Institut für Parallele und Verteilte Systeme

Abteilung Maschinelles Lernen und Robotik

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Bachelorarbeit Nr. 2462

Automatisiertes Testen und Analyse einer Pick'n'Place Anwendung am Beispiel des PR-2 Roboters

Kim Peter Wabersich

Studiengang:	Technische Kybernetik
Prüfer:	Prof. Dr. rer. nat. Marc Toussaint
Betreuer:	Dr.-Ing. Ingo Lütkebohle
begonnen am:	12.03.2014
beendet am:	12.08.2014
CR-Klassifikation:	H.3.4

Inhaltsverzeichnis

1 Einleitung **5**
- 1.1 Zielsetzung der Arbeit . . . 5
- 1.2 Überblick . . . 6

2 Systemkonzept **7**
- 2.1 Anforderungen . . . 7
- 2.2 Optimierungselement . . . 8
  - 2.2.1 Realisierung . . . 9
  - 2.2.2 Beispiele . . . 10
- 2.3 Zusammenschluss von Optimierungselementen . . . 11
  - 2.3.1 Ausführung . . . 11
- 2.4 Kontinuierliche Bewertungsfunktion . . . 12
- 2.5 Fehlerklassifizierungsfunktion . . . 13

3 Automatisiertes Testen bzw. Optimieren **15**
- 3.1 Formulierung des Optimierungsproblems . . . 15
- 3.2 Globale Optimierung auf Basis maschinellen Lernens . . . 16
  - 3.2.1 Modell . . . 16
  - 3.2.2 Berechnung der Hyperparameter . . . 18
  - 3.2.3 Vorhersage von Funktionswerten . . . 19
  - 3.2.4 Modellfehler . . . 21
  - 3.2.5 Heuristiken . . . 21
  - 3.2.6 Hinweise zur Implementierung . . . 25
- 3.3 Synthetische Testfunktionen . . . 26
- 3.4 MINIMAX Optimierungsstrategie . . . 27

4 Beispielapplikation: Pick'n'Place **29**
- 4.1 Statische Szenengenerierung . . . 29
- 4.2 Pick'n'Place am PR-2 . . . 30
  - 4.2.1 MoveIt! . . . 30
  - 4.2.2 Ablauf . . . 32
  - 4.2.3 Implementierung . . . 32
  - 4.2.4 Objekterkennung . . . 33

5 Evaluation und Optimierung **35**
- 5.1 Objekterkennung . . . 35
- 5.2 MoveIt! . . . 37

5.3 Optimale Wahl des Planungsalgorithmus 41

6 Schluss **43**
6.1 Zusammenfassung 43
6.2 Ausblick 43

A Verwendete Softwareversionen **45**

Kapitel 1

Einleitung

Moderne Roboterframeworks wie ROS (**R**obotic **O**perating **S**ystem) beinhalten eine Vielzahl an Modulen für die Entwicklung von Roboterapplikationen. Innerhalb des letzten Jahres gewann das Softwarepaket MoveIt! auf Basis von ROS zur Handlungsplanung und Aktionsausführung zunehmend an Bedeutung. Unabhängig von der Roboterplattform ermöglicht es die effiziente Entwicklung neuer Roboterapplikationen. Anhand des PR-2 Roboters existiert ein erstes Benchmarksystem[1] innerhalb des MoveIt! Frameworks, mit dem sich jedoch nur einzelne Planungsalgorithmen analysieren lassen. Wie es typisch für solche Benchmarksysteme ist, können nur einzelne Szenen statisch vorgegeben werden. Um praxisnahe Aussagen über eine MoveIt! basierte Roboterapplikation zu erhalten ist daher ein Testsystem wünschenswert, welches die Wahrnehmung der Umgebung durch die Robotersensorik sowie die Ausführung von Aktionen mitberücksichtigt. Zudem ist es von Vorteil, gewisse Freiheitsgrade in der Umgebung beim Testen der Roboterapplikation zu „erlauben“, um damit deren Performance nicht nur anhand weniger Szenarien zu bewerten. Diese Freiheitsgrade können durch ein Optimierungsverfahren gezielt dazu genutzt werden, unterschiedliche Aspekte durch entsprechende Bewertungsfunktionen automatisiert zu untersuchen. Nimmt man dieses Testsystem als Bewertungsmetrik, so können Optimierungen der Roboterapplikation nach dem MINIMAX-Prinzip durchgeführt werden.

1.1 Zielsetzung der Arbeit

Als Applikation für das Testsystem soll eine einfache Pick'n'Place Applikation auf Basis von MoveIt! für den PR-2 Roboter (Abbildung 1.1) implementiert werden. Mit diesem Testsystem soll es insbesondere möglich sein, die Wahrnehmung und die Ausführung von Aktionen durch MoveIt! an einem konkreten Roboter in der Simulation zu bewerten.
Da es im typischen Einsatzbereich humanoider Roboter eine Vielzahl an freien Parametern gibt, ist eine endliche Anzahl an vordefinierten Szenarien für einen repräsentativen Benchmark ungeeignet. Vielmehr soll sich die Umgebung, in der sich der Roboter befindet, möglichst so verändern, dass potentielle Schwächen der Applikation „automatisiert “ zum Vorschein gebracht werden können. Dies soll durch eine Optimierung freier Umgebungsparameter bzgl. einer Bewertung der Applikation ge-

[1] `http://moveit.ros.org/wiki/Benchmarking`, zuletzt geprüft am 31.7.2014.

schehen. Anhand des Beispiels der Pick'n'Place Anwendung wäre es z.B. von Interesse, welche Planungsalgorithmen zeiteffizienter mit Hindernissen im Arbeitsbereich umgehen können als andere. Dieser Grundgedanke der „bewussten" Modifikation der Umgebung durch ein Optimierungsverfahren soll generalisiert und als allgemein anwendbares ROS-Modul implementiert werden. Darüber hinaus soll es möglich sein, analog zum automatisierten Testen von Applikationen, diese zu optimieren. Die Optimierung erhält dann als Bewertungsmetrik das Ergebnis eines unterlagert ermittelten Testfalls.

1.2 Überblick

Die Arbeit lässt sich inhaltlich in drei Abschnitte unterteilen. Zuerst wird auf das erarbeitete Konzept zum automatisierten Testen und Optimieren von Roboterapplikationen innerhalb von ROS eingegangen.
Im Anschluss daran wird das verwendete Optimierungsverfahren als Kern des Test- und Optimierungssystems erläutert, hergeleitet und anhand synthetischer Optimierungsprobleme getestet.
Als letzter Punkt folgt die Implementierung und beispielhafte Evaluation und Optimierung der Pick'n'Place Applikation mit dem PR-2 Roboter.

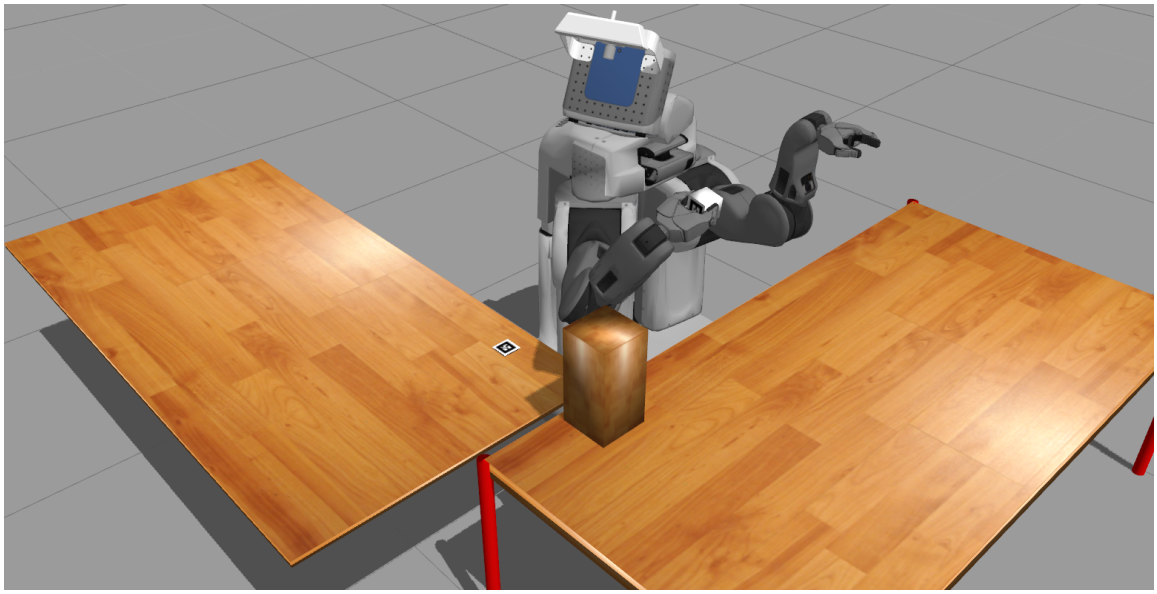


Abbildung 1.1: Beispielszenario für die Pick'n'Place Anwendung mit dem PR-2.

Kapitel 2

Systemkonzept

2.1 Anforderungen

Am Beispiel der Pick'n'Place Applikation soll das System in der Lage sein, automatisiert Tests bezüglich vorgegebener Kriterien durchzuführen. Dazu müssen Umgebungsparameter veränderbar sein, die in Wechselwirkung mit der Applikation stehen. Bei dem Greifen und Abstellen von Objekten könnte z.B. die Position von Hindernissen oder der Winkel eines Lichteinfalls modifiziert werden. Ein beendeter Durchlauf soll durch eine Nutzen- bzw. Bewertungsfunktion nach seiner Güte bewertet werden. Auf Basis dieser Bewertung kann ein neuer Satz an Umgebungsparametern für den nachfolgenden Durchlauf generiert werden.
Neben einer kontinuierlichen Bewertungsfunktion soll eine diskrete gut/schlecht Klassifikation den Erfolg eines Durchlaufs bewerten. Dies kann beispielsweise durch Toleranzgrenzen realisiert werden. Die zugrunde liegende Idee ist schematisch in Abbildung 2.1 gezeigt. Je nach Bewertungsfunktion existiert eine Dualität zwischen auto-

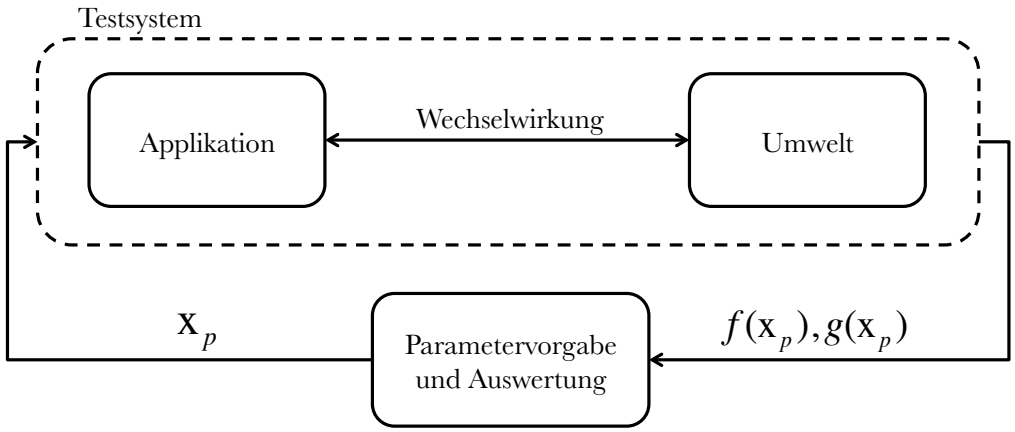


Abbildung 2.1: Schematische Darstellung der Grundidee des automatisierten Test- und Optimierungssystems, bestehend aus einem Testsystem und einer Komponente für die Generierung von Testfälle sowie deren Auswertung.

matisierten Testen und dem Optimieren einer Applikation. Deshalb wird im Weiteren einheitlich von Optimierung und Optimierungselementen gesprochen. Das Konzept kann daher auch als verallgemeinerte Optimierungsschnittstelle für ROS aufgefasst

werden. In dieser Arbeit wurde das so erarbeitete System dazu genutzt automatisiertes Testen und Optimieren einer Pick'n'Place Applikation am Beispiel des PR-2 Roboters in der Simulation zu ermöglichen.

Im Folgenden werden Parametermengen sowie Bewertungs- und Fehlerfunktion definiert, die für alle nachfolgenden Kapitel von Relevanz sind.
Die Menge

$$\mathcal{P}_p \subset \mathbb{R}^{d_p} \tag{2.1}$$

als Teilmenge des $\mathbb{R}^{d_p}$ hat die geometrische Form eines Hyperrechteckes, definiert durch minimale $\boldsymbol{x}_{p,min}$ und maximale $\boldsymbol{x}_{p,max}$ Werte jedes einzelnen Parameters $x_{p,i}, i = 1, 2, .., d_p$. Auf das Testen bezogen hat der Optimierer innerhalb dieses Raumes die Möglichkeit, die Umgebung so zu verändern, dass es die Applikation möglichst „schwer" hat. Von Außen betrachtet bildet

$$f : \mathcal{P}_p \to \mathbb{R} \tag{2.2}$$

die veränderbaren Umgebungs- oder Applikationsparameter auf eine skalare Bewertung der Applikation ab. Im Fall der Pick'n'Place Applikation könnte dies die benötigte Zeit der Ausführung oder die erreichte Genauigkeit zwischen Ist- und Sollposition des Gegenstandes sein.
Zur gut/schlecht Klassifikation eines Applikationsdurchlaufes bildet die Funktion

$$g : \mathcal{P}_p \to \{0, 1\} \tag{2.3}$$

die veränderbaren Umgebungs- oder Funktionsparameter auf eins ab, sofern ein Applikationsdurchlauf erfolgreich war. Bei nicht erfolgreichem Durchlauf auf null. Dadurch können beispielsweise vorgegebene Toleranzen, wie die maximale Abweichung zwischen einer Soll- und Istposition berücksichtigt werden.
Die Applikation und ihre Umgebung stellt von außen betrachtet eine „Black-Box"-Funktion[1] dar (Abb. 2.1). Diese „Black-Box" wird im Folgenden als *Testsystem* bezeichnet.
Bei der Pick'n'Place Applikation stellt das Robotersystem mit der Funktion „greife und platziere ein Objekt" die Applikation dar. Die Umwelt ist eine „statische Szene" in der die Applikation agiert. Wie beispielsweise in Abbildung 1.1 gezeigt.

2.2 Optimierungselement

Um das Test- und Optimierungssystem allgemein anwenden zu können, ist eine Kapselung des Gesamtkonzeptes notwendig. Einzelne, funktionale Einheiten werden als Optimierungselemente (kurz: Elemente) bezeichnet. Diese Elemente stellen einen allgemeinen Rahmen für funktional-abgrenzbare Einheiten dar. Funktional-abgrenzbar bedeutet hier, dass eine solche Einheit mindestens einen Zustand besitzt der wiederum mindestens veränderbar oder „messbar"[2] sein muss. Wie in Abbildung 2.2 gezeigt,

[1] Black-Box-Funktion bedeutet, dass keine interne (analytischen) Mechanismen bekannt sind. Die Funktion kann nur aufgerufen werden.

[2] „Messbarer" Zustand bedeutet im Fall der Simulation, dass dieser mit hinreichender Genauigkeit in der Simulation berücksichtigt und zugänglich ist.

bekommt ein Element den Vektor $\boldsymbol{x_{p,i}} \in \mathcal{P}_{p,i}$ wobei $\mathcal{P}_{p,1} \cup \mathcal{P}_{p,2} \cup .. \cup \mathcal{P}_{p,n} = \mathcal{P}_p$ als Eingang und liefert seinen internen Zustand $\boldsymbol{x_{o,i}} \in \mathbb{R}^{d_{o,i}}$ der Dimension $d_{o,i}$ als Ausgang.

Anmerkung: Da nur ein Teil aller internen Zustände modifizierbar ist, gleichzeitig jedoch alle modifizierbaren Zustände die Parameter repräsentieren gilt folglich $d_o \geq d_p$.

Auf Basis von $\boldsymbol{x_{o,i}}$ werden innerhalb des Testsystems aus Abb. 2.1 f und g berechnet.

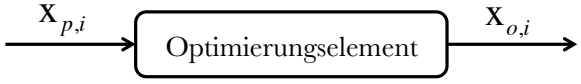


Abbildung 2.2: Einzelnes Optimierungselement erhält veränderbare Parameter als Eingang und liefert Zustandsattribute als Ausgang.

2.2.1 Realisierung

Um bereits entwickelte Applikationen ohne großen Aufwand testen und optimieren zu können, wurde eine abstrakte Basisklasse für Optimierungselemente sowohl für Python wie auch für C++ implementiert, siehe Abb. 2.3.

