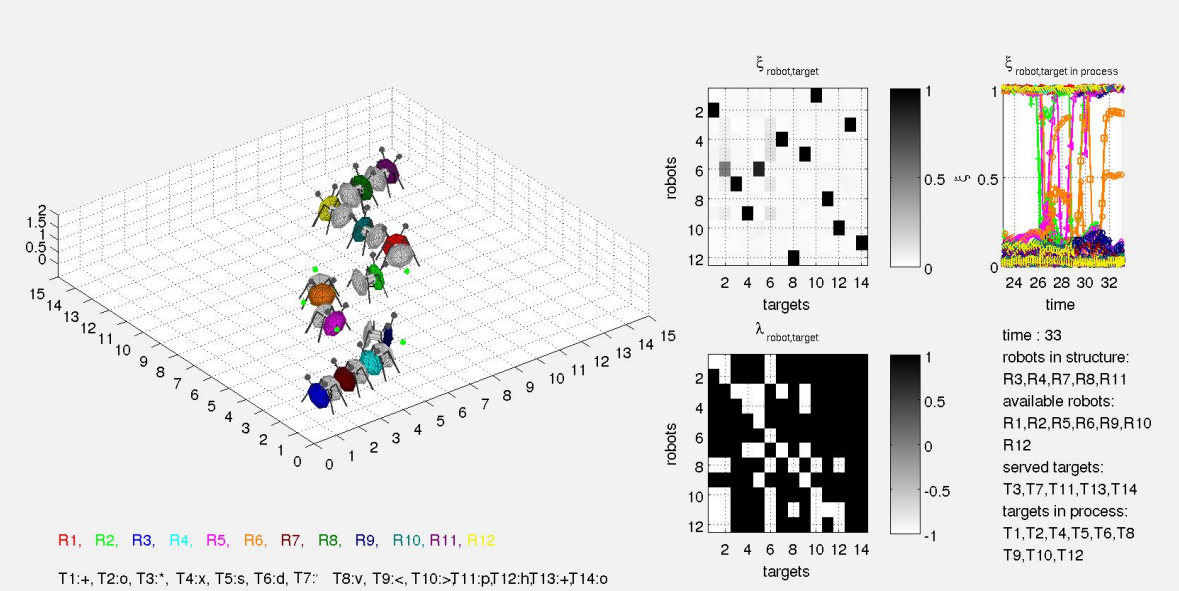

Abbildung 6.23.: Beispiel „Paralleles Bilden zweier Linien (3.Methode)“: Endgültige Zielzuordnung

In Abb. 6.23 ist die Situation zu sehen, in der bereits die endgültige Zielzuordnung besteht und Anfangsteile der beiden Linien gebildet sind und diese auch bereits ihre Ziele erreicht haben. Roboter 6 ist noch nicht sehr fest der Docking-Station 5 zugeordnet und der Wert $\xi_{6,2}$ noch nicht ganz abgefallen. Roboter 6 wird auch auf Grund seiner Nähe zu Roboter 2 als dessen Nachfolger angesehen. In der Folge wird dies aber nicht mehr der Fall sein, da sich Roboter 2 immer weiter von Roboter 6 entfernen wird.

Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (3.Methode)“

Als letztes Beispiel soll gezeigt werden, wie die Aufgabe der paarweisen Startaufstellung bewältigt wird. Die Startaufstellung und die Ziele sind identisch zu den vorangegangenen Beispielen mit den beiden anderen Methoden zur Zyklenvermeidung.

In Abb. 6.24 ist nach zwei Simulationssekunden zu sehen, dass die Zyklenvermeidung die Zielzuordnung eines Roboters am Auflösen ist. Die des anderen Paares besteht noch fort. Damit zögert diese Methode das Eingreifen am weitesten hinaus.

Die Roboter bilden die beiden Ketten $(2,1)$ und $(4,3)$ und bewegen sich - wie bisher bekannt - auf das Ziel in der Mitte des Feldes zu. Allerdings kommt es dort zu einem Verlust des Zieles von Roboter 4. In Abb. 6.25 ist dies zu sehen. Auf Grund der Kollisionsvermeidung steht Roboter 4 kurzzeitig seinem Nachfolger entgegengesetzt und die Docking-Station von Roboter 4 befindet sich in der Nähe von Roboter 2. Dadurch wird Roboter 2 als Nachfolger von Roboter 4 angesehen. Da Roboter 1 sein Ziel (die Docking-Station von Roboter 2) bereits

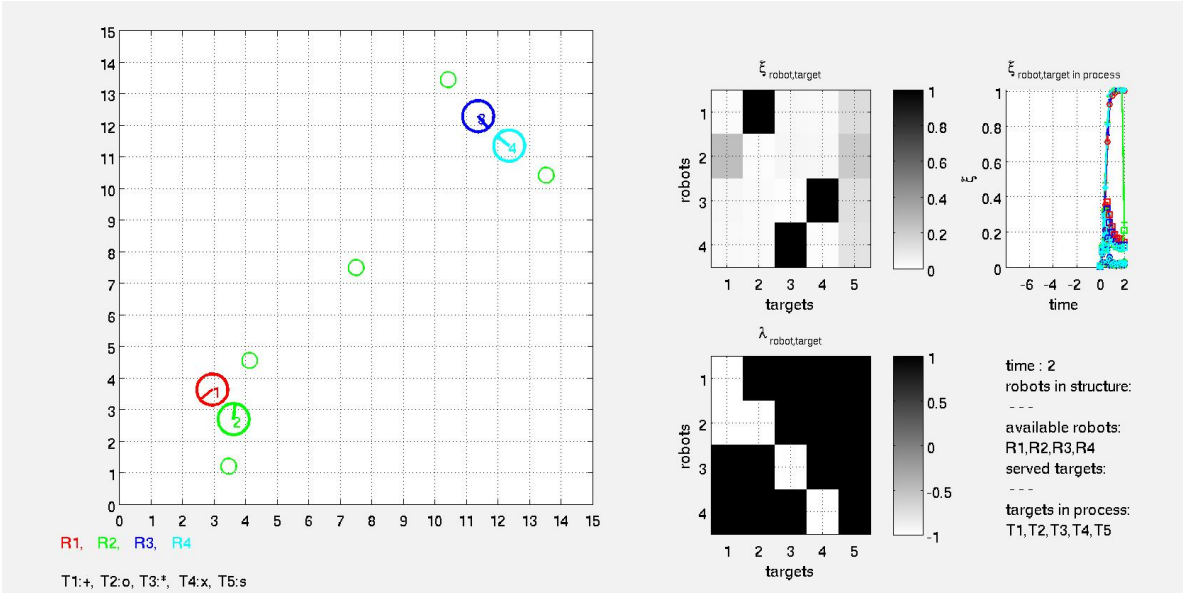

Abbildung 6.24.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (3.Methode)“: Zyklenvermeidung ist bei den Paaren am Eingreifen

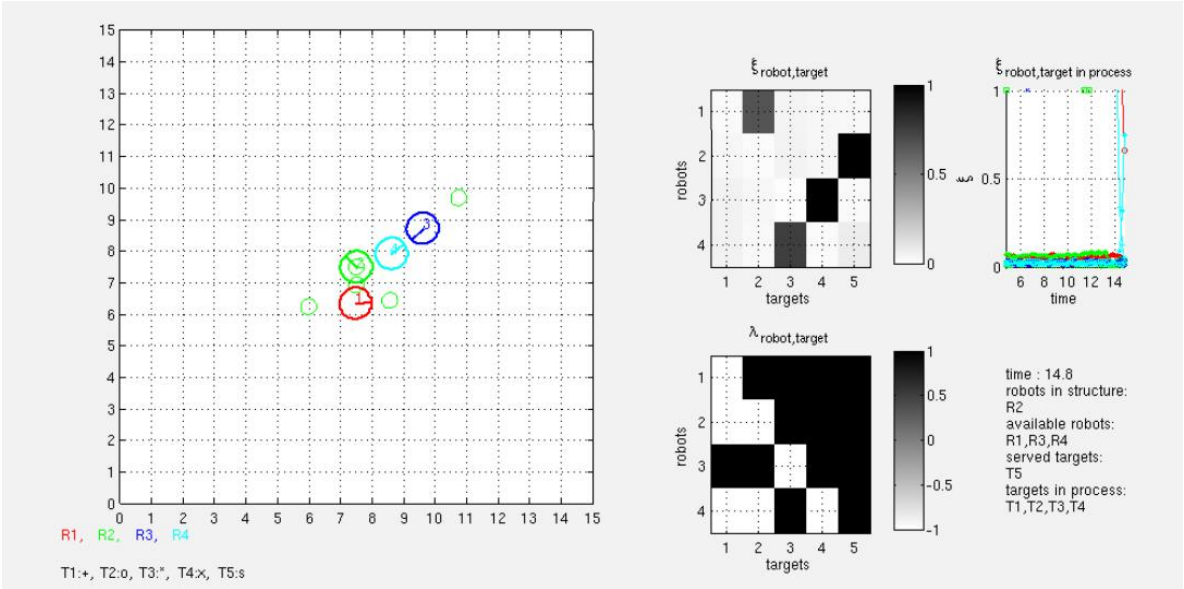

Abbildung 6.25.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (3.Methode)“: Unerwünschter Zielverlust auf Grund ungünstiger Position

fast erreicht hat, wird er als Nachfolger von Roboter 2 angesehen. Damit gilt die Docking-Station von Roboter 1 auch als unerlaubtes Ziel von Roboter 4. Dies führt zum Verlust des

Ziels von Roboter 4. In der Folge bildet sich eine Rotation gegen den Uhrzeigersinn der beiden Roboter 3 und 4 aus, die daher kommt, dass der Roboter, der Roboter 1 als Ziel hat, dies verliert, wenn er der Kette aus Roboter 2 und 1 zu nahe kommt.

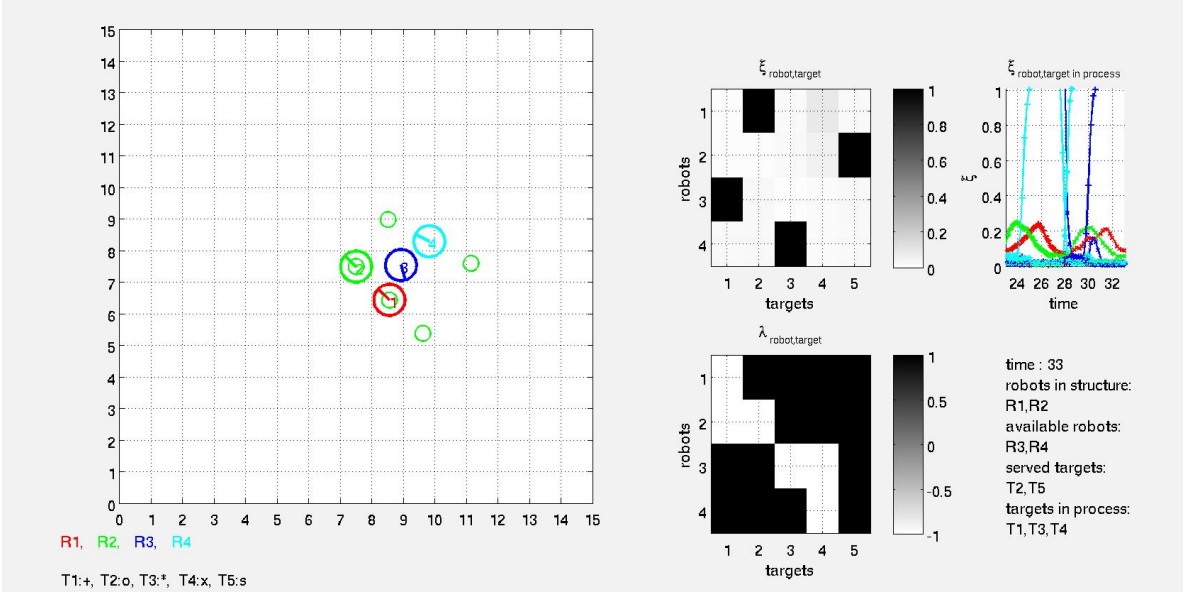

Abbildung 6.26.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (3.Methode)“: Zielzuordnung noch korrekt

Die Situation vor einem solchen Zielverlust ist in Abb. 6.26 zu sehen. Roboter 3 hat Roboter 1 als Ziel und wird von Roboter 4 gefolgt. Eine Simulationssekunde später (siehe Abb. 6.27) ist zu sehen, dass Roboter 3 wegen der Kollisionsvermeidung die Richtung ändern musste, und damit die Positionen der Docking-Stationen nahe an die Positionen der anderen Roboter rückten. Fälschlicherweise wird als Reihenfolge der Roboter die Kette $(4, 3, 2, 1)$ angesehen. Damit verliehrt Roboter 3 Roboter 1 als Ziel.

Diese fehlerhaften Annahmen verschwinden aber durch kleinste Bewegungen (insbesondere Drehungen) der Roboter wieder und so kann eine neue Zielzuordnung gefunden werden. In Abb. 6.28 ist die endgültige Zielzuordnung zu sehen. Da nun Roboter 4 Roboter 1 als Ziel hat, und nicht so dicht an der bestehenden Kette vorbeifahren muss, kommt es zu keinen weiteren Abstoßungen mehr und das Ziel kann erfolgreich bedient werden. Dieses Beispiel zeigt, dass es durch ungünstige Positionen der Roboter und der Docking-Stationen zu Problemen durch fehlerhafte Annahmen über die Zielzuordnungen kommen kann. Da aber die Roboter in Bewegung sind, werden diese fehlerhaften Annahmen zerstört und es kann sich eine neue Zielzuordnung aufbauen. Man kann davon ausgehen, dass es unwahrscheinlich ist, dass eine Gruppe von Robotern immer in exakt die selben Bewegungsabläufe und damit exakt die selben Zielzuordnungen fällt, ohne stabile Strukturen zu erreichen.

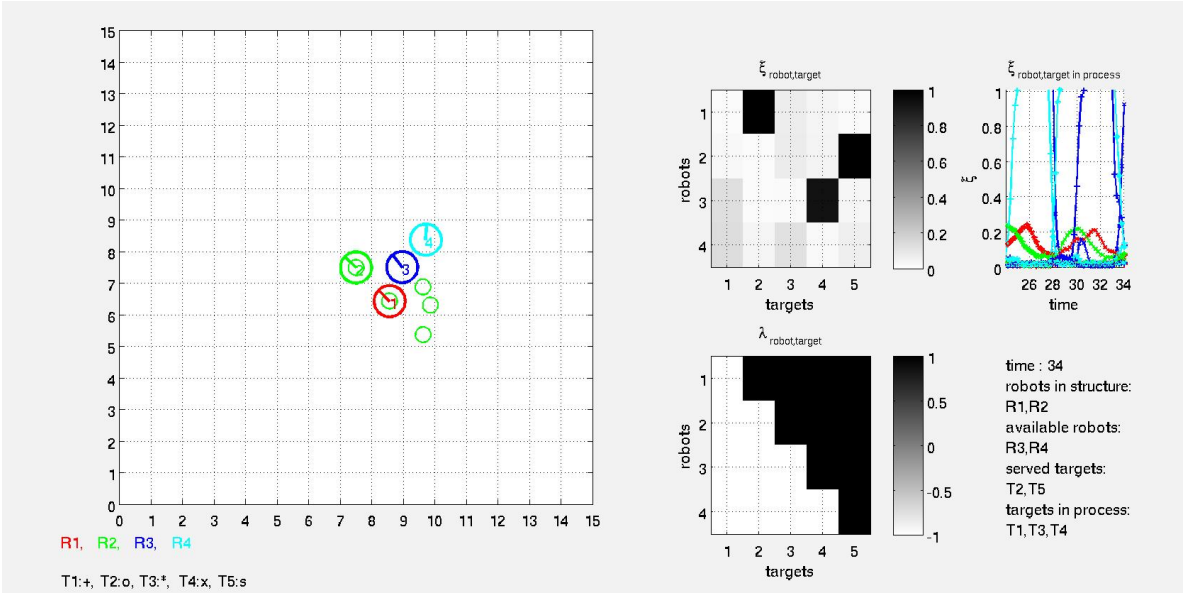

Abbildung 6.27.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (3.Methode)“: Verlust der Zielzuordnung durch ungünstige Position

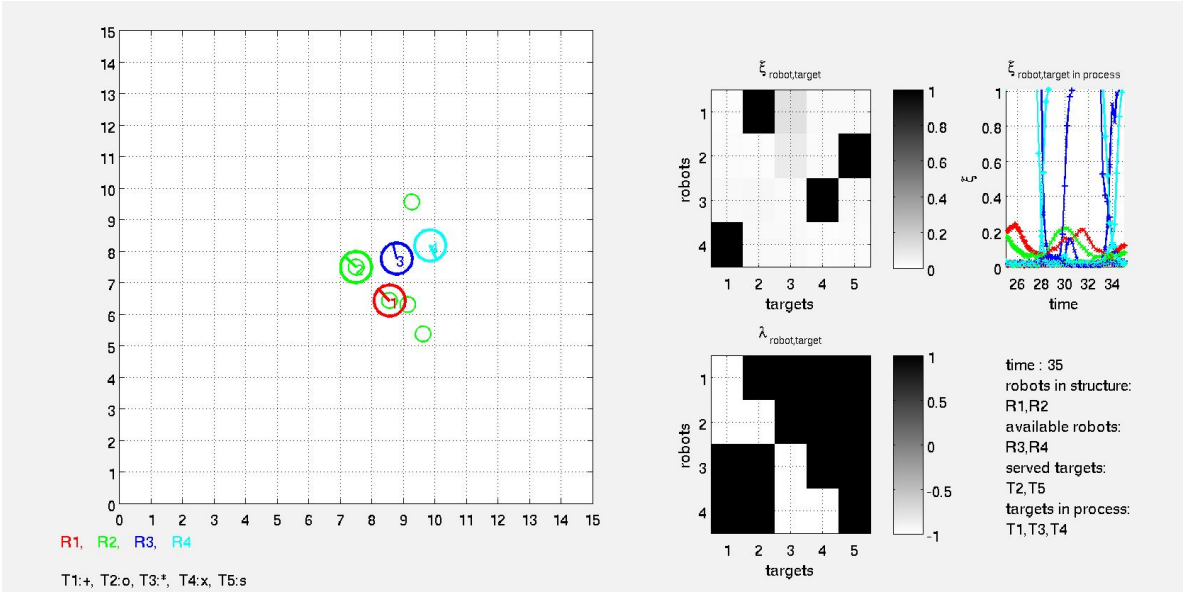

Abbildung 6.28.: Beispiel „Paralleles Bilden einer Linie bei paarweiser Startaufstellung (3.Methode)“: Neue Zielzuordnung nach Zielverlust

6.3.4. Bewertung der Lösung

Es konnte gezeigt werden, dass es möglich ist, die parallele Strukturbildung ausschließlich mit Hilfe der Positionen der Roboter und deren Docking-Stationen zu realisieren. Darüber hinaus konnte gezeigt werden, dass Roboterpaare recht spät von der Zyklenerkennung aufgebrochen werden. Damit kann erreicht werden, dass sich zunächst kleinere Strukturen bilden, die sich zu größeren zusammenschließen. Allerdings konnten auch einige Probleme in den Beispielen aufgezeigt werden: Wenn die Roboter sehr eng beisammen stehen, werden unter Umständen die nächsten Ziele durch die Zyklenvermeidung ausgeschlossen. Dies kommt daher, dass die nahe beisammen stehenden Roboter als Zyklus (oder wenigstens als Kette) angesehen werden und damit nicht mehr alle Vorgänger-Nachfolger-Relationen zur Verfügung stehen.

