

# Anregungen für Bewegungsstrategien

(Vorsicht: Ausdauer und Leidenschaft bei der Programmierung sind Voraussetzung!)

## ■ Kollisionen mit der Wand sind schlecht

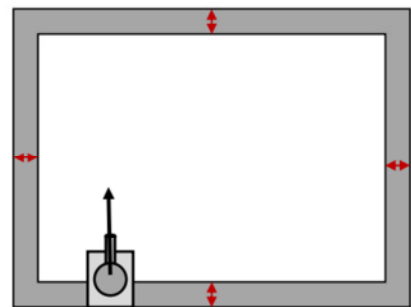
- Sie kosten Energie
- Der Roboter bewegt sich kurz nicht und ist somit leichtes Ziel für Gegner

## ■ Das Spielfeld hat 4 Wände

- In der Regel: gesonderte Betrachtung alle vier Wände

## ■ Achtung:

- Der Körper des Roboters benötigt Platz.
- Daher: Um Kollision zu vermeiden sollte der Roboter nicht näher als **18 Pixel** an die Wand kommen!

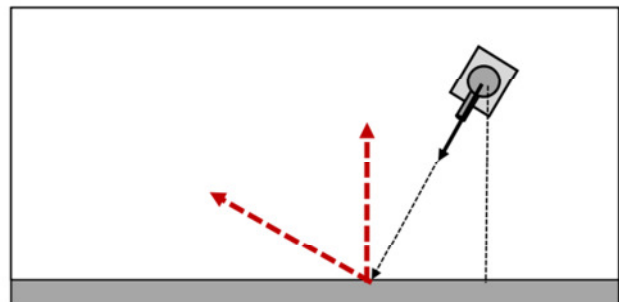


## ■ Kollisionsvermeidung

- Um Energieverlust zu vermeiden

## ■ Idee:

- Wann:  
Wenn Roboter nur noch 18 Pixel von der Wand entfernt ist
- Was:  
Ändere Bewegungsrichtung (z.B. senkrecht von der Wand weg oder in einem anderen Winkel)



## ■ Überlegungen:

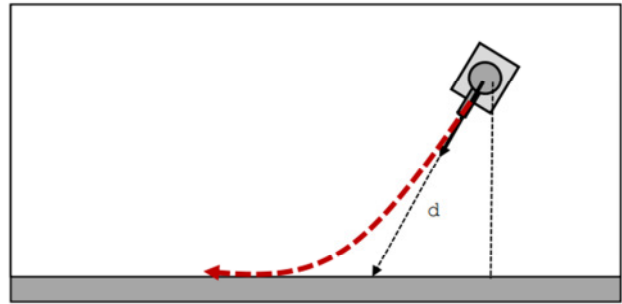
- Man untersucht am besten alle 4 Wände getrennt.  
Zudem auch Annähern an die Wand von links bzw. von rechts.
- Hilfreiche Methoden:  
`getHeading()`, `getX()`, `getY()`, `battleFieldWidth()`, `battleFieldHeight()`
- Zu ermitteln:
  - Mit welchem Winkel nähert sich Roboter der Wand an?
  - Wie weit muss er sich drehen, um im gewünschten Winkel weiter zu fahren?

## ■ „Wall-Smoothing“

- Kollision mit Wand und „Still-Stehen“ zu vermeiden
- Besser geeignet für AdvancedRobots, kann aber auch mit Robots gemacht werden

## ■ Idee:

- Wann:  
Wenn Fahrtweg zu einer Wand kleiner als bestimmter Wert ist
- Was:  
Die Bewegungsbahn soll dann in einer Kurve verlaufen, die sich quasi an die Wand anschmiegt