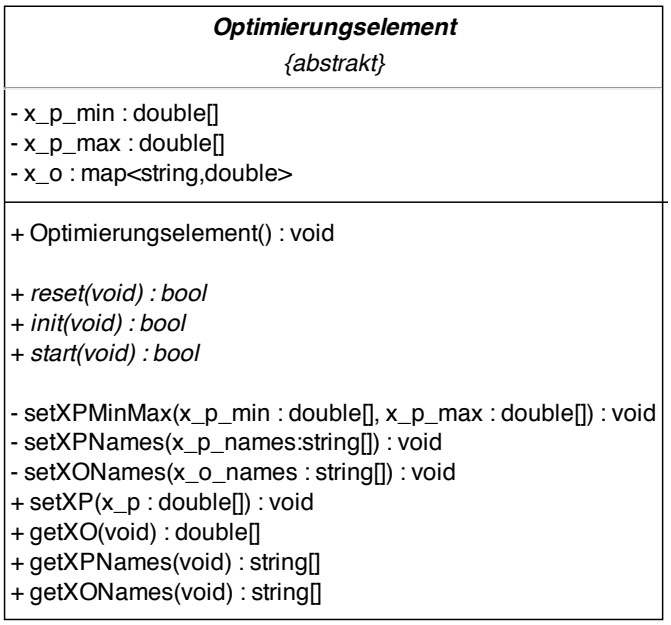


Abbildung 2.3: Vereinfachtes Klassendiagramm eines Optimierungselementes.

Durch den Konstruktor (`Optimierungselement()`) werden alle als `public` definierten Methoden als sogenannte *services* (Abbildung 2.4) angeboten. Diese ermöglichen eine direkte Interaktionen zwischen zwei laufenden Prozessen (*nodes*) innerhalb von ROS. Angelehnt an das Konzept von „Webservices“ beginnt eine Interaktion mit einem „request“, von *node a* an *node b*, auf den eine „response“ von b folgt. Die Datenstrukturen des „requests“ und der „response“ können nicht dynamisch gewählt werden, sondern müssen zum Zeitpunkt der Implementierung festgelegt werden.

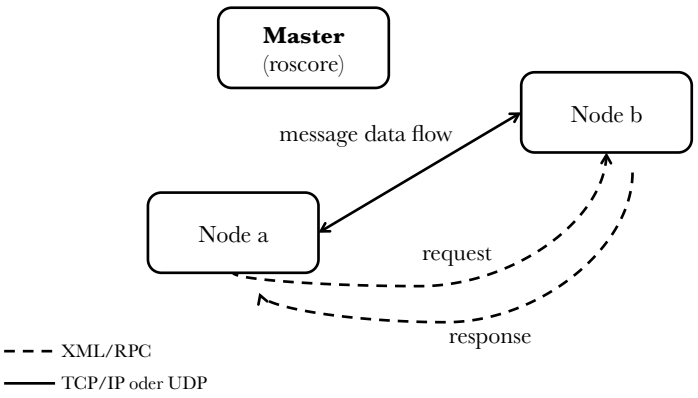


Abbildung 2.4: Schematische Darstellung eines service-Aufrufs in ROS.

Durch standardisierte Namen für die *services* eines Elements können diese automatisiert über eine Vielzahl an Optimierungselementen hinweg von einem zentralen *node* aufgerufen werden. Dieses Prinzip ist in Abbildung 2.5 veranschaulicht.

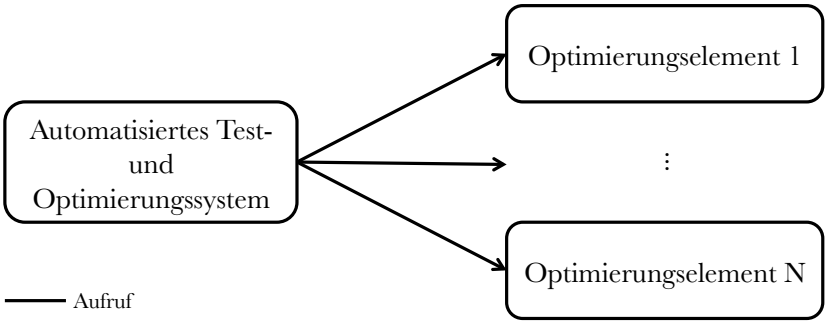


Abbildung 2.5: Aufruf von standardisierten *services* von Optimierungselementen.

Dadurch, dass die Namen der Zustände mit dem Service `get_XONames()` abgefragt werden können, kann der Anwender bei der Erstellung von Bewertungs- (Abschnitt 2.4) und Klassifizierungsfunktion (Abschnitt 2.5) mit einer Templategenerierung unterstützt werden.

2.2.2 Beispiele

Wie in Abbildung 2.6 dargestellt fungiert als Beispiel für eine funktional-abgrenzbare Optimierungseinheit zum Einen die statische Umgebung des Roboters (Abschnitt 4.1). Diese beinhaltet sämtliche Objekte, die zeitlich invariant sind und direkten Einfluss auf den Aktionsbereichs des Roboters haben. Zum Anderen ist das Pick'n'Place System (Abschnitt 4.2) mit dem PR-2 als Plattform ein Beispiel für eine Applikationsumgebung.

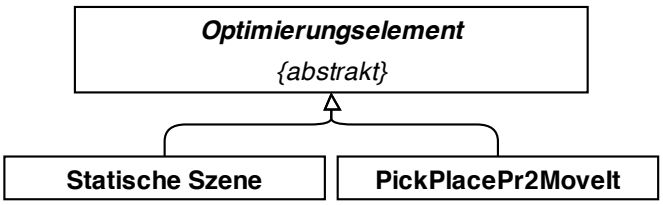


Abbildung 2.6: Beipiele Pick'n'Place und Statische Szene als Realisierung von Optimierungselementen.

2.3 Zusammenschluss von Optimierungselementen

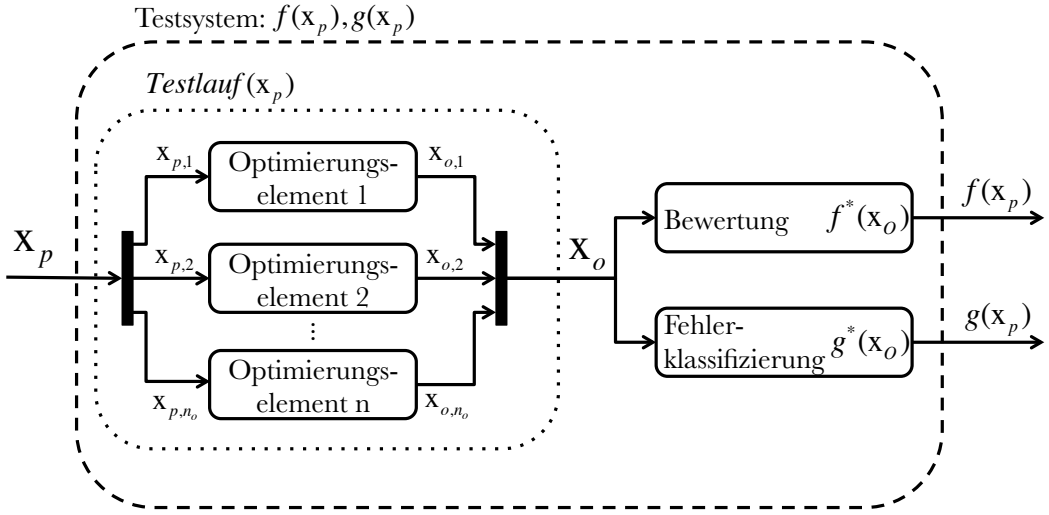


Abbildung 2.7: Zusammenschluss von Optimierungselementen in Testsystem.

Abbildung 2.7 zeigt den Zusammenschluss mehrerer Optimierungselemente als Teil der internen Mechanismen des Testsystems. Die Parametervorgabe $\boldsymbol{x_p}$ wird in $\boldsymbol{x_{p,i}}$ mit $i = 1, .., n_o$ für die jeweiligen Optimierungselemente aufgeteilt.
Die internenen Zustände $\boldsymbol{x_{o,i}}$ mit $i = 1, .., n_o$ als Ausgänge der Optimierungselemente werden zu $\boldsymbol{x_o}$ zusammengefasst. Nun kann auf Basis von $\boldsymbol{x_o}$ die Bewertung sowie Fehlerklassifikation durchgeführt werden.
Im Vergleich zur einzelnen Bewertungen von Elementen hat dieser Ansatz den Vorteil, dass interne Zustände verschiedener Elemente für die Bewertung in Wechselwirkung gesetzt werden können. Ein Beispiel hierfür wäre ein Vergleich von der Position eines Objektes in der Simulation, mit der erkannten Position desselben durch eine Objekterkennung.

2.3.1 Ausführung

Algorithmus 1 zeigt die Ausführung eines Verbundes von Optimierungselementen als Aufruf des Testsystems der im Folgenden als *Testlauf* bezeichnet wird.

Algorithm 1 Aufruf des Testsystems

```
1: procedure TESTLAUF(x_p)
2:     teile x_p in n-Vektoren x_{p,i} auf
3:     error ← 0
4:     for i − tesElement ∈ Optimierungselemente ∧ error = 0 do
5:         error ← initialisiere i-tes Element mit x_{p,i}
6:         if error then
7:             return Fehler bei Initialisierung
8:         end if
9:     end for
10:    for i − tesElement ∈ Optimierungselemente ∧ error = 0 do
11:        [x_{o,i}, error] ← starte i-tes Element
12:        if error then
13:            return Fehler bei Ausführung
14:        end if
15:    end for
16:    vereine x_{o,i} zu x_o
17:    return x_o
18: end procedure
```

2.4 Kontinuierliche Bewertungsfunktion

Um flexibel bei der Festlegung einer Bewertungsfunktion zu sein, basiert diese auf der Vereinigung aller internen Zustände $\boldsymbol{x}_o$ der Optimierungselemente. Sie ist durch

$$f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}}(\boldsymbol{x}_o) = \frac{1}{d_m}\boldsymbol{\omega}^T[\mathbf{Z}\boldsymbol{\phi}(\mathbf{M}\boldsymbol{x}_o)] \tag{2.4}$$

definiert[3] und wurde wie in Abb. 2.8 gezeigt in fünf Berechnungsschritte aufgeteilt:

1. *Lineare Wechselwirkung:* Die Matrix

   $$\mathbf{M} \in \mathbb{R}^{d_M \times d_o} \tag{2.5}$$

   wird als „Wechselwirkungsmatrix“ bezeichnet und bildet d_o-internen Zustände aller Optimierungselemente auf d_M-lineare Zusammenhänge ab.

2. *Nichtlineare Features:* Die nichtlineare Funktion

   $$\boldsymbol{\phi} : \mathbb{R}^{d_M} \to \mathbb{R}^{d_\phi} \tag{2.6}$$

   wirkt auf die von (2.5) erzeugten, linearen Zusammenhänge. Mit (2.6) können nichtlineare Eigenschaften wie Betrag oder quadratischer Fehler ohne Einschränkungen in der Bewertungsfunktion berücksichtigt werden. Das Ergebnis ist ein Vektor mit d_ϕ-Bewertungsmerkmalen.

[3]Für eine konsistente Darstellung wurde in Abb. 2.8 die Bezeichnung $f^*(\boldsymbol{x}_o)$ bzw. $g^*(\boldsymbol{x}_o)$ verwendet, da die diese im Gegensatz zu der Grundidee in Abb. 2.1 Funktionen von dem vereinigten, internen Zustandsvektor $\boldsymbol{x}_o$ sind.

3. *Skalierung:* Um eine normierte Gewichtung der Merkmale zu ermöglichen, wird eine Skalierung durch Multiplikation der Bewertungsmerkmale mit der Matrix

$$\mathbf{Z} \in \mathbb{R}^{d_\phi \times d_\phi} \text{ mit } z_{ii} = \frac{1}{\max\limits_{t \in T} \mid \phi(\mathbf{M}\boldsymbol{x}_o)_i \mid}, \; z_{ij} = 0 \; \forall \; i \neq j \tag{2.7}$$

vorgenommen. Ihre Elemente z_{ii} enthalten die invertierten maximalen Werte jedes nichtlinearen Bewertungsmerkmales über alle Testläufe T hinweg. Dies wird rückwirkend auf den gesamten Datensatz an Auswertungen angewandt.

4. *Gewichtung:* Mittels

$$\boldsymbol{\omega}^T \in \; [0, 1] \tag{2.8}$$

als Gewichtungsvektor werden die skalierten Bewertungsmerkmale auf eine skalare Größe abgebildet. Dazu werden diese mit dem Gewichtungsvektor $\boldsymbol{\omega}^T$ aus (2.8) mutlipliziert.

5. *Normierung:* Das skalare Ergebnis aus der Gewichtung wird mittels einer Division durch d_ϕ auf einen absoluten Maximalwert von eins normiert.

Mit der so definierten Bewertungsfunktion ist über eine Variation des Gewichtungsvektors die Analyse eines Pareto-Optimums möglich. Ein Beispiel aus der Pick-'n'Place Applikation ist eine Optimierung des quadratischen Fehlers der Soll-Position eines Objekts bei gleichzeitiger Optimierung der Ausführungsgeschwindigkeit. Im Optimum variieren die Lösungen über die Gewichtungen $\boldsymbol{\omega}^T$ der widersprüchlichen Anforderungen.

2.5 Fehlerklassifizierungsfunktion

Die Fehlerklassifikation[3] $g^*_{\boldsymbol{M},\boldsymbol{\Phi},\mathbf{c}}(\boldsymbol{x}_o)$ berechnet sich aus den elementweisen Ungleichungen

$$c_i - (\boldsymbol{\phi}(\mathbf{M}\boldsymbol{x}_o))_i \geq 0 \tag{2.9}$$

und ist durch $\boldsymbol{M}, \boldsymbol{\Phi}$ sowie den Toleranzgrenzen

$$\mathbf{c} \in \mathbb{R}^{d_\phi} \tag{2.10}$$

definiert. Wie in Abb. 2.8 gezeigt, findet eine elementweise Prüfung der Ungleichung (2.9) statt. Sind alle Ungleichungen erfüllt, wird eine eins zurückgeliefert, ansonsten eine null. Das automatisierte Testen in Kapitel 3 ist darauf konzipiert, die Parametermenge (2.1) so zu wählen, dass nur in Ausnahmefällen fehlerhafte Durchläufe zustande kommen. Bei einem fehlerhaften Testlauf kann mit T_e festgelegt, wie viele erneute Versuch bei gleichem Parametervektor $\boldsymbol{x}_p$ unternommen werden dürfen, bis das Ergebnis des Testlaufes übernommen wird.

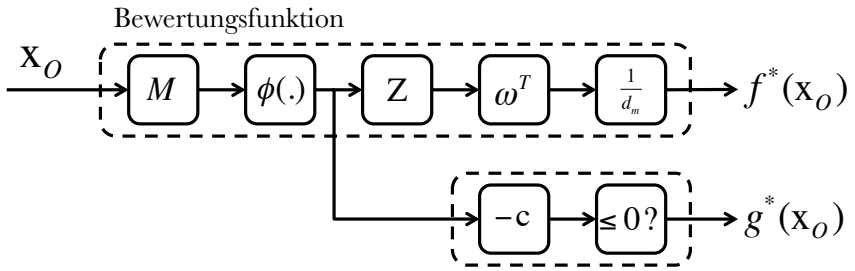


Abbildung 2.8: Blockschaltbild der Bewertungs- und Fehlerfunktion eines Testdurchlaufes. Oben ist die kontinuierliche Bewertungsfunktion abgebildet, darunter die diskrete gut/schlecht Klassifizierungsfunktion.

Kapitel 3

Automatisiertes Testen bzw. Optimieren

3.1 Formulierung des Optimierungsproblems

Durch das in Kapitel 2, Abschnitt 2.1 eingeführte Konzept wird das automatisierte Testen und Optimieren von der spezifischen Roboterapplikation abstrahiert.
Dadurch ist eine vollständig getrennte Betrachtung zwischen der zu testenden Anwendung und der Optimierung möglich. Die Optimierung lässt sich als klassisches, globales Black-Box Optimierungsproblem betrachten:

$$y_{opt} = \underset{\boldsymbol{x}}{\operatorname{argmin}} f(\boldsymbol{x}),\ \boldsymbol{x} \in \mathcal{P}_p.$$

Die Bewertungsfunktion 2.2 besitzt folgende wesentliche Eigenschaften:

- f ist nichtlinear, insbesondere existieren i.A. lokale Optima.
- Der Such- bzw. Parameterraum ist durch ein Hyperrechteck (Definition 2.1) definiert.
- Die Funktion kann mit einer stochastischen Ungenauigkeit behaftet sein. Als praktisches Beispiel stelle man sich Sensorrauschen oder zufallsbasierte Planungsalgorithmen vor.
- Die Funktion f besitzt keine formale Glattheit.
- Ein Funktionsaufruf ist „teuer“, da jeweils eine komplette Ausführung der Applikation stattfinden muss, welche typischerweise Ausführungszeiten im Minutenbereich benötigt.

Das zugrundeliegende Optimierungsverfahren ist der zentrale Bestandteil des hier vorgestellten Test- und Optimierungssystems. Daher werden in den folgenden Abschnitten die Grundlagen für dieses hergeleitet, sowie dessen praktische Umsetzung erläutert und verifiziert.