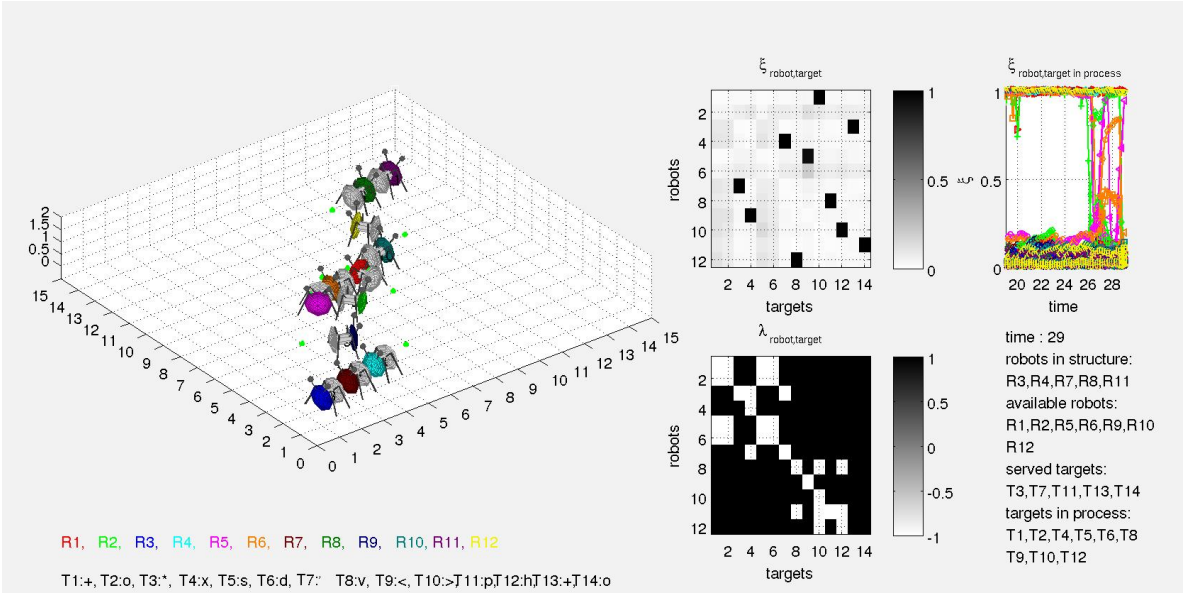

Abbildung 6.29.: Beispiel „Paralleles Bilden zweier Linien (3.Methode)“: Vielausschlüsse wegen zu nahe stehender Roboter

In Abb. 6.29 ist eine solche Situation zu sehen. Die Roboter 1, 2, 5 und 6 stehen so dicht beisammen, dass sie als Zyklus angesehen werden. So können sich keinerlei Vorgänger-Nachfolger-Relationen unter ihnen aufbauen. Im Fall von Roboter 1 und Roboter 5 ist dies kein Problem, da beide bereits ein anderes Ziel verfolgen. Im Fall der Roboter 2 und 6 ist dies nicht der Fall, und darüber hinaus gibt es in dieser Situation auch kein Ziel anderes Ziel, das bisher von keinem Roboter verfolgt wird. Daher werden diese beiden Roboter kein Ziel finden können, so lange dieser Zyklus angenommen wird. Durch das Fortbewegen der Roboter 1 und 5 wird die Situation in der Folge aber dahingehend entschärft, dass die Roboter nicht mehr so dicht stehen und keine zyklische Abhängigkeit mehr angenommen wird. Dadurch können sich für die Roboter 2 und 6 neue Zielverteilungen einstellen.
Es konnte auch gezeigt werden, dass eine gewisse Robustheit gegenüber Zielverlusten

besteht. Zielverluste durch fehlerfafte Annahmen sind unvermeidbar, da es keine Garantien geben kann, dass die aus den Abständen gewonnenen Informationen mit den tatsächlichen Zielzuordnungen übereinstimmen. Durch die Zielverluste entstehen Zeitverluste, und daher ist dieses Verfahren in der Regel langsamer im Strukturaufbau als das Verfahren, das die Matrix ξ zur Grundlage für die Zielzuordnungen der Zyklusvermeidung hat. Die Zielverluste auf Grund falscher Annahmen können aber auch als eine gewisse Chance interpretiert werden. Wenn sie auftreten, heißt das andererseits, dass ein Roboter näher an einer Docking-Station steht, die er eigentlich nicht verfolgt, als an seinem eigentlichen Ziel. Daher kann es durchaus sinnvoll sein, dass dieser Roboter ein neues Ziel sucht, da das Ziel, das er verfolgt, nicht optimal für ihn ist.

6.4. Verschieben einer parallel gebildeten Linie

Bei der sequentiellen Strukturbildung hatten wir gesehen, dass eine gebildete Linie von ihrem Standort aus zu einem anderen Ort verschoben werden kann. Auch bei parallel gebildeten Linien oder ganzen Strukturen kann dies wünschenswert sein.

6.4.1. Idee des Verfahrens

Das Verschieben einer parallel gebildeten Linie soll analog zur Verschiebung einer sequentiell gebildeten Linie erfolgen (vgl.5.4). Auch hier soll nach dem Fertigstellen der Linie, also wenn sich alle Roboter innerhalb der gebildeten Struktur befinden, das erste vorgegebene Ziel deaktiviert und das neue Ziel aktiviert werden. Als Methode zur Zyklenvermeidung wurde die erste, also mit Nutzung der Matrix ξ, gewählt.

6.4.2. Matlab Funktionen

Da die Ziele durch die Funktion *cheackPosition* aktiv oder inaktiv geschaltet werden, ist dies die einzige Funktion, die adaptiert werden muss. Alle anderen Funktionen können entsprechend der Beschreibung aus 6.1 unverändert übernommen werden.

cheackPosition

Um ganze Sequenzen von Zielen nacheinander freischalten zu können, wurde eine statische Variable „missionIndex“ eingeführt. Diese wird immer dann um 1 erhöht, wenn sich alle Roboter in Strukturen befinden. Entsprechend des Wertes dieser einen Variablen werden in diesem Fall dann die vorgegebenen Ziele aktiv oder inaktiv geschaltet.

6.4.3. Beispiele

Das erstellte Beispiel sieht vor, eine Linie von sechs zufällig verteilten Robotern an einem Startpunkt zu bilden, und diese dann an einen anderen Punkt zu verschieben.

Nach zwei Simulationssekunden ist die Zielverteilung bereits abgeschlossen (siehe Abb. 6.30). Danach gibt es keine Veränderungen dieser Verteilung, und die Linie bildet sich am ersten Ziel (Nr.7) aus. Das Verhalten beim Ausschalten dieses Ziels und Hinzuschalten des Ziels Nr.8 ist vollkommen analog zum bereits bekannten Verhalten aus 5.4. Es kommt hier auch wieder zu den bereits dort beschriebenen Schwingungen beim Verlassen des alten und Erreichen des neuen Ziels.

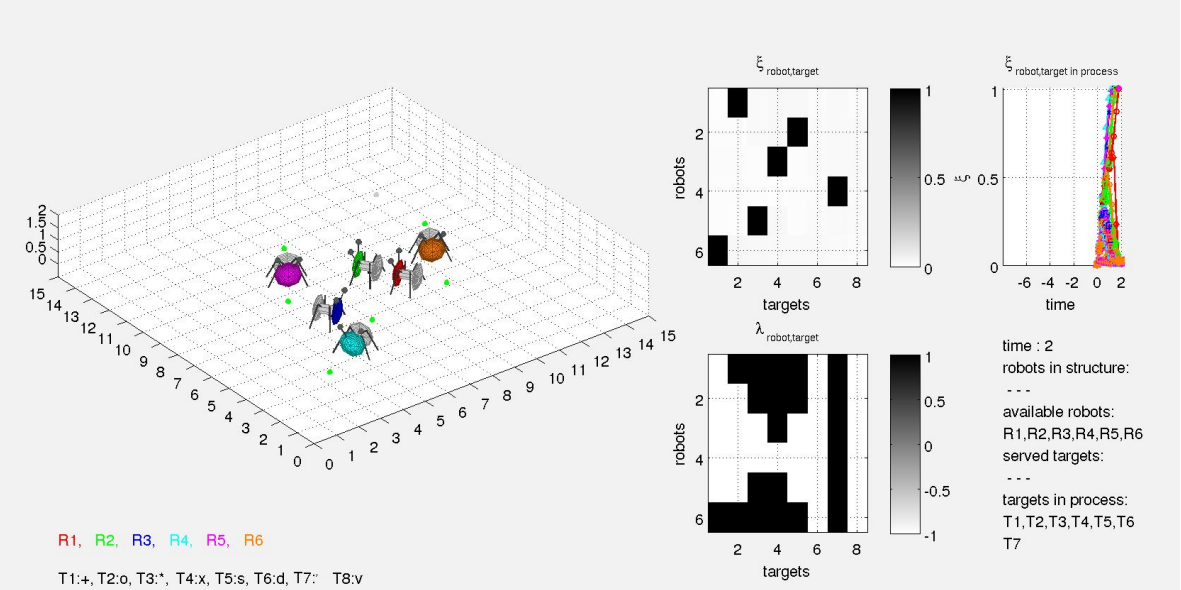

Abbildung 6.30.: Beispiel „Verschieben einer parallel gebildeten Linie“: Abgeschlossene Zielverteilung

6.4.4. Bewertung der Lösung

Es konnte gezeigt werden, dass das Verschieben einer parallel gebildeten Linie (oder allgemeiner einer parallel gebildeten Struktur), analog zum bekannten Verfahren der Verschiebung einer sequentiell gebildeten Linie, möglich ist. Es ist sogar in gewisser Hinsicht einfacher, da nicht sichergestellt werden muss, dass die Docking-Stationen der Roboter aktiv bleiben, wenn ein Roboter selbst kein Ziel besetzt hält.
Es ist über das Gezeigte hinaus auch vorstellbar, mehrere Linien zu bilden, und diese nach ihrer Fertigstellung an mehrere neue Ziele zu verschieben. Hierzu müssen nur mehrere aktive Ziele gleichzeitig deaktiviert und die neuen Ziele alle gleichzeitig aktiviert werden. Allerdings kann es hierbei vorkommen, dass der erste Roboter einer Linie nicht ein vorgegebenes Ziel als neues Ziel auffasst, sondern die Docking-Station eines Roboters am Ende einer Linie.
Der erste Roboter kann auf Grund der für ihn unerlaubten Docking-Station des letzten Roboters diesen nicht als Ziel auffassen. Damit sind Zyklenbildungen in der Art, wie sie in 5.8 erwünscht waren, ausgeschlossen. Bei der Verwendung der Abstände der Roboter zur Zyklenvermeidung (3. Methode) gilt dies ebenfalls. Allerdings kann es beim Bewegen der Roboter hier dazu kommen, dass die Nachfolgerbeziehungen nicht mehr richtig erkannt werden. Dies kann dazu führen, dass der erste Roboter doch ein anderes Ziel findet, sollte er nicht schnell genug das neue aktiv geschaltete Ziel gefunden haben. In diesem Fall könnten sich zwischenzeitlich doch Zyklen bilden, die aber, wenn die Abstände klein genug geworden sind, erkannt und aufgebrochen werden können.

7. Robustheit

In allen vorangegangenen Kapiteln wurden unterschiedliche Verfahren zur Strukturbildung vorgestellt. All diesen Verfahren ist eine Problematik gemein: Eine bereits gebildete, nicht konvexe Struktur ist für einen Roboter unüberwindbar. Dies liegt daran, dass die Roboter zur Kollisionsvermeidung nur lokales Wissen haben und bestrebt sind, den kürzesten Weg zu ihrem Ziel zu wählen.

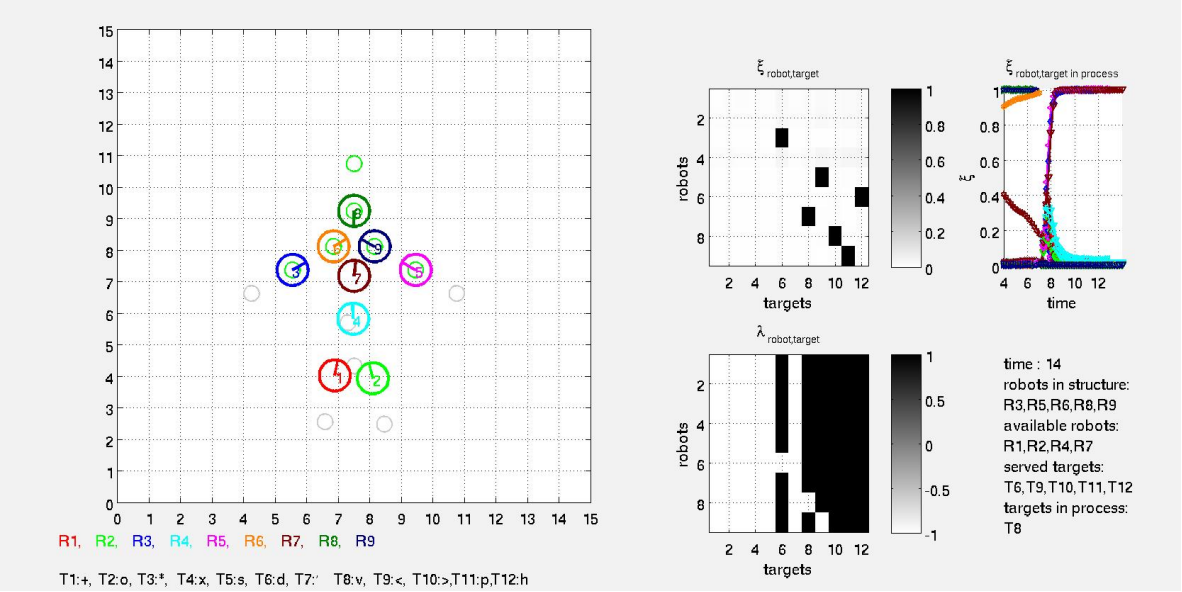

Abbildung 7.1.: Beispiel „Deadlock bei der sequentiellen Bildung eines gleichmäßigen Sterns“: Roboter 7 kann sein Ziel (Nr. 8) nicht erreichen

Eine solche Situation ist in Abb. 7.1 zu sehen. Bei diesem Beispiel handelt es sich um das sequentielle Bilden eines Sterns mit gleich langen Zacken (vgl.5.3). Roboter 7 kann sein Ziel, die Docking-Station von Roboter 8 nicht erreichen, weil die Roboter 6 und 9 den Weg versperren. Zwei nahe stehende kreisförmige Roboter bilden eine konkave Struktur, die nicht überwunden werden kann, wenn die Fahrtrichtung orthogonal zur Verbindungslinie der Mittelpunkte der beiden Roboter ist. Es hat sich zwar gezeigt, dass auf Grund von Abstoßungen Roboter durch eine geringfügig zu kleine Lücke zwischen zwei Robotern schlüpfen können, aber die Lücke in dem gezeigten Beispiel ist in diesem Falle zu klein. Das Nichterreichen eines der Ziele führt bei diesem Beispiel zum vollständigen Stillstand der Strukturbildung, weil keine weitere Docking-Station freigeschaltet wird.

Auf Grund der Problematik der bisherigen Unüberwindbarkeit von nicht konvexen Strukturen sollen in diesem Kapitel hierfür Möglichkeiten aufgezeigt werden. Den vorgestellten Methoden ist gemein, dass der Tausch von Zielen unter Robotern angestrebt wird. Hierbei soll der blockierte Roboter sein Ziel mit einem der blockierenden Roboter tauschen.

7.1. Robustheit der Strukturbildung mit vorgegebenen Raumpositionen

Im ersten Teil soll die Strukturbildung an vorgegebenen Zielen gegenüber Deadlocks robust gestaltet werden. Zur Erinnerung sei nochmals erwähnt, dass bei dieser Methode die Positionen der Ziele unabhängig von den Positionen der Roboter sind. Das bedeutet, dass es für die Positionen der Ziele anderer Roboter keine Auswirkungen hat, wenn sich ein Roboter bewegt. Diese Tatsache vereinfacht den Tausch von Zielen sehr, da beim Tausch von Zielen nicht auf eventuelle Nachfolger eines Roboters geachtet werden muss.

7.1.1. Grundidee der Realisierung von Robustheit

Grundsätzlich sollte es ein Ziel sein, in den Zielfindungsprozess möglichst wenig einzugreifen. Daher sollte nach Möglichkeit die Anzahl der von außen veränderten Werte möglichst gering sein. Eine Möglichkeit, die zum Zieltausch zwei Roboter führen kann, ist, die jeweiligen Ziele der beiden Roboter für diese beiden Roboter kurzfristig zu unerlaubten Zielen zu machen. Dies führt zum Absinken der entsprechenden ξ-Werte und damit zum Zielverlust der Roboter. Das Finden des anderen Ziels funktioniert selbstständig, da der blockierende Roboter sich näher am ehemaligen Ziel des blockierten Roboters befindet und dieses dann bedienen wird. Für den blockierten Roboter bleibt nun das frei gewordene Ziel.
Dieser Ansatz macht einen einfachen Eindruck, aber es werden einige Fragen aufgeworfen:

- Wann soll das Verfahren des Zieltauschs eingeleitet werden?
- Woher sind die Ziele der Roboter explizit bekannt, sodass diese für die betreffenden Roboter inaktiv geschaltet werden können?
- Wie lange soll ein solches Ziel inaktiv sein, und wie soll dies sichergestellt werden?

Wann soll das Verfahren des Zieltauschs eingeleitet werden?

Um das Verfahren des Zieltauschs einzuleiten, muss ein (möglicher) Deadlock erkannt werden. Die hier verfolgte Variante sieht vor, genau dann von einem sinnvollen Zieltausch auszugehen, wenn zwei Roboter einander im Weg stehen. In diesem Fall muss es sich zwar nicht zwangsläufig um einen Deadlock handeln, aber die Wege zu den getauschten Zielen sind in ihrer Summe kürzer als die Summe der Wege zu den ungetauschten Zielen. Daher ist diese Definition sinnvoll. Es bleibt die Frage, wie erkannt werden kann, dass sich zwei Roboter im Weg stehen. Dies kann mit Hilfe der Kollisionsvermeidung beantwortet werden:

Ist die Distanz zwischen zwei Robotern so klein, dass die Kollisionsvermeidung in die Bewegung der Roboter eingreifen muss, so kann man von einer Behinderung der Roboter untereinander ausgehen.

Woher sind die Ziele der Roboter explizit bekannt?

Die Zielzuordnungen sind in diesem Fall ausschließlich der Matrix ξ zu entnehmen. Die Positionen der Roboter ergeben hier keinerlei Aufschluss, welcher Roboter welches Ziel verfolgt. Das Maximum einer Zeile i der Matrix ξ kann also als das vorrangig verfolgte Ziel des Roboters i angesehen werden. An dieser Stelle kann ein Schwellwert eingeführt werden, damit der Zieltausch nur bei einer Überschreitung dieses Wertes erfolgen soll. Es ist allerdings nicht erforderlich, diesen Wert - und damit eine weitere Fallunterscheidung - einzuführen, da es keinen Qualitätsverlust der Strukturbildung nach sich zieht, wenn ein Roboter ein ohnehin sehr wenig verfolgtes Ziel verliert.

Wie lange soll ein solches Ziel inaktiv sein, und wie soll dies sichergestellt werden?

Es ist klar, dass es nicht ausreichend sein kann, ein Ziel eines Roboters für nur einen Simulationsschritt inaktiv zu schalten, damit der Roboter dieses verliert und sich ein neues Ziel suchen kann. Daher muss dieses Ziel für einen längeren Zeitraum inaktiv geschaltet werden, um so durch die gekoppelten Selektionsgleichungen ein starkes Abfallen des entsprechenden ξ-Wertes zu erreichen. Allerdings sollte jedes Ziel nach einer festgelegten Zeit wieder für jeden Roboter ein erlaubtes Ziel sein, damit die Einschränkungen nicht zu groß sind und so den Selbstorganisationsprozess zu stark manipulieren. Es soll also für jede unerlaubte Roboter-Ziel-Relation eine inkrementierte Variable geben, die angibt, für wie viele Simulationsschritte diese Relation noch unerlaubt ist. Da in der Phase, in der die Roboter ihre Ziele verlieren, nur geringfügige Bewegungen vollzogen werden, bleiben die Positionen der Roboter nahezu unverändert. Das bedeutet aber, dass die Abstände der Roboter ebenfalls untereinander unverändert sind und damit der Zieltausch-Prozess erneut immer wieder eingeleitet würde. Daher muss es auch für jedes Paar von Robotern eine dekrementierte Variable geben, die angibt, ob eine Kollision zum Auslösen des Zieltausch-Prozesses führen darf.

7.1.2. Matlab Funktionen

Da sich das Verfahren nur in Details von dem in 4.1 unterscheidet, sollen hier nur die Modifikationen der Funktionen erläutert werden.

navigation

Die für viele Simulationen genutzte Funktion *cheackPosition* muss zur Schaffung der Robustheit gegenüber Deadlocks erweitert werden. Die Erweiterung sieht insbesondere auch die beiden Rückgabewerte „lambdaRobustness“ und „collisionMatrix“ vor. Die Matrix „lambdaRobustness“ enthält die Information, für wie viele Iterationsschritte ein λ-Wert negativ gesetzt werden soll. Sie ist daher bei Simulationsbeginn mit Nullen besetzt. In der Matrix "collisionMatrix“ wird festgehalten, vor wie vielen Simulationsschritten die letzte Kollision zwischen zwei Robotern stattfand (Genau genommen werden nach einer Kollision die Werte entsprechend gesetzt und dann in jedem Simulationsschritt dekrementiert, bis sie wieder auf den Wert 0 gesunken sind. Daher muss diese Matrix auch die Nullmatrix bei Simulationsbeginn sein.).
Das Verfahren sieht nun vor, alle Paare von Robotern zu bilden und deren tatsächlichen Abstand zueinander zu überprüfen. Wird hier ein Einschreiten der Kollisionsvermeidung erkannt und die letzte Kollision der beiden Roboter ist lange genug her (d.h. der entsprechende Wert in der Matrix „collisionMatrix“ ist gleich 0), so werden die Werte für „collisionMatrix“ gesetzt und die jeweiligen Ziele der beiden Roboter bestimmt. Mit diesen Informationen können die Roboter-Ziel-Relationen durch Setzen der entsprechenden Werte in der Matrix „lambdaRobustness“ für eine bestimmte Anzahl an Simulationsschritten als unerlaubt geschaltet werden.
Eine weitere Modifikation der Funktion *navigation* besteht im Inkrementieren der Werte von Matrix „lambdaRobustness“ und im Decrementieren der Matrix „collisionMatrix“, sofern deren Werte ungleich 0 sind. Dies kann durch Matrixoperationen geschehen.

setLambda

Die Funktion *setLambda* wurde in der Form erweitert, dass überall dort, wo die Matrix „lambdaRobustness“ einen negativen Wert beinhaltet, in der Matrix „lambda“ der Wert -1 gesetzt wird.

scene_104

Da zwischen den Funktionen *setLambda* und *navigation* die Matrix „lambdaRobustness“ kommuniziert werden muss, ist es erforderlich die Funktion *scene_104* dahingehend zu modifizieren, dass sie diese Parameterübergabe realisiert. Darüber hinaus werden hier die Initialisierungen der zusätzlichen Variablen durchgeführt. Damit bei der Erstellung der Beispiele die neuen Algorithmen erst greifen, wenn bestehende Ziele bedient sind, wurde eine weitere Variable eingeführt, die es ermöglicht, diese Algorithmen hinzuzuschalten.

