

# Roboternavigation mit Potenzialfeldern

**Intelligente Robotik WS2023/24**  
**Praktische Arbeit**

Carl Schünemann (Mat.Nr. 00107827)

27. Dezember 2023

# Inhaltsverzeichnis

<b>1 Ziel der Implementierung</b>	<b>1</b>
<b>2 Ausführung der Implementierung</b>	<b>2</b>
2.1 Vorbereitung der Ausführungsumgebung . . . . .	2
2.2 Ausführung des Jupyter Notebooks . . . . .	2
<b>3 Roboterbewegung im Occupancy Grid</b>	<b>5</b>
<b>4 Konfigurationsraum</b>	<b>6</b>
<b>5 Berechnung der Potenzialfelder</b>	<b>8</b>
5.1 Anziehende & Abstoßende Potenziale . . . . .	8
5.2 Wavefront Potenziale . . . . .	9
<b>6 Roboternavigation im Kraftfeld</b>	<b>11</b>
6.1 Berechnung der Gradienten . . . . .	11
6.1.1 Gradienten an Grenzen und Hindernissen . . . . .	12
6.1.2 Behandlung lokaler Maxima . . . . .	12
6.2 Gradientenabstieg . . . . .	13
<b>7 Diskussion</b>	<b>14</b>
7.1 Lokale Minima und Oszillationen . . . . .	14
7.2 Interpolationsartefakte . . . . .	15
<b>Literatur</b>	<b>16</b>

# 1 Ziel der Implementierung

Die vorliegende Arbeit dokumentiert die praktische Umsetzung eines Planungssystems zur Roboternavigation im Rahmen der Vorlesung “Intelligente Robotik” im Wintersemester 2023/24.

Die Implementierung erfolgte in Python, wobei ein Jupyter Notebook als zentraler Einstiegspunkt dient. Ein rechteckiger Roboter variabler Größe navigiert in einem statisch vorgegebenen Occupancy Grid mit beliebig konfigurierbaren Hindernissen von einem Start- zum Zielpunkt. Die kollisionsfreie Roboternavigation wird durch die Transformation der Roboterbewegung in einen dreidimensionalen Konfigurationsraum nach einem Ansatz von Yunfeng und Chirikjian ermöglicht [6]. Die Routenplanung von Start- zu Zielpunkt basiert auf Potenzialfeldern, deren Berechnung sowohl mit anziehenden und abstoßenden Potenzialen als auch dem Wavefront-Algorithmus implementiert wurde. Die Roboternavigation erfolgt durch das Gradientenabstiegsverfahren in den Kraftfeldern der Potenzialfelder. Diverse grafische Darstellungen visualisieren die Berechnungen und Roboternavigation.

Nachfolgendes Kapitel dient als Kurzanleitung zur Ausführung des Programms. Die weiteren Kapitel befassen sich mit dem theoretischen Hintergrund der Implementierung.

## 2 Ausführung der Implementierung

Die Roboternavigation mit Potenzialfeldern wurde in Python als *Jupyter Notebook* implementiert. Das Programm ist nach Vorbereitung der Ausführungsumgebung in sechs sequenziellen Schritten ausführbar.

### 2.1 Vorbereitung der Ausführungsumgebung

Das Jupyter Notebook ist entweder als lokal ausführbar oder ohne benötigte lokale Installationen über *Google Colab* in der Cloud ausführbar.

#### Lokal

Um das Programm lokal auszuführen, muss das [GitHub Repository](#) geklont werden. Das Jupyter Notebook `robot-navigation-potential-fields-local.ipynb` ist der zentrale Einstiegspunkt des Programms. Die Implementierung wurde mit *Visual Studio Code*, kurz *VSCode*, als Ausführungs-umgebung des Notebooks getestet. Dazu wird Python ab Version 3.8 vorausgesetzt (getestet mit Version 3.8.18). Isolierte *Python Environments* können über *Anacoda Navigator* installiert werden. Zur Ausführung eines Jupyter Notebooks wird der *ipykernel* benötigt. Dieses Paket kann entweder nach Öffnen des Notebooks in *VSCode* über das Pop-up installiert werden oder über die Kommandozeile von Anaconda: `conda install -n <environment_name> ipykernel`.

#### Google Colab

Alternativ kann das Notebook ohne lokale Installationen in [Google Colab](#) ausgeführt werden. Hierfür wird nur ein Google Account benötigt.

### 2.2 Ausführung des Jupyter Notebooks

Das Jupyter Notebook unterteilt das Programm in sechs sequenziell auszuführende Schritte. Ein *Ausführungsschritt* entspricht einer Überschrift, die eine auszuführende *Zelle* gruppier. Per Klick auf das Pfeilsymbol links neben der Überschrift wird die Zelle ausgeklappt. Per Klick auf den Knopf links neben der Zelle oder über die Tastenkombination **Strg + Enter** wird der Python Code ausgeführt:

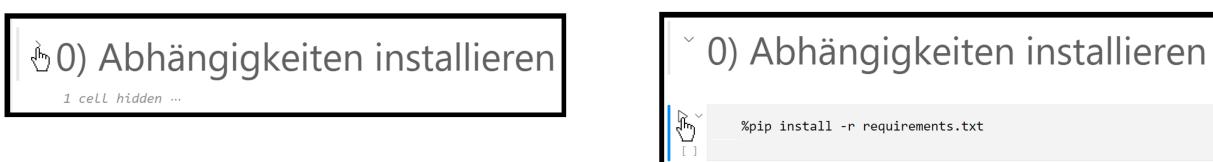


Abbildung 2.1: Nach Aufklappen der Überschrift eines Ausführungsschritts (links) kann der Python Code in der Zelle ausgeführt werden (rechts).

## 0) Abhängigkeiten installieren

Bevor das Python Programm ausgeführt werden kann, müssen alle notwendigen Bibliotheken installiert werden. Im Falle der lokalen Ausführung werden mit `pip install -r requirements.txt` die benötigten Pakete installiert: *Numpy* und *Scipy* unterstützen mathematische Berechnungen während *matplotlib* zur Visualisierung dient. Nach Installation der Pakete müssen VSCode und Anaconda Navigator komplett neu gestartet werden.

Beim Notebook für Google Colab wird zusätzlich das Repository geklont. Ein Neustart wird nicht benötigt.