3.2 Globale Optimierung auf Basis maschinellen Lernens

Ein besonders wichtiger Aspekt ist die Wahl des zugrundeliegenden Optimierungsverfahrens. Genetische Algorithmen scheinen aufgrund der Analogie zu der natürlichen Evolution gut geeignet zu sein. Bei der Umsetzung jedoch stößt man hierbei sehr schnell auf das praktische Problem, dass ein Applikationsdurchlauf bis zu einer Minute oder länger in Anspruch nimmt und damit eine einzelne Iterationen mit einer Population von z.B. 60 Individuen ohne Parallelisierung ca. eine Stunde in Anspruch nehmen würde und damit 80 oder mehr Iterationen sehr zeitintensiv ausfallen. Um dieses Problem zu umgehen und damit dem Konzept einen praktikablen Nutzen zu verleihen, wurde eine Optimierung auf Basis maschinellen Lernens verwendet.
Hierbei wird anhand bisheriger Applikationsdurchläufe versucht, die Bewertungsfunktion zu erlernen bzw. zu interpolieren. Auf dieser Basis lässt sich dann vorhersagen, wo vermutlich ein Optimum liegen könnte. Zudem lassen sich Aussagen darüber treffen, mit welcher Sicherheit solche Situationen vorhergesagt werden können. Diese beiden Tatsachen lassen sich zu einem höchst effizienten Optimierungsverfahren bezüglich der Anzahl an Testläufen kombinieren. Darüber hinaus ist es durch das modulare Konzept der Bewertungsfunktion anhand der Zustände (vgl. 2.4) möglich, mit einem bestehenden Datensatz an Testläufen eine Optimierung bzgl. weiteren Bewertungsfunktionen ohne zusätzliche Testläufe durchzuführen. Andererseits kann durch dieses Optimierungskonzept ein solcher Datensatz als Ausgangspunkt weiterer Testläufe bzgl. anderer Bewertungsfunktionen verwendet werden.

3.2.1 Modell

Die Bewertungsfunktion wird als stochastischer Prozess angenommen. Auf Basis dieser Annahme wird der zugrunde liegende Mechanismus der Bewertungsfunktion als Gauss'scher Prozess anhand von Testdaten identifiziert. Allgemeine Vorteile einer solchen Lernmethode sind ohne formale Beweise:

- Kompensation von „Clustering"-Effekten: Naheliegende Punkte im Parameterraum werden eher als einzelne Punkte betrachtet.
- Gewisses Maß an numerischer Robustheit, welches für die allgemeine Anwendbarkeit im Kontext dieser Arbeit ausreichend ist.
- Erweiterbar auf Verarbeitung verrauschter Datensätze.

Die Herleitung basiert im Wesentlichen auf [Jon01], da diese durch eine vergleichsweise intuitive Herangehensweise motiviert ist. Sie trägt wesentlich zum Verständnis und zur Anwendbarkeit des Gesamtsystems bei und wird daher ausführlich formuliert.

Wir beginnen mit grundsätzlichen Annahmen, aus denen wir eine statistische Formulierung herleiten können:

- Ohne eine tatsächliche Funktionsauswertung y_i an einer Stelle $\boldsymbol{x}_i$ ist der Funktionswert $y_i = f(\boldsymbol{x}_i)$ „unsicher". Diese Unsicherheit sei normalverteilt mit Mittelwert μ und Streuung σ.

- D.h. die Funktion $f(\boldsymbol{x})$ hat einen typischen Funktionswert μ und variiert mit großer Wahrscheinlichkeit um $[\mu - 3\sigma, \mu + 3\sigma]$.
- Des Weiteren soll die Funktion ein gewisses Maß an Struktur in Form einer Glattheit aufweisen, d.h. nahegelegene Funktionsauswertungen y_i und y_j an den Stellen $\boldsymbol{x}_i$ und $\boldsymbol{x}_j$ mit $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ stehen in einer gewissen Beziehung zueinander.

Anders formuliert gilt:

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \text{ „klein “} \leftrightarrow \text{ Hohe Korrelation zwischen } y_i \text{ und } y_j.$$

Eine mathematische Formulierung hierfür liefert das sogenannte „Krigingmodell“ nach [Jon01]:

$$Corr_{ij} = Corr[Y(\boldsymbol{x}_i), Y(\boldsymbol{x}_j)] = e^{-\sum_{l=0}^{d_p} \theta_l |x_{i_l} - x_{j_l}|^{p_l}} \tag{3.1}$$

Diese freien Parameter werden in der Literatur auch „Hyperparameter“ genannt. Es ist wichtig, den Einfluss der Parameter θ_l und p_l in (3.1) zu verstehen, da diese das Modell charakterisieren.
Ist $\boldsymbol{x}_i \equiv \boldsymbol{x}_j$ so soll die Korrelation maximal sein. Gleichung (3.1) wird in diesem Fall gleich 1. Entgegengesetzt betrachtet gilt: $\lim_{\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \to \infty} Corr[Y(\boldsymbol{x}_i), Y(\boldsymbol{x}_j)] \to 0$, d.h. „unendlich“ weit entfernte Funktionsauswertungen sind unkorreliert.
Der Parameter p_l ist ein Maß für die „Glattheit“ entlang einer Dimension des Parameterraums. Für ein p_l nahe 0 ist eine Modellierung hochgradig nicht-stetiger Funktionen möglich. Für ein p_l nahe 2 werden unendlich oft differenzierbare Funktionen modelliert. Abbildung 3.1 zeigt den Einfluss von p_l entlang einer Dimension. Der Parameter θ_l kann als Maß für die „Aktivität“ des stochastischen Prozessen entlang der l-ten Dimension interpretiert werden. Für große θ_l nimmt die Korrelation bei Verlassen der Umgebung eines Datenpunktes schneller ab als bei kleinen Werten für θ_l.

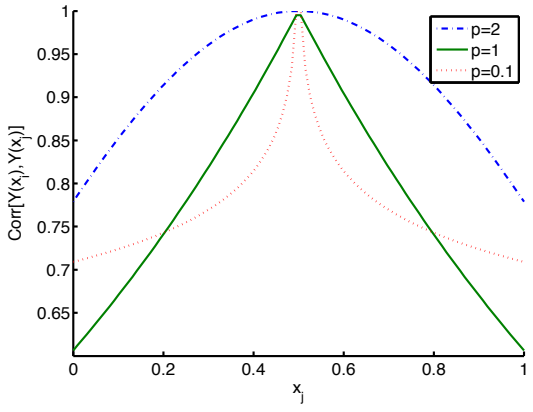


Abbildung 3.1: $Corr[Y(\boldsymbol{x}_i), Y(\boldsymbol{x}_j)]$ für $d_p = 1$, $\boldsymbol{x}_i = 0,5$ und $p = 0, 1$, $p = 1$, $p = 2$.

Fügt man alle „unsicheren“ $Y(\boldsymbol{x}_i)$ in einem Vektor

$$\mathbf{Y} = \begin{pmatrix} Y(x_1) \\ .. \\ .. \\ Y(x_n) \end{pmatrix} \tag{3.2}$$

zusammen, so hat dieser einen Mittelwert $\mu\mathbf{1}$, mit $\mathbf{1}$ als Spaltenvektor, besetzt mit n-Einsen und einer Kovarianzmatrix, bestehend aus $\sigma^2\mathbf{R}$ wobei $R^*_{ij} = Corr_{ij}$ und $\mathbf{R} = \mathbf{R}^* + \mathbf{I}\sigma_D^2$. Dabei beschreibt σ_D^2 die Unsicherheit der Daten durch dessen zugrundeliegenden stochastischen Prozess. Der Vektor $\mathbf{Y}$ ist abhängig von $\boldsymbol{\theta}, \boldsymbol{p}, \mu$ und σ. Er stellt eine statistische Beschreibung der Funktion f als Gauss'schen Prozess dar.

3.2.2 Berechnung der Hyperparameter

Die Parameter $\boldsymbol{\theta}, \boldsymbol{p}$ lassen eines anhand eines Datensatzes $[(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)]^T$ mittels der Maximum-Likelihood Methode durch eine unterlagerte Optimierung effizient bestimmen. Maximieren von

$$\mathcal{L}(\mu, \sigma, \boldsymbol{\theta}, \boldsymbol{p}) = \frac{1}{(2\pi)^{\frac{n}{2}}(\sigma^2)^{\frac{n}{2}} \mid \mathbf{R} \mid^{\frac{1}{2}}} e^{\frac{-(\mathbf{y}-\mathbf{1}\mu)^T\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2}} \tag{3.3}$$

ist unter Vernachlässigung konstanter Terme nach [Jon01] äquivalent zu minimieren von

$$log\mathcal{L}(\mu, \sigma, \boldsymbol{\theta}, \boldsymbol{p}) = -\frac{n}{2}\log(\sigma^2) - \frac{1}{2}\log(\mid \mathbf{R} \mid) - \frac{-(\mathbf{y}-\mathbf{1}\mu)^T\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2}. \tag{3.4}$$

Mit $\frac{\partial log\mathcal{L}}{\partial \sigma^2} = 0$ und $\frac{\partial log\mathcal{L}}{\partial \mu} = 0$ folgen optimale Parameter für μ und σ:

$$\hat{\mu} = \frac{\mathbf{1}^T\mathbf{R}^{-1}\mathbf{y}}{\mathbf{1}^T\mathbf{R}^{-1}\mathbf{1}} \tag{3.5}$$

$$\hat{\sigma}^2 = \frac{-(\mathbf{y}-\mathbf{1}\mu)^T\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{n} \tag{3.6}$$

welche eingesetzt in (3.4) die Eliminierung der freien Parameter μ und σ^2 zu einer konzentrierten Form von Gleichung (3.4)

$$klog\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{p}) = -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(\mid \mathbf{R} \mid) \tag{3.7}$$

erlaubt. Durch (3.7) reduziert sich die Modellidentifikation auf eine Optimierung von $2d_p$-Parametern: $\boldsymbol{\theta}$ und $\boldsymbol{p}$. Bei Festlegen von $p \equiv 2$ (Analogon zu radialen Basisfunktionen) sind lediglich d_p Parameter für eine Identifikation zu optimieren. Zur Bestimmung dieser Parameter wird der L-BFGS-B Algorithmus verwendet.[1]

L-BFGS-B Algorithmus

Im Folgenden werden die Grundzüge des L-BFGS-B Algorithmus zur Minimierung von (3.7) erklärt. Für eine genauere Beschreibung siehe [BLNZ95]. Unter der Annahme, dass die zu optimierende Zielfunktion $f(\boldsymbol{x})$ differenzierbar ist, lässt sich diese im

[1] In [Ras06] wird aufgezeigt, dass der teuerste Teil aus Gleichung 3.7 das Invertieren von $\mathbf{R}$ mit einer Laufzeit von $\mathcal{O}(n^3)$ ist. Durch partielles Ableiten von (3.4) kann gezeigt werden, dass auf Basis des berechneten $\mathbf{R}^{-1}$ die Berechnung des Gradienten lediglich eine Laufzeit von $\mathcal{O}(n^2)$ pro Hyperparameter aufweist. Damit wäre eine Effizienzsteigerung bei der Berechnung der Hyperparamter möglich, was im Rahmen dieser Arbeit nicht notwendig ist, da n maximal 100 - 200 beträgt.

Bereich eines Punktes $(\boldsymbol{x}_t, f_t)$ im Parameterraum anhand einer Taylorreihe mittels finiter Differenzen näherungsweise zu

$$m_t(\boldsymbol{x}) = f_t + \nabla f_t^T(\boldsymbol{x} - \boldsymbol{x}_t) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_t)^T \mathbf{H}_t(\boldsymbol{x} - \boldsymbol{x}_t) \tag{3.8}$$

berechnen, mit approximiertem Gradient ∇f_t und Hessematrix $\mathbf{H}_t$. Durch Nullsetzen von (3.8) ergibt sich das Optimum der durch die Taylorreihe modellierten Zielfunktion zu

$$\boldsymbol{x}_{opt} = \boldsymbol{x}_t - \alpha \mathbf{H}^{-1} \nabla f_t \tag{3.9}$$

mit α als „Vertrauensparameter“ bzgl. der Approximation durch die Taylorreihe. In dem L-BFGS-B Algorithmus nach [BLNZ95] wird $\mathbf{H}^{-1}$ so bestimmt, dass diese stets positiv definit ist. Damit bewegt man sich für ein $\alpha > 0$ zwangsläufig in eine absteigende Richtung. Die Wahl von α basiert auf dem „line search“ Verfahren, bei dem nach [Tou14] ganz Allgemein das Problem $\text{argmin}_{\alpha \geq 0} f(\boldsymbol{x} + \alpha \boldsymbol{d})$ für eine beliebige Richtung $\boldsymbol{d}$ ($\widehat{=}\mathbf{H}^{-1}\nabla f_t$) gelöst wird. Hierzu wird α (mit $\alpha_0 = 1$) um ϱ_α iterativ verkleinert, solange

$$f(\boldsymbol{x} + \alpha d) > f(\boldsymbol{x}) + \varrho_{ls} \nabla f(\boldsymbol{x})^T (\alpha \boldsymbol{d}) \tag{3.10}$$

erfüllt ist. Dabei beschreibt ϱ_{ls} die minimale gewünschte Minimierung von $f(\boldsymbol{x})$.
In einem hochdimensionalen Parameterraum ist für unsere Zwecke die Berechnung von $\mathbf{H}$ durch finite Differenzen anhand (3.17) aufwändig. Noch aufwändiger jedoch ist die Invertierung von $\mathbf{H}$. Hier liefert der BFGS Algorithmus eine Methode zur Approximation von $\mathbf{H}$ anhand bisheriger Funktionsauswertungen. Nach [Tou14] lässt sich die Grundidee herleiten, indem man sich ausgehend von zwei Gradientenapproximationen $\nabla f(\boldsymbol{x}_1)$ und $\nabla f(\boldsymbol{x}_2)$ an den Stellen $\boldsymbol{x}_1$ und $\boldsymbol{x}_2$ die Hilfsdifferenzen $\boldsymbol{y} = \nabla f(\boldsymbol{x}_2) - \nabla f(\boldsymbol{x}_1)$ und $\boldsymbol{\delta} = \boldsymbol{x}_2 - \boldsymbol{x}_1$ bildet und durch Auflösen von $\mathbf{H}\boldsymbol{\delta} \stackrel{!}{=} \boldsymbol{y}$ nach $\mathbf{H}$ bzw. $\mathbf{H}^{-1}$

$$\mathbf{H} = \frac{\boldsymbol{y}\boldsymbol{y}^T}{\boldsymbol{y}^T\boldsymbol{\delta}} \tag{3.11}$$

bzw.

$$\mathbf{H}^{-1} = \frac{\boldsymbol{\delta}\boldsymbol{\delta}^T}{\boldsymbol{\delta}^T\boldsymbol{y}} \tag{3.12}$$

erhält. In [BLNZ95] ist eine Erweiterung des BFGS Algorithmus gezeigt, die nur begrenzt viele Punkte für die Berechnung von $\mathbf{H}$ im Programmspeicher behält. Für die Bestimmung der Hyperparameter müssen die zu optimierende Parameter in einer einfachen Grenze $\boldsymbol{x}_{min}, \boldsymbol{x}_{max}$ liegen. Dies wird als zusätzliche Bedingung bei der Bestimmung von α berücksichtigt. Der resultierende L-BFGS-B Algorithmus ist in Algorithmus 2 skizziert.