7.1.3. Beispiele

Es sollen hier zwei Beispiele vorgeführt werden. Bei beiden Beispielen wurden die Ziele von Robotern, die kollidierten, für jeweils 10 Simulationsschritte zu unerlaubten Zielen geschaltet. Die Anzahl der Simulationsschritte zwischen zwei Kollisionen sollte 40 betragen.

Beispiel „Robustheit bei vorgegebenen Zielen und senkrechtem Aufprallen auf eine Linie“

Bei diesem Beispiel soll ein Roboter senkrecht auf eine Linie von Robotern stoßen, da er ein Ziel auf der gegenüberliegenden Seite verfolgt.

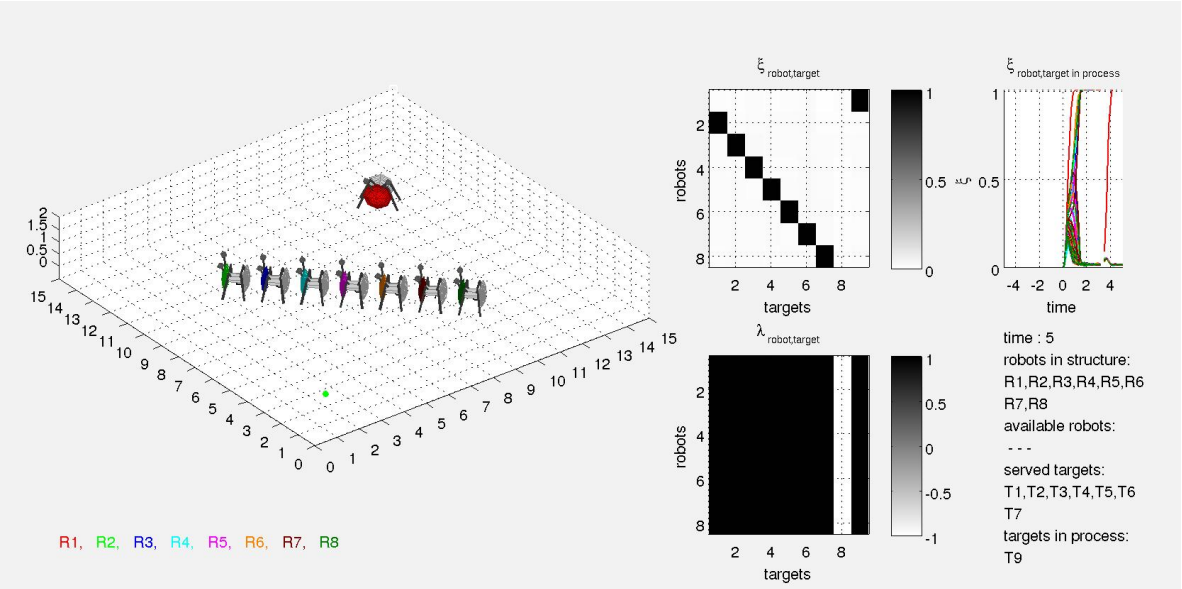

Abbildung 7.2.: Beispiel „Robustheit bei vorgegebenen Zielen und senkrechtem Aufprallen auf eine Linie“: Linie versperrt den Weg zum Ziel

In Abb. 7.2 ist diese Situation zu sehen. Roboter 1 bewegt sich auf sein Ziel (Nr. 8) auf der anderen Seite der Linie zu. Die Roboter innerhalb der Linie stehen auf ihren Zielen und so dicht, dass Roboter 1 nicht zwischen ihnen hindurch kann. Einige Simulationssekunden später trifft Roboter 1 auf Roboter 5 (siehe Abb. 7.3). Es ist zu sehen, dass die Werte $\lambda_{1,8}$ und $\lambda 5,4$ gleich -1 sind, und dadurch die Werte $\xi_{1,8}$ und $\xi_{5,4}$ deutlich gesunken sind. Da die Werte für λ nur für 10 Simulationsschritte negativ sind, fallen die ξ-Werte bei der ersten Kollision nicht so weit ab, dass die Ziele bereits als verloren gelten können. Dies führt zum erneuten Ansteigen der alten Werte, nachdem die λ-Werte wieder zu 1 geworden sind. In der Folge kommt es zu weiteren Kollisionen, die ein ähnliches Verhalten nach sich ziehen. Auf diese Wiese sinken die ξ-Werte der jeweiligen Zielzuordnungen und die neuen Zielzuordnungen können sich ausbilden. Allerdings kann es durchaus vorkommen, dass die Ziele mehrfach hin und her getauscht werden. Dies ist in Abb. 7.4 zwischen der

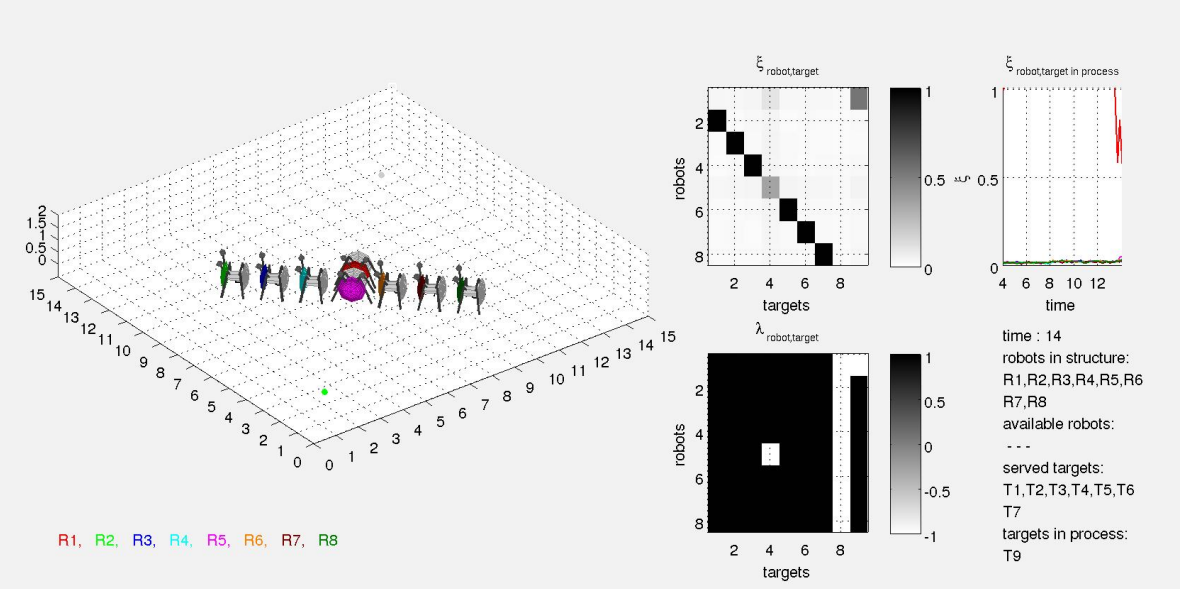

Abbildung 7.3.: Beispiel „Robustheit bei vorgegebenen Zielen und senkrechtem Aufprallen auf eine Linie“: Erster Zusammenstoß

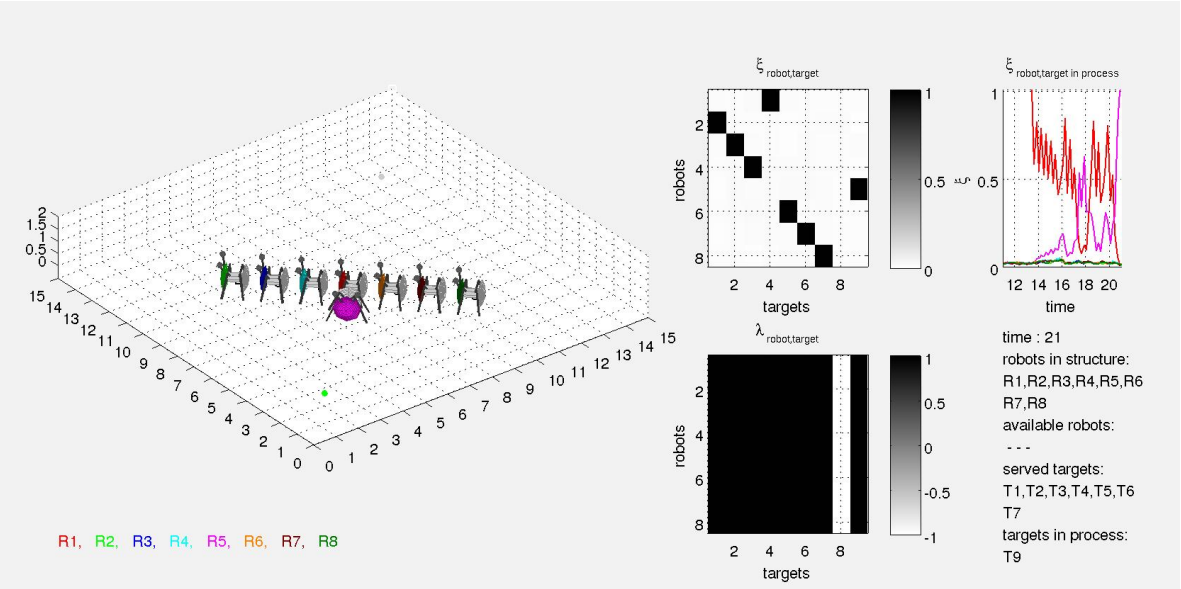

Abbildung 7.4.: Beispiel „Robustheit bei vorgegebenen Zielen und senkrechtem Aufprallen auf eine Linie“: Beendeter Zieltausch

Simulationssekunde 17 und 18 im Diagramm zu sehen. Hierzu kann es kommen, wenn durch eine erneute Kollision nicht der alte, sondern der neue Wert vermindert wird. Dies

tritt dann auf, wenn der neue Wert bereits den alten Wert überschritten hat und somit das neue Ziel als das vorrangige Ziel angesehen wird. Ebenfalls ist in Abb. 7.4 der vollzogene Zieltausch zu sehen. Roboter 5 bedient nun das Ziel 8 und Roboter 1 hat die Position in der Linie besetzt.

Beispiel „Robustheit bei vorgegebenen Zielen und frontales Aufprallen auf eine Linie“

Im zweiten Beispiel ist die Situation gegeben, dass ein Roboter frontal auf eine Linie aufprallt. Die Idee dahinter ist, dass alle Roboter in der Linie um eine Position nach hinten rücken sollen und somit der letzte Roboter in der Linie das neue Ziel übernehmen soll. Inspiriert ist dieses Beispiel vom Einspringen eines Roboters für einen in einer Struktur ausgefallenen Roboter. Hier kann es auch oftmals sinnvoller sein, dass die umgebenden Roboter um eine Position aufrücken, sodass die Lücke, in die ein neuer Roboter einspringen soll, am Anfang oder am Ende der Struktur entsteht.

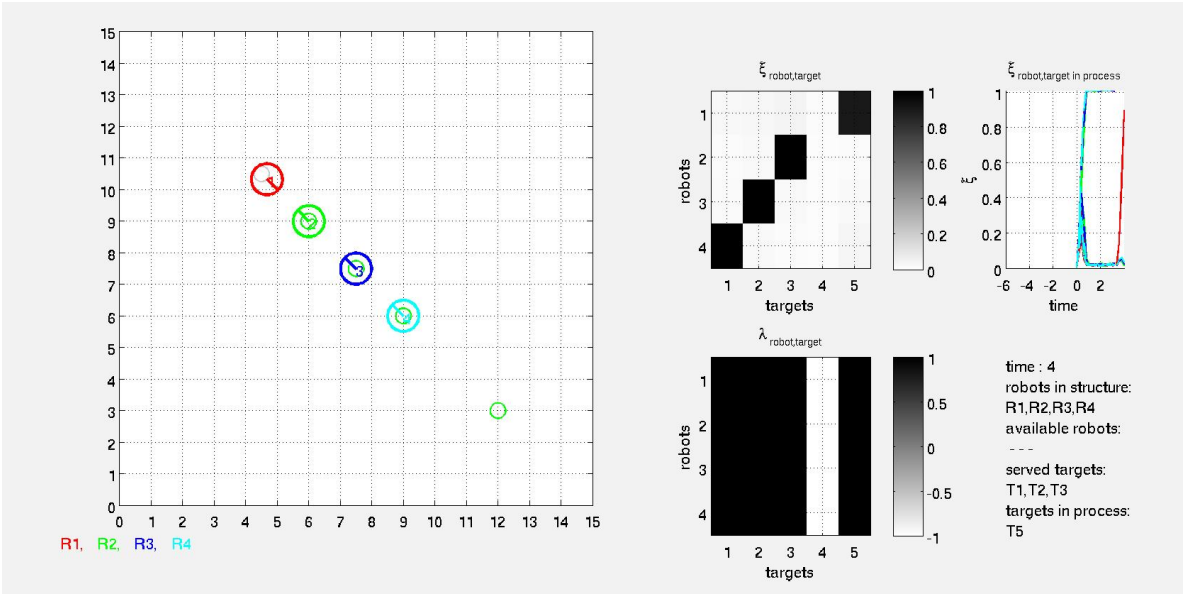

Abbildung 7.5.: Beispiel „Robustheit bei vorgegebenen Zielen und frontales Aufprallen auf eine Linie“: Linie versperrt den Weg zum Ziel

Roboter 1 hat in Abb. 7.5 das Ziel Nr. 5 mit den Koordinaten $(12, 3, 0)$. Auf dem direkten Weg dorthin liegt die Linie aus den Robotern 2 (auf Ziel Nr. 3), 3 (auf Ziel Nr. 2) und 4 (auf Ziel Nr. 1). In Abb. 7.6 ist die Situation nach mehreren Kollisionen zu sehen. Die Werte $\xi_{1,5}$ und $\xi_{2,5}$ sind mittlerweile etwa gleich groß, sodass Roboter 1 und Roboter 2 etwa gleich stark das Ziel 5 verfolgen. Es ist auch zu sehen, dass Kollisionen mit Roboter 3 stattgefunden haben, und dieser das Ziel 2 nicht mehr vorrangig verfolgt. Im weiteren Verlauf verliert Roboter 2 das Ziel 3 vollständig zu Gunsten von Roboter 1. Dies ist in Abb. 7.7 zu sehen. Hier ist ebenfalls zu beobachten, dass sich ein ähnliches Verhalten zwischen den Robotern 2,

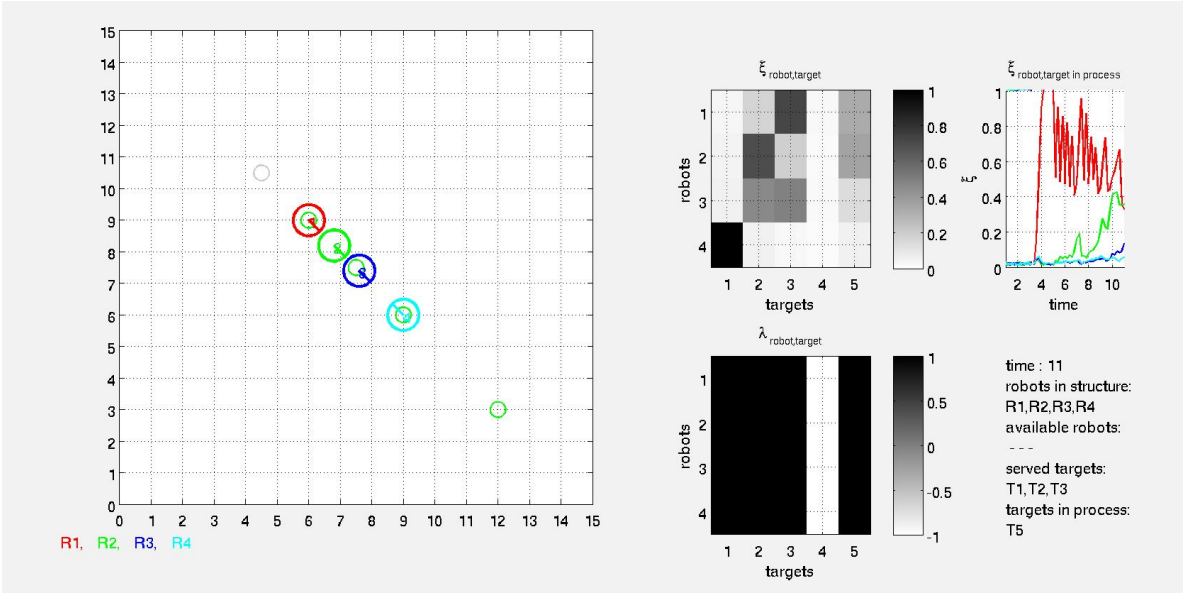

Abbildung 7.6.: Beispiel „Robustheit bei vorgegebenen Zielen und frontales Aufprallen auf eine Linie“: Nach mehreren Kollisionen.

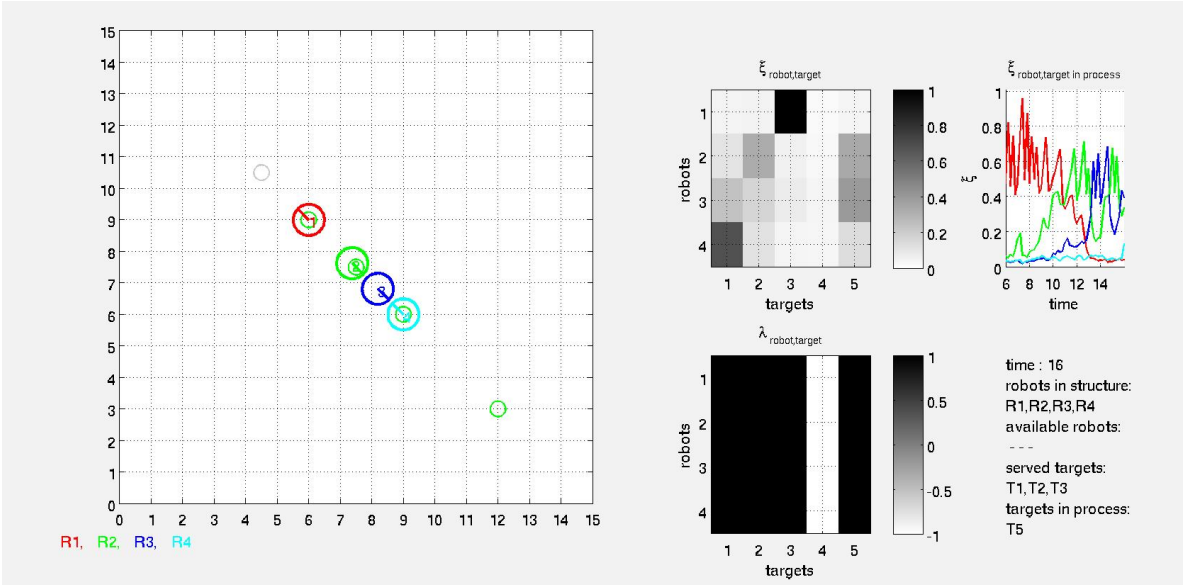

Abbildung 7.7.: Beispiel „Robustheit bei vorgegebenen Zielen und frontales Aufprallen auf eine Linie“: Roboter 1 hat bereits den Zieltausch vollzogen.

3 und 4 abspielt und somit Roboter 4 das Ziel Nr. 1 verlieren wird. Diesem steht dann Ziel

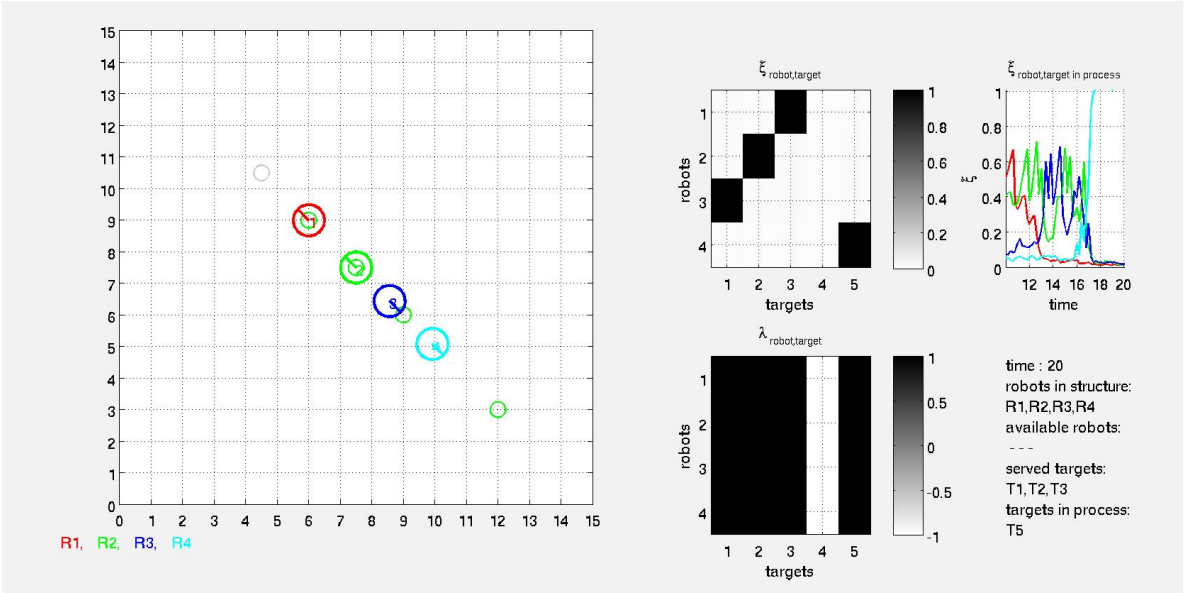

Abbildung 7.8.: Beispiel „Robustheit bei vorgegebenen Zielen und frontales Aufprallen auf eine Linie“: Der Zieltausch ist beendet.