## 1) Szenario auswählen

Dieser Ausführungsschritt gruppert als Ausnahme mehrere Zellen. Eine Zelle entspricht hier einem vorkonfiguriertem *Szenario*. Dort werden alle Parameter des Programms eingestellt, beispielsweise die Dimensionen des Roboters, dessen Start- und Zielposition und die Position der Hindernisse. Die Bedeutung der Parameter sind als Kommentare dokumentiert, sodass neue Szenarien konfiguriert werden können. Zu beachten ist, dass nur die Parameter der zuletzt ausgeführten Zelle übernommen werden.

```
# Dimensionen Roboter
robot_width = 1 # Breite in X-Richtung
robot_length = 2 # Länge in Y-Richtung

# Rotationsschritte (Grad)
rotation_step = 45

# Start- und Zielpunkt festlegen: (X,Y,Rotationzustand)
# => Rotation in Grad = Rotationzustand * 'rotation_step'
start_point = (2, 13, 0)
goal_point = (13, 3, 0)
current_position=start_point # Initialisierung der aktuellen Position

# Potenzialfunktion (disjunkte Auswahl)
wavefront_potential_function = False
attraction_repulsion_potential_function = True
attraction_weight = 2 # Nur relevant für 'attr._repul._pot._function = True'
repulsion_weight = 0 # Nur relevant für 'attr._repul._pot._function = True'

# Dimensionen Occupancy Grid
occupancy_grid_width = 17 # Breite des Occupancy Grid in X-Richtung
occupancy_grid_length = 17 # Länge des Occupancy Grid in Y-Richtung
occupancy_grid = generate_empty_occupancy_grid(
    occupancy_grid_width=occupancy_grid_width,
    occupancy_grid_length=occupancy_grid_length
)

# Hindernisse im Occupancy Grid:
# Aufgespanntes Rechteck von x bis x + 'obstacle_width' und y bis y + 'obstacle_length'
occupancy_grid = add_obstacle(
    occupancy_grid=occupancy_grid,
    obstacle_width=3,
    obstacle_length=3,
    x=7,
    y=7)

# Berechnung der Plots für 3) und 4) aktivieren bzw. deaktivieren wenn CPU-intensiv
optional_plots = True # Nicht empfohlen für Occu. Grid. w/h > 30 oder rotation_step < 30
gradient_descent_plots = False # Nicht empfohlen für Occu. Grid. w/h > 20 oder rotation_step < 90
```

Abbildung 2.2: Über die Parameter wird die gesamte Programmausführung eines Szenarios konfiguriert.

## 2) Berechnungen durchführen

Dieser Ausführungsschritt führt die in den nachfolgenden Kapiteln ausführlich beschriebenen mathematischen Berechnungen durch. Diese werden für das Gradientenabstiegsverfahren benötigt. Änderungen am Python Code dieser Zelle sind nicht notwendig.

### **3) Plots ausgeben (optional)**

Als einziger Ausführungsschritt, ist die Berechnung und Visualisierung der grafischen Darstellungen optional. Da dieser Schritt sehr rechenaufwändig sein kann, verhindert der im Ausführungsschritts 1) gesetzte Parameter `optional_plots=False` bei bestimmten vorkonfigurierten Szenarien die Ausführung dieser Zelle.

### **4) Gradientenabstieg initialisieren**

Bedingt durch die matplotlib-Bibliothek muss bei animierten grafische Darstellungen die Initialisierung von der Ausführungslogik auf mindestens zwei Zellen aufgeteilt werden. Diese Zelle initialisiert somit das Gradientenabstiegsverfahren und dessen animierte Visualisierung. Wenn im Ausführungsschritt 1) der Parameter `gradient_decent_plots=True` gesetzt wird, werden zusätzliche Plots berechnet. Analog zum Parameter `optional_plots=False` des vorherigen Ausführungsschritts wird dies aufgrund des hohen Ressourcenverbrauchs für gewissen Szenarien nicht empfohlen.

### **5) Gradientenabstieg starten**

Beim Ausführen dieser Zelle wird das Gradientenabstiegsverfahren gestartet. Im Plot des vorherigen Ausführungsschritts wird die Roboternavigation animiert. Um ein weiteres Szenario auszuführen, müssen die Ausführungsschritte ab 1) wiederholt werden.

### 3 Roboterbewegung im Occupancy Grid

Die physikalische Umgebung des Roboters wird diskretisiert durch das *Occupancy Grid*, ein binärer, zweidimensionaler Raum aus der Vogelperspektive. Implementiert mit der Python Bibliothek *Numpy*, hat das Boolean Array `occupancy_grid[Y] [X]` die Länge `occupancy_grid_length`, die Breite `occupancy_grid_width` und den Ursprung links oben. Jede Koordinate ist entweder durch ein Hindernis belegt (`occupancy_grid[Y] [X] == False`) oder frei von Hindernissen (`occupancy_grid[Y] [X] == True`).

Im Occupancy Grid gibt es drei Roboterpositionen: `start_point`, `current_position` und `goal_point`.

```
occupancy_grid =
[[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T],
[T, T, T]]
```

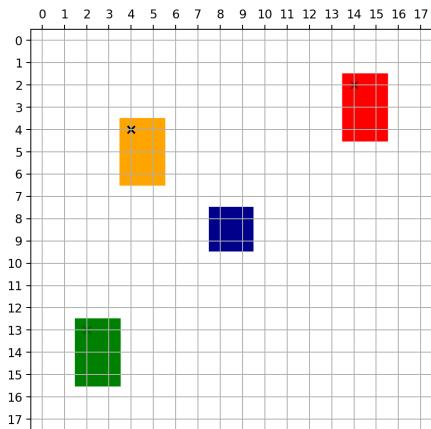


Abbildung 3.1: Ein einfaches Occupancy Grid mit vier Hindernissen, dem `start_point` bei ( $X=2$ ,  $Y=13$ ), der `current_position` bei ( $X=4$ ,  $Y=4$ ) und dem `goal_point` bei ( $X=14$ ,  $Y=2$ )

Die Dimension des Roboters wird durch die Variablen `robot_width` und `robot_length` definiert. Pro Verarbeitungseinheit kann sich der Roboter relativ zum *Ankerpunkt* der aktuellen Position entweder durch eine Translation oder Rotation im Occupancy Grid bewegen:

- **Translation** nach links ( $x-1$ ), rechts ( $x+1$ ), oben ( $y-1$ ) und unten ( $y+1$ )
- **Rotation** um den Ankerpunkt. Bei einer Rotation von  $0^\circ$  liegt der Ankerpunkt in der linken oberen Koordinate innerhalb der Roboterdimensionen.