3.2.3 Vorhersage von Funktionswerten

Nun leiten wir her, auf welche Weise anhand des identifizierten Modells eine Prädiktion an einem Punkt $y^* = f(\boldsymbol{x}^*)$ durchgeführt werden kann. Anders formuliert wollen wir zu einem gegebenen $\boldsymbol{x}^*$ den nach unserem Modell wahrscheinlichsten zugehörigen Wert y^* berechnen. Hierzu erweitern wir den bestehenden Datensatz um $\boldsymbol{x}^*$ als festen

Algorithm 2 L-BFGS-B Algorithmus (skizziert)

procedure L-BFGS-B($\boldsymbol{x}_0$,$\boldsymbol{x}_{min}$,$\boldsymbol{x}_{max}$,ϵ)
  $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
  Initialisiere $\mathbf{H}$
  while $||\boldsymbol{\delta}||_\infty \geq \epsilon$ **do**
    $\boldsymbol{d} = -\alpha \mathbf{H}^{-1} \nabla f_t$
    Berechne α mit modifizierter line search respektive $\boldsymbol{x}_{min}$ und $\boldsymbol{x}_{max}$
    $\boldsymbol{\delta} \leftarrow \alpha \boldsymbol{d}$
    $\nabla f_1 =$ mit finite Differenzen approximierter Gradient an $\boldsymbol{x}$
    $\nabla f_2 =$ mit finite Differenzen approximierter Gradient an $\boldsymbol{x} + \delta$
    $\boldsymbol{y} = \nabla f_2 - \nabla f_2$
    $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{\delta}$
    Berechne $\mathbf{H}^{-1}$
  end while
  return $\boldsymbol{x}$
end procedure

und y^* als freien Parameter, um bei unveränderten Hyperparametern die Gleichung (3.3) bzgl. y^* zu optimieren. Auf diese Weise erhalten wir den wahrscheinlichsten Wert von y^* für ein gegebenes $\boldsymbol{x}^*$ anhand unseres Modells.
Die durch den erweiterten Datensatz $(\boldsymbol{x}^*, y^*)$ neu entstandenen Gleichungen lauten

$$\tilde{\mathbf{y}} = (\mathbf{y}, y^*)^T \tag{3.13}$$

$$\tilde{\mathbf{R}} = \begin{pmatrix} \mathbf{R} & \mathbf{r} \\ \mathbf{r}^T & 1 \end{pmatrix}, \mathbf{r} = \begin{pmatrix} Corr[Y(x^*), Y(x_1)] \\ Corr[Y(x^*), Y(x_2)] \\ .. \\ Corr[Y(x^*), Y(x_n)] \end{pmatrix}. \tag{3.14}$$

Setzt man diese in (3.4) ein, so verbleibt nur ein Term, welcher von y^* abhängt:

$$log\mathcal{L}(y^*) = -\frac{-(\tilde{\mathbf{y}} - \mathbf{1}\mu)^T \tilde{\mathbf{R}}^{-1} (\tilde{\mathbf{y}} - \mathbf{1}\mu)}{2\hat{\sigma}^2} + (..) \tag{3.15}$$

Mittels der expliziten Formel für das Invertieren zusammengesetzter Matrizen nach [The83] lässt sich $\tilde{\mathbf{R}}^{-1}$ in Abhängigkeit von $\mathbf{R}^{-1}$ und $\mathbf{r}$ explizit berechnen, in (3.15) einsetzen und nach y^* ableiten, um für y^* eine analytische Optimalitätsbedingung durch Nullsetzen zu erhalten.
Hierzu löst man

$$-\frac{1}{\hat{\sigma}^2(1 - \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r})}(y^* - \hat{\mu}) + \frac{\mathbf{r}^T \mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu})}{\hat{\sigma}^2(1 - \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r})} = 0 \tag{3.16}$$

nach y^* auf und erhält letztendlich folgende Vorhersageformel:

$$\hat{y}(\boldsymbol{x}^*) = \hat{\mu} + \mathbf{r}^T \mathbf{R}^{-1}(\mathbf{y} - 1\tilde{\mu}). \tag{3.17}$$

3.2.4 Modellfehler

Um den Fehler einer Vorhersage $y^* = f(\boldsymbol{x}^*)$ abschätzen zu können, betrachten wir, wie bei der Vorhersage selbst, die Gleichung (3.4) noch etwas genauer: Stellen wir uns vor, (3.4) sei sehr flach. Damit ist offensichtlich, dass das gefundene Optimum und damit y^* mit relativ hoher Wahrscheinlichkeit auch einen anderen Wert annehmen könnte. Ist (3.4) um $\boldsymbol{x}^*$ jedoch sehr stark gekrümmt, lässt dies auf eine hohe Sicherheit bzw. Eindeutigkeit von y^* schließen vgl. Abbildung 3.2.

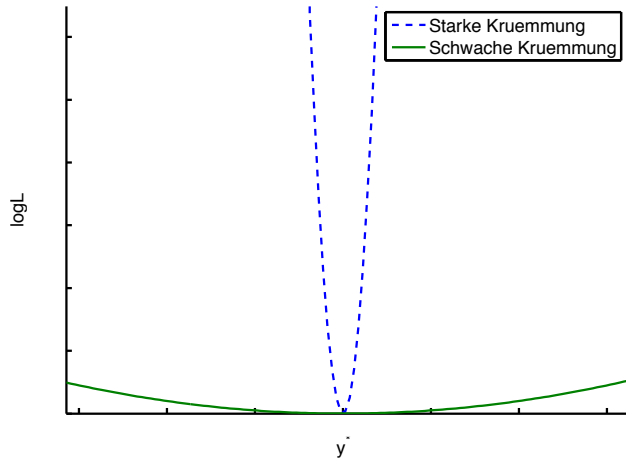


Abbildung 3.2: Bei schwacher Krümmung ist optimales y^* weniger eindeutig, als im Vergleich zu einer starken Krümmung.

D.h. der prädizierte Modellfehler sollte umgekehrt proportional zur Krümmung von (3.4) sein:

$$\epsilon(\boldsymbol{x}^*) = \hat{\sigma}^2(1 - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{r}). \tag{3.18}$$

Dies entspricht genau der Herleitung in [Sch98]. Nach [Sas02] ergibt sich der genaue Fehler durch Einbeziehung der Ungenauigkeit von $\hat{\mu}$ zu der Varianz des Modells bei $\boldsymbol{x}^*$ und lautet:

$$\sigma^2(\boldsymbol{x}^*) = \hat{\sigma}^2 \left(1 - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{r} + \frac{(1 - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{r})^2}{\mathbf{1}^T\mathbf{R}^{-1}\mathbf{1}}\right). \tag{3.19}$$

Die Berechenbarkeit der Varianz (3.19) ist der entscheidende Vorteil, der die Verwendung stochastischer Modelle für unsere Zwecke motiviert. Mittels $\sigma(x^*)$ lässt sich eine Suchheuristik für eine „gute“ Wahl neuer Funktionsparameter formulieren.[2]

3.2.5 Heuristiken

Der Zweck einer Heuristik besteht darin auf Basis des vorhanden Wissens über die Bewertungsfunktion, abgebildet durch den Gauss'schen Prozess, eine Aussage über

[2]In der Literatur werden für globale Optimierungsverfahren auf Basis Gauss'scher Prozesse oft mittelwertfreie Datensätze angenommen. Man gelangt mittels der Z-Transformation von obiger Darstellung zu Darstellungen wie z.B. in [Ras06].

Algorithm 3 Optimierung Testsystem mit UCB-Heuristik

procedure GP-UCB($\boldsymbol{D}_0, \mathcal{P}_p, \boldsymbol{p}_{min}, Testlauf, f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}}, g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}}, T, T_e$)
  for $t = 1, t \leq T$ **do**
    $\boldsymbol{d}$ = skalierter Datensatz $\boldsymbol{D}_{t-1}$ ▷ Für diesen Iterationsschritt.
    $\text{argmin}_{\boldsymbol{\theta}>0;1,99>\boldsymbol{p}\geq\boldsymbol{p}_{min}}\, klog\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{p})$ (3.7) ▷ Interpoliere.
    Wähle $\boldsymbol{x}_t = \text{argmin}_{\boldsymbol{x}\in\mathcal{P}_p}\, h_t(\boldsymbol{x})$ ▷ Liefert $\boldsymbol{x}_t$.
    for $t = 1, t \leq T_e$ **do**
      $\boldsymbol{x}_o = Testlauf(\boldsymbol{x}_t)$ ▷ $\boldsymbol{x}_t$ entspricht einem $\boldsymbol{x}_p$ aus (2.1).
      Funktionsauswertung $y_t = f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}}(\boldsymbol{x}_o)$
      if $g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}}(\boldsymbol{x}_o)$ **then**
        Beende diese Schleife
      end if
    end for
    Erweitere $\boldsymbol{D}_{t-1}$ um $(\boldsymbol{x}_t, \boldsymbol{x}_o, y_t)$ zu $\boldsymbol{D}_t$
  end for
  Wähle $\boldsymbol{x}^* = \text{argmin}_{x\in\mathcal{P}_p}\, \hat{y}(\boldsymbol{x}^*)$ (3.17) ▷ Suche Optimum in Modell.
  $\hat{y}^* = \hat{y}(\boldsymbol{x}^*)$ (3.17) ▷ Berechne unskaliertes, interp. Optimum.
  return $[\boldsymbol{x}^*, f(\boldsymbol{x}^*), \hat{y}^*, \boldsymbol{D}_T]$ ▷ Liefere gefundenes Optimum und erweiterten Datensatz.
end procedure

das Potential bisher ungetesteter Gebiete im Parameterraum zu treffen. Hierbei existiert ein „Trade-off“ zwischen der Exploration bisher unerschlossener Gebiete und eher bekannten Gebieten, die aufgrund einer hohen Güte bzgl. der Bewertungsfunktion vielversprechend für die weitere Suche erscheinen.
In der Literatur sind zahlreiche Heuristiken für die globale Optimierung mittels Gauss'scher Prozesse zu finden. Im Rahmen dieser Arbeit wurden zwei Ansätze implementierung und evaluiert.

Gaussian Process Upper Confidence Bound

Die „Gaussian Process Upper Confidence Bound“ ist nach [SKKS12] als zu minimierende Funktion durch

$$h_t(\boldsymbol{x}^{**}) = \hat{y}(\boldsymbol{x}^{**}) - \sqrt{\beta_t}\sigma(\boldsymbol{x}^{**}) \tag{3.20}$$

mit $\boldsymbol{x}^{**}$ als nächsten Testpunkt im Parameterraum und β_t als „Designparameter“, gegeben. Durch β_t ist in dieser Heuristik der eingangs genannte „Trade-off“ explizit wählbar. Nach [SKKS12] wurde

$$\beta_t = 2\log\left(\frac{t^2 2\pi^2}{3\delta}\right) + 2d_p \log\left(t^2 d_p \sqrt{\log\left(\frac{4d_p}{\delta}\right)}\right) \tag{3.21}$$

mit $\delta = 0,1$ gewählt. Der resultierende Algorithmus für eine Optimierung des Testsystems nach Kapitel 2 durch die GP-UCB mit einem Anfangsdatensatz $\boldsymbol{D}_t$ und T Iterationen ist in Algorithmus 3 skizziert.

Einstufen-Heuristik

Durch [Jon01] motiviert, wurde zusätzliche eine weniger etablierte Heuristik als Alternative zu der GP-UCB verwendet. Diese bestimmt in einem unterlagerten Optimierungsschritt sowohl die Hyperparameter wie auch den neuen Auswertungspunkt $\boldsymbol{x}^*$ gleichzeitig. Das Verfahren wurde in [NHQ09] erfolgreich getestet.
Die zugrundeliegende Idee[Jon01] basiert darauf, dass man ein Optimum y^*_{opt} rät. Anschließend erweitert man das Modell aus Abschnitt 3.2.1 um diese zusätzliche Information y_{opt} mit einem noch unbekannten korrespondierenden Parameter $\boldsymbol{x}^*$. Aufgrund der zusätzlichen Information für den Wert des angenommenen Optimums ist die Suche nach $\boldsymbol{x}^*$ insofern verbessert, dass der integrative Optimierungsschritt implizit nach der plausibelsten Stelle im Parameterraum $\mathcal{P}_p$ für das geratene Optimum auf Basis bestehender Daten $\boldsymbol{D}_{t-1}$ sucht.
Ein Vorteil ist hierbei, dass kein expliziter „Designparameter“ wie β_t aus (3.20) für die Heuristik gewählt werden muss. Implizit ist der geratene Wert für y^*_{opt} jedoch eine Analogie zu einem solchen Parameter.
Falls keine Vorstellung des Optimums vorhanden ist, gibt es nach [Gut01] ein Verfahren für die Berechnung von fünf potentiellen y^*_{opt} auf Basis des Modells, die das Minimum der aktuellen Interpolation unterbieten, gegeben durch

$$y^*_{opt_k} = y_t - w_k(\max_i f(\boldsymbol{x}_i) - y_t),\ w_k = 1 - \frac{k}{N^2},\ k = 0, 1, 2, 3, 4. \tag{3.22}$$

Hierbei ist y_t das auf Basis der Interpolation gefundene Optimum und $\max_i f(\boldsymbol{x}_i)$ der größte Funktionswert innerhalb des aktuellen Datensatzes. Die sich so ergebenden Werte variieren zwischen kleinen Abweichungen zu y_t, die lokalen Optima entsprechen, bis zu größeren Abweichungen für y_t für das gobale Optimum. Damit ist es möglich den unterlagerten Optimierungsschritt der Heuristik parallel mit einem Tupel von y^*_{opt} durchzuführen.

Die erweiterten Elemente mit $\mathbf{m}$ als bedingten Mittelwert

$$\mathbf{m} = \mathbf{1}\mu + \mathbf{r}(y^*_{opt} - \mu) \tag{3.23}$$

und der bedingten Korrelationsmatrix $\mathbf{C}$ gegeben durch

$$\mathbf{C} = \mathbf{R} - \mathbf{r}\mathbf{r}^T \tag{3.24}$$

ergeben analog hergeleitet zu Abschnitt 3.2.2 die zu optimierende Funktion

$$kClog\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{p}, \boldsymbol{x}^*) = -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(\mid \mathbf{C} \mid) \tag{3.25}$$

mit

$$\hat{\mu}_C = \frac{(\mathbf{1} - \mathbf{r})^T\mathbf{C}^{-1}(\mathbf{y} - \mathbf{r}y^*_{opt})}{(\mathbf{1} - \mathbf{r})^T\mathbf{C}^{-1}(\mathbf{1} - \mathbf{r})} \tag{3.26}$$

und

$$\hat{\sigma}^2_C = \frac{(\mathbf{y} - \mathbf{r}y^*_{opt} - (\mathbf{1} - \mathbf{r})\hat{\mu_C})^T C^{-1}(\mathbf{y} - \mathbf{r}y^*_{opt} - (\mathbf{1} - \mathbf{r})\hat{\mu_C})}{n}. \tag{3.27}$$

Algorithm 4 Optimierung mit Einstufen-Heuristik

1: **procedure** GP-ONE($\boldsymbol{D}_0, \mathcal{P}_p, \boldsymbol{p}_{min}, Testlauf, f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}}, g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}}, T, T_e$)
2: $\boldsymbol{d}$ = skalierter Datensatz $\boldsymbol{D}_{t-1}$ ▷ Für anfängliche Interpolation.
3: $\text{argmin}_{\boldsymbol{\theta}>0;1{,}99>\boldsymbol{p}\geq\boldsymbol{p}_{min}}\, klog\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{p})$ (3.7) ▷ Interpoliere.
4: **for** $t = 1, t \leq T$ **do**
5: $\boldsymbol{d}$ = skalierter Datensatz $\boldsymbol{D}_{t-1}$ ▷ Für diesen Iterationsschritt.
6: Berechne $y^*_{opt,t}$ (3.22) ▷ Schätze Optimum.
7: $\text{argmin}_{\boldsymbol{\theta}>0;1{,}99>\boldsymbol{p}\geq\boldsymbol{p}_{min};\boldsymbol{x}_t\in\mathcal{P}_p}\, kClog\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{p}, \boldsymbol{x}^*)$ (3.25) ▷ Liefert $\boldsymbol{x}_t$.
8: **for** $t = 1, t \leq T_e$ **do**
9: $\boldsymbol{x}_o = Testlauf(\boldsymbol{x}_t)$ ▷ $\boldsymbol{x}_t$ entspricht einem $\boldsymbol{x}_p$ aus (2.1).
10: Funktionsauswertung $y_t = f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}}(\boldsymbol{x}_o)$
11: **if** $g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}}(\boldsymbol{x}_o)$ **then**
12: Beende diese Schleife
13: **end if**
14: **end for**
15: Erweitere $\boldsymbol{D}_{t-1}$ um $(\boldsymbol{x}_t, \boldsymbol{x}_o, y_t)$ zu $\boldsymbol{D}_t$
16: **end for**
17: Wähle $\boldsymbol{x}^* = \text{argmin}_{x\in\mathcal{P}_p}\, \hat{y}(\boldsymbol{x}^*)$ (3.17) ▷ Suche Optimum in Modell.
18: $\hat{y}^* = \hat{y}(\boldsymbol{x}^*)$ (3.17) ▷ Berechne unskaliertes, interp. Optimum.
19: **return** $[\boldsymbol{x}^*, f(\boldsymbol{x}^*), \hat{y}^*, \boldsymbol{D}_T]$ ▷ Liefere gefundenes Optimum und erweiterten Datensatz.
20: **end procedure**

Der entsprechende Algorithmus ist in Algorithmus 4 gezeigt. Im Vergleich zu der GP-UCB Heuristik wird bei der Einstufen-Heuristik der unterlagerte Optimierungsschritt wesentlich komplexer, da der neue Parameterraum neben den Hyperparametern $\boldsymbol{\theta}$ und $\boldsymbol{p}$ zusätzlich $\boldsymbol{x}^*$ beinhaltet. In [NHQ09] wird eine dreiteilige Strategie für den unterlagerten Optimierungsschritt vorgeschlagen, die in Algorithmus 5 aufgezeigt ist. Obwohl diese Annahme in [NHQ09] nicht getroffen wurde, ist zu bemerken, dass das Setzen von $\boldsymbol{x}$ zu einem univarianten Werte für alle Dimensionen nur bei einem identischen Parameterraum entlang allen Dimensionen d_p sinnvoll ist, da sonst ungültige Startwerte für die Suche nach $\boldsymbol{x}^*_p$ gewählt werden können.