Nr. 5 als nächstes Ziel zur Verfügung. Dies zeigt Abb. 7.8. Roboter 4 bewegt sich auf Ziel 5 und Roboter 3 die letzte verbliebene Strecke auf Ziel 1 zu.

7.1.4. Bewertung der Lösung

In diesem Abschnitt konnte gezeigt werden, dass die vorgestellte Methode geeignet ist, Deadlocks zu überwinden. Es sollen allerdings noch einige Bemerkungen zu dieser Methode gemacht werden:
Bei der Erkennung der Deadlocks könnte, anstatt den tatsächlichen Abstand der Roboter zu betrachten, die Variable „frSum“ betrachtet werden. Diese gibt den Vektor an, mit dem jeder Geschwindigkeitsvektor der Roboter zur Kollisionsvermeidung korrigiert werden muss. Wenn der Betrag dieses Vektors größer als Null ist, so muss eine Kollision vorliegen. Vorteilhaft an dieser Methode wäre, dass anhand der reinen Position nicht ständig von weiteren Kollisionen ausgegangen werden würde. Dieses Vorgehen hat allerdings drei entscheidende Nachteile:
Erstens muss dies für jeden Roboter einzeln überprüft werden.
Zweitens kann aus der Tatsache, dass dies bei zwei Robotern der Fall ist, nicht gefolgert werden, dass diese beiden miteinander kollidiert sind. Es muss also ohnehin eine weitere Methode geben, um herauszufinden, welche Roboter miteinander kollidierten.
Der dritte Nachteil ist, dass die Geschwindigkeit eines im Weg stehenden Roboters nicht immer korrigiert werden muss, und damit fällt es nicht auf, dass dieser an einer Kollision beteiligt ist. Dies ist dann der Fall, wenn ein Abbremsen oder Ausweichen eines anderen

Roboters vollständig genügt.
Beim frontalen Aufprallen eines Roboters auf eine Linie gibt es zwei wichtige Punkte, die erwähnt werden sollten: Stehen die Roboter innerhalb der Linie zu dicht beisammen, können sich gar keine sinnvollen Zielverteilungen mehr ausbilden, weil zu viele Kollisionen stattfinden und alle Ziele verloren gehen. Darüber hinaus sollten die Linien nicht zu lang sein, weil es bei jedem Roboter zu einer leichten seitlichen Verschiebung durch Abstoßungen kommen kann. Diese führen unter Umständen zum seitlichen Ausbrechen eines Roboters, der dann das hinter der Linie liegende Ziel direkt ansteuern kann.

7.2. Robustheit der sequentiellen Strukturbildung

In Analogie zur Robustheit der Strukturbildung bei vorgegebenen Zielen kann man die Robustheit der sequentiellen Strukturbildung gestalten. Auch hier sollen durch Setzen der entsprechenden λ-Werte zwei Roboter ihre Ziele tauschen. Anders als bei der Strukturbildung mit vorgegebenen Zielen sind hier die Positionen nicht aller Ziele von den Positionen der Roboter unabhängig. Das heißt, eine Docking-Station befindet sich immer hinter einem Roboter. Daher muss beim Tausch von Zielen auch darauf geachtet werden, dass die Nachfolger eines Roboters an den anderen Roboter übergeben werden. Dies soll natürlich nicht durch Setzen der Roboter-Ziel-Zuordnungen, sondern selbstorganisiert stattfinden.

7.2.1. Grundidee der Realisierung von Robustheit

In Anlehnung an die Realisierung der Robustheit bei vorgegebenen Zielen sollen auch hier zwei kollidierende Roboter einen Tausch ihrer Ziele durchführen. Da aber davon ausgegangen werden muss, dass sich an den Docking-Stationen dieser beiden Roboter weitere Roboter angelagert haben, müssen diese ebenfalls getauscht werden. Im Folgenden soll davon ausgegangen werden, dass es sich bei einem um einen einzelnen Roboter handelt, an dessen Docking-Station keine weiteren Roboter angedockt sind. Dies stellt eine erhebliche Erleichterung des Problems dar, da nur eine Nachfolgerkette übergeben werden muss und nicht gleichzeitig zwei. Daher muss nur die Docking-Station des einzelnen Roboters aktiv und die des anderen inaktiv geschaltet werden.
An dieser Stelle stellt sich die Frage, wie lange die Docking-Stationen so geschaltet sein sollen. Zur Erinnerung sei nochmal erwähnt, dass die Werte, mit denen die Zielzuordnungen aufgebrochen werden sollen, nach einer festgelegten Zeit wieder verschwinden sollen und damit die Ziele wieder für jeden Roboter erlaubt sind. Ein ähnliches Vorgehen ist im Fall der Freischaltung der Docking-Stationen aus folgendem Grund nicht sinnvoll: Da die Struktur, die ein Hindernis bildet, sequentiell gebildet wurde, ist es für die hintere Teilstruktur essentiell, dass eine aktive Docking-Station in diesem Umfeld besteht. Wäre diese nicht vorhanden, würde die hintere Teilstruktur kollabieren. Daher ist es sinnvoll, die eine Docking-Station aktiv und die andere inaktiv zu schalten. Um ein wiederholtes Hin- und Herschalten der Docking-Stationen auf Grund von wiederholten Kollisionen zu vermeiden, soll hierzu das Auslösen der Umschaltung analog zur Auslösung des Zieltauschs realisiert werden.

7.2.2. Matlab Funktionen

Die Funktionen *navigation* und *scene_212* wurden in Anlehnung an die bereits aus Abschnitt 7.1 bekannten Änderungen durchgeführt.

navigation

Die Funktion *navigation* ist bis auf ein kleines Detail identisch zur Funktion *navigation* aus 7.1: Mit Hilfe der Menge der aktiven Ziele kann herausgefunden werden, welcher der beiden an der Kollision beteiligten Roboter eine aktiv geschaltete Docking-Station besitzt. Diese wird aus der Menge der aktiven Ziele entfernt und die des anderen Roboters hinzugefügt. Dieses Vorgehen darf ebenfalls nur dann durchgeführt werden, wenn dies entsprechend des jeweiligen Eintrags in der Matrix „collisionMatrix“ erlaubt ist. Es hat sich herausgestellt, dass hier wesentlich längere Wartezeiten erforderlich sind als bei der Methode an vorgegebenen Zielen. Es ist klar, dass durch diese Modifikation auch die aktiven Ziele zu den Übergabeparametern zählen müssen.

scene_212

Die aus 7.1 bekannten Änderungen wurden in der Funktion *scene_212* umgesetzt und so die Funktion *scene_206* (sequentielle Bildung beliebig vieler Linien mit vorgegebenen Startpositionen der Roboter) modifiziert.

7.2.3. Beispiele

Beispiel: „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“

Das Beispiel sieht eine sequentiell gebildete Linie vor, auf die ein einzelner Roboter senkrecht aufprallt, weil das Ziel, das er verfolgt, auf der anderen Seite der Linie liegt. Zwischen zwei Kollisionen, die zum Auslösen des Zieltauschs und zum Tausch der aktiven Docking-Station führen dürfen, liegen 400 Simulationsschritte, was 400 Simulationssekunden entspricht. Die Anzahl der Simulationsschritte, während denen Ziele als unerlaubte Ziele aufgefasst werden, beträgt 20 Schritte (dies entspricht $\frac{1}{5}$ Simulationssekunden).

Die Situation vor der ersten Kollision ist in Abb. 7.9 zu sehen. Roboter 1 ist auf dem Weg zu Ziel Nr. 10. Senkrecht zu diesem Weg befindet sich die Kette der Roboter 2 bis 7. In Abb. 7.10 ist die Situation kurz nach der Kollision von Roboter 1 mit Roboter 4 zu sehen. Durch die größere Anzahl von Simulationsschritten, in denen die Werte $\lambda_{1,10}$ und $\lambda_{4,3}$ gleich -1 gesetzt sind, kommt es zu einem wesentlich schnelleren Abfallen der Zielzuordnungen. Daher können sich die neuen Zielzuordnungen wesentlich schneller neu aufbauen. Dies hat die positive Folge, dass es zu keinen weiteren Kollisionen (die einen erneuten Zieltausch auslösen) der Roboter 1 und 4 kommt, weil Roboter 4 bereits auf dem Weg zu seinem neuen Ziel Nr. 10 ist. Ebenfalls ist zu sehen, dass die Docking-Station 1 aktiv und die Docking-Station inaktiv geschaltet ist. Das führt dazu, dass Roboter 5 sein Ziel verloren hat und auf der Suche nach einem neuen Ziel, die Docking-Station 1 gefunden hat. Die Nachfolger von Roboter 5 verfolgen ihren Vorgänger auf dem Weg zu diesem Ziel. Roboter 1 hat kurze Zeit später die Docking-Station 3 erreicht und so kann

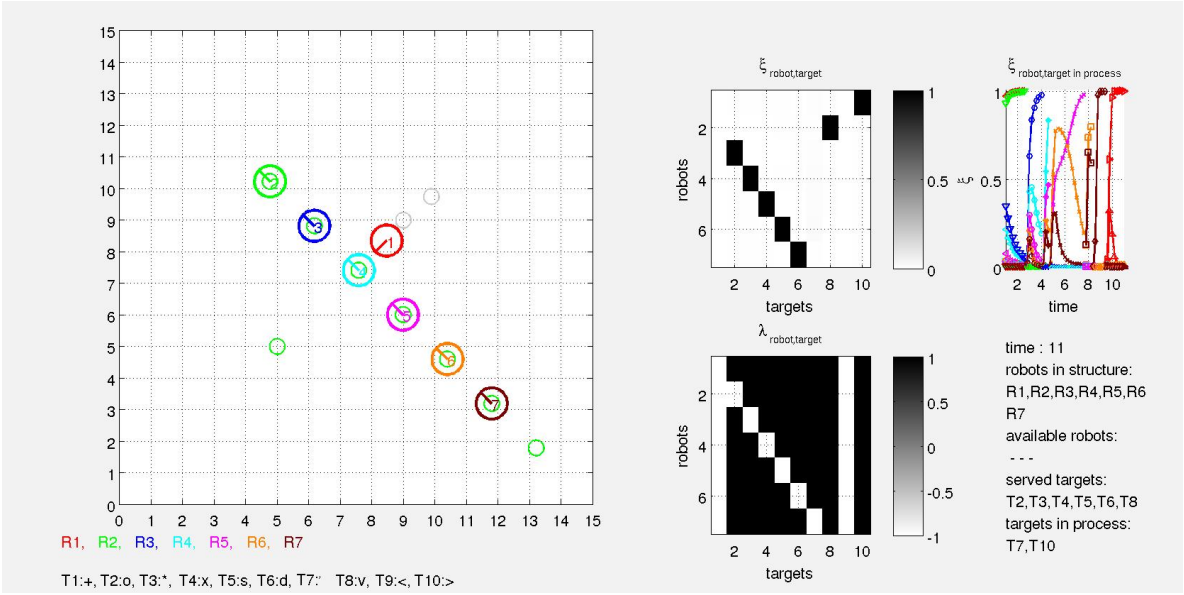

Abbildung 7.9.: Beispiel „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“: Linie versperrt den Weg zum Ziel

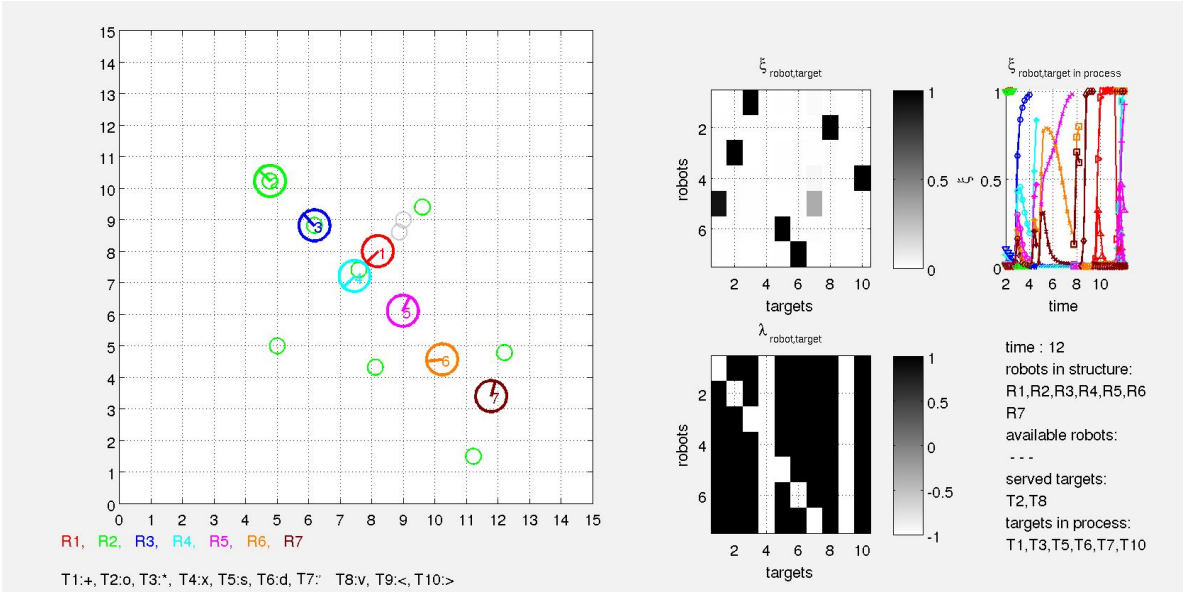

Abbildung 7.10.: Beispiel „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“: Erste Kollision hat den Zieltausch eingeleitet

die Stabilisierung der hinteren Teillinie beginnen (siehe Abb. 7.11. Der hintere Teil der Linie schwingt in ähnlicher Form wie bei der Verschiebung einer sequentiell gebildeten

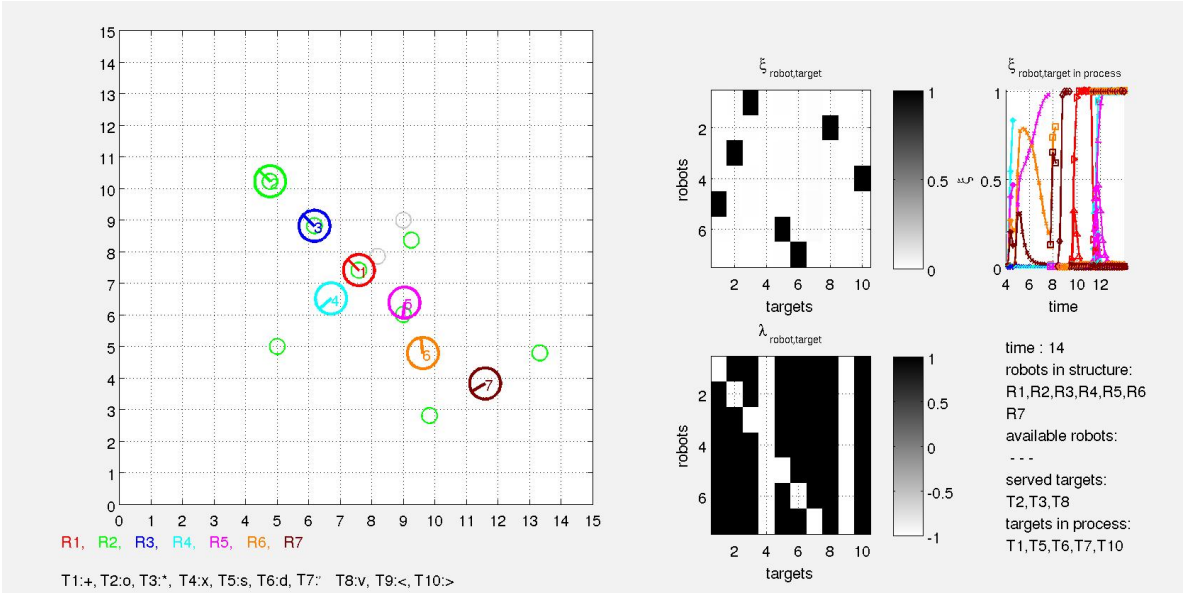

Abbildung 7.11.: Beispiel „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“: Hintere Teillinie muss stabilisiert werden

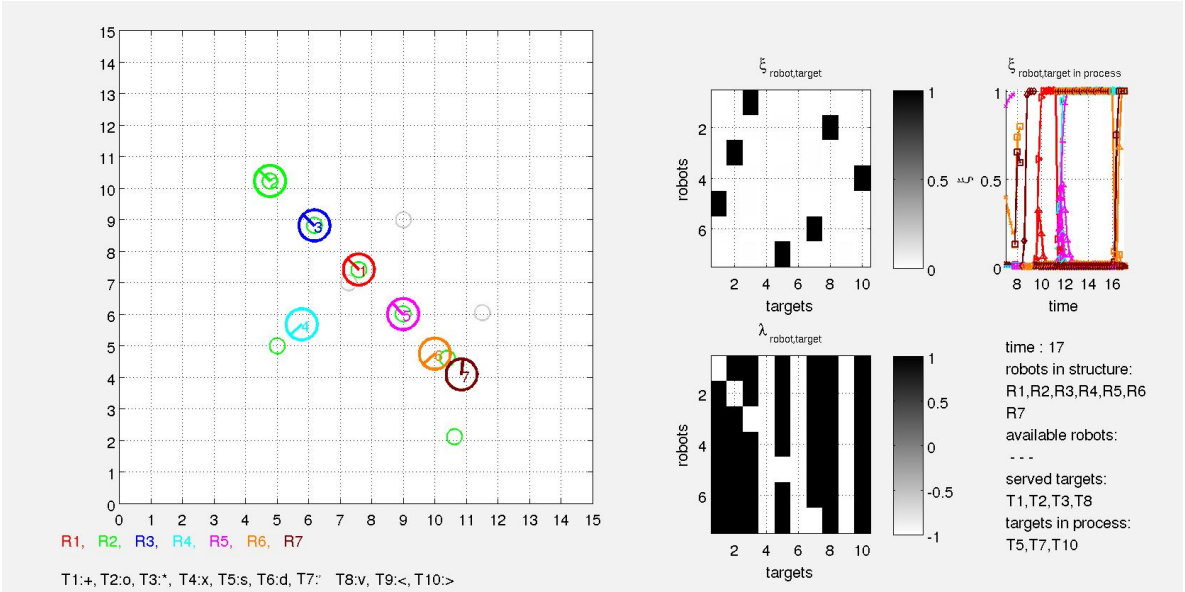

Abbildung 7.12.: Beispiel „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“: Zieltausch in der hinteren schwingenden Teillinie

Linie. Allerdings kommt hier eine erschwerende Tatsache hinzu. Es kann in der Folge der Schwingung zu Kollisionen der Roboter untereinander kommen. Diese führen dann

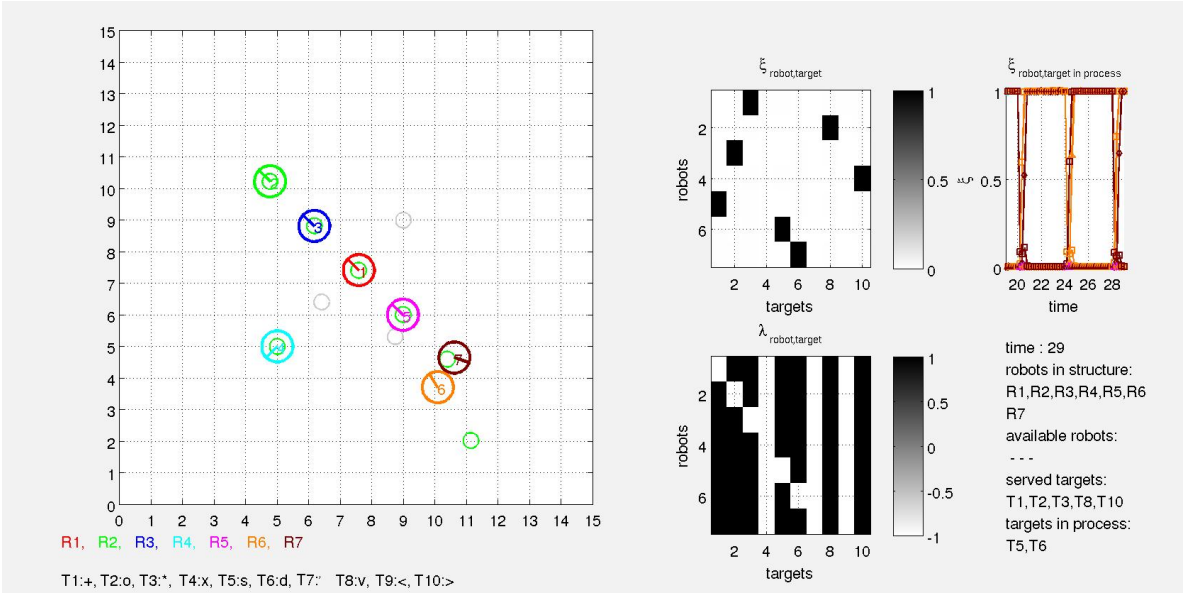

Abbildung 7.13.: Beispiel „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“: Vermehrter Tausch von Zielen in der hinteren Teillinie