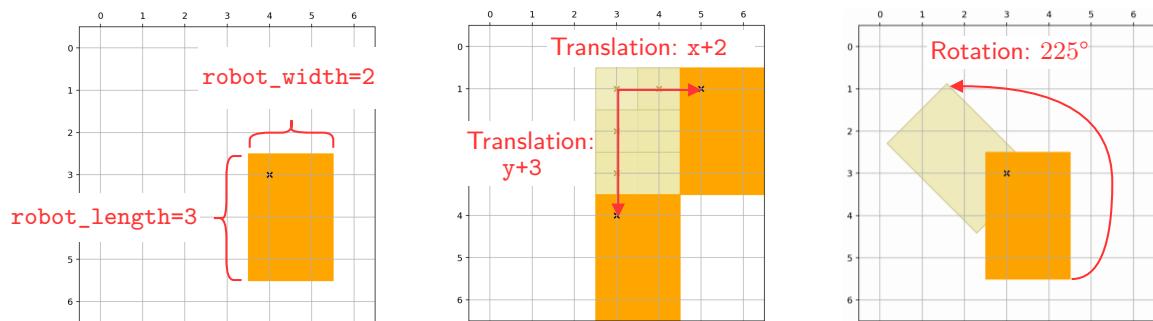


Abbildung 3.2: Bewegung eines Roboters mit variabler Dimension durch Translation und Rotation.

## 4 Konfigurationsraum

Bei einer Roboterlänge und -breite größer als eins können gewisse Punkte im Occupancy Grid nicht erreicht werden, ohne Hindernisse oder Grenzen zu überdecken. Deshalb wird die Roboterbewegung im *Konfigurationsraum* in eine Punktbewegung (`robot_width = robot_length = 1`) transformiert. Dazu wird jedes Hindernis im Occupancy Grid um die Roboterdimensionen erweitert. [4]

Je nach Rotation ändern sich die überdeckten Koordinaten im Occupancy Grid. Um den Verbrauch freier Koordinaten um das Hindernis zu minimieren, unterteilen Yunfeng und Chirikjian die Rotation in diskrete Zustände [6]. Die Variable `rotation_step` gibt als Teiler von  $360^\circ$  an, um wie viel Grad sich der Roboter pro Bewegungsschritt drehen kann, wodurch sich  $\text{rotations} = 360 / \text{rotation\_step}$  unterschiedliche Rotationszustände ergeben.

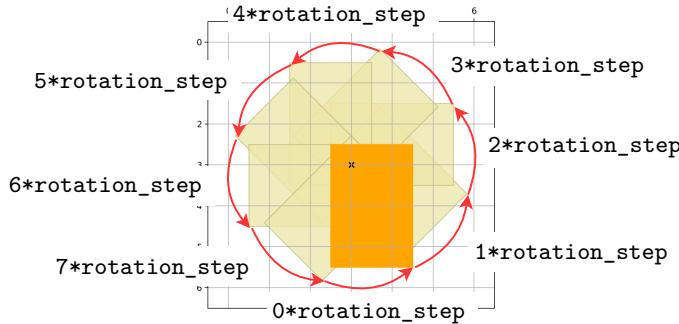


Abbildung 4.1: Für `rotation_step=45` ergeben sich  $(360^\circ \div 45^\circ) = 8$  Rotationszustände.

Für jeden Rotationszustand wird ein *erweitertes Occupancy Grid* generiert. Dazu wird eine Maske des Robotermodells erstellt, rotiert und punktgespiegelt. Pro Rotationszustand wird in einer Kopie des ursprünglichen Occupancy Grids jedes Hindernis sowie jede Grenzkoordinate um diese Maske erweitert.

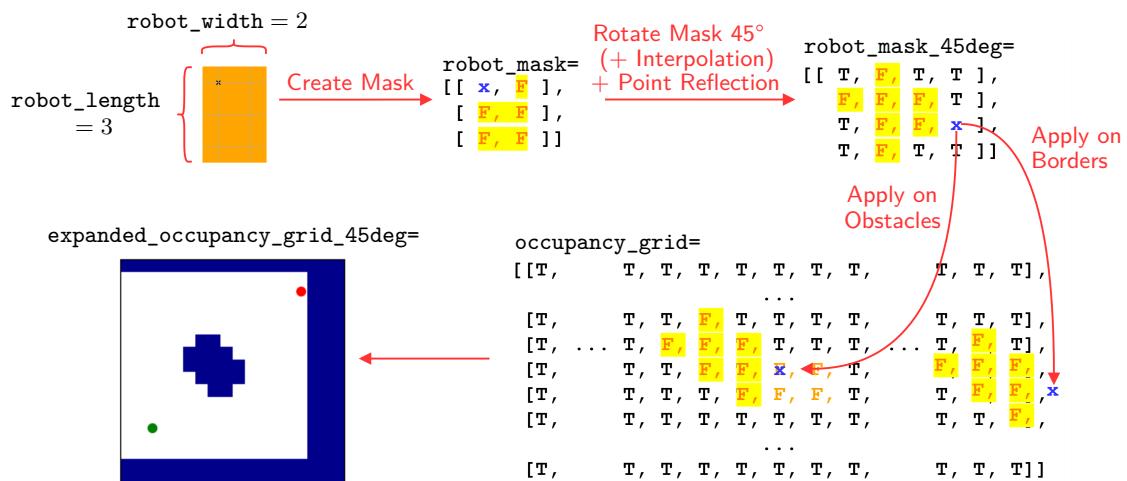


Abbildung 4.2: Erzeugung eines erweiterten Occupancy Grid für eine Rotation um  $45^\circ$

Der Konfigurationsraum `configuration-space[rotation] [y] [x]` mit den Dimensionen fasst die erweiterten Occupancy Grids in der Dimension `[rotation]` zusammen:

- $(\text{rotation} + 1) \% \text{ rotations} \hat{=} \text{Rotation um rotation\_step gegen den Uhrzeigersinn}$
- $(\text{rotation} - 1) \% \text{ rotations} \hat{=} \text{Rotation um rotation\_step im Uhrzeigersinn}$