Genetische Algorithmen

Die Optimierungsprobleme bei der UCB-Heuristik in Algorithmus 3, Zeile 5 und bei der Einstufen-Heuristik in Algorithmus 5, Zeile 6 sowie bei der Optimumsuche im Modell ($\text{argmin}_{x\in\mathcal{P}_p}\, \hat{y}(\boldsymbol{x}^*)$) besitzen im Allgemeinen viele lokale Optima und sind daher nicht mit herkömmlichen Gradientenabstiegsverfahren lösbar. Deshalb wurden für diese Teilprobleme Genetische Algorithmen verwendet, die sich am Vorbild der biologischen Evolution orientieren und damit der Klasse der Evolutionären Algorithmen zuzuordnen sind.
Mögliche Lösungen eines Problems werden als Individuen einer Population modelliert, so dass die genetischen Operationen Selektion, Rekombination und Mutation angewendet werden können. Die Selektion basiert auf einer Fitnessbewertung einzelner Individuen, so dass sich die Population über mehrere Generationen hinweg

Algorithm 5 Unterlagerte Optimierung bei Einstufen-Heuristik

procedure UNTERLAGERTE ONE-STEP OPTIMIERUNG($\mathcal{P}_p, \boldsymbol{p}_{min}, \mathbf{C}_t$)
  Setze $\boldsymbol{\theta}$ und $\boldsymbol{x}_t^*$ univariant für alle Dimensionen zu $\boldsymbol{\theta}_{uni}$ und $\boldsymbol{x}_{t,uni}^*$.
  $\text{argmin}_{\boldsymbol{\theta}_{uni};\boldsymbol{x}_{t,uni}^* \in \mathcal{P}_p} kClog\mathcal{L}(\boldsymbol{\theta}_{uni}, 1, 99. * \mathbf{1}, \boldsymbol{x}_{t,uni}^*)$ (3.25) ▷ 2-dimensionales Optimierungsproblem, liefert Startwert $\boldsymbol{x}^*$ und $\boldsymbol{\theta}$.
  Setze $\boldsymbol{\theta}$ und $\boldsymbol{p}$ univariant für alle Dimensionen zu $\boldsymbol{\theta}_{uni}$ und $\boldsymbol{p}_{uni}$.
  $\text{argmin}_{\boldsymbol{\theta}_{uni}>0;1,99>\boldsymbol{p}_{uni}\geq\boldsymbol{p}_{min}} kClog\mathcal{L}(\boldsymbol{\theta}_{uni}, \boldsymbol{p}_{uni}, \boldsymbol{x}_{t,uni}^*)$ (3.25) ▷ 2-dimensionales Optimierungsproblem, liefert Startwert $\boldsymbol{\theta}$ und $\boldsymbol{p}$.
  $\text{argmin}_{\boldsymbol{\theta}>0;1,99>\boldsymbol{p}\geq\boldsymbol{p}_{min};\boldsymbol{x}_t\in\mathcal{P}_p} kClog\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{p}, \boldsymbol{x}^*)$ (3.25) ▷ Löse komplettes Optimierungsproblem auf Basis der gefundenen Startewerte durch univariante Teiloptimierungen.
  return $\boldsymbol{\theta}$,$\boldsymbol{p}$ und $\boldsymbol{x}_t^*$
end procedure

einem über die Fitnessfunktion definierten Optimum annähert. Algorithmus 6 zeigt den entsprechenden Pseudocode in Anlehnung an [RN05].

Algorithm 6 Genetischer Algorithmus

procedure GENETISCHER ALGORITHMUS($Population, FitnessFkt$)
  while Individuum fit genug oder maximale Anzahl an Generationen erreicht **do**
    $neue\_Population \leftarrow$ leere Menge
    for $i = 1, i \leq len(Population), t = t + 1$ **do**
      $x \leftarrow ZufaelligeAuswahl(Population, FitnessFkt)$
      $y \leftarrow ZufaelligeAuswahl(Population, FitnessFkt)$
      $kind \leftarrow Rekombination(x, y)$
      if *kleine Mutationswahrscheinlichkeit* **then**
        $kind \leftarrow Mutation(kind)$
      end if
      Füge *kind* zu $neue\_Population$ hinzu
    end for
    return Bestes Individuum in Population gemäß $FitnessFkt$
  end while
end procedure

3.2.6 Hinweise zur Implementierung

Die Implementierung wurde auf Basis von Python durchgeführt. Mittels der „numpy“ Bibliothek lassen sich mit Python in Analogie zu Matlab sehr effizient Matrizenoperationen wie Determinantenberechnung oder Matrixinvertierung durchführen.
Zur Generierung eines Startparametersatzes wurde eine einfache Rastersuche implementiert. Dies kann durch Methoden des *Experimental Desgin* verbessert werden, wie in [NHQ09] beschrieben.
Für die unterlagerten Optimierungen der Hyperparameter wurde eine Implementierung in „scipy“ des „L-BFGS-B“ Optimierungsverfahren auf Basis einer FORTRAN

Implementierung nach [ZBLN97] verwendet. Für das unterlagerte Optimierungsproblem der UCB bzw. Einstufen-Heuristik wurde ein genetischer Algorithmus aus der „pyevolve"-Bibliothek verwendet. Genaueres siehe Anhang A. Das Hauptproblem bei der praktischen Umsetzung ist die Invertierung der **R** bzw. **C** Matrix, sofern Testpunkte im Parameterraum sehr nahe beinander liegen. Die in [NHQ09] vorgeschlagene Methode zur Verbesserung der numerischen Stabilität durch adaptives Vergrößern der Diagonalelemente lässt sich als eine adaptive Rauschaufschaltung interpretieren, die Eingangs in Abschnitt 3.2.1 erwähnt wurde. Da in [NHQ09] jedoch die zu optimierende Funktion (3.7) bzw. (3.25) mit einem „großen" Offset in solchen, numerisch instabilen Bereichen beaufschlagt wird, entsteht dadurch ein zusätzlicher „Desginparameter", der gewählt werden muss. Daher wurde in der Implementierung dieser Mechanismus zwar auch berücksichtigt, um einen Absturz zu vermeiden, jedoch wird prinzipiell von verrauschten Datensätzen ausgegangen, was sich in der Praxis als unkritisch herausgestellt hat und damit kaum eine Verfälschung der Ergebnisse durch einen „Offset" auftreten. In der Praxis hat sich herausgestellt, dass oftmals eine konstante Wahl für p_l über alle Dimensionen hinweg hinreichend ist.

3.3 Synthetische Testfunktionen

Im Folgenden werden synthetische Referenzprobleme[3] anhand mit der GP-UCB- und GP-Einstufen-Heuristik getestet.
Zuerst wurde validiert, dass der unterlagerte Genetische Algorithmus die Optima der jeweiligen Probleme findet, um ein Fehlschlagen aufgrund der unterlagerten Optimierung auszuschließen. Die Testfunktionen sind so gewählt, dass unterschiedliche Problemcharakteristika getestet werden. Die Funktionen sind in Abbildung 3.3 über den Parameterraum gemäß Tabelle 3.1 geplottet.
Es wird eine maximale Anzahl von 200 Iterationen festgelegt. Da es von entscheidenter Bedeutung ist, wie viele Iterationen für die Optimierung benötigt werden, wird als Gütemaß die Anzahl der Funktionsaufrufe zur Erreichung von 1% und 0.1% des Optimums verwendet. Um den Fokus auf die Heuristik in noch unbekannten Gebieten zu legen wurden dem Optimierungsverfahren zu Beginn der Optimierung keine vorhandenen Messungen zur Verwendung übergeben. Die Ergebnisse sind in Tabelle 3.1 zusammengefasst. Es zeigt sich, dass unter diesen Voraussetzungen die

Name	d_p	**Parameterraum**	**Bereich**	1% **UCB**	1% **OS**	0,1% **UCB**	0,1% **OS**
Gramacy & Lee	1	$0,5 \; - \; 1,5$	6,1	13	3	15	17
Rosenbrock	2	$[-0,5 \; -0,5] \; - \; [1,5 \; 1,5]$	2,8	8	4	47	33
Schaffer F6	2	$[-6,0 \; -6,0] \; - \; [6,0 \; 6,0]$	1,0	64	E	21	E
Rastrigin 2D	2	$[-1,0 \; -1,0] \; - \; [1,0 \; 1,0]$	21,0	31	40	55	E
EAOM	2	$[-2,0 \; -2,0] \; - \; [7,0 \; 7,0]$	1,0	33	*E*	34	E

Tabelle 3.1: Auswertungen. Mit „E" gekennzeichnete Zellen stehen für ein fehlschlagen bei 200 Iterationen.

Einstufen-Heuristik bei den zweidimensionalen Problemen Schwierigkeiten beim Auf-

[3]Entnommen aus `http://www.sfu.ca/~ssurjano`, zuletzt geprüft am 31.7.2014.

finden des globalen Optimums aufweist und damit eine höhere Sensitivität gegenüber einem anfänglichen Datensatz hat, zugleich jedoch effizienter innerhalb einer lokalen, unbekannten Umgebung konvergiert.

3.4 MINIMAX Optimierungsstrategie

Stehen Optimierungselemente (siehe Kapitel 2) in „signifikanter“ Wechselwirkung zueinander, so lässt sich die Optimierung eines Elements als *Spiel* auffassen.
Am Beispiel der Pick'n'Place Applikation kann es von Interesse, mit welcher Geschwindigkeit der Kopf geschwenkt werden soll, um möglichst schnell alle Objekte, Ziele und Hindernisse mit ausreichender Genauigkeit zu erkennen. Da die Erkennung neben der Geschwindigkeit von der Position und Ausrichtung des Objektes abhängt, eignet sich dieses Beispiel sehr gut.
Um die Anpassung der Geschwindigkeit gezielt in schwierigen Situationen vorzunehmen sollte die Umgebung, bezeichnet als „MIN“, möglichst schwierige Situationen für die Objekterkennung realisieren. Die Objekterkennung, bezeichnet als „MAX“, soll durch die Optimierung im statistischen Mittel besonders gute Resultate erzielen. Schlechte Resultate von MAX bedeuten also aus Sicht von MIN ein gutes Resultat - und umgekehrt. Der resultierende Algorithmus ist in Algorithmus 7 gezeigt.

Bemerkung zur Skalierung der kontinuierlichen Bewertungsfunktion

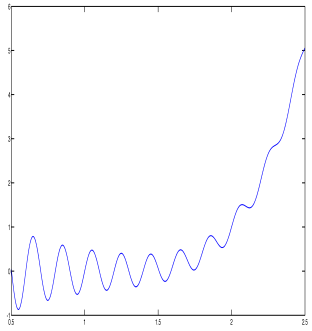
Werden bei der MINIMAX-Optimierung Merkmale mit unterschiedlicher Größenordnung und/oder Einheit zusammengeführt, so muss die Skalierung manuell über einen neuen Gewichtungsvektor $\boldsymbol{\omega}^*$ erfolgen, da MAX nur das skalierte Ergebnis von MIN erhält. Der Anwender muss daher in diesem Fall

$$\boldsymbol{\omega}^* = \boldsymbol{Z}_{manuell}\boldsymbol{\omega} \tag{3.28}$$

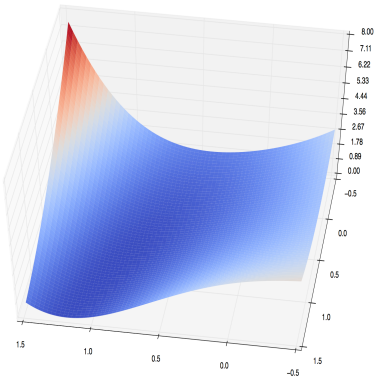
manuell vorgeben.

Algorithm 7 MINIMAX Optimierungsstrategie

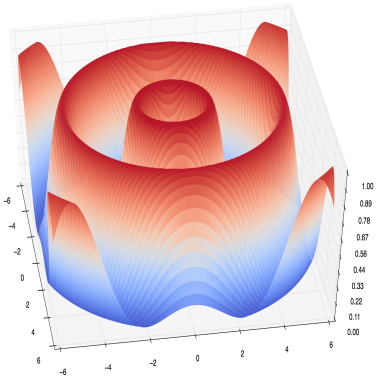
1: **procedure** MINIMAX-Optimierung($\boldsymbol{D}_0$, $\mathcal{P}_{p,MIN}$, $\mathcal{P}_{p,MAX}$, $\boldsymbol{p}_{min,MIN}$, $\boldsymbol{p}_{min,MAX}$, $Testlauf$, $f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}^*}$, $g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}}$, T_{MIN}, T_{MAX}, T_e)
2: f_{MIN} ←GP-UCB/ONE($\mathbf{0}, \mathcal{P}_{p,MIN}, \boldsymbol{p}_{min,MIN}, Testlauf, f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}^*}, g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}}$, T_{MIN}, T_e)
3: **return** GP-UCB/ONE($\boldsymbol{D}_0, \mathcal{P}_{p,MAX}, \boldsymbol{p}_{min,MAX}, -f_{MIN}, f^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{\omega}^*}(\boldsymbol{x}_p) = \boldsymbol{x}_p, g^*_{\boldsymbol{M},\boldsymbol{\Phi},\boldsymbol{c}} = \mathbf{1}, T_{MAX}, T_e$) ▷ Liefere gefundenes MINIMAX Optimum zurück.
4: **end procedure**



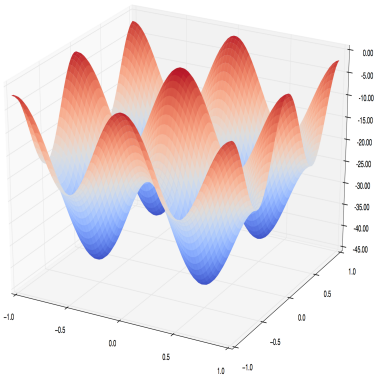
(a) GramacyLee: Vielzahl an lokalen Optima, die sich teilweise nur geringfügig von dem globalen Optimum unterscheiden.



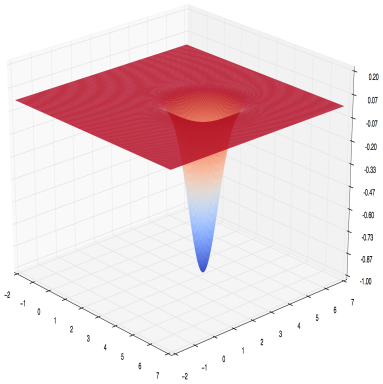
(b) Rosenbrock: Funktion mit einem globalen Optimum, das in einer flachen Umgebung liegt.



(c) SchafferF6: Stetige Funktion, ähnlich einer konzentrischen Wellenausbreitung mit einem globalen Optimum in der Mitte.



(d) Rastrigin 2D: Weist in dem betrachteten Bereich viele ähnliche lokale Minima auf.



(e) EAOM:Sehr flache Funktion mit einem charakteristischen Optimum.

Abbildung 3.3: Synthetische Testfunktionen für die GP-UCB und GP-ONE Optimierung.

Kapitel 4

Beispielapplikation: Pick'n'Place

4.1 Statische Szenengenerierung

Die statische Szenengenerierung erfüllt zwei Hauptaufgaben. Zum Einem das Verwalten verschiedener statischer Szenen. Zum Anderen muss sie als Optimierungselement Änderungen durch den Optimierer anwenden können und den aktuellen Zustand der Szene zur Verfügung stellen. Dies lässt sich weitgehenst systematisch automatisieren. Für den Fall der statischen Umgebung sind alle Arten von statischen Objekten, die sich in der Simulation befinden, von Interesse. So lassen sich beim Erstellen der Beschreibung einer statischen Szene alle relevanten Daten aus einer bereits laufenden Simulation extrahieren. Der Nutzer muss lediglich entscheiden, welche Parameter für die Beschreibung der Szene von Relevanz sind. Innerhalb dieser Menge an Parametern muss dann festgelegt werden, welche für den Optimierer in welchen Wertebereichen modifizierbar sein sollen. Die Erstellung einer statischen Szene ist in Abbildung 4.1 skizziert.

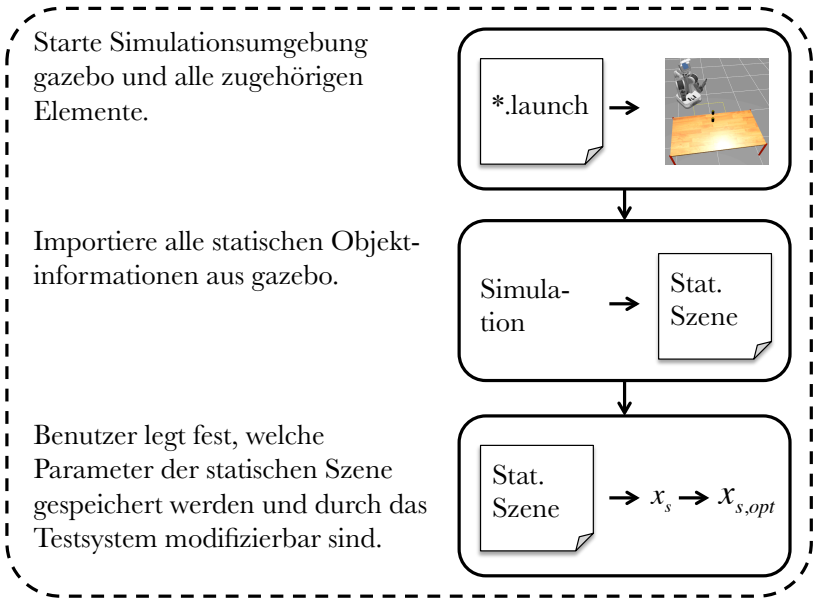


Abbildung 4.1: Ablauf der Erstellung einer Statischen Szene.

Eine erstellte Szene kann als Optimierungselement ausgeführt werden und stellt damit

alle notwendigen *services* aus in Kapitel 2, Abschnitt 2.2 bereit. Beim Aufruf des *service* `init()` wird die gespeicherte, statische Szene geladen. Mit `start()` werden die durch das Testsystem festgelegten Parameter in der Simulation angewandt.