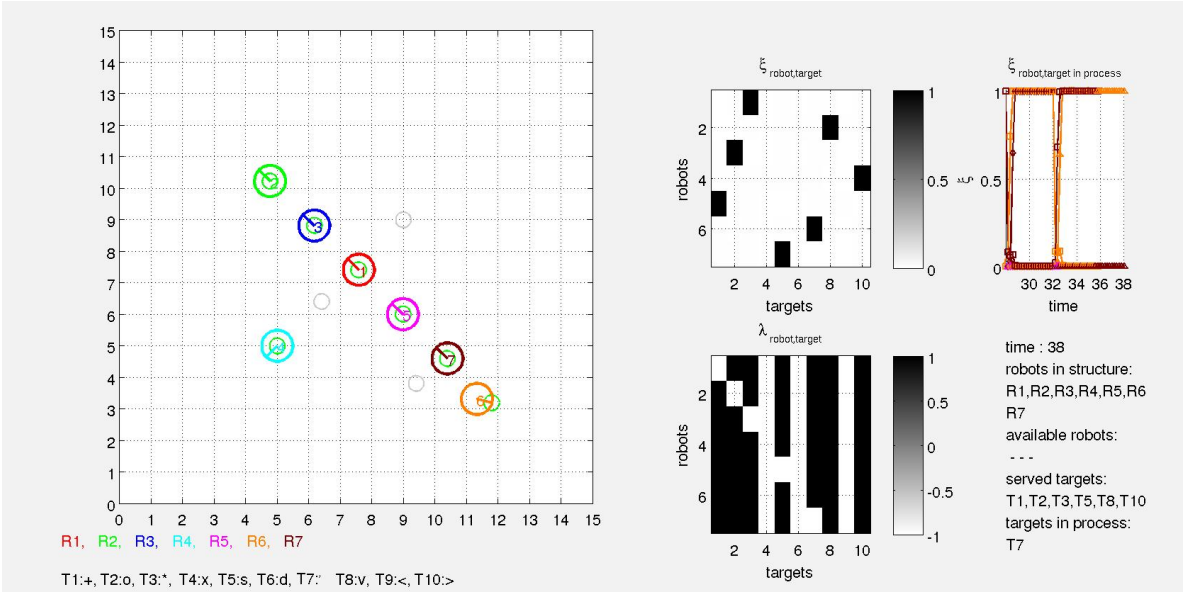

Abbildung 7.14.: Beispiel „Robustheit sequentieller Linienbildung und senkrechtem Aufprallen auf eine Linie“: Kurz vor Fertigstellung der Linie

zum weiteren Tauschen von Zielen. In Abb. 7.12 ist die Kollision der Roboter 6 und 7 zu

sehen. Deutlich ist hier, dass die vormals aktive Docking-Station 6 nun inaktiv, und die vormals inaktive Docking-Station 7 nun aktiv ist. Beim Versuch, die Ziele zu erreichen, kommt es immer wieder zu solchen Kollisionen. Hieran wird deutlich, warum die Anzahl der Simulationsschritte, in denen Kollisionen zweier Roboter nicht zu einem Zieltausch führen dürfen, deutlich gesteigert werden musste. In der Abb. 7.13 ist im Zeitdiagramm genau zu sehen, dass der Zieltausch alle vier Simulationssekunden, also nach genau 400 Simulationsschritten, stattfindet. Im Gesamten findet aber eine Annäherung der Roboter an die Docking-Station 5 statt, sodass dieser Prozess abbricht (sofern die Anzahl der Schritte ohne erneute Auslösung des Zieltauschs groß genug gewählt wurde). Roboter 7 besetzt in diesem Beispiel die Docking-Station 5 und Roboter 6 kann die von Roboter 7 besetzen (siehe Abb. 7.14).

7.2.4. Bewertung der Lösung

Es konnte gezeigt werden, dass es auch bei der sequentiellen Bildung von Linien möglich ist, einzelne Roboter darin auszutauschen und so eine Robustheit gegenüber Deadlocks zu schaffen. Ein frontales Aufprallen auf eine bestehende Linie kann allerdings zu Problemen führen: Die hintere Teillinie wird auf die Docking-Station des neuen Roboters zusteuern. Diese Tatsache wird den Abstand zwischen ihr und den kollidierten Robotern verkürzen. Dieser verkürzte Abstand könnte weitere Probleme verursachen. Aus Zeitgründen konnten hierzu leider keine Experimente gemacht werden. Diese wären sicherlich eine interessante Aufgabe für die Zukunft.
Eine weitere interessante Frage ist, wie ein Tausch von zwei Nachfolgerketten realisiert werden könnte. Das heißt, wenn nicht ein einzelner Roboter auf eine Linie prallt, sondern eine ganze Kette. Wünschenswert wäre hier, dass in diesem Fall die den zusammengestoßenen Robotern nachfolgenden Teillinien ausgetauscht würden. Hier kann die entstehende Schwingung in den hinteren Teillinien zu Problemen führen.
Eine dritte wichtige Frage für die Zukunft lautet: Ist es sinnvoll, die Zeiten, in denen kein Zieltausch nach Kollisionen durchgeführt werden darf, und die Zeiten, in denen Ziele als verbotene Ziele definiert werden, dynamisch zu gestalten? Hierzu müssten Untersuchungen unternommen werden, wie der genaue Einfluss dieser Werte auf die Ausbildung von stabilen Strukturen ist, wie diese klassifiziert werden können, und wie diese anhand der diskreten Simulationssituation ausgewählt werden können.

7.3. Robustheit der parallelen Strukturbildung

Es liegt nahe, die Robustheit in parallel gebildeten Strukturen analog zu der in sequentiell gebildeten zu realisieren. Hierbei besteht allerdings die grundsätzliche Eigenschaft, dass zunächst alle Docking-Stationen der Roboter aktiv sind. Um nun die hintere Teilstruktur, die an den Roboter angedockt ist, der aus der Struktur herausgelöst werden soll, abzuhängen, muss von dieser Grundregel abgewichen werden. Es muss also genauso verfahren werden, wie bei der sequentiellen Variante, nur dass ausschließlich die eine Docking-Station inaktiv geschaltet werden muss. Die Docking-Station des einzelnen Roboters muss nicht aktiv geschaltet werden, da sie dies ohnehin bereits ist. Alles Andere funktioniert in vollkommen analoger Art und Weise.
Als einzige interessante Frage bleibt, wann die deaktivierte Docking-Station wieder aktiviert werden darf und sollte. Hier könnte man sich an den Zeitpunkten der unerlaubten Ziele orientieren und die deaktivierte Docking-Station zeitgleich mit diesen wieder freigeben.
Aus Zeitgründen konnten hierzu keine Beispiele erstellt werden, aber es sollte klar sein, dass dieses Verfahren funktionsfähig ist.

8. Zusammenfassung und Ausblick

Im Kapitel 2 wurde das Konzept der gekoppelten Selektionsgleichungen vorgestellt, um die Grundlage der ausgewählten Selektionsprozesse zu schaffen. Hier sind die Verbindungen zum wissenschaftlichen Kontext zu sehen.
Die im Zuge dieser Arbeit entstandene Software wurde in Kapitel 3 vorgestellt und die wichtigsten Funktionen erläutert. Das Zusammenspiel und Ineinandergreifen der einzelnen Funktionen wurde verdeutlicht und abschließend eine Bewertung der erstellten Software vorgenommen.
Um erste Strukturbildungen einzuführen, wurde in Kapitel 4 ein Ansatz verfolgt, bei dem Ziele im Raum vorgegeben sind. An diesen einfachen Beispielen konnte die Arbeitsweise des Selektionsprozesses illustriert werden. Ausgehend von dieser ersten Lösung wurde ein Verfahren entwickelt, bei dem die Ziele der Roboter nicht von Anfang an bekannt sind sondern während der Strukturbildung hinzukommen. Bei diesem selbstorganisierten, sequentiellen Verfahren sind somit keine strikten Vorgaben mehr erforderlich und die Struktur entsteht aus sich selbst heraus durch die Anwendung sehr einfacher Regeln. Es konnten mit diesem Verfahren Linien, Brücken und Polygone erzeugt und verschoben werden.
Als Weiterentwicklung der sequentiellen konnte die parallele Methode zur Strukturbildung in Kapitel 6 präsentiert werden. Hier werden Ziele nicht nacheinander, sondern gleichzeitig von den Robotern angesteuert. Das dabei entstehende zentrale Problem der Zyklenbildung konnte auf drei verschiedene Arten gelöst werden:
Die erste Methode nutzt hierzu die von der Zielverteilung her existierenden Informationen. Um eine Eigenschaft dieser Methode zu unterbinden, wurde ein zweiter Lösungsansatz entwickelt, bei dem die Zyklenvermeidung nur im Nahbereich der Roboter aktiv sein soll. Es konnte gezeigt werden, dass hiermit das gewünschte Ergebnis erreicht werden kann. Die dritte vorgestellte Methode verzichtet zur Zyklenvermeidung auf Informationen der Zielverteilung. Sie stützt sich ausschließlich auf die Informationen, die aus den Positionen der Roboter und der Ziele zu gewinnen sind. Die Robustheit dieses Verfahrens gegenüber unvermeidlichen Fehlinterpretationen der Positionen konnte an Beispielen dargestellt werden.
Kapitel 7 beschäftigt sich mit der Robustheit der drei Strukturbildungsverfahren gegenüber Deadlocks. Es konnten Strategien entwickelt werden, die den Austausch von einzelnen Robotern aus Linien ermöglichen, wenn diese den Weg versperren. Hierbei ist klar geworden, dass diese Aufgabe sehr schwer zu lösen ist und relativ viele Eingriffe in das allgemeine Verfahren erfordert.
Anhand von Beispielen wurden die Vor- und Nachteile sowie die allgemeinen Eigenschaften aller Lösungswege diskutiert. Eine kurze Bewertung jeder vorgestellten Lösung soll ein späteres Weiterentwickeln der Methoden erleichtern.

Ausblick

Die bisher erzeugten Strukturen sind allesamt relativ einfacher Art. Daher wäre es für die Zukunft sicherlich interessant, komplexere Strukturen als Linie, Brücken oder Polygone zu entwickeln. Es wurden zwar Sterne erzeugt, diese waren jedoch aus mehreren einfachen Linien aufgebaut. Wie bereits angedeutet, könnten Sterne ebenfalls dadurch entstehen, dass jeder Roboter mehrere Docking-Stationen besitzt. Welche dieser Stationen aktiv geschaltet werden darf, würde davon abhängen, welches Ziel der jeweilige Roboter bedient.
Dieser Ansatz führt sehr schnell auf das Bilden hierarchischer Strukturen: Jeder Roboter könnte verschiedenste Arten von Docking-Stationen besitzen. In Abhängigkeit, an welche Art von Docking-Station ein Roboter andockt, werden seine eigenen Docking-Stationen frei geschaltet.
Allgemein kann man sagen, dass ein weites Forschungsgebiet im Bereich der hierarchischen Strukturen liegt. Sie können oftmals zu Robustheit führen, weil beim Ausfall einzelner Teilstrukturen diese durch andere, im Voraus gebildete Ersatzstrukturen ersetzt werden können. Sicherlich von ebenfalls großem Interesse sind alle Möglichkeiten der Erzielung von Robustheit, insbesondere jene, bei denen die Eingriffe in die Strukturbildungsprozesse sehr klein sind und damit eine große Allgemeingültigkeit erzielt werden kann.
Eine große Herausforderung für die Zukunft ist die Übertragung der Ergebnisse auf reale Robotersysteme, um hier ihre Funktionstüchtigkeit unter Beweis zu stellen. Selbstverständlich werden diese zunächst einfache Strukturen bilden, aber als Fernziel sind Roboterstrukturen zu sehen, die dem Menschen Werkzeug und Unterstützung sein können.

A. Graphiken

In diesem Teil des Anhangs sind Graphiken von zwei Simulationen zu finden. Die erste Simulation ist der sequentielle Bau einer Brücke (siehe Abschnitt 5.5). In diesem Beispiel soll eine Brücke zwischen zwei Punkten gebildet werden. Hierzu stehen insgesamt 7 Roboter zur Verfügung. Von diesen werden am Ende nur 6 benötigt worden sein. Die Abbildungen A.1 bis A.6 zeigen diesen Brückenbau.
Die zweite Folge von Abbildungen (A.7 bis A.18) zeigen den parallelen Bau einer Linie. Hierbei wurden zur Zyklenvermeidung die Werte der Matrix ξ verwendet (1.Methode siehe Abschnitt 6.1). In diesem Beispiel wurden insgesamt 9 zufällig verteilte Roboter zu einer Linie an einem Ziel bewegt. Die Abblidungen illustrieren die Zielverteilung und die anschließende Fahrt der Roboter sehr genau.

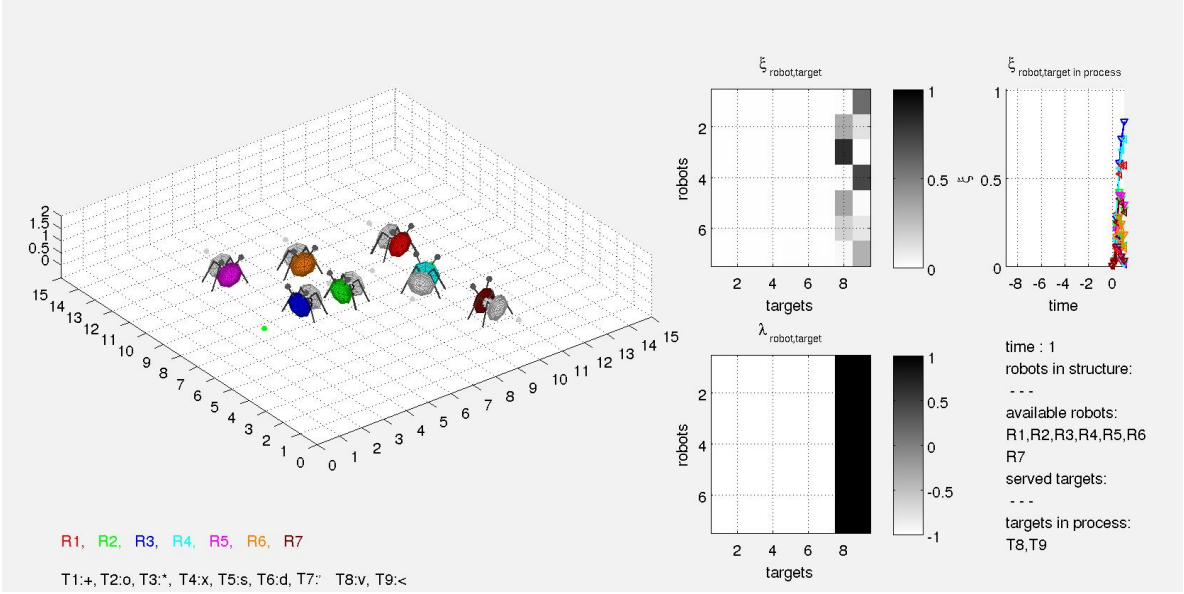

Abbildung A.1.: „Sequentielle Bildung einer Brücke“: Situation am Anfang der Simulation.

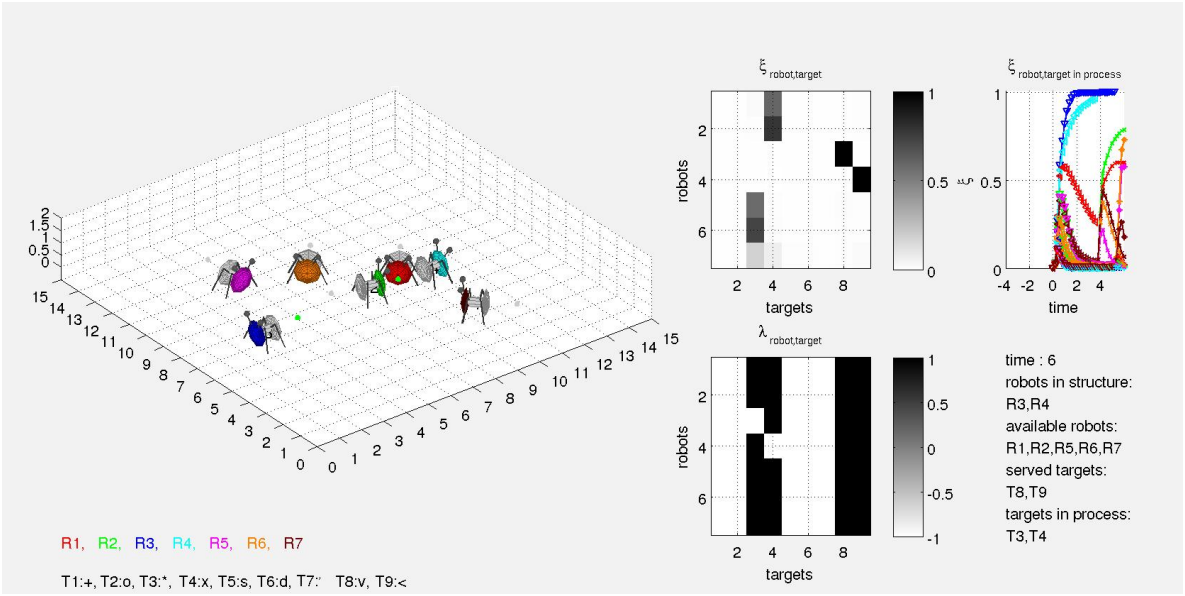

Abbildung A.2.: „Sequentielle Bildung einer Brücke“: Die Brückenköpfe sind eingefügt.

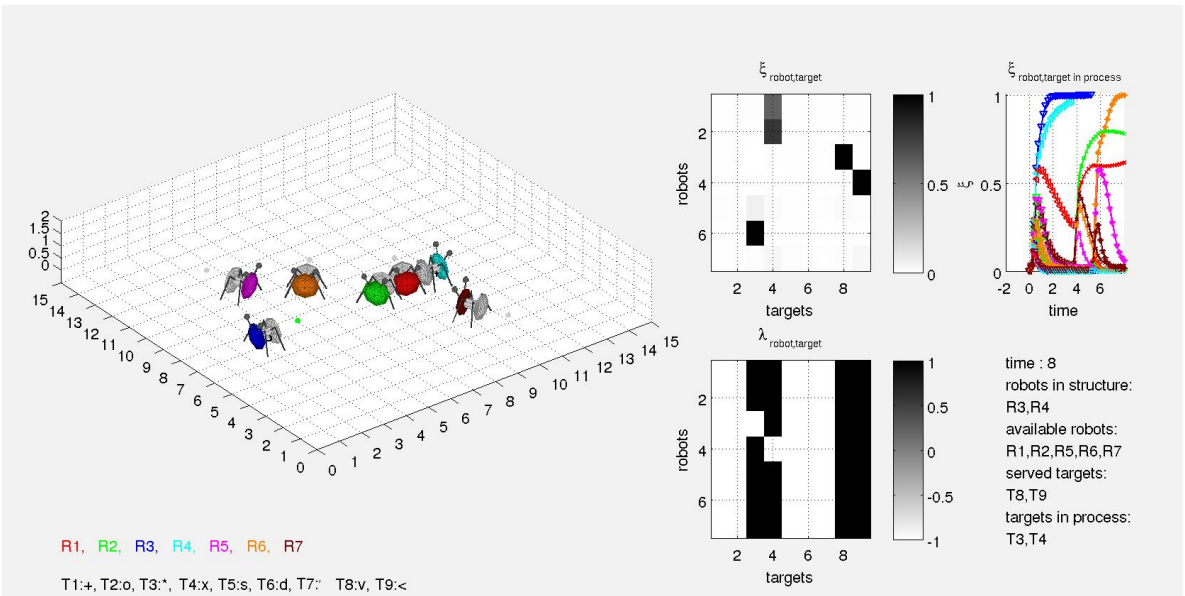

Abbildung A.3.: „Sequentielle Bildung einer Brücke“: Wettstreit der Roboter 1 und 2 um Ziel.