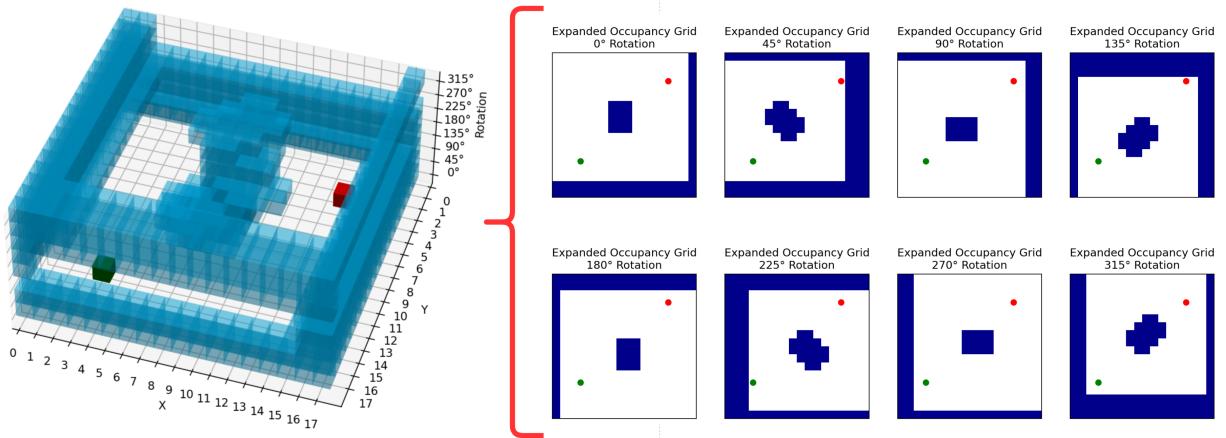


Abbildung 4.3: Die erweiterten Occupancy Grids bilden den Konfigurationsraum

Der berechnete Konfigurationsraum `rotations*occupancy_grid_length*occupancy_grid_width` nach Yunfeng und Chirikjian ermöglicht über die Punktbewegung im dreidimensionalen Raum eine kollisionsfreie Translation und Rotation im Occupancy Grid bei minimalem Verbrauch freier Koordinaten [6]. Die Position des Roboters sowie dessen Rotationszustand im Occupancy Grid wird dabei im Konfigurationsraum über die Dimensionen `rotation`, `y`, `x` referenziert.

# 5 Berechnung der Potenzialfelder

Die Grundlage der Pfadplanung zu einem Zielpunkt bildet in dieser Implementierung die *Potenzialfeldmethode*: Der Konfigurationsraum entspricht einem skalaren *Potenzialfeld*, wobei jede Koordinate eine potenzielle Energie  $U(x, y, \text{rotation})$  besitzt, ausgedrückt durch eine reelle Zahl. Je höher die potenzielle Energie einer Koordinate, desto weiter ist der Punkt auf dem aktuellen Pfad vom Ziel entfernt. Die Berechnung des Potenzialfelds `potential[rotation][y][x] = U(x, y, rotation)` erfolgt mit *Potenzialfunktionen*. [7]

## 5.1 Anziehende & Abstoßende Potenziale

Khatib schlug 1986 vor, das Potenzialfeld in Analogie zu Gravitationsfeldern zu berechnen [3]. Der zu erreichende Zielpunkt wirkt mit einem *anziehenden* (engl. *attractive*) Potenzial auf eine Koordinate  $(y, x)$ . Dieses Potenzial ist in allen Rotationsebenen gleich:

$$U_{Attr, Ziel}(x, y) = \sqrt{(y - y_{Ziel})^2 + (x - x_{Ziel})^2}$$

Bei den getesteten Implementierungsszenarien wurden bessere Ergebnisse festgestellt, wenn das anziehenden Potenzial erst normiert und anschließend gewichtet wird. Mit `attraction_weight = 1` hat das anziehende Potenzial einen Wertebereich von  $[0; 1]$ . Mit `attraction_weight < 1` wird die obere Grenze des Wertebereichs verringert, mit `attraction_weight > 1` vergrößert:

$$U_{Attr, Ziel}(x, y)_{norm} = \frac{U_{Attr, Ziel}(x, y)}{\max\{U_{Attr, Ziel}\}} * \text{attraction\_weight}$$

Die Hindernisse im Konfigurationsraum wirken auf jede Koordinate  $(x, y, \text{rotation})$  mit einem *abstoßendem* (engl. *repulsive*) Potenzial. Dazu zählen auch die Hindernisse aus der benachbarten Rotationsebene  $((\text{rotation}+1) \% \text{rotations})$  sowie  $((\text{rotation}-1) \% \text{rotations})$ . Mit `repulsion_weight = 0` hat das abstoßende Potenzial einen Wertebereich von  $[0; 1]$ . Werte `repulsion_weight > 0` verringern die obere Grenze des Wertebereichs gegen 0: [5]

$$U_{Repul, Hindernis}(x, y, \text{rotation}) = \frac{1}{\text{repulsion\_weight} + \sqrt{(y - y_{Hindernis})^2 + (x - x_{Hindernis})^2 + (\text{rotation} - \text{rotation}_{Hindernis})^2}}$$

Yujiang und Huilin definieren das abstoßende Potenzial einer Koordinate als kleinsten Abstand zu allen Hindernissen [7]:

$$U_{Repul}(x, y, \text{rotation}) = \min_{\forall \text{ Hindernis} \in \text{occupancy\_grid}} \{U_{Repul, Hindernis}(x, y, \text{rotation})\}$$

Die gesamte potenzielle Energie einer Koordinate entspricht der Kombination beider Potenziale. In dieser Implementierung wurden anziehendes und abstoßendes Potenzial gemäß Khalib addiert [3]:

$$U(x, y, \text{rotation}) = U_{Attr, Ziel}(x, y)_{norm} + U_{Repul}(x, y, \text{rotation})$$

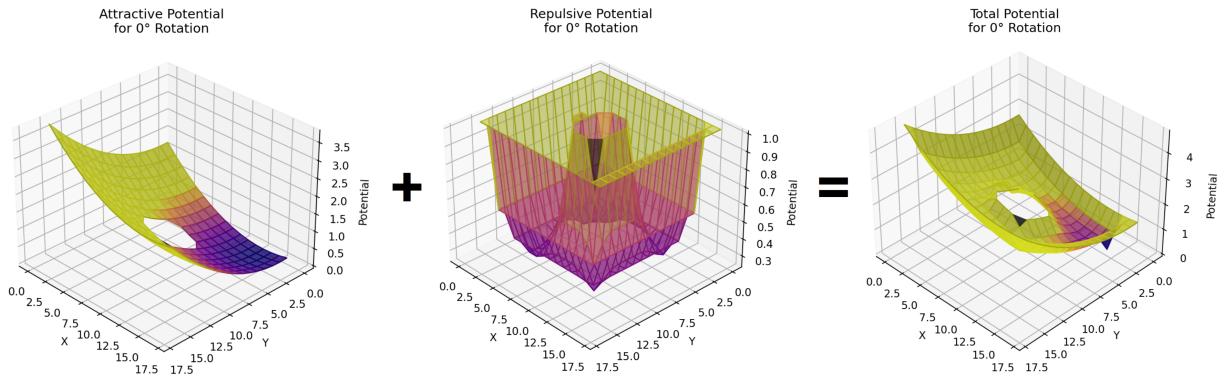


Abbildung 5.1: Die Berechnung des Gestamtpotenzial für die  $0^\circ$ -Rotationsebene des Konfigurationsraums mit `attraction_weight=5` und `repulsion_weight=0`.