4.2 Pick'n'Place am PR-2

4.2.1 MoveIt!

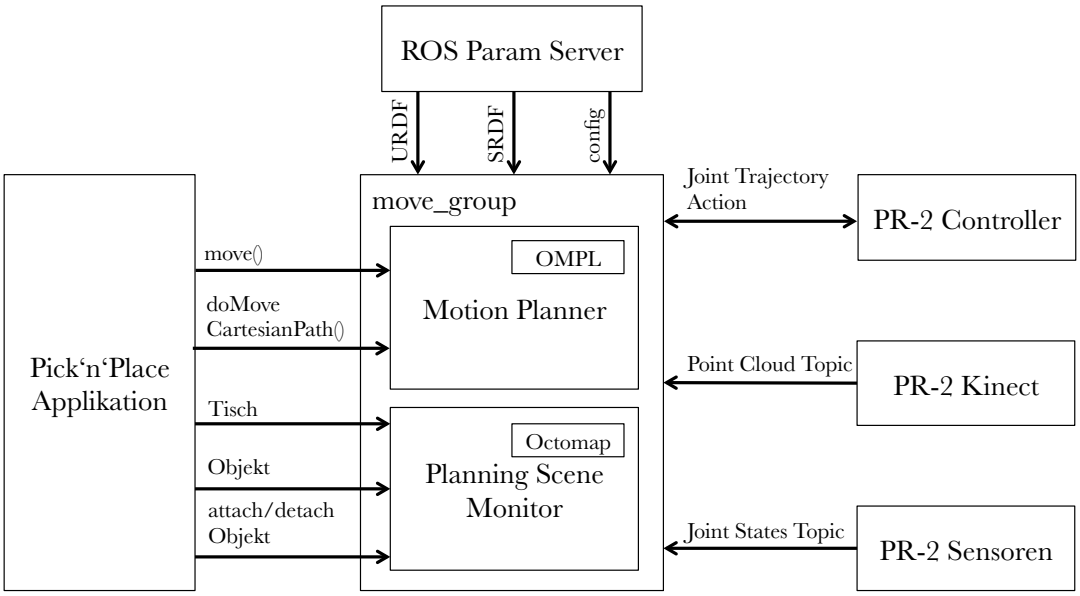


Abbildung 4.2: Konzept der Pick'n'Place Applikation mit MoveIt! und dem PR-2 Roboter.

Abbildung 4.2 zeigt das Konzept der Pick'n'Place Applikation zusammen mit den Grundprinzipien von MoveIt![1].
In der Mitte ist die *move_group* abgebildet. Sie vereint sämtliche Funktionalitäten von MoveIt!. Die Konfiguration der *move_group* erfolgt über den *ROS Param Server*.
Die Kommunikation mit dem Roboter erfolgt über drei Schnittstellen. Die Regler des Roboters werden über die *Joint Trajectory Action* angesteuert. Die Umgebungssensorinformationen werden von dem Roboter über die *Point Cloud Topic* als Point Cloud [RC11] der *move_group* bereitgestellt. In der Pick'n'Place Applikation wurde hierfür der *Xbox-Kinect* Sensor verwendet. Alle sonstigen Sensorinformationen, die nicht zur Wahrnehmung gehören kommunizieren über die *Joint State Topic* mit der *move_group*. Hier werden ebenfalls die Koordinatentransformationen durch die *TF-Library* [QCG+09] von ROS durchgeführt.
Die *move_group* verwaltet die ankommenden Informationen des Roboters in einer Planungszene (*Planning Scene Monitor*). Die über *Joint States Topic* gelieferten Informationen bilden zusammen mit der *URDF/SRDF* Beschreibung des Roboters eine

[1]Weitere Informationen unter `http://moveit.ros.org/documentation/`, zuletzt geprüft am 31.7.2014.

interne Modellierung der aktuellen Roboterkonfiguration. Daten aus der *Point Cloud Topic* werden in der sogenannten *Octomap* verarbeitet. Diese modelliert dreidimensionale Umgebungsstrukturen durch würfelförmige *Voxels*. Neben den sensorischen Eingängen des Roboters kann der Anwender die Planungszene mit Objekten befüllen. Für die in der Einleitung gezeigten Szene ist die Visualisierung der Planungszene mit *Rviz*[2] in Abbildung 4.3 gezeigt.
Das *move_group* Interface erlaubt es diese Objekte mit einem Endeffektor des Roboters zu verknüpfen, wenn es z.B. gegriffen wurde. Diese Verknüpfungen werden bei der Planung von Bewegungen berücksichtigt.
Durch ein Plugin Interface erfolgt die eigentliche Bewegungsplanung mit der **O**pen **M**otion **P**lanning **L**ibrary (OMPL). Diese besteht aus zahlreichen zufallsbasierten Bewegungsplanungsalgortihmen. Eine detailierte Beschreibung ist in [SMK10] zu finden. Für die Planung einer Bewegung kann entweder die Position und Ausrichtung eines Endeffektors oder eine gewünschte Konfiguration der Robotergelenke angegeben werden. In der Planung werden automatisch die Umgebung und mögliche Kollisionen mit dem Roboter selbst durch ein Interface mit der Planungszene von MoveIt! berücksichtigt. Des Weiteren können sogenannte „kinematic contraints" vorgegeben werden. Auf diese Weise kann z.B. erzwungen werden, dass beim Greifen eines befüllten Bechers dessen Ausrichtung während der Bewegung konstant bleibt. Als Ergebnis liefert die Planung eine Trajektorie mit den maximalen Geschwindigkeiten und Beschleunigungen.

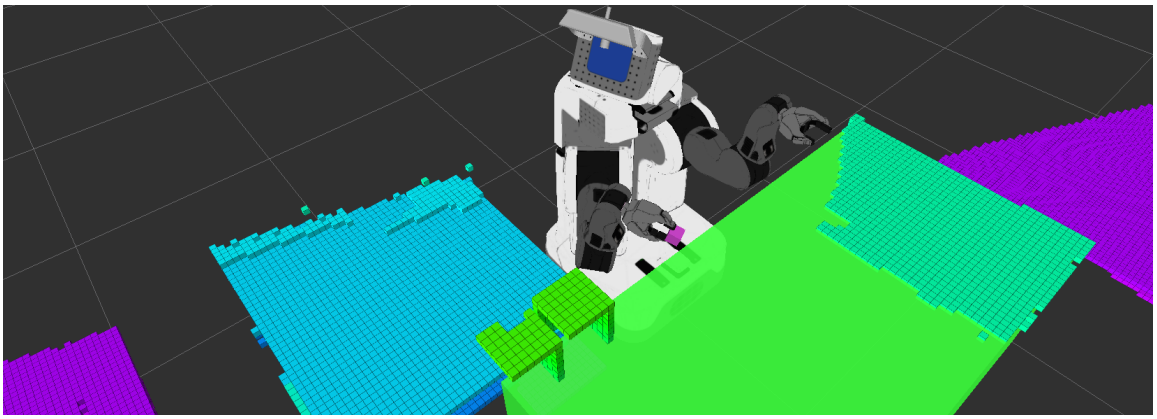


Abbildung 4.3: Beispiel für eine mit Rviz visualisierte MoveIt! Planungszene, zu der in Kapitel 1, Abb. 1.1 gezeigten Situation. Der Planungszene wurde ein Tischobjekt hinzugefügt und das gegriffene Objekt mit dem Greifer verlinkt. Die Position der Box am linken Ende des Tisches variiert. Anhand der doppelten Abbildung der Box ist zu erkennen, dass verdeckte Bereiche in der *Octomap* erhalten bleiben, bis diese wieder erfasst werden können.

Für die Pick'n'Place Applikation wird die *move_group* C++ Schnittstelle verwendet. Sie dient als Interface für die *move_group*.

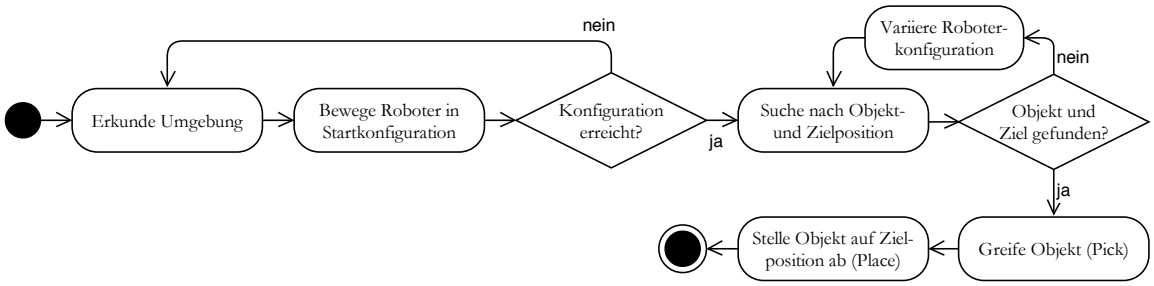


Abbildung 4.4: Aktivitätsdiagramm der Pick'n'Place Applikation.

4.2.2 Ablauf

Abbildung 4.4 zeigt das Aktivitätsdiagramm einer Pick'n'Place Aktion. Die Rückkopplungsschleifen sind dadurch begründet, dass die Wahrnehmung der Umgebung von der robotereigenen Konfiguration abhängt. In dem einfachen Beispiel der Pick'n' Place Applikation reduziert sich dieser Einfluss auf das Verdecken der Umgebung durch die Arme.
Nachdem eine Durchführung der Applikation gestartet wurde, wird zunächst mittels Schwenken des Roboterkopfes der aktuelle Arbeitsbereich von MoveIt! in der *Octomap* modelliert. Anschließend wird der Roboter in eine definierte Ausgangsposition gefahren. Nun kann der Arbeitsraum des Roboters nach Objekten und Zielen abgesucht werden. Wurde nach einer Suchsequenz kein Objekt-Ziel Paar gefunden, so werden die Arme des PR-2 in eine andere Stellung gebracht, um eine Verdeckung durch diese zu vermeiden. Die Armbewegung beim Greifen eines Objektes (Pick) erfolgt durch MoveIt! und ist in Ablauf 4.5 gezeigt. Das Abstellen (Place) erfolgt analog dazu.

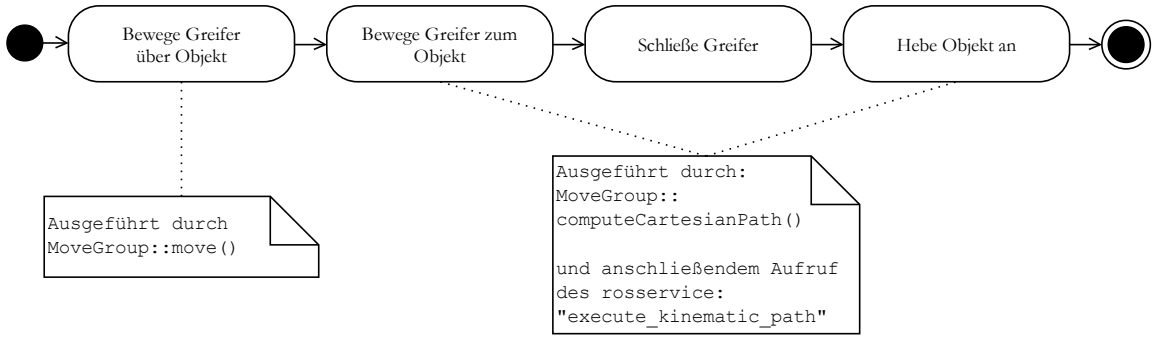


Abbildung 4.5: Aktivitätsdiagramm des „Aufheben" eines Objektes.

4.2.3 Implementierung

Abbildung 4.6 zeigt das Klassendiagramm der Pick'n'Place Applikation. Wie in Kapitel 2, Abbildung 2.6 gezeigt, ist die abstrakte Basisklasse `PickPlacePr2MoveIt` ein Optimierungselement. Durch die MoveIt! eigenen Funktionalitäten sind darin die

[2] *Rviz* ist fester Bestandteil von ROS und ermöglicht zwei- und dreidimensionale Visualisierungen von z.B. Point Clouds, Roboter, Koordinatensysteme, etc.

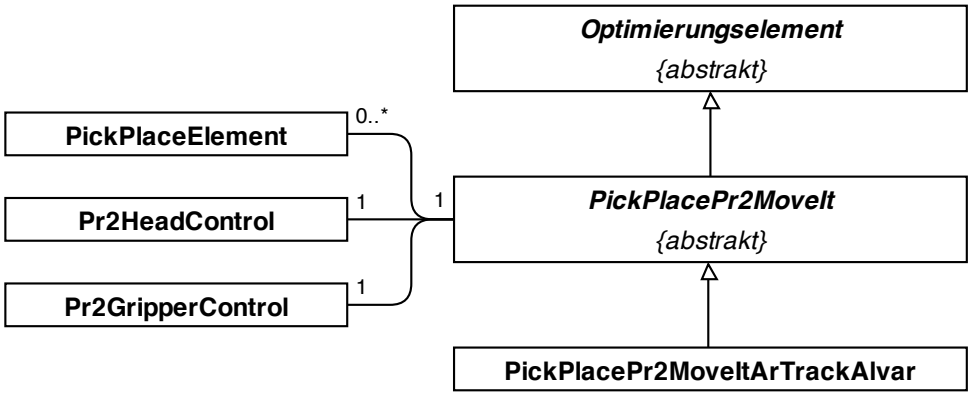


Abbildung 4.6: Vereinfachtes Klassendiagramm der Pick'n'Place Applikation mit MoveIt! und dem AR Track Alvar Modul.

komplexen Verhalten, wie das Greifen nach einem Objekt oder dem Anfahren in die Ausgangsposition implementiert. Sie verwendet außerdem noch die PR2-spezifische Implementierung der Kopf- und Greifersteuerung. Die vereinfachte Klasse mit den wesentlichen Merkmalen ist in 4.7 gezeigt.

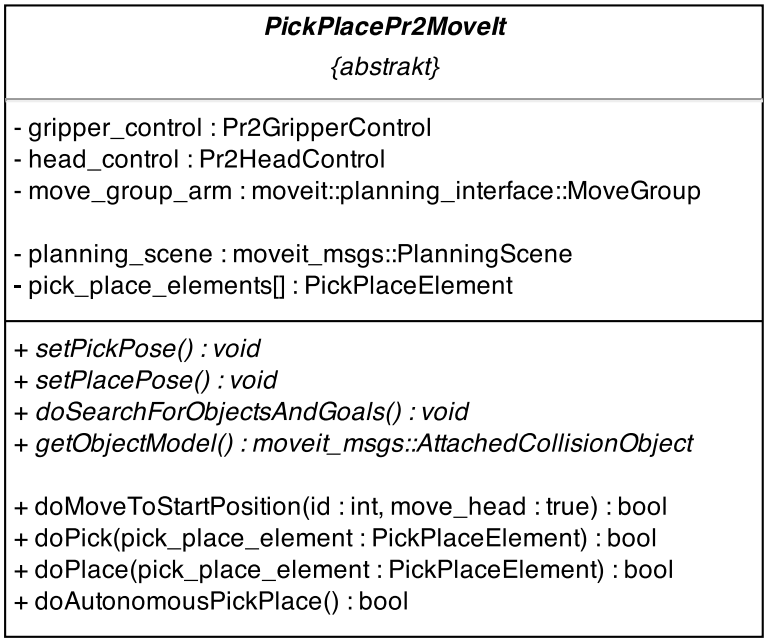
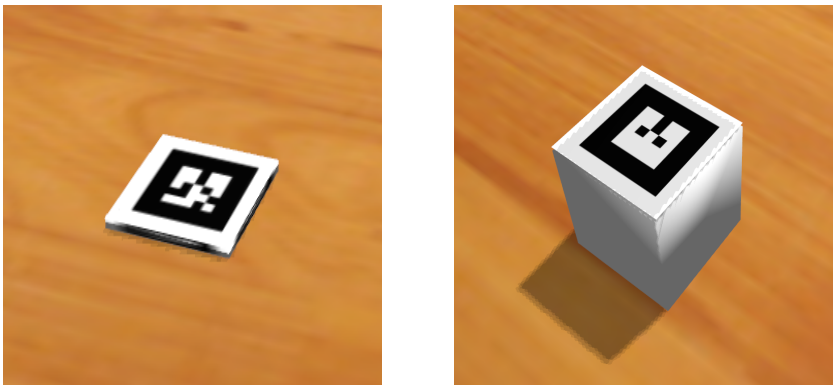


Abbildung 4.7: Vereinfachtes Klassendiagramm der Pick'n'Place Applikation.

4.2.4 Objekterkennung

Während der Implementierung hat sich herausgestellt, dass die Funktionstüchtigkeit von MoveIt! sehr eng mit der Wahrnehmung der Umgebung zusammenhängt. Das Konzept mit der *Octomap* für die Modellierung der Umgebung hat sich in der verwendeten Version von MoveIt! (siehe Anhang A) nur als bedingt nützlich erwiesen. Dies liegt daran, dass die gesamte Umgebung mit einer einheitlichen Auflösung

durch die *Octomap* reproduziert wird. Für das Greifen des Objektes werden genaue Maße benötigt, um eine Planung unter Berücksichtigung von Kollisionen durchführen zu können. Selbst die Oberfläche, auf der das Objekt steht, muss genauer in der Planungsszene modelliert sein, als es durch die *Octomap* ohne wesentliche Performanceeinbußen möglich ist. Daher müssen der MoveIt! Planungsszene entsprechende Modelle aus einer externen Quelle zur Verfügung gestellt werden. Zudem beinhalten einzelne Voxels keinerlei Informationen über deren Zuverlässigkeit. Dadurch entstehen oft Situationen, in denen eine „falsche" Kollision erkannt wird, wenn ein Bereich der Umgebung, z.B. durch den Roboter selbst, verdeckt wird.
Da die Wahrnehmung/Objekterkennung nicht Bestandteil dieser Arbeit ist, wurde dieses Problem durch eine statische Definition von Objekten in Verbindung mit Markern gelöst. Das ROS-Modul AR-Track-Alvar[3] ist ein Wrapper für das Alvar Softwarepaket für die Entwicklung von „augmented reality" Applikationen. Es wurde dazu genutzt, um die Position und Ausrichtung von einzelnen *AR-Markern* wie in Abbildung 4.8 gezeigt, zu erkennen. Bei der Pick'n'Place Anwendung wurden damit das Objekt, wie auch die Zielposition (Abbildung 4.8) für die Objekterkennung markiert.