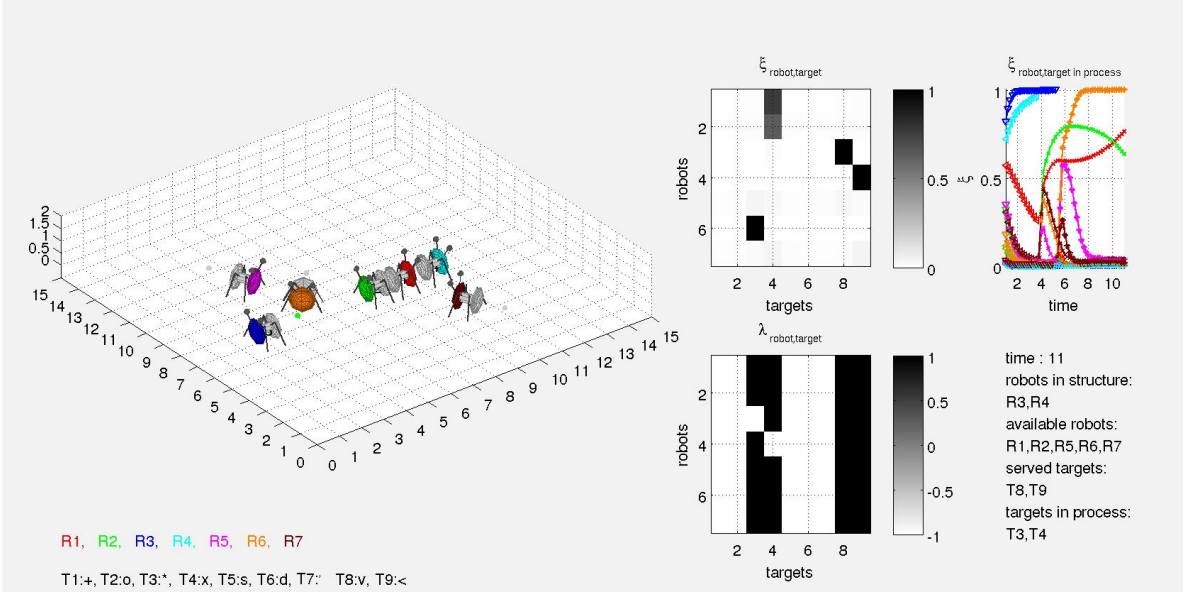

Abbildung A.4.: „Sequentielle Bildung einer Brücke“: Einfügen der Roboter 2 und 6.

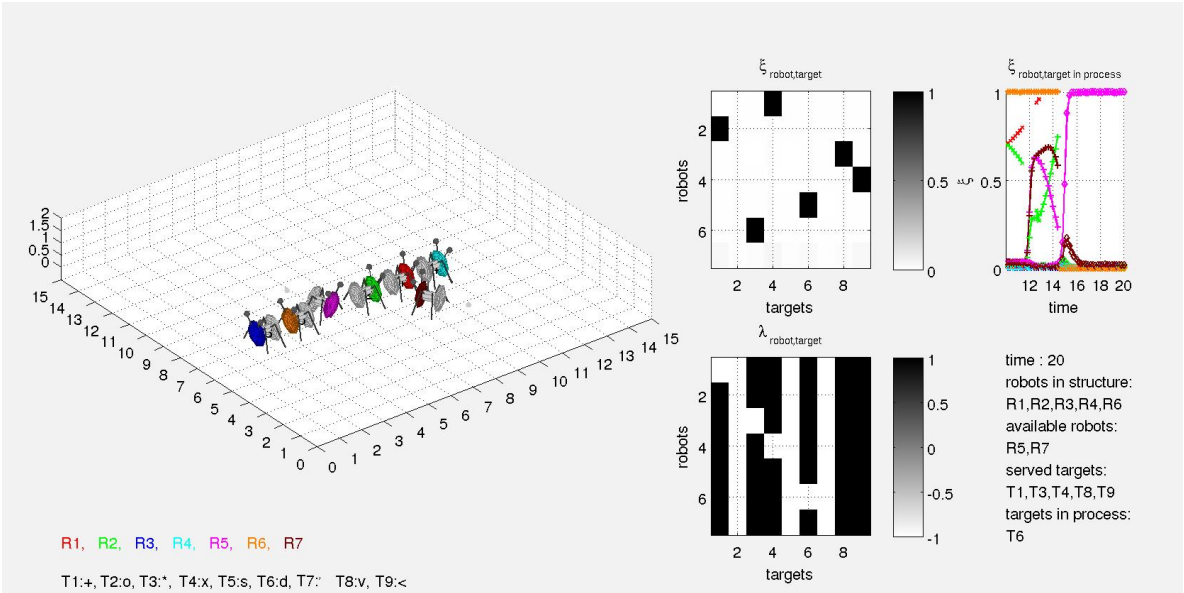

Abbildung A.5.: „Sequentielle Bildung einer Brücke“: Kurz vor Schließung der Brücke.

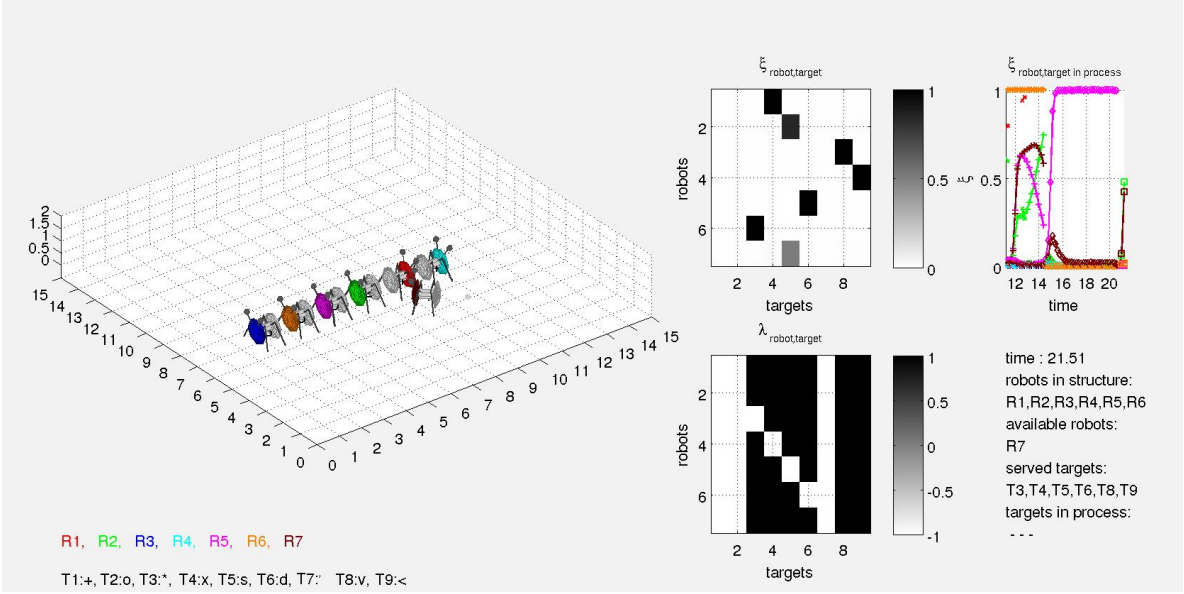

Abbildung A.6.: „Sequentielle Bildung einer Brücke“: Situation am Ende der Simulation.

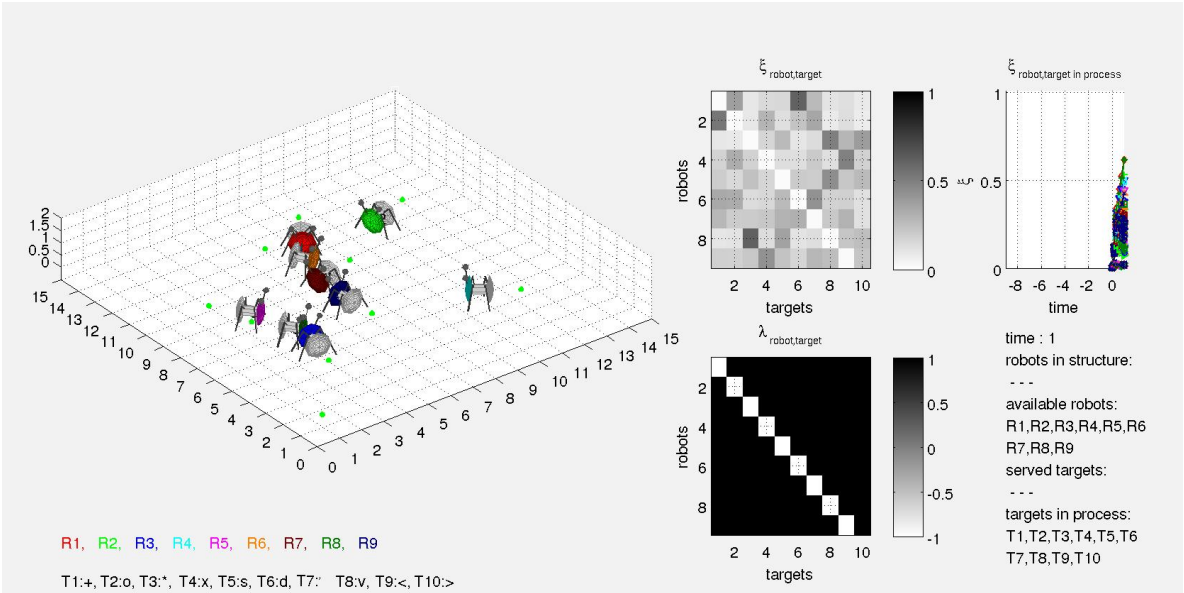

Abbildung A.7.: „Parallele Bildung einer Linie (1.Methode)“: Am Anfang der Simulation.

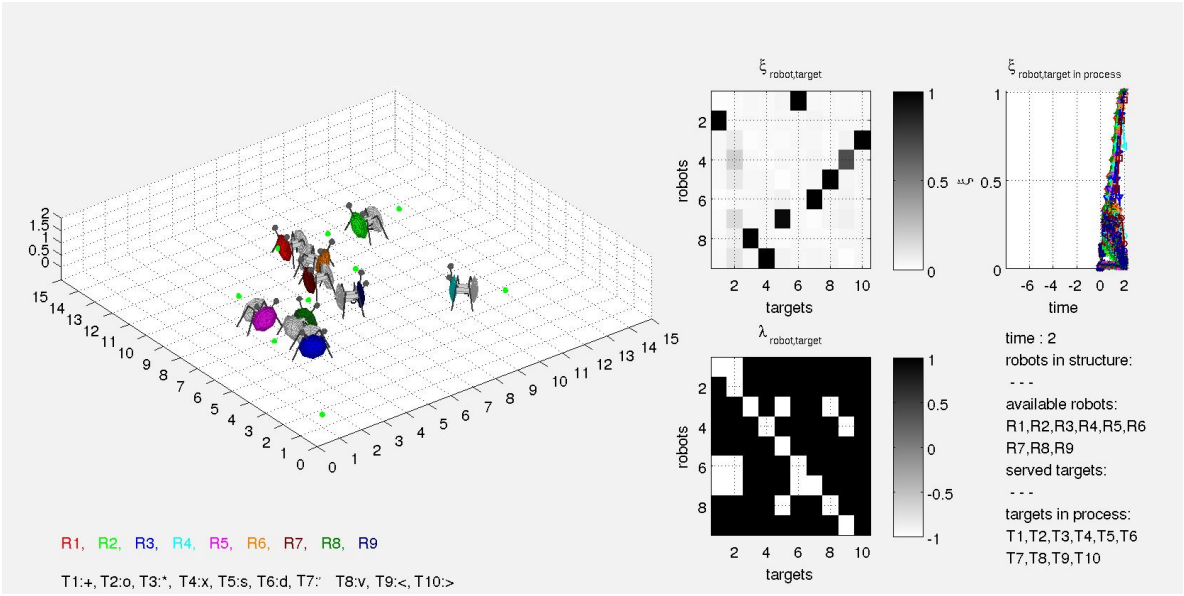

Abbildung A.8.: „Parallele Bildung einer Linie (1.Methode)“: Erste Ketten bestehen.

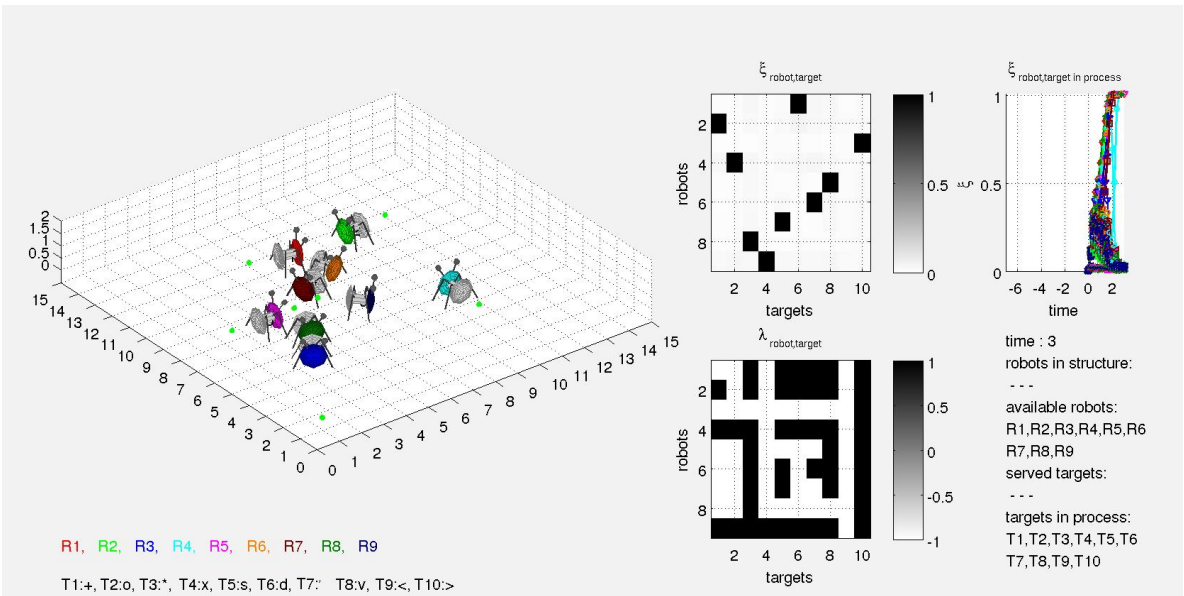

Abbildung A.9.: „Parallele Bildung einer Linie (1.Methode)“: Alle Zielzuordnungen sind ausgebildet.

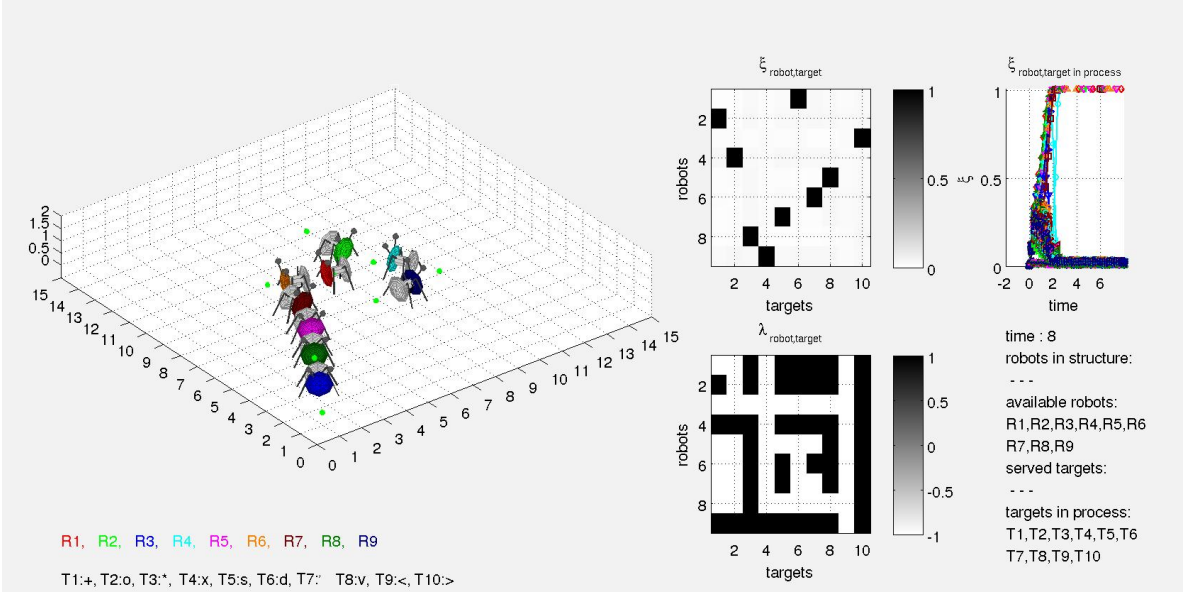

Abbildung A.10.: „Parallele Bildung einer Linie (1.Methode)“: Der Anfang der Kette ist deutlich zu erkennen.

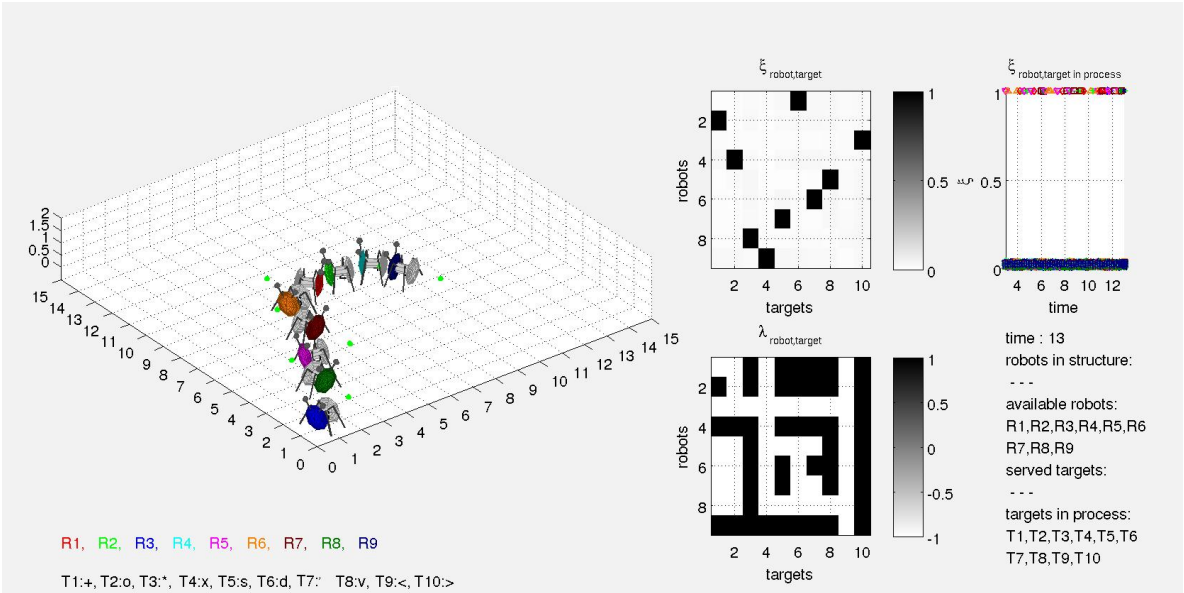

Abbildung A.11.: „Parallele Bildung einer Linie (1.Methode)“: Ausbildung einer Schwingung beim Erreichen des Ziels.

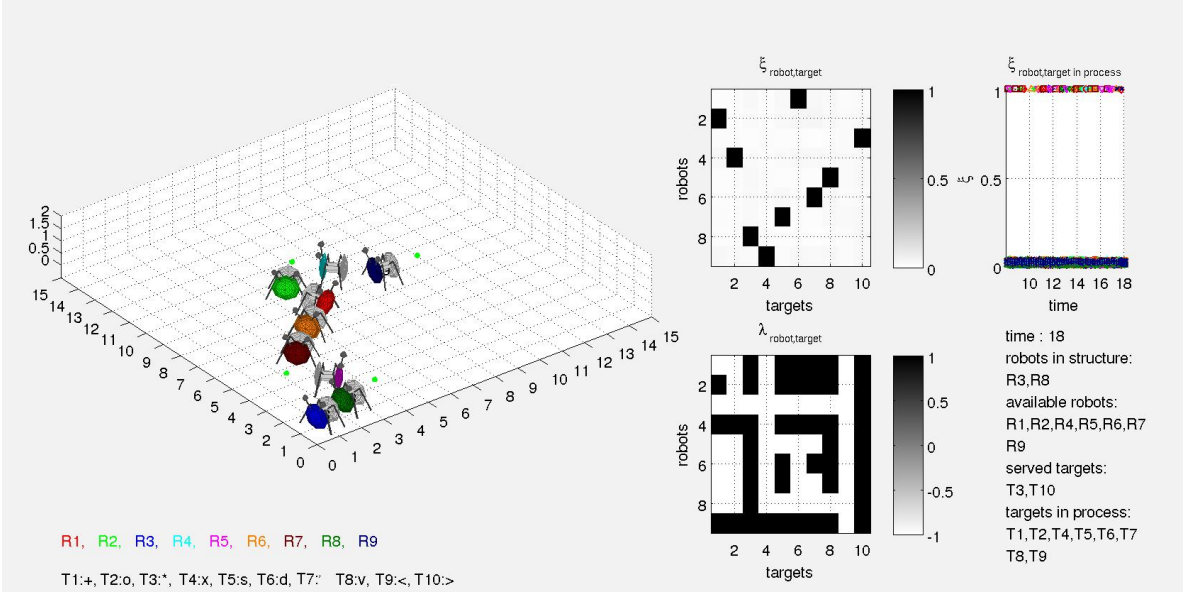

Abbildung A.12.: „Parallele Bildung einer Linie (1.Methode)“: Ausrichten der Kette am Ziel nach 18 Sekunden.