## 5.2 Wavefront Potenziale

Choset stellt eine Potenzialfunktion unter Verwendung der Breitensuche ausgehend vom Zielpunkt vor. Beim sogenannten *Wavefront-Algorithmus* entsprechen die Koordinaten des Konfigurationsraums den Knoten der Breitensuche, wobei jeder besuchte Knoten das pro Breitensuchenebene streng monoton steigende Potenzial erhält: [1]

---

### Algorithm 1 Wavefront-Algorithmus

---

```

1: Initialisierung:
2:   Jeder Punkt  $U(x, y, \text{rotation}) = 0$ 
3:   Warteschlange  $Q := \{((x_{\text{Ziel}}, y_{\text{Ziel}}, \text{rotation}_{\text{Ziel}}), 2)\}$ 

4: while  $Q \neq \emptyset$  do
5:    $((x, y, \text{rotation}), \text{potential}) \leftarrow Q$ 
6:    $U(x, y, \text{rotation}) = \text{potential}$ 
7:   Nachbarn  $N := \{(x-1, y, \text{rotation}), (x+1, y, \text{rotation}), \dots, (x, y, (\text{rotation}-1) \% \text{rotations})\}$ 
8:   for  $(x_{\text{Nachbar}}, y_{\text{Nachbar}}, \text{rotation}_{\text{Nachbar}}) \leftarrow N$  do
9:     if  $0 \leq x_{\text{Nachbar}} < \text{occupancy\_grid\_width}$ 
10:      and  $0 \leq y_{\text{Nachbar}} < \text{occupancy\_grid\_height}$ 
11:      and  $\text{computational\_space}[\text{rotation}_{\text{Nachbar}}][y_{\text{Nachbar}}][x_{\text{Nachbar}}] = \text{True}$  then
12:         $((x_{\text{Nachbar}}, y_{\text{Nachbar}}, \text{rotation}_{\text{Nachbar}}), \text{potential} + 1) \rightarrow Q$ 
13:      end if
14:    end for
15:  end while

```

---

## 5 Berechnung der Potenzialfelder

---

Somit breitet sich ausgehend vom Zielpunkt als Quelle das ansteigende Potenzial bildlich als *Wellenfront* im Konfigurationsraum aus. Koordinaten in Hindernisnähe erhalten auf entgegengesetzter Seite der Ausbreitungsrichtung höhere Potenziale.

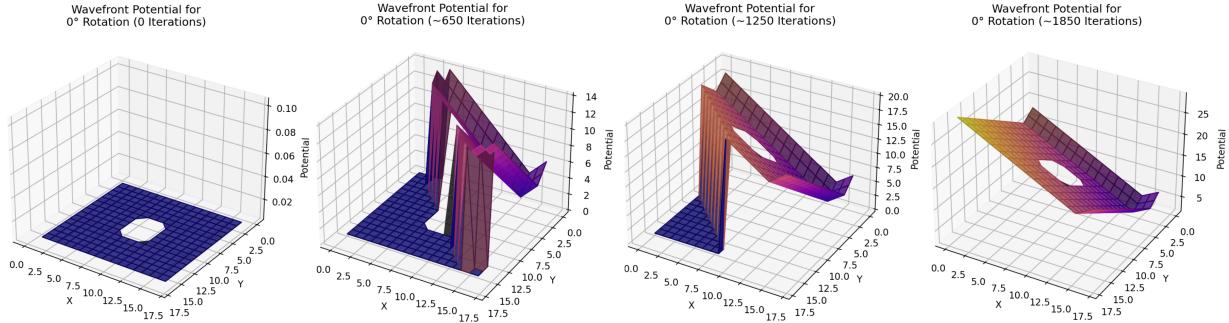


Abbildung 5.2: Die Berechnung des Gestamtpotenzial für die  $0^\circ$ -Rotationsebene des Konfigurationsraums mit `attraction_weight = 5` und `repulsion_weight = 0`.

Unabhängig von der gewählten Potenzialfunktion wird das Potenzialfeld im dreidimensionalen Konfigurationsraum für jede Rotationsebene berechnet. Somit hat das Potenzialfeld die gleichen Dimensionen wie der Konfigurationsraum:

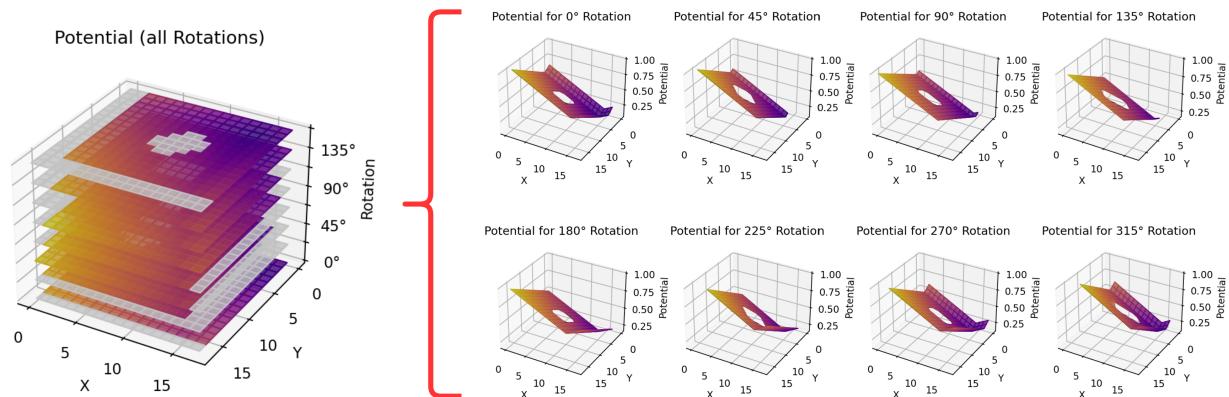


Abbildung 5.3: Das Potenzialfeld wird im dreidimensionalen Konfigurationsraum berechnet.