(a) AR Marker auf einer Scheibe als Objekt in gazebo.

(b) AR Marker zur Markierung des Objekts für die Pick'n'Place Aktion.

Abbildung 4.8: AR Marker für die Objekterkennung in der Simulation.

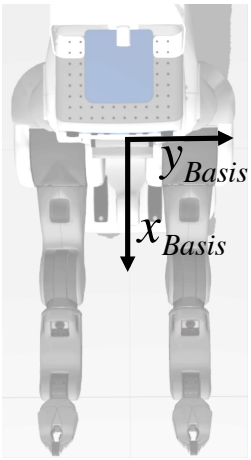
[3] `http://wiki.ros.org/ar_track_alvar`, zuletzt geprüft am 30.7.2014.

Kapitel 5

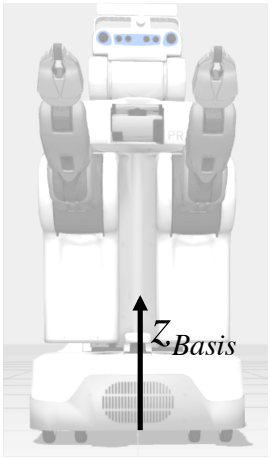
Evaluation und Optimierung

5.1 Objekterkennung

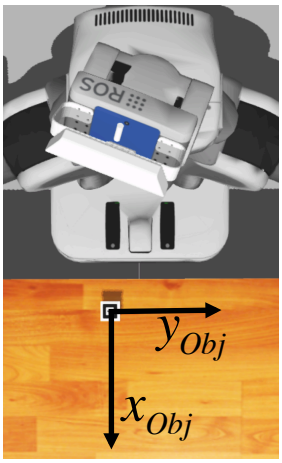
Auf der Erkennung der Objekt- und Zielposition basiert unter Anderem die Planung und Ausführung der Pick'n'Place Applikation durch MoveIt!. Deshalb wird zuerst analysiert, ob in dem betrachteten Bereich das AR-Track-Alvar Modul die notwendige Genauigkeit bereitstellt.



(a) x/y Achse PR-2 Basisplattform.



(b) z Achse PR-2 Basisplattform.



(c) Objektkoordinatensystem.

Abbildung 5.1: Basis- und Objektkoordinatensystem.

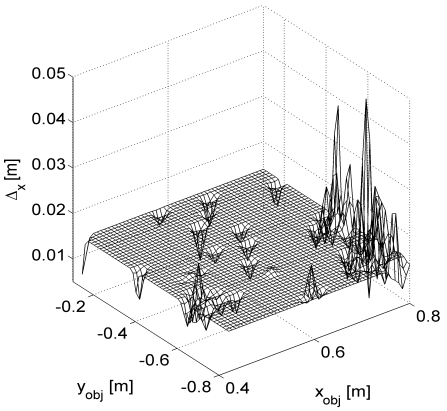
Für eine Evaluation des AR-Track-Alvar Moduls in der Simulation wurde die Szene aus Abbildung 5.1 erstellt. Die freien Parameter des Objekts sind $\boldsymbol{x}_p = (x_{obj}, y_{obj})^T$ - dessen x,y Koordinaten, transformiert auf das roboterfeste Koordinatensystem der Basisplattform. Der Parameterraum wurde respektive des Arbeitsraumes des rechten Armes zu $\boldsymbol{x}_p = [0,42 \quad -0,80] \times [0,80 \quad -0,10]$ gewählt. Aufgrund der exakten Symmetrie bzgl. der x/z Ebene in der Simulation wird nur eine Seite betrachtet. Die verwendete Bewertungsfunktion ist durch den quadratischen Fehler

$$f^*(\boldsymbol{x}_o) = \Delta_{\boldsymbol{x}} = \left\| \begin{pmatrix} x_{obj} \\ y_{obj} \\ (z_{obj} - \epsilon) \end{pmatrix}_{AR} - \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \end{pmatrix}_{sim} \right\|_2^2 \tag{5.1}$$

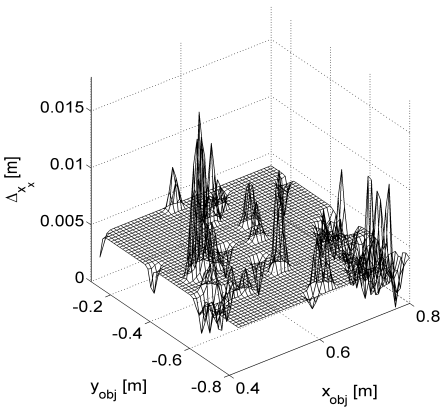
definiert. Der Offset ϵ entspricht der halben Objekthöhe, da der Marker wie in Abbildung 4.8 oben auf dem Objekt angebracht ist und aus der Simulation der Mittelpunkt geliefert wird.
Das automatisierte Testen wurde mit 80 Iterationsschritten und der GP-UCB Heuristik bezüglich (5.1) ausgeführt. Anschließend wurden auf Basis dieses Datensatzes weitere Merkmale analysiert ohne neue Simulationsdurchläufe durchzuführen. Die Ergebnisse sind der Tabelle 5.1 zu entnehmen.
Zur Analyse sind in Abbildung 5.2 die Kriging-Modelle der unterlagert arbeitenden Optimierung dargestellt. Die Skalierung nach Abschnitt 2.4, Kapitel 2, wurde zur besseren Lesbarkeit rückgängig gemacht, ebenso die Quadratur ($\Phi(.) = (.)^2 \rightarrow \Phi^{-1} = (.)^{0,5}$). Dargestellt ist damit der Fehler $\Delta\ [m]$ als betragsmäßige Abweichung zwischen erkannter und tatsächlicher Koordinate in euklidischer Metrik.
Die Tabelle 5.1 zeigt für die Bewertungsfunktion 5.1 sowie für die Fehler $\Delta_{\boldsymbol{x}_y}$ und $\Delta_{\boldsymbol{x}_z}$ ein intuitives Ergebnis am Rande des Parameterraums. Auffällig ist, dass für $\Delta_{\boldsymbol{x}_x}$ eine Position in einem anderen Bereich ermittelt wurde. Dieses Phänomen lässt sich dadurch begründen, dass sich $\Delta_{\boldsymbol{x}_x}$ im Gegensatz zu $\Delta_{\boldsymbol{x}_y}$ und $\Delta_{\boldsymbol{x}_z}$ über den ganzen Parameterraum hinweg unbeständig zeigt, insbesondere an dem ermittelten Punkt $(0,49 \quad 0,47)$. Dies wird auch anhand der Abbildungen in 5.2 deutlich. Darin ist zu erkennen, dass die Varianz des angenommenen stochastischen Prozesses für $\Delta_{\boldsymbol{x}_x}$ sehr groß ist. Anhand des Krigingmodells spiegelt sich diese Tatsache in extrem großen Werte für θ_l bzgl. aller Bewertungsfunktionen wieder. In diesem Fall ist die Vorhersage von weiteren Optima bzw. Testfällen bzgl. anderer Metriken nur bei einem starken Zusammenhang dieser mit der Ursprünglichen im linearen Sinne möglich.
Insgesamt zeigt sich anhand der durchschnittlichen Abweichung, außer am Rande des Arbeitsbereiches, eine ausreichende Genauigkeit für die Applikation.

Bewertungsfunktion	**Ergebnis**
(5.1)	$(0,80 \quad -0,62)$
$\Delta_{\boldsymbol{x}_x} = \lvert x_{AR} - x_{Obj} \rvert^2$	$(0,49 \quad -0,47)$
$\Delta_{\boldsymbol{x}_y} = \lvert y_{AR} - y_{Obj} \rvert^2$	$(0,78 \quad -0,70)$
$\Delta_{\boldsymbol{x}_z} = \lvert z_{AR} - (z_{Obj} + \epsilon) \rvert^2$	$(0,75 \quad -0,72)$

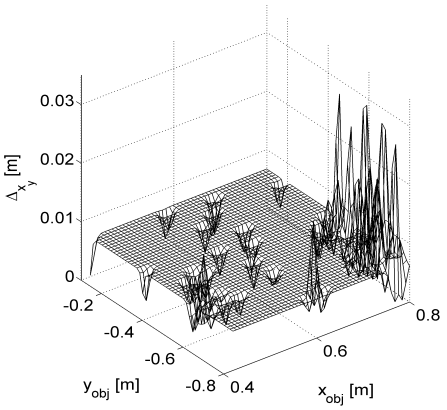
Tabelle 5.1: Ergebnisse des Testsystems beim Testen der Objekterkennung.



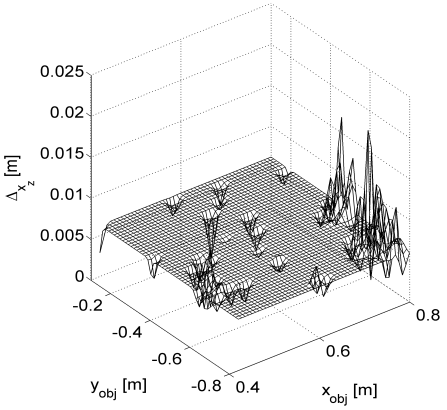
(a) Bewertungsfunktion (5.1).



(b) $||x_{AR} - x_{Obj}||^2$.



(c) $||y_{AR} - y_{Obj}||^2$.

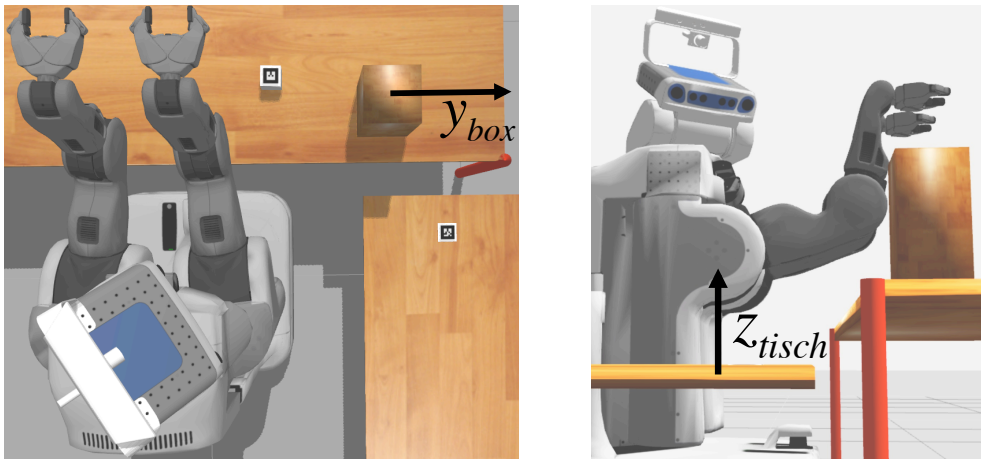


(d) $||z_{AR} - (z_{Obj} + \epsilon)||^2$.

Abbildung 5.2: Anhand der Testdaten des automatisierten Testlaufs bzgl. (5.1) als Bewertungsfunktion berechnete Ergebnisse. Die Färbung entspricht 3σ gemäß (3.19).

5.2 MoveIt!

Zur Analyse der Applikation mit MoveIt! wurde das in der Einleitung, Abbildung 1.1 gezeigte Szenario erstellt. Zielstellung der Pick'n'Place ist es das Objekt, das frontal auf dem Tisch liegt (6cm×6cm×6cm, 100 Gramm) auf der Tischoberfläche zur Rechten des Roboters abzustellen. Das genaue Ziel ist durch den weiteren AR-Marker gekennzeichnet. Freie Parameter in der Szene sind die vertikale Position des Tisches zur Rechten des Roboters z_{tisch} sowie die horizontale Position der Hindernis-Holzbox entlang einer Dimension y_{box} (siehe Abbildung 5.3), transformiert auf das Basiskoordinatensystem, vgl. Abbildung 5.1. Der Parameterraum wurde zu $\boldsymbol{x}_o \in [0,37\ -0,80] \times [0,47\ -0,65]$ respektive des Arbeitsraumes des rechten Armes und der Durchführbarkeit bzgl. des Hindernisses definiert.
Dadurch kann MoveIt! mittels z_{tisch} auf Eigenschaften in Grenzbereichen der Armlänge und mit y_{obj} auf das Verhalten bei einem Hindernisobjekt in der einfachsten



(a) Vogelperspektive. (b) Seitliche Perspektive.

Abbildung 5.3: Versuchsaufbau automatisiertes Testen von MoveIt!.

Trajektorie analysiert werden. Für die verwendete *move_group* „right_arm“ wurde der standardmäßig aktivierte Planer *Lazy Bi-directional Kinematic Planning by Interior-Exterior Cell Exploration (LBKPIECE)* [ŞK09][BK00] verwendet. Das automatisierte Testen wurde mit der GP-UCB Heuristik und 50 Testläufen durchgeführt.

Hinweise zu den Ergebnissen

Da sich MoveIt! überwiegend im „alpha“ Status befindet, sind einige Funktionalitäten nicht robust. Hierzu gehört insbesondere das Aktualisieren der Planungszene. Neben den in Kapitel 4.2.4 erläuterten Problemen bzgl. der Umgebungsrepräsentation durch die *Octomap* gibt es vor allem bekannte, funktionale Probleme bei der Aktualisierung der Umgebungsrepräsentation durch die *planning_scene*.
Diese Probleme machen sich durch nicht unmittelbar begründbare Fehlausführungen der Pick'n'Place Applikation bemerkbar. Somit sind die hier vorgestellten Ergebnisse eine Analyse der verwendeten Version von *MoveIt!* im Entwicklungsstatus und lassen damit nur entsprechende Rückschlüsse zu. Zudem treten bei der verwendeten Version von *gazebo* gelegentlich Probleme bei gegriffenem Objekt durch den Roboter auf. Diese treten durch „driften“ des Objekts trotz hoher simulierter Reibung und

Bewertungsfunktion	**Optimum**	$3\sigma_{kriging}$	$\theta_{y_{box}}$	$\theta_{z_{tisch}}$
Planung	$(0,45 \quad -0.70)$	$0,32$	„∞“	6706
Ausführung	$(0,40 \quad -0,88)$	$0,067$	„∞“	4106
Applikation (5.2)	$(0,44 \quad -0,70)$	$0,20$	8186	7433

Tabelle 5.2: Ergebnisse des Testsystems beim Testen der MoveIt! Pick'n'Place Applikation. Mit „∞ “ gekennzeichnete Werte symbolisieren sehr große Werte, die als Konstanten eine Instabilität des Algorithmus beim Streben gegen unendlich verhindern.

maximaler Schließkraft des Greifers auf. Um diese Effekte zu minimieren werden bei Toleranzverletzung bis zu 2 Wiederholungen des Testlaufs vorgenommen, bevor das Ergebnis übernommen wird.

Planung

Die Planung wird durch die Summe der benötigten Planungszeit für Endeffektorbewegungen hin zum Objekt und zum Greifen des Objekts bewertet. In Tabelle 5.2 ist das Ergebnis dokumentiert. Es zeigt sich, dass eine hohe Position des Tisches in Verbindung mit einer relativ nahen Positionierung des Hindernisses an dem zu greifendem Objekt eine verhältnismäßig lange Planungszeit in Anspruch nimmt. In Abbildung 5.4 ist dies auch zu erkennen. Es zeigen sich vor allem mit zunehmendem y_{box} mehrfache Spitzen mit ~ 1 Sekunde bzgl. der durchschnittlichen Planungszeit von $\sim 0,6$ Sekunden. Der stärkere Einfluss der y_{box} Dimension ist auch anhand des identifizierten Krigingparameters θ_{box} aus Tabelle 5.2 ersichtlich. Der erhöhte Wert bei $\sim (0,36 \quad -0,88)^T$ lässt auf eine längere Planungszeit bei größerer Distanz schließen.