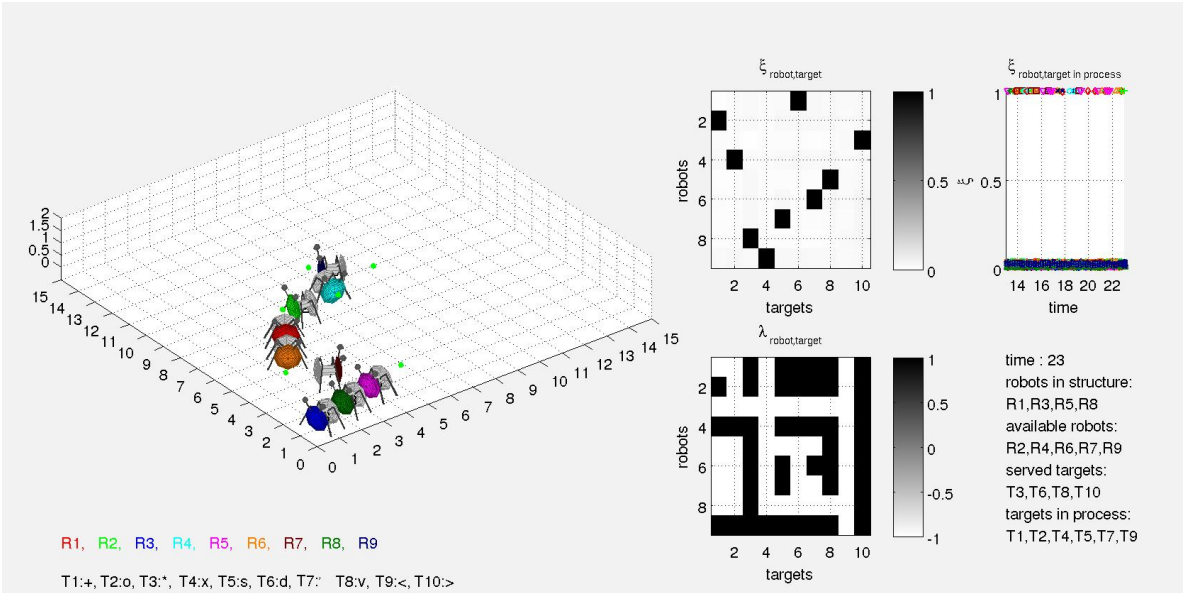

Abbildung A.13.: „Parallele Bildung einer Linie (1.Methode)“: Ausrichten der Kette am Ziel nach 23 Sekunden.

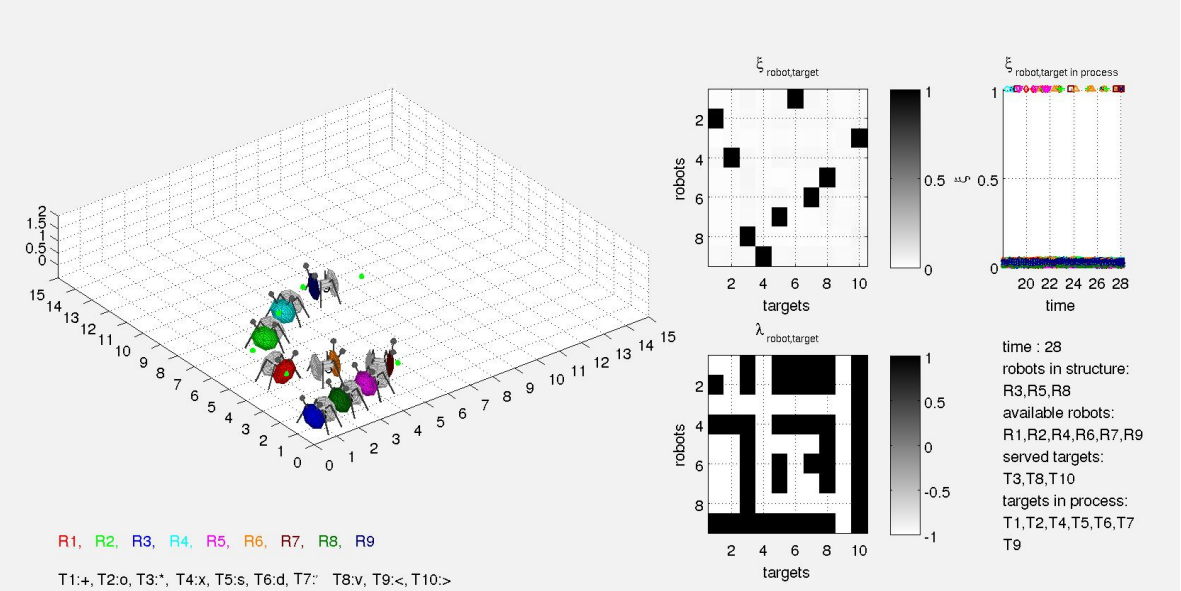

Abbildung A.14.: „Parallele Bildung einer Linie (1.Methode)“: Ausrichten der Kette am Ziel nach 28 Sekunden.

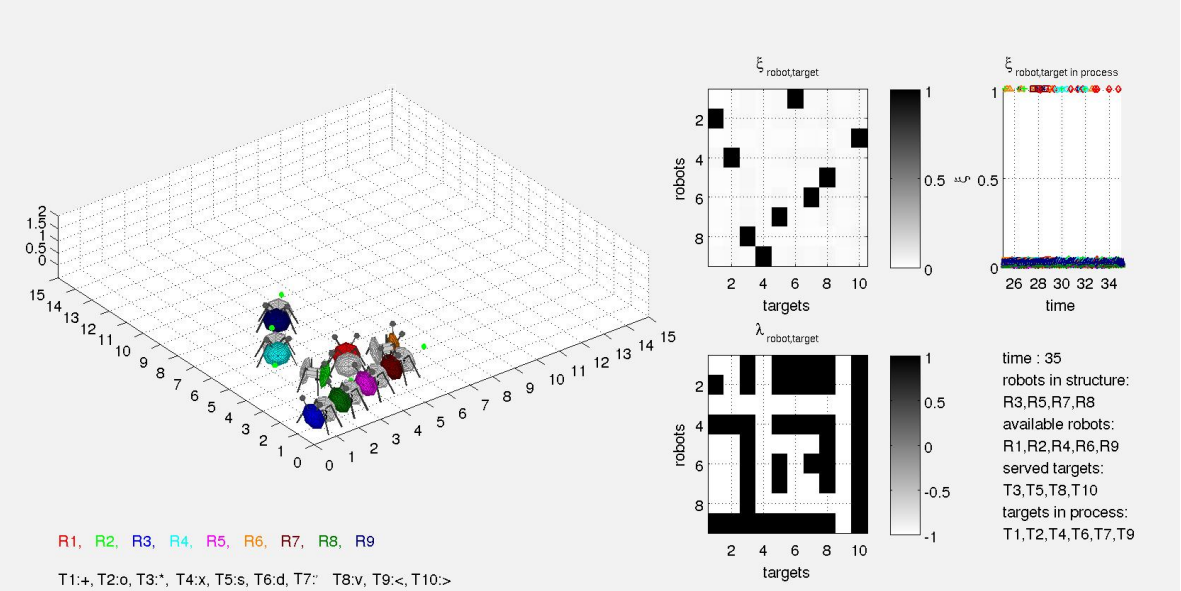

Abbildung A.15.: „Parallele Bildung einer Linie (1.Methode)“: Ausrichten der Kette am Ziel nach 35 Sekunden.

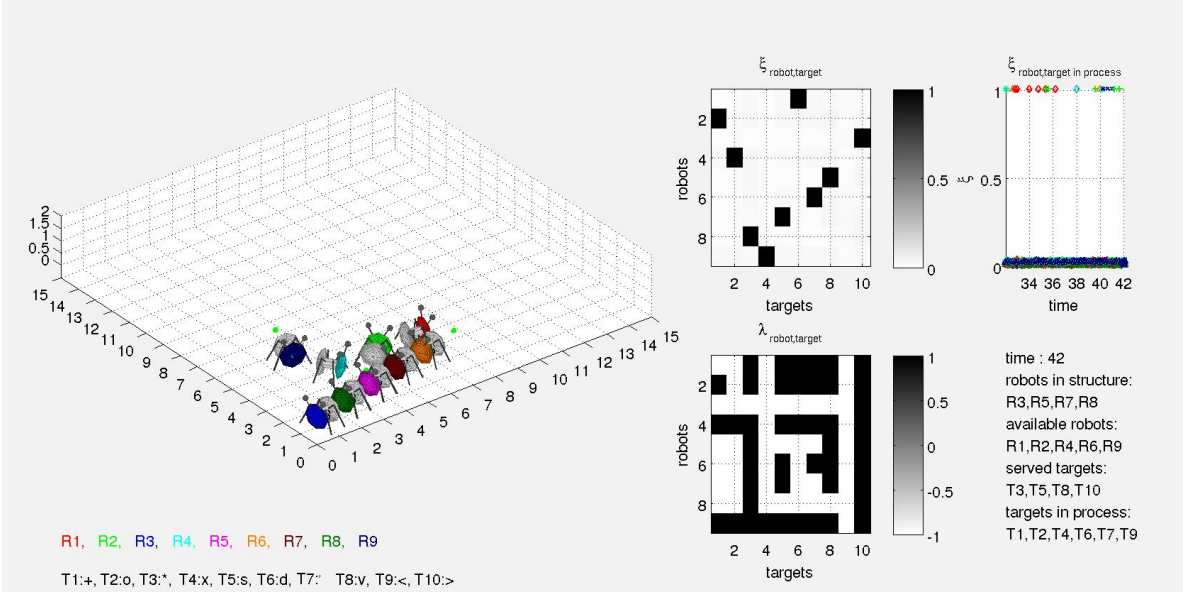

Abbildung A.16.: „Parallele Bildung einer Linie (1.Methode)“: Ausrichten der Kette am Ziel nach 42 Sekunden.

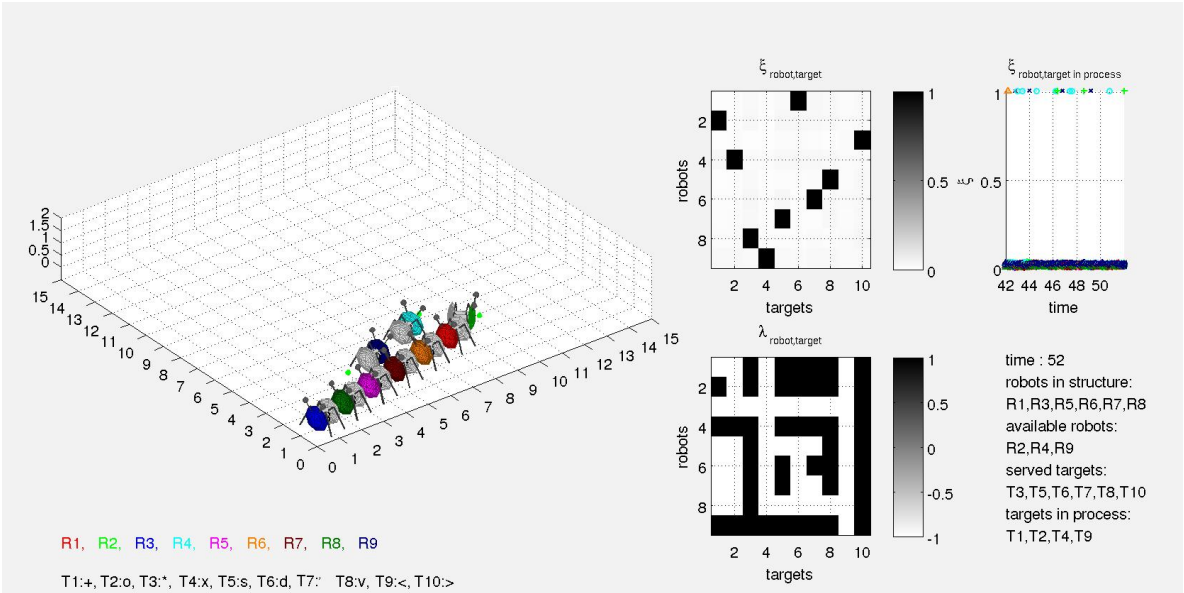

Abbildung A.17.: „Parallele Bildung einer Linie (1.Methode)“: Ausrichten der Kette am Ziel nach 52 Sekunden.

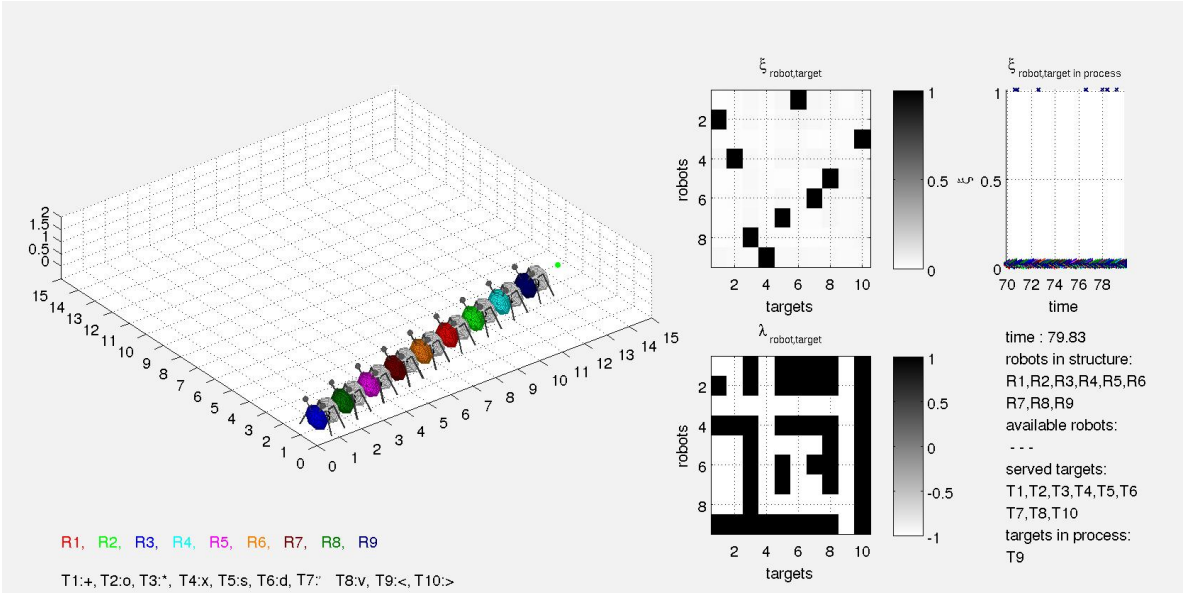

Abbildung A.18.: „Parallele Bildung einer Linie (1.Methode)“: Endgültige Situation.

B. Ein „Hello World“-Programm mit vielen Bugs

In diesem Teil der Arbeit wird ein „Hello World“- Programm vorgestellt, das relativ viele Bugs enthält. Das Programm basiert auf der Strukturbildung mit im Raum vorgegebenen Zielen. Somit sind alle λ 1. Weil die Anzahl der benötigten Roboter nicht konstant ist, aber die Anzahl der Ziele nicht verändert werden sollte, wurden „Dummy“-Ziele an zufälligen Positionen eingefügt. Diese haben eine andere Ausrichtung und so sind die Roboter auf diesen Zielen durch ihr Aufstellen leicht zu erkennen. Der Abstand zwischen den Robotern wurde ausreichend groß gewählt, um es anderen Robotern zu ermöglichen, zwischen zwei Robotern hindurchzukommen.