# 6 Roboternavigation im Kraftfeld

Die Roboternavigation basierend auf der Potenzialfeldmethode entspricht der Bewegung eines Körpers im *Gravitationsfeld* des Potenzialfelds.

## 6.1 Berechnung der Gradienten

Das Gravitationsfeld entspricht dem negativen Gradientenfeld des Potenzialfelds: Auf jede Koordinate im Gravitationsfeld wirkt Gravitationskraft zerlegt in die drei kartesische Komponenten des Konfigurationsraums. Jedes dieser drei Kraftfelder  $F_{x/y/rotation}(x, y, rotation)$  gibt für die jeweilige Dimension des Konfigurationsraums die Größe der Gravitationskraft in der Koordinate  $(x, y, rotation)$  an [3]:

$$F_x(x, y, rotation), F_y(x, y, rotation), F_{rotation}(x, y, rotation) = -\nabla U(x, y, rotation)$$

Um die Gradienten zwischen dem letzten Rotationszustand und  $0^\circ$  zu berechnen, wird die erste und letzte Rotationsebene an das jeweils andere Ende der Dimension potential[rotation] kopiert:

```
potential_padded = np.concatenate([potential[rotations - 1], potential, potential[0]])
```

Die kopierten Rotationsebenen werden nach Berechnung der Kraftfelder entfernt:

```
force_field_rotation, force_field_y, force_field_x = -np.gradients(potential_padded)[1:rotation-1]
```

Somit entspricht  $force\_field\_rotation[rotation] > 0$  einer Kraft in Richtung  $force\_field\_rotation[0]$  und  $force\_field\_rotation[0] < 0$  einer Kraft in Richtung  $force\_field\_rotation[rotation]$ .

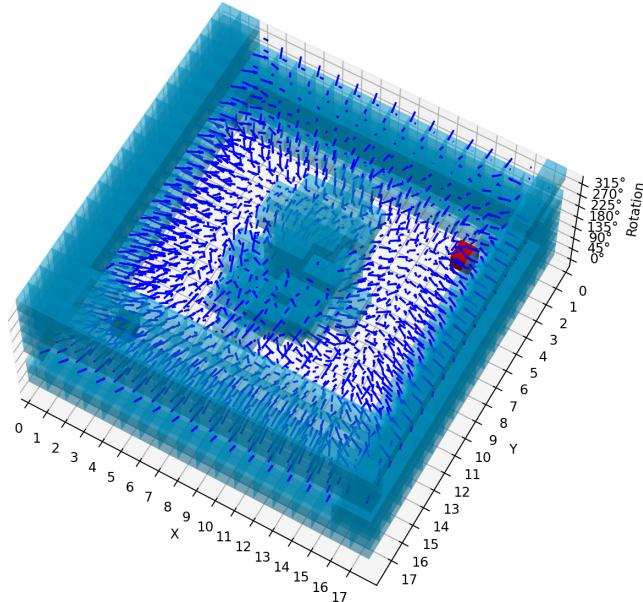


Abbildung 6.1: Darstellung des negativen Gradientenfelds als Kraftpfeile im Konfigurationsraum.

### 6.1.1 Gradienten an Grenzen und Hindernissen

Um die Grenzen des Konfigurationsraums einzuhalten, werden unzulässige Kräfte an den X- und Y-Grenzen auf 0 gesetzt. Durch dieses manuelle Abschneiden der Kräfte können an den Grenzen lokale Minima entstehen.

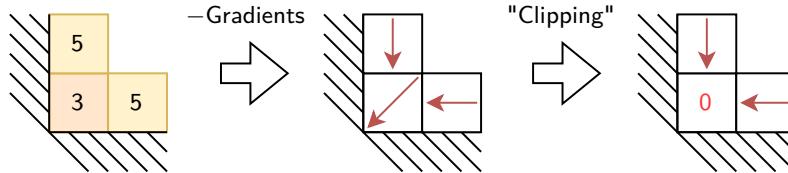


Abbildung 6.2: Das Setzen der Gradienten in Grenznähe auf 0 kann zu lokalen Minima führen.

Im Numpy Potenzialfeld `potential` haben Hindernisse das Potenzial `np.nan`. Somit berechnet `np.gradients()` für jede an ein Hindernis grenzende Koordinate den Gradienten `np.nan`. Um die Gravitationskräfte an Hindernissen analog zu Kräften an Grenzen des Konfigurationsraums zu interpretieren, müssen diese manuell berechnet werden. Sollte die berechnete Kraft einer Dimension in Richtung des Hindernisses zeigen, wird diese ebenfalls auf 0 gesetzt. Nachfolgende Tabelle zeigt dies beispielhaft für Hindernisse im Kraftfeld der X-Achse `force_field_x`:

Koordinate Hindernis	<code>force_field_x[rotation][y][x] = -1*...</code>	= 0, wenn ...
$x + 1$ (rechts)	<code>potential[z, y, x] - potential[z, y, x-1]</code>	$< 0$
$x - 1$ (links)	<code>potential[z, y, x+1] - potential[z, y, x]</code>	$> 0$
$x + 1$ und $x - 1$ (rechts und links)	0	N/A

Wird die Kraft einer Dimension auf 0 gesetzt, wobei die Kräfte der anderen Kraftfelder an dieser Koordinate ebenfalls 0 sind, entsteht ein lokales Minimum oder Plateau.

### 6.1.2 Behandlung lokaler Maxima

Im Unterschied zu einem lokalen Minimum oder Plateau ist bei einem lokalen Maximum das Potenzial der Nachbarn streng monoton niedriger als das der aktuellen Koordinate. In diesen Fällen wird zu einem der beiden Nachbarn ein künstliches Hindernis gesetzt und der Gradient neu berechnet.

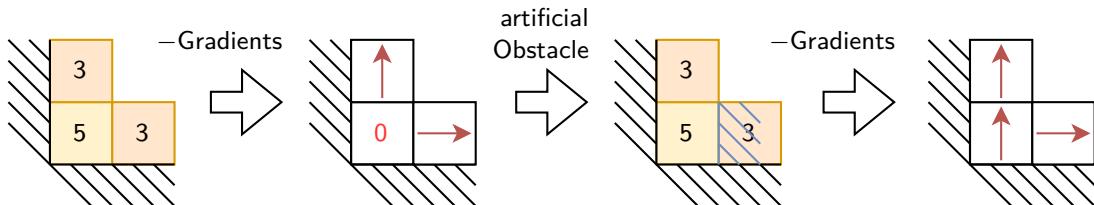


Abbildung 6.3: Lokale Maxima können durch künstliche Hindernisse behoben werden.