## ■ Informationen:

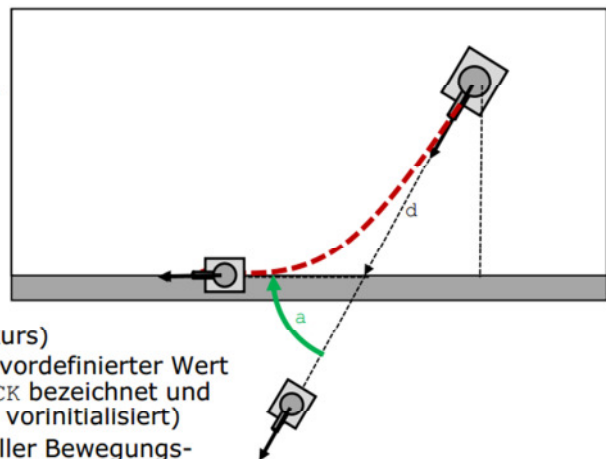
- Es wird der Abstand  $d$  vom Roboter in Fahrtrichtung zur Wand benötigt. Dieser wird am besten in einer separaten Methode ermittelt. Dabei kann auch gleich der Abstand nach hinten zur nächsten Wand ermittelt werden, da dieser auch oft hilfreich sein kann

## ■ „Wall-Smoothing“

- Wir zeigen hier eine simple Strategie, die nur vom Abstand  $d$  und dem Auftreffwinkel des Roboters abhängt
- Es gibt kompliziertere Ansätze, die z.B. auch die Position des Gegners berücksichtigen

### Vorgehen:

- Berechne Abstand  $d$  auf der Bewegungsbahn zur Wand (siehe Exkurs)
- Falls  $d$  kleiner ist als ein bestimmter, vordefinierter Wert (diese Variable wird oft als `WALL_STICK` bezeichnet und auf einen Wert zwischen 120 und 160 vorinitialisiert)
- Ermittle den Winkel  $a$  zwischen aktueller Bewegungsrichtung und Zielrichtung
- Nun implementiere eine While-Schleife:  
Solange der Abstand zur Wand noch nicht kleiner 18 Pixel ist:  
Drehe Roboter noch ein wenig in Richtung Zielrichtung



**Wichtig:** Separate Behandlung aller 4 Wände

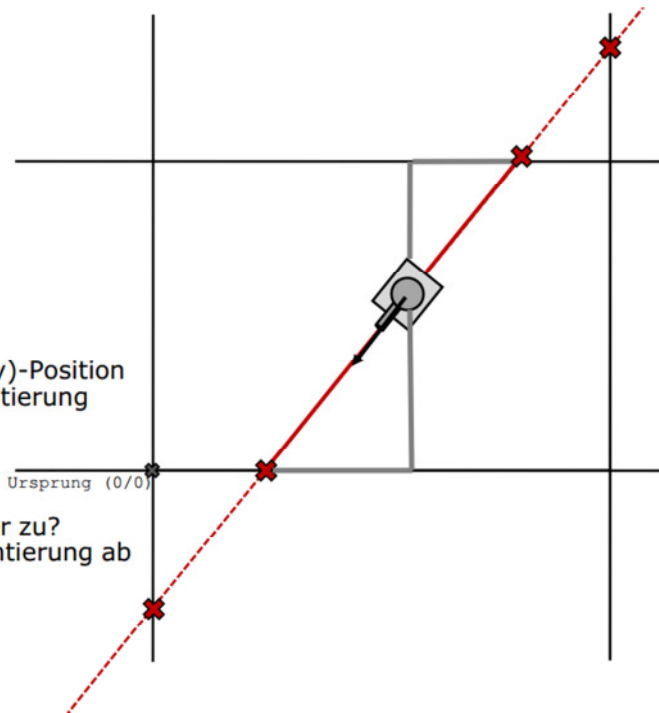
```
double a = ...;
double distanceToWall = ...;
while(distanceToWall > 18) {
    setTurnRight(a*0.1)
}
```

**Vorsicht:** Diese Strategie eignet sich am besten für Advanced Robots!

- Berechnung des Abstandes nach vorne und hinten zur nächsten Wand

- Ansatz:

- Wichtig ist hierbei die aktuelle  $(x,y)$ -Position des Roboters und seine abs. Orientierung
  - Viele Fallunterscheidungen nötig.
  - Auf welche Wand fährt der Roboter zu? Dies hängt nicht nur von der Orientierung ab sondern auch von seiner Position!



Quelle: Robocode Seminar der Universität Erlangen

<https://www5.cs.fau.de/lectures/ss-11/problemorientiertes-programmieren-robocode-robocode/>

**Viel Erfolg !**