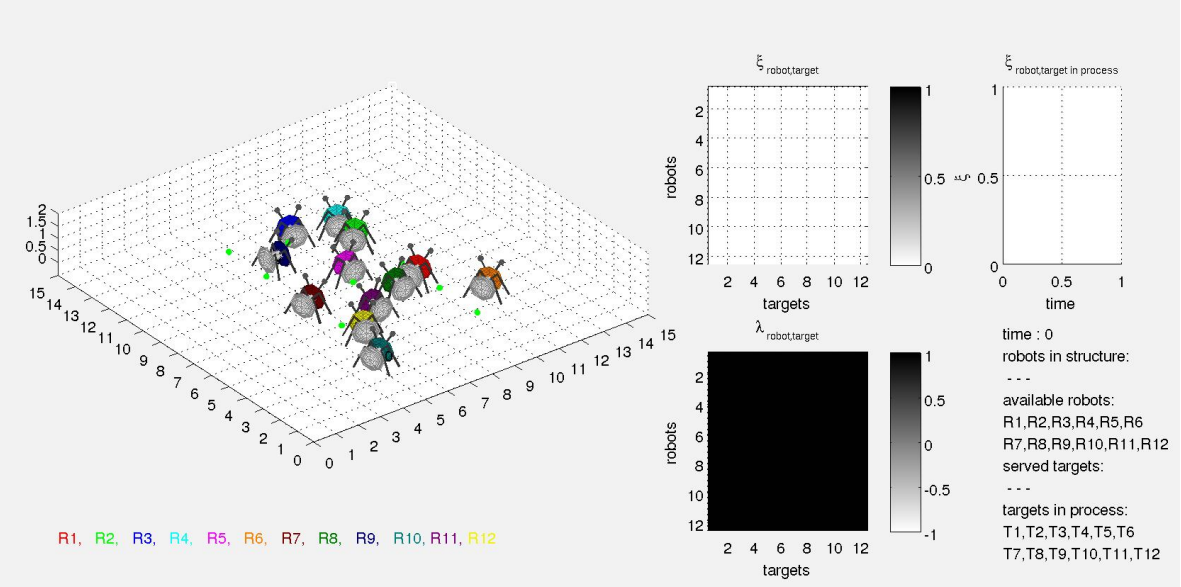

Abbildung B.1.: „Hello World“: Am Anfang der Simulation

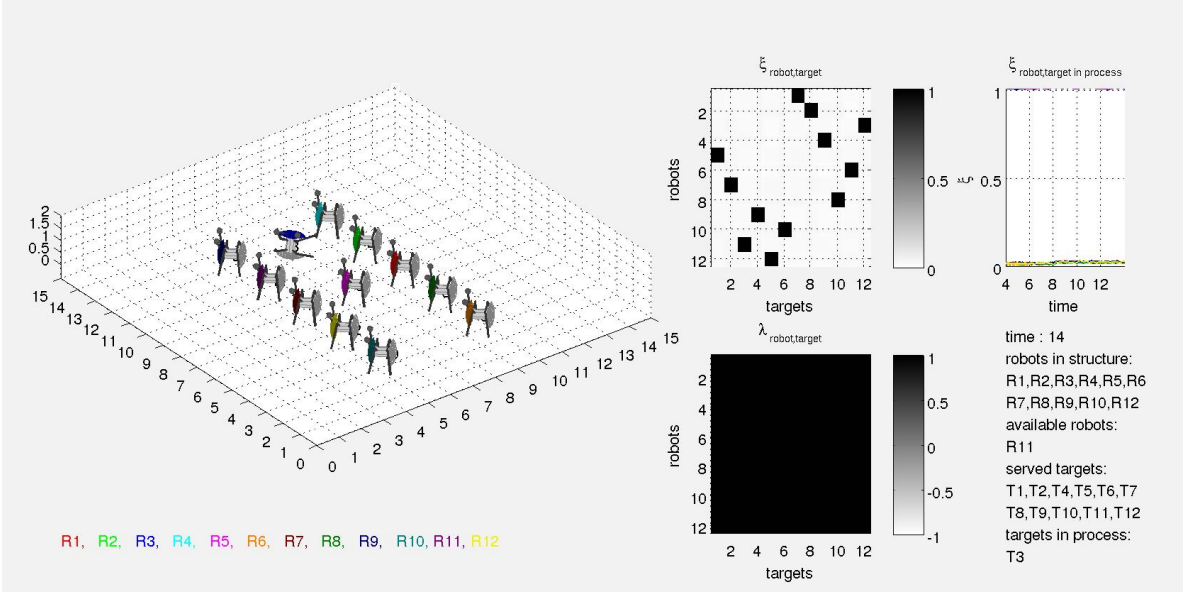

Abbildung B.2.: „Hello World“: Das H

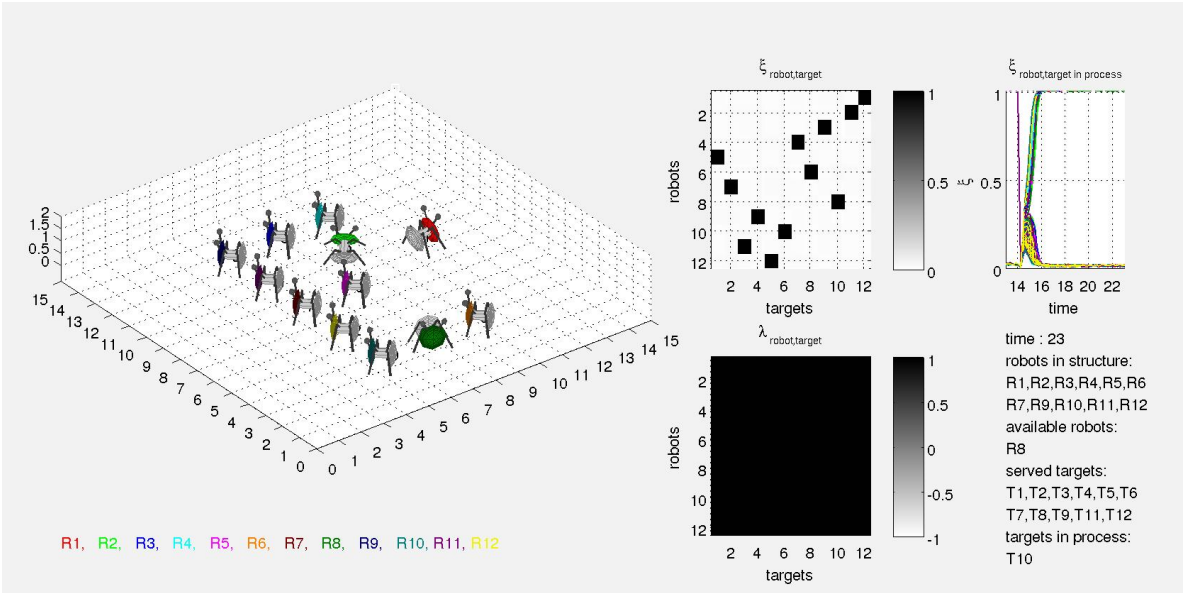

Abbildung B.3.: „Hello World“: Das E

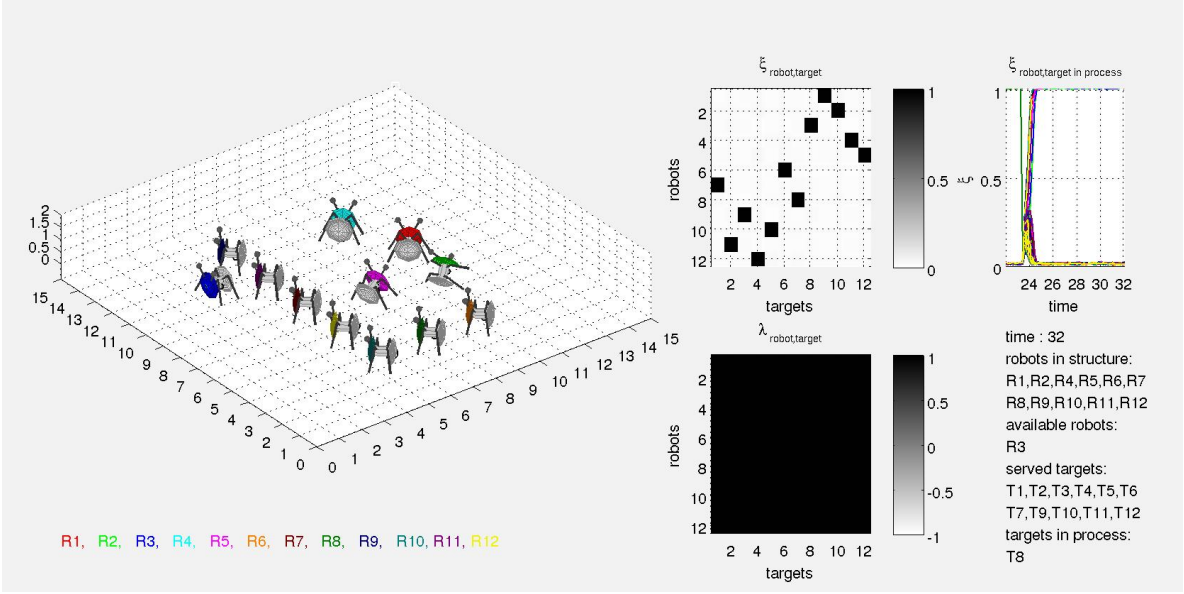

Abbildung B.4.: „Hello World“: Das erste L

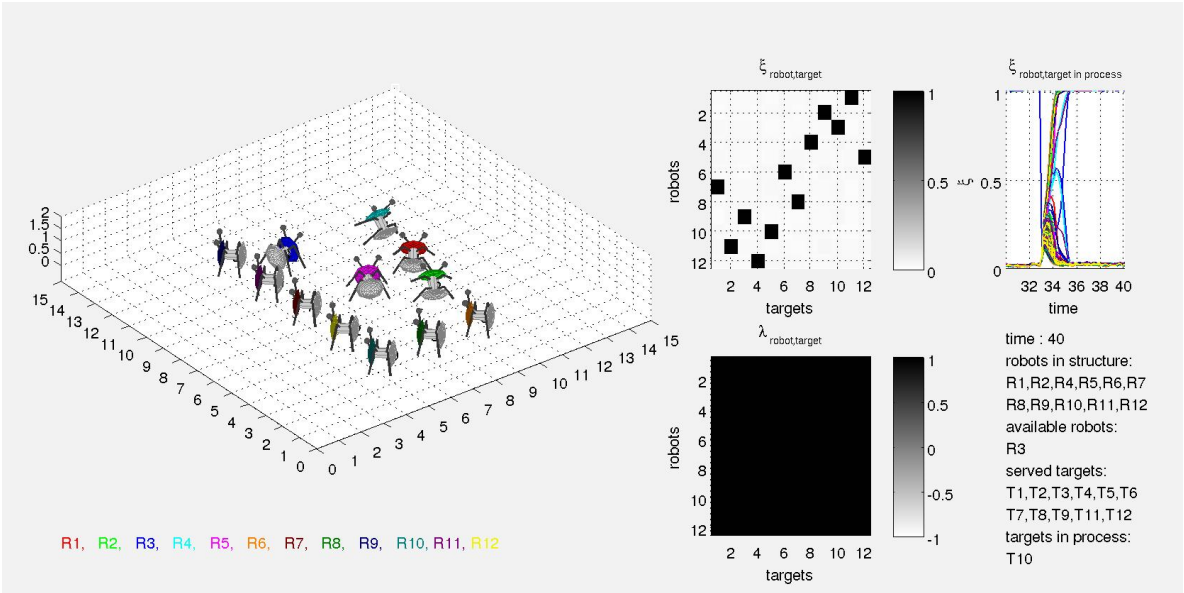

Abbildung B.5.: „Hello World“: Das zweite L

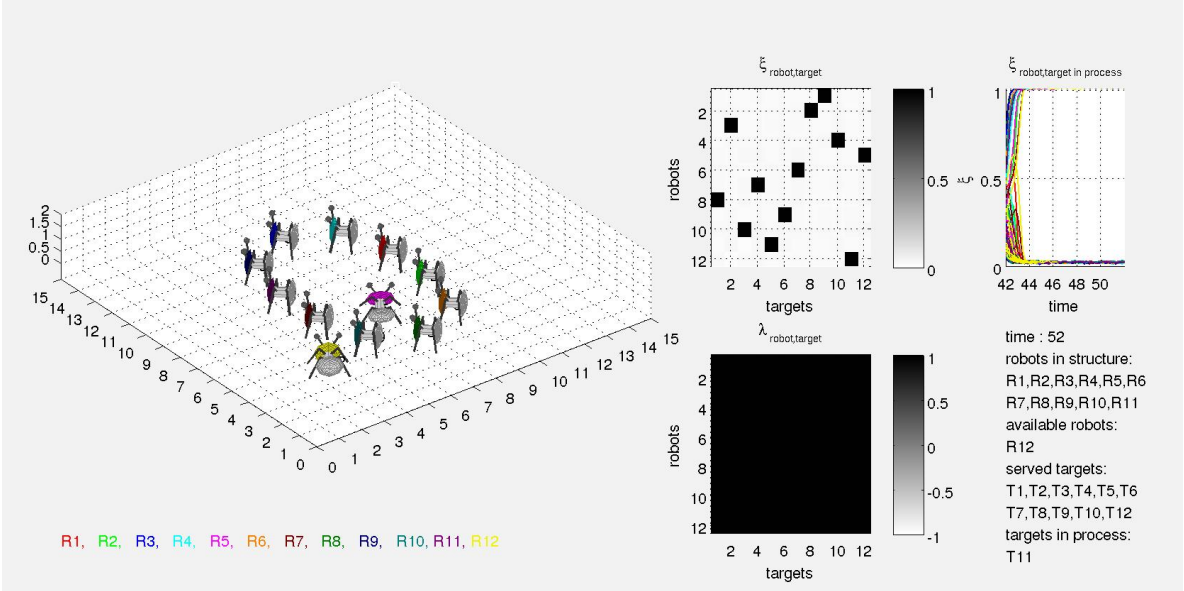

Abbildung B.6.: „Hello World“: Das erste O

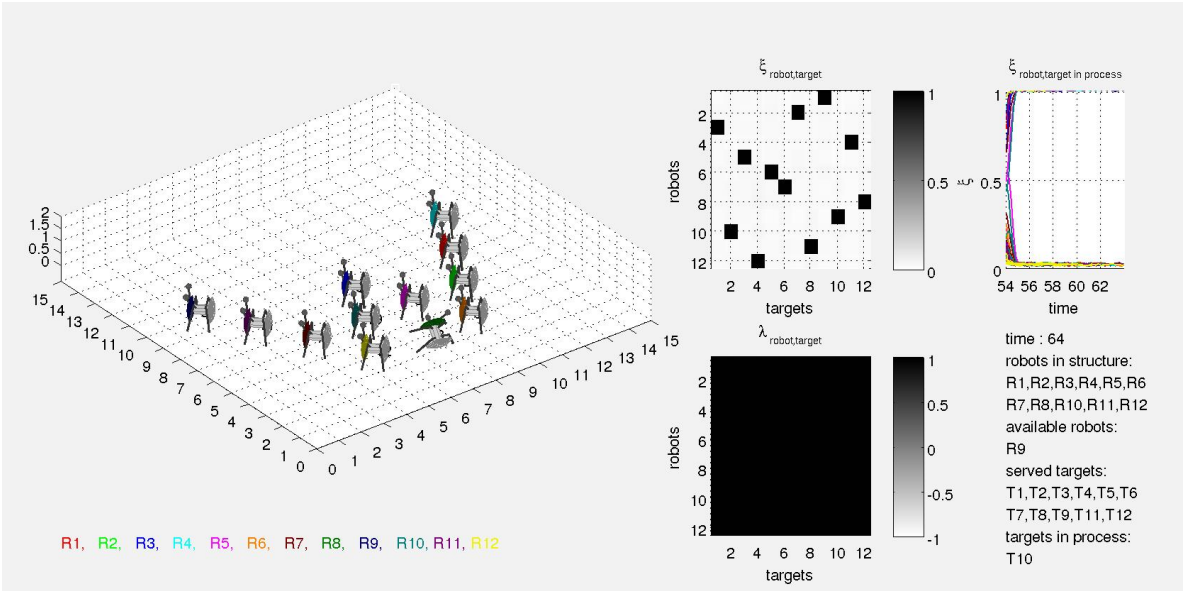

Abbildung B.7.: „Hello World“: Das W

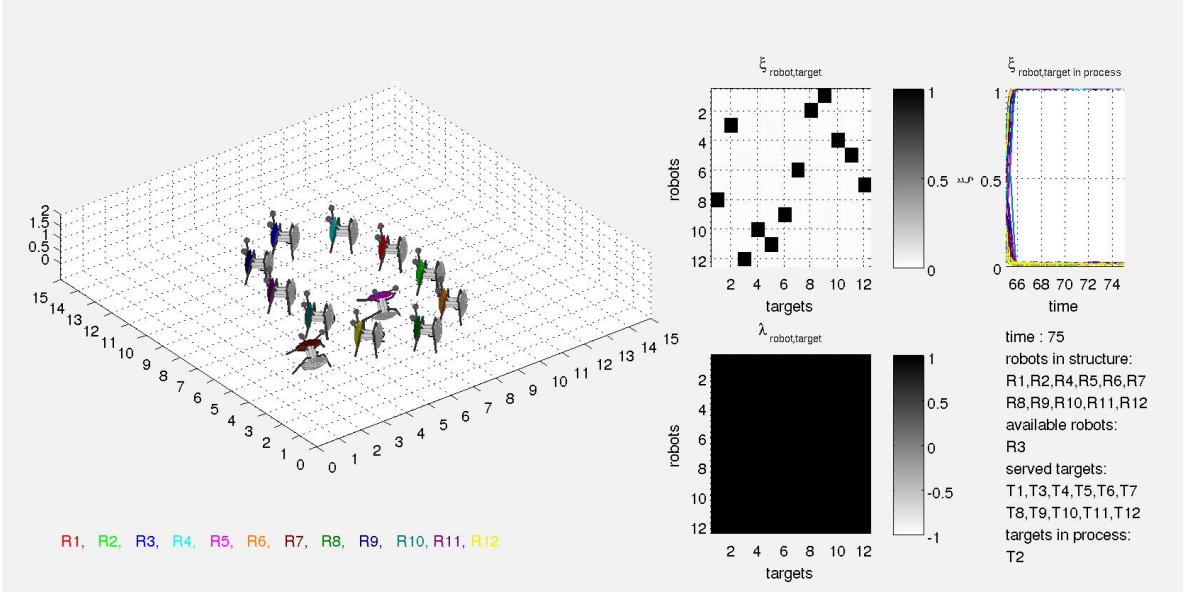

Abbildung B.8.: „Hello World“: Das zweite O

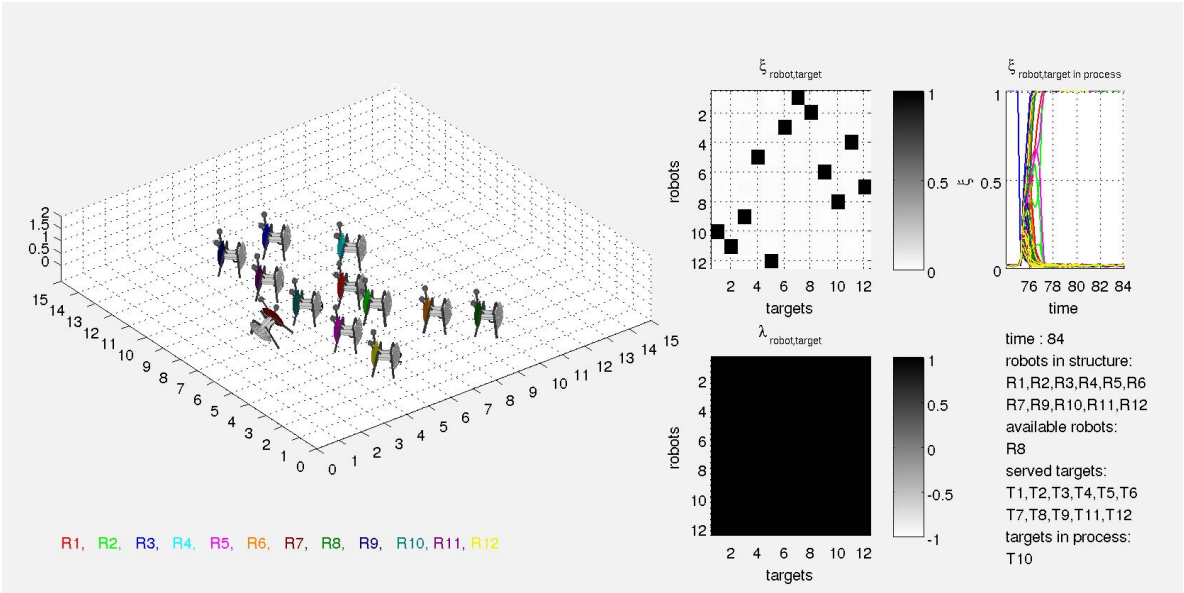

Abbildung B.9.: „Hello World“: Das R

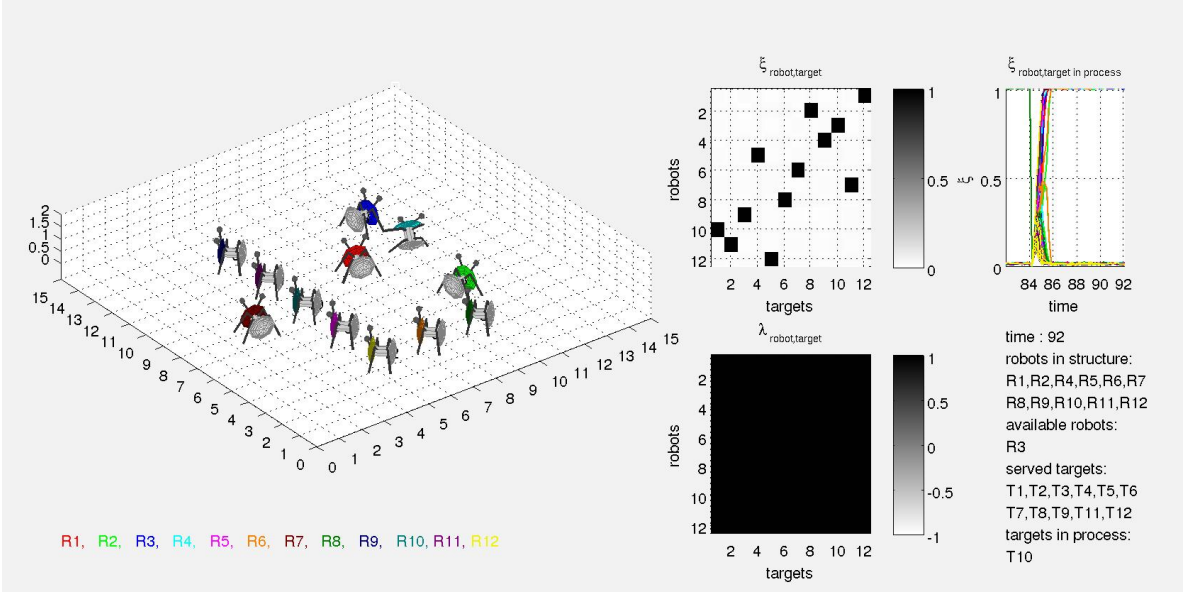

Abbildung B.10.: „Hello World“: Das dritte L

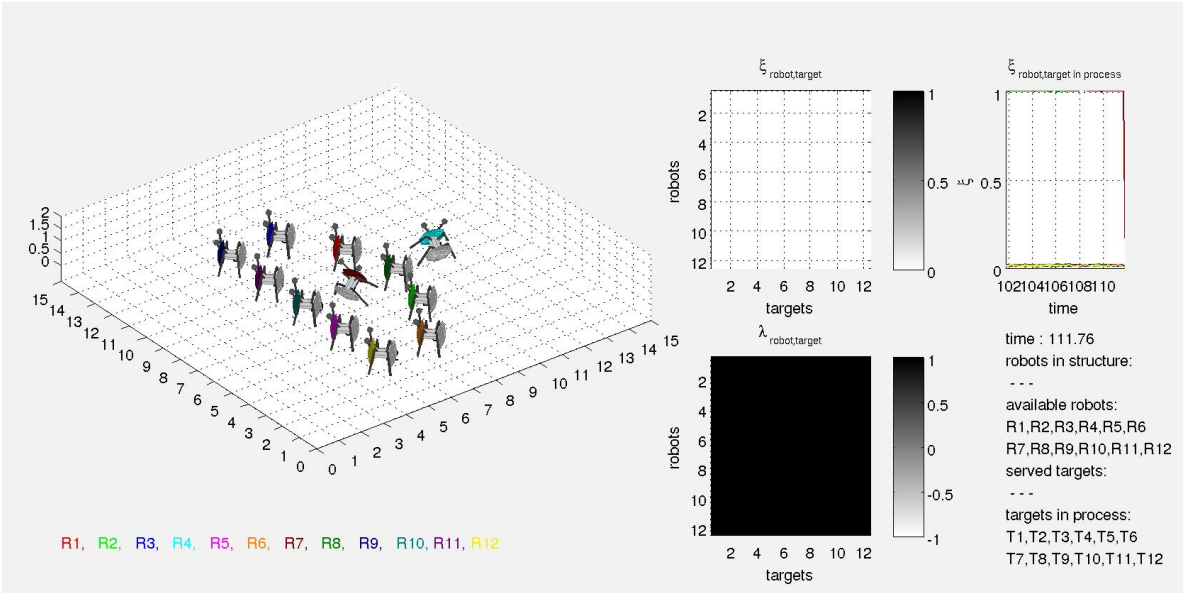

Abbildung B.11.: „Hello World“: Das D

Literaturverzeichnis

[J.S98] J.Starke, M.Schanz: Dynamical system approaches to combinatorial optimization. In: D.-Z.Du, P.Pardalos (Hrsg.): *Handbook of Combinatorial Optimization* Bd. 2, Kluver Academic Bublisher, 1998, S. 471–524 (Zitiert auf Seite 12)

[J.S02] J.Starke: Dynamical Assignments of Distributed Autonomous Robotic Systems to Manufacturing Targets Considering Environmental Feedbacks. In: *International Symposium on Intelligent Control* IEEE, 2002, S. 678–683 (Zitiert auf den Seiten 13, 14 und 15)

[J.S11] J.Starke, T.Fukuda C.Ellsaesser: Self-organized control in cooperative robots using a pattern formation principle. In: *Physics Letters* (2011) (Zitiert auf den Seiten 14, 15 und 19)

[M.S04] M.Schanz: IPVS Uni Stuttgart, 2004. – Forschungsbericht (Zitiert auf den Seiten 11 und 12)

[R.L] R.Lafrenz, U.-P.Käppeler O.Zweigle H.Rajaie F.Schreiber P. M.Schanz: Evaluating robustness of coupled selection equations using sparse communication / University of Stuttgart. – Forschungsbericht (Zitiert auf Seite 12)

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Martin Weber)