## 6.2 Gradientenabstieg

Die berechneten Gradientenfelder ermöglichen die Roboternavigation mit dem *Gradientenabstiegsverfahren*. Pro Verarbeitungsschritt wird die nächste Koordinate basierend auf der betragsmäßig größten Gravitationskraft in  $F_x(x, y, \text{rotation})$ ,  $F_y(x, y, \text{rotation})$  und  $F_{\text{rotation}}(x, y, \text{rotation})$  gewählt, bis eine bereits besuchte Koordinate oder eine Koordinate mit Kräftegleichgewicht  $F_x(x, y, \text{rotation}) = F_y(x, y, \text{rotation}) = F_{\text{rotation}}(x, y, \text{rotation}) = 0$  erreicht wird:

---

### Algorithm 2 Gradientenabstiegsverfahren

---

```

1: Initialisierung:
2:   ( $x_{\text{current}}, y_{\text{current}}, \text{rotation}_{\text{current}}$ ) = start_point
3:   Visited  $V \leftarrow \emptyset$ 

4: while true do
5:   if ( $x_{\text{current}}, y_{\text{current}}, \text{rotation}_{\text{current}}$ ) = goal_point then
6:     return Ziel gefunden
7:   end if
8:   force_field_x/y/rotation[rotationcurrent] [ycurrent] [xcurrent] = 0
9:   (dx, dy, drotation)  $\leftarrow \max(|\text{force\_field}_x/y/\text{rotation}[rotation_{\text{current}}] [y_{\text{current}}] [x_{\text{current}}]|)$ 
10:  if (dx, dy, drotation) = (0, 0, 0) then
11:    return Lokales Minimum oder Plateau
12:  end if
13:  ( $x_{\text{current}}, y_{\text{current}}, \text{rotation}_{\text{current}}$ )  $\leftarrow (x_{\text{current}} + dx, y_{\text{current}} + dy, rotation_{\text{current}} + drotation)$ 
14:  if ( $x_{\text{current}}, y_{\text{current}}, \text{rotation}_{\text{current}}$ )  $\in V$  then
15:    return Lokales Minimum
16:  else
17:    ( $x_{\text{current}}, y_{\text{current}}, \text{rotation}_{\text{current}}$ )  $\rightarrow V$ 
18:  end if
19: end while

```

---

Gemäß der in Kapitel 3 definierten Roboterbewegungen darf pro Verarbeitungsschritt nur eine Translation um eine Einheit entlang der Dimensionen des Konfigurationsraums durchgeführt werden:  $(dx, dy, drotation) \in \{(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1)\}$

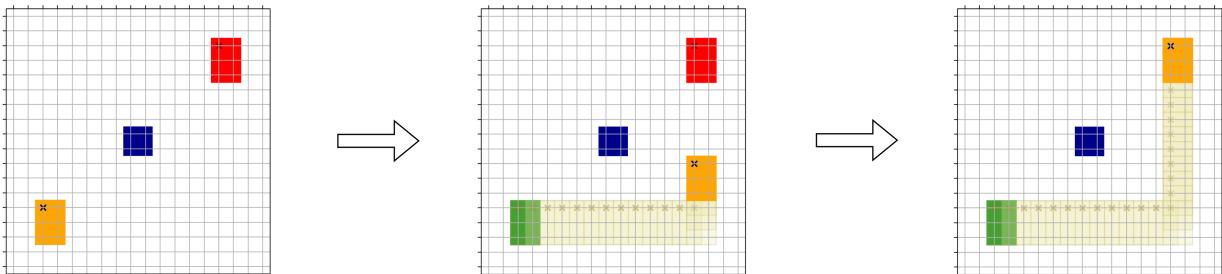


Abbildung 6.4: Roboternavigation über das Gradientenabstiegsverfahren im Kraftfeld des Wavefront-Potenzialfelds.

# 7 Diskussion

## 7.1 Lokale Minima und Oszillationen

Bei konkaven Hindernissen in Potenzialfeldern berechnet mit anziehendem und abstoßendem Potenzial, hat das Gradientenabstiegsverfahren die Eigenschaft zur Konvergenz zu einem lokalen Minimum [7]. Davon betroffen sind Occupancy Grids, deren Hindernisse Polygone mit überstumpfen Winkeln ( $180^\circ \leq \text{Winkel} \leq 360^\circ$ ) bilden. Bekannte Beispiele in der Literatur sind C- und U-förmige Polygone [2, 7]:

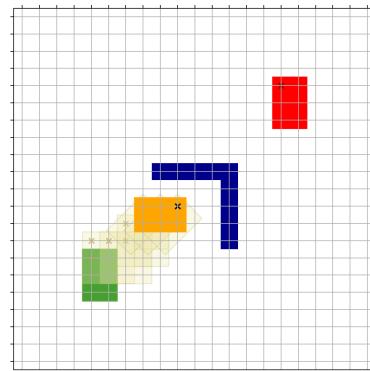


Abbildung 7.1: Für konkave Hindernisse in Potenzialfeldern berechnet mit anziehendem und abstoßendem Potenzial konvergiert das Gradientenabstiegsverfahren zu einem lokalen Minimum.

Weiterhin sind im Kraftfeld von Potenzialfeldern berechnet mit anziehendem und abstoßendem Potenzial zwischen Hindernisengstellen Oszillationen möglich. Der Roboter wechselt bei dieser Art des lokalen Minimums zwischen zwei Koordinaten mit entgegengerichteten Gradienten. Insbesondere kleine Occupancy Grids sind von diesem Effekt betroffen, da die X- und Y-Grenzen als Hindernisse interpretiert werden. Abbildung 7.2 zeigt diesen Effekt anhand des Potenzialfelds aus Kapitel 5.1. Die Erhöhung des Abstandes zwischen den Grenzen des Occupancy-Grid und Hindernissen vermeidet dieses Problem.

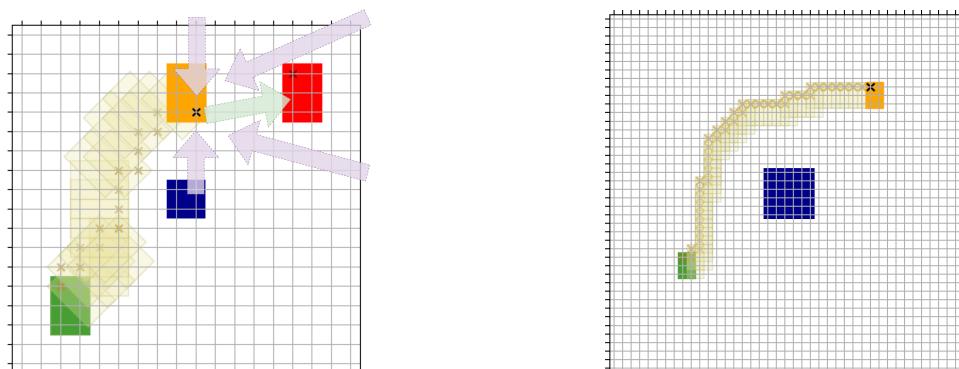


Abbildung 7.2: Bei Engstellen oszillieren die abstoßenden Kräfte zwischen Hindernissen (links), bei hinreichend Abstand nicht (rechts).