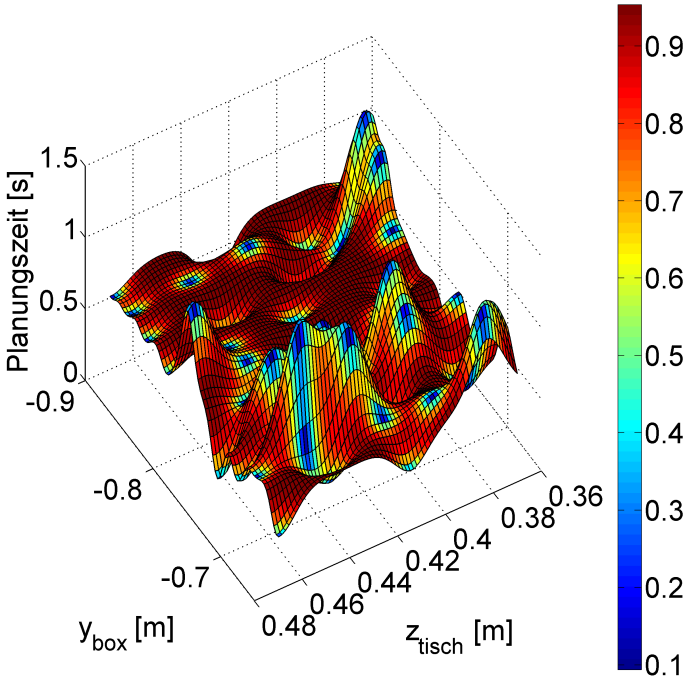


Abbildung 5.4: Interpoliertes Ergebnis der Planungszeit. Die Färbung entspricht 3σ gemäß (3.19).

Ausführung

Zur Auswertung der Ausführung des generierten Plans wurde die für die Pick'n'Place Aktion benötigte Zeit abzüglich der kumulierten Planungszeiten verwendet. Davon ausgeschlossen ist die benötigte Zeit bis zur Erkennung von Objekt und Ziel. Das Ergebnis aus Tabelle 5.2 zeigt ein unerwartetes Ergebnis, da sich weder das Hindernis

in der unmittelbaren Trajektorie, noch die Tischoberfläche am Rande des Arbeitsbereiches befindet. Es hat sich herausgestellt, dass durch ungünstiges Verdecken des Zielbereiches durch den Roboterarm eine erhöhte Tischposition im vorangegangenen Versuch dazu führt, dass dieser Bereich in der *Octomap* nicht aktualisiert wird. Deshalb verbleiben *Voxels* aus dem vorangegangenem Durchlauf, die zu hohen Ausführungszeiten aufgrund sehr umständlicher Bewegungen führen. MoveIt! verfügt bereits über eine *allowLooking( )* Funktion, die wie sich herausgestellt hat für gewöhnlich Bewegungen noch nicht funktionsfähig ist. Diese Funktion kann auch deshalb noch nicht funktionieren, da die Voxels keine Informationen über deren Zuverlässigkeit beinhalten.

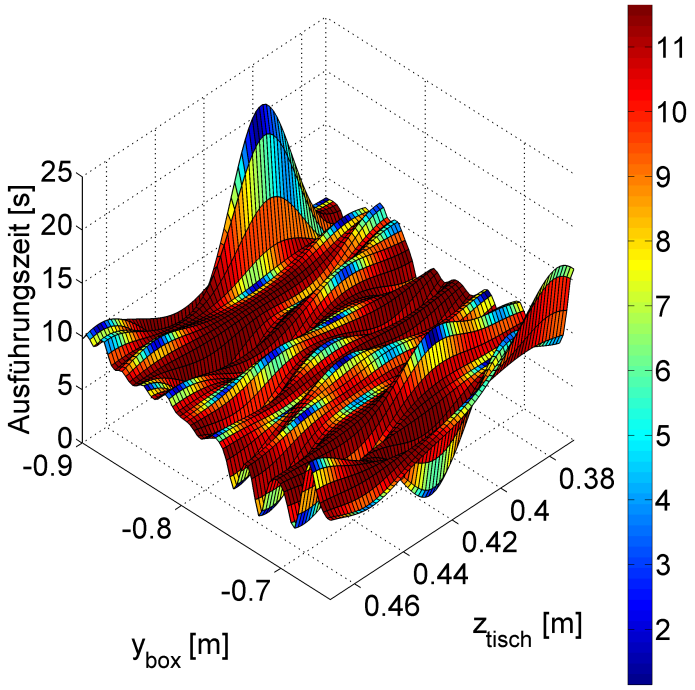


Abbildung 5.5: Interpoliertes Ergebnis der Ausführungszeit. Die Färbung entspricht 3σ nach (3.19).

Applikation

Zur Bewertung der Applikation wurde die Bewertungsmetrik gemäß Abschnitt 2.4, Kapitel 2 mit

$$\boldsymbol{\Phi}(T_P, T_A, \boldsymbol{x}_{obj}, \boldsymbol{x}_{ziel}) = \begin{pmatrix} T_P \\ T_A \\ ||\boldsymbol{x}_{obj} - \boldsymbol{x}_{ziel}||_2^2 \end{pmatrix} \tag{5.2}$$

und $\boldsymbol{w} = [1 \quad 1 \quad 1]$ gewählt, wobei T_P und T_A der Planungs- bzw. Ausführungszeit entsprechen. Der Abstand zwischen Soll- und Istposition kann hier nicht als Genauigkeitsmaß interpretiert werden, da Situationen identifiziert werden, in denen keine erfolgreiche Ausführung möglich ist. Er repräsentiert vielmehr unlösbare Situationen für das Testsystem. Das Ergebnis ist in Tabelle 5.2 und Abbildung 5.6 ersichtlich. Neben den bisherigen Einflüssen von T_P und T_A zeigen sich durch $||\boldsymbol{x}_{obj} - \boldsymbol{x}_{ziel}||_2^2$

zusätzliche Schwierigkeiten bei einer hohen Tischposition. Dies hängt mit dem maximalen Winkel des Schultergelenks des PR-2 zusammen. Dadurch entstehen gelegentlich Situationen, in denen die Auflösung der *Octomap* nicht hoch genug ist, um eine kollisionsfreie Trajektorie zu finden.

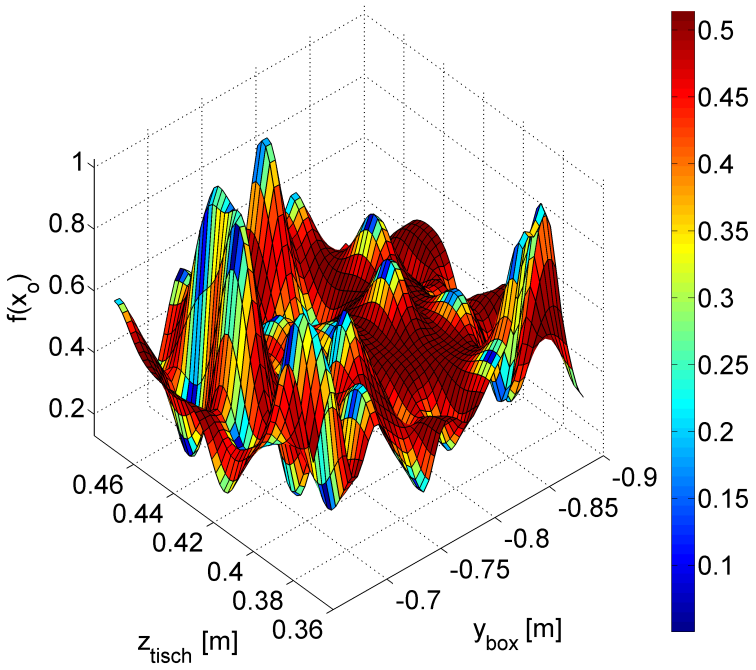


Abbildung 5.6: Interpoliertes Ergebnis der Applikation gemäß 5.2. Die Färbung entspricht 3σ gemäß (3.19).

5.3 Optimale Wahl des Planungsalgorithmus

Tabelle 5.4 zeigt die standardmäßig verfügbaren Planungsalgorithmen in MoveIt! durch die *OMPL*. Ziel ist es eine optimale Wahl eines Planungsalgorithmus in der in Abbildung 5.3 abgebildeten Szene zu treffen. Auf Basis der Bewertungsfunktion nach 5.1 mit $\boldsymbol{\omega} = [1 \quad 2 \quad 0]$[1] wurde das in Abschnitt 3.4 vorgestellte Konzept der MINIMAX Optimierung angewandt.
Dabei dient die GP-UCB mit einem Eckpunktedatensatz und 4 Iterationen (=8 Testläufe, da $d_o = 2$) als Bewertungsmetrik. Verwendet wird diese von einer überlagerten GP-Einstufen Optimierung auf Basis eines 6-Punkte Rasters und 6 weiteren Iterationen bzgl. der ID=$\lfloor \tau_{ID} \rfloor$ mit $\tau_{ID} \in (-0,5 \quad 5,5)$ als Parameterraum. Durch das 6 Punkteraster wird jeder Algorithmus einmal durch das unterlagerte Testsystem ausgewertet, bevor die weiteren Auswertungen auf Basis der Einstufen-Heuristik beginnen. Die kontinuierlichen Vorgaben der Einschritt-Heuristik werden gerundet dem Datensatz hinzugefügt. Es wurde der *Rapidly-exploring Random Trees Connect*[KL00]

[1]Nach Abschnitt 3.4 wurden die Diagonalelemente von $\mathbf{Z}_{manuell}$ aus Gleichung(3.28) zu $Z_{manuell,11} = 1$, $Z_{manuell,22} = 0,1$ und $Z_{manuell,33} = 1$ gewählt.

(RRTConnect) als beste Wahl durch die MINIMAX Optimierung ermittelt. Das Ergebnis ist in Abbildung 5.7 zu sehen. Interessant ist hierbei die Tatsache, dass der von MoveIt! standardmäßig festgelegte Planungsalgorithmus das mit Abstand schlechteste Resultat erzielt hat. Ein Vergleich zu der in Abschnitt 5.2 verwendeten Szene ist in Tabelle 5.4 gezeigt.

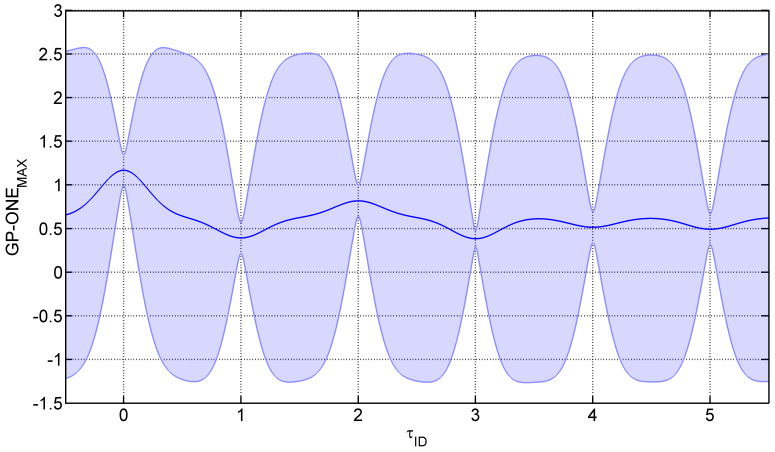


Abbildung 5.7: Interpoliertes Ergebnis der MINIMAX Optimierung. Das Band stellt die Unsicherheit des Kriging-Modells durch 3σ gemäß (3.19) dar.

Verwendet	**ID**	**Planungsalgorithmus**
✓	0	Lazy Bi-directional Kinematic Planning by Interior-Exterior Cell Exploration (LBKPIECE)
✓	1	Bi-directional Kinematic Planning by Interior-Exterior Cell Exploration
✓	2	Probabilistic RoadMap
✗	-	Rapidly-exploring Random Trees*
✗	-	Rapidly-exploring Random Trees
✗	-	Transition-based Rapidly-exploring Random Trees
✓	3	Rapidly-exploring Random Trees Connect (RRTConnect)
✓	4	Single-query Bi-directional Lazy collision checking planner
✓	5	Expansive Space Trees

Tabelle 5.3: Verwendung und Zuweisung ID/Planungsalgorithmen.

Planungsalgorithmus	$\hat{\boldsymbol{\mu}}_P$**[s]**	**max. Planungszeit [s]**	$\hat{\boldsymbol{\mu}}_A$**[s]**	**max. Ausführungszeit [s]**
LBKPIECE	$0,616$	$1,408$	$8,064$	$22,839$
RRTConnect	$0,237$	$0,793$	$5,772$	10.408

Tabelle 5.4: Vergleich LBKPIECE und RRTConnect mit mittlerer Planungs- und Ausführungszeit $\hat{\boldsymbol{\mu}}_P$ und $\hat{\boldsymbol{\mu}}_A$ sowie maximaler Planungs- und Ausführungszeit.

Kapitel 6

Schluss

6.1 Zusammenfassung

Es wurde ein generisches Konzept zum automatisierten Testen und Optimieren von Roboterapplikationen auf Basis von ROS vorgestellt und implementiert. Dieses basiert auf einem „state-of-the-art“ Optimierungsverfahren, welches es zusammen mit dem modularen Konzept der Bewertungsfunktion erlaubt, nach einer abgeschlossenen Optimierung weitere Analysen anhand des bestehenden Datensatzes durchzuführen. Datensätze von Optimierungen innerhalb eines Testszenarios können überdies zusammengeführt und als Basis für eine neue Optimierung bzgl. einer anderen Bewertungsfunktion verwendet werden. Außerdem ist es möglich, mehrere Komponenten eines Systems in eine rekursive Optimierungskette anzuordnen und mittels einer MINIMAX Optimierungsstrategie „konkurrierend“ zu testen und gleichzeitig zu optimieren.
Es wurde eine einfache Pick'n'Place Applikation auf Basis von MoveIt! zusammen mit dem AR-Track-Alvar Modul für die Objekt- und Zielerkennung implementiert. Zusammen dem entwickelten Test- und Optimierungssystem wurden Aspekte der Objekterkennung, Umgebungswahrnehmung sowie Handlungsplanung und Aktionsausführung automatisiert getestet. Daraus hat sich unter Anderem ergeben, dass MoveIt! insbesondere Schwächen gegenüber der Umgebungswahrnehmung aufweist. Darüber hinaus wurde auf Basis des Testsystems eine automatisierte Wahl der verfügbaren Planungsalgorithmen in der Standardkonfiguration von MoveIt! durchgeführt.

6.2 Ausblick

Klassifizierung

Eine relativ einfach Erweiterung des Systems wäre analog zu der Fehlerklassifizierungsfunktion eine diskrete Bewertungsfunktion, die neben einem schlechten Durchlauf aufgrund von Toleranzverletzungen eine diskrete Bewertung eines Testlaufs ermöglicht. Auf Basis Gauss'scher logistischer Regression nach [Ras06] ließe sich damit eine automatisierte Analyse des Systems durchführen. Dabei würde man als Heuristik für den nächsten Testpunkt die Minimierung der „Unsicherheit“ des Modells verwenden. Mit dem gelernten Klassifikator ließen sich anhand der diskreten Bewertungsmetrik gezielt Situationen erzeugen, die einer definierten „Schwierigkeit“ entsprechen.

Dynamische Umgebungen

Mit der fortschreitenden Entwicklung von hochautomatisierten/autonomen Robotersystemen werden immer mehr solcher Systeme in (hoch)dynamischen Umgebungen agieren. Beispiele hierfür wäre die verzögerungsfreie Interaktionen mit Menschen, autonome Fahrzeuge, Drohnen, etc. Das erarbeitete Konzept kann durch eine Erweiterung dynamischer Szenen dazu verwendet werden solche Systeme automatisiert zu testen. Durch das generische Konzept wäre es möglich zur Erweiterung der statischen Umgebung ein oder mehrere autonome Systeme in Optimierungselemte zu kapseln und deren Parameter dem Testsystem zur Verfügung zu stellen.

Anhang A

Verwendete Softwareversionen

Paket	**Version**
moveit	0.5.8
pr2_moveit	0.5.7
pr2_gazebo	2.0.2
ar_track_alvar	0.4.1
roslib	1.10.10
rospy	1.10.10
actionlib	1.10.3
pluginlib	1.9.24
gazebo_ros	2.3.5
control_msgs	1.2.0
geometry_msgs	1.10.6
std_srvs	1.10.10
python	2.7.3
numpy	1.8.1
scipy	0.14.0
pyevolve	0.6

Tabelle A.1: Softwareversionen der direkten Abhängigkeiten von ROS-hydro und verwendete Pythonbibliotheken.

Literaturverzeichnis

[BK00] Robert Bohlin and EE Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[BLNZ95] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[Gut01] H-M Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19(3):201–227, 2001.

[Jon01] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[KL00] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[NHQ09] K. Holmström Nils-Hassan Quttineh. Implementation of a one-stage efficient global optimization (ego) algorithm. Technical report, Linköping University, 2009.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2), 2009.

[Ras06] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[RN05] Stuart Russell and Peter Norvig. Ai a modern approach. *Learning*, 2(3):4, 2005.

[Sas02] Michael James Sasena. *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations*. PhD thesis, University of Michigan, 2002.

[Sch98] Matthias Schonlau. *Computer experiments and global optimization.* PhD thesis, University of Waterloo, 1998.

[ŞK09] Ioan A Şucan and Lydia E Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009.

[SKKS12] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *Information Theory, IEEE Transactions on*, 58(5):3250–3265, 2012.

[SMK10] IA Sucan, M Moll, and LE Kavraki. The open motion planning library (ompl), 2010.

[The83] Henri Theil. Linear algebra and matrix methods in econometrics. *Handbook of Econometrics*, 1:5–65, 1983.

[Tou14] Marc Toussaint. Introduction to optimization - constrained optimization. Vorlesung, 2014.

[ZBLN97] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, den 11.08.2014

Kim Peter Wabersich