Im Gegensatz dazu verhindert die streng monotone Potenzialänderung des Wavefront-Algorithmus die Konvergenz zu lokalen Minima, sofern sich das Ziel an einem für den Roboter physikalisch erreichbaren Punkt befindet.

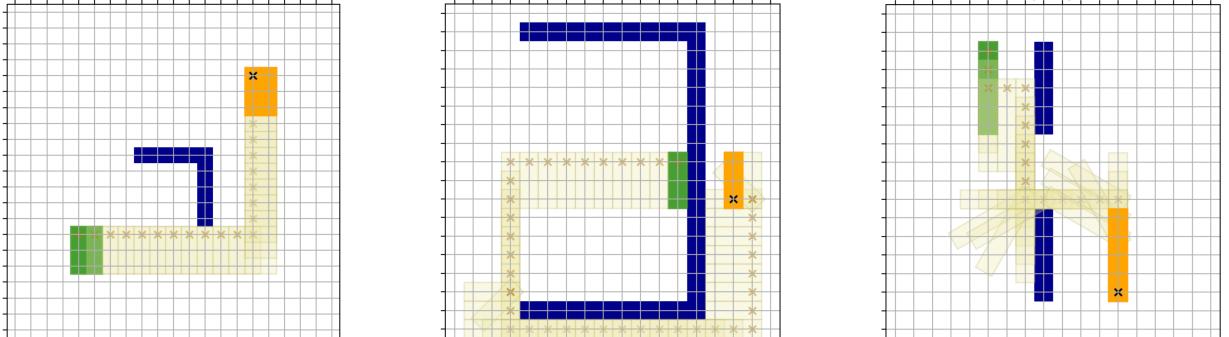


Abbildung 7.3: Das Gradientenabstiegsverfahren in Kraftfeldern der Wavefront-Potenzialfelder konvergiert zum Zielpunkt.

## 7.2 Interpolationsartefakte

Wie in Kapitel 4 beschrieben, wird bei der Berechnung des Konfigurationsraums die Roboterrotation in `rotation_steps` diskretisiert. Dazu wird für das Robotermodell eine Array-Maske über die Matrixrotation `sklearn.rotate()` gedreht.

Insbesondere bei kleinen Roboterdimensionen mit kleinen Rotationswinkeln entstehen hier *Interpolationsartefakte*. Dies zeigt sich bei der Visualisierung der Roboternavigation durch die Überdeckung von Hindernissen mit dem Robotermodell. Yunfeng und Chirikjian empfehlen deshalb, für kleine Rotationswinkel die Roboterdimensionen sowie das Occupancy Grid zu skalieren [7]. Somit wird die Auflösung der Robotermaske erhöht und die Artefakte der Interpolation verringert.

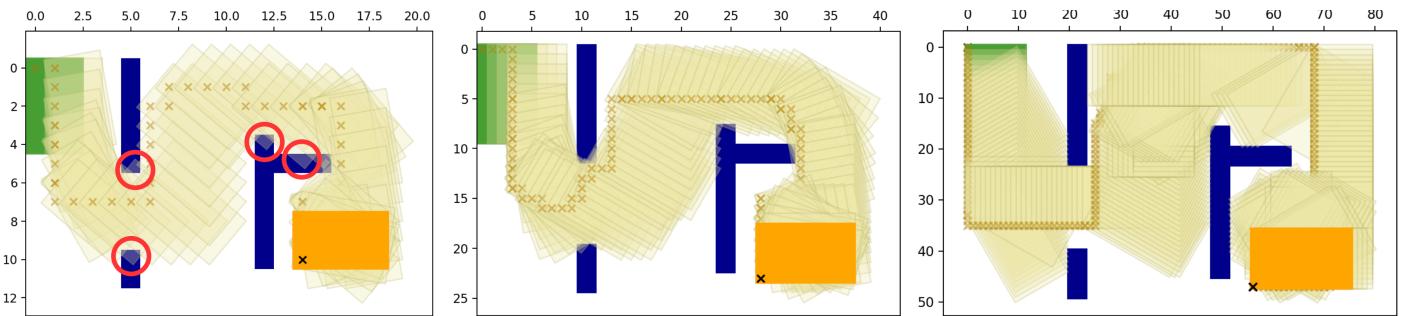


Abbildung 7.4: Die sukzessive Verdopplung der Dimensionen des Robotermodells und Occupancy Grids verringert die Interpolationsartefakte.

# Literatur

- [1] Howie Choset. *Robotic Motion Planning: Potential Functions*. letzter Zugriff am 23.12.2023.  
CMU School of Computer Science: Robotics Institute 16-735. 2007. URL: [https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field\\_howie.pdf](https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf).
- [2] Ayesha Maqbool, Alina Mirza und Farkhanda Afzal.  
„Modified 2-Way Wavefront (M2W) Algorithm for Efficient Path Planning.“  
In: *Int. J. Comput. Intell. Syst.* 14.1 (2021), S. 1066–1077.
- [3] Khatib Oussama. „Real-time obstacle avoidance for manipulators and mobile robots“.  
In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Bd. 2. 1985,  
S. 500–505.
- [4] Eric Roberts. *Robotics: Useful Definitions*. letzter Zugriff am 27.12.2023.  
Stanford University: Department of Computer Science. 1999.  
URL: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/definitions.html>.
- [5] Johann Schweiger. *Intelligente Robotik: Kapitel 5.3 Navigation*. Wintersemester 2023/24.  
Technische Hochschule Ingolstadt: Fakultät Informatik. 2023.
- [6] Yunfeng Wang und Gregory S Chirikjian. „A new potential field method for robot path planning“.  
In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Bd. 2. 2000, S. 977–982.
- [7] Zhang Yujiang und Li Huilin.  
„Research on mobile robot path planning based on improved artificial potential field“.  
In: *Mathematical Models in Engineering* 3.2 (2017), S. 135–144.