Pad planning voor mobiele robots

Bepalen van pad in **bekende omgeving** voor mobiele robots

Voor meeste taken is **pad** zelf niet belangrijk, alleen `beginpositie` en `eindpositie`

Vereisten voor een pad:

- `effectief`: eind- of doelpositie wordt bereikt
- `efficiënt`: niet al te lang
- `collision-free`: zonder botsingen

Pad planning probleem

Gegeven:

- robot
- in `bekende` omgeving
- met `statische` obstakels

Bereken:

effectief, efficiënt en collision-free pad van beginpositie naar eindpositie

Vrijheidsgraden van robot

Elke positie van robot kan beschreven worden door een aantal `onafhankelijke` **parameters**:

aantal **vrijheidsgraden** of **degrees of freedom DOF**

Voorbeelden:

- `2D-plotter`: pen kan in twee richtingen bewegen $\longrightarrow$ twee coördinaten zijn genoeg om positie aan te geven: (x, y) $\longrightarrow$ **2-dof**
- `auto`: niet alleen positie, maar ook oriëntatie $\longrightarrow$ **3-dof**
- `arm`: schouder 3-dof, elleboog 1-dof, pols 3-dof $\longrightarrow$ **7-dof**

Pad plannen in configuration space $\mathcal{C}$

Er zijn veel verschillende robots

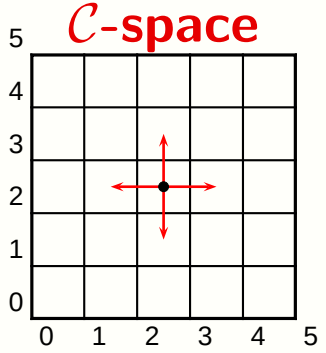
Om te zorgen dat voor al deze verschillende robots algemene pad planning methodes ontwikkeld kunnen worden:

Configuration space $\mathcal{C}$

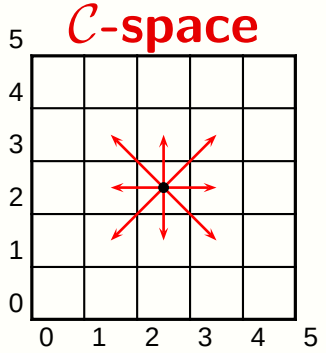
- Elke robot wordt gerepresenteerd door een `bewegend punt`
- Een **configuration q** van een robot is specificatie van zijn toestand (positie) in dofs (**q** $= (x, y, \theta)$ voor auto)
- De **configuration space** $\mathcal{C}$ bestaat uit alle mogelijke toestanden **q**

Configuration spaces $\mathcal{C}$ van 2D-plotters

`2D-plotter`:
directe besturing van x- **of** y-positie
in $\mathcal{C}$-**space**:
4 richtingen met directe buren



`2D-plotter`:
directe besturing van x- **en** y-positie
in $\mathcal{C}$-**space**:
8 richtingen met directe buren



Simplificaties voor 2D-plotter

- `Configuratie ruimte` voor 2D-plotter bestaat uit alle posities in (x, y) vlak die we kunnen bereiken
- **Pad plannen** voor 2D-plotter gewoon in (x, y) vlak
- **Obstakels** worden met zwarte vakjes aangegeven
- `Representeren` van omgeving kan met zwarte en witte vakjes

Toepassingen van pad planning voor 2D-plotter

- pad planning met `grafen`
- pad planning met `potentiaal velden`

Pad planning met grafen

Gegeven: beginpositie en eindpositie

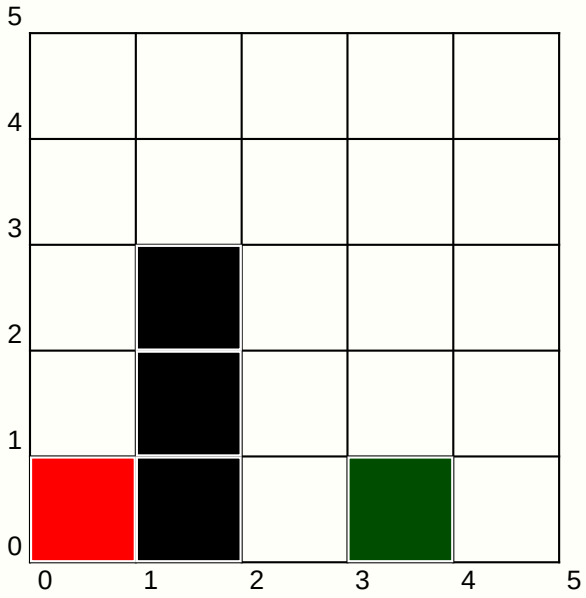
Gevraagd: `kortste pad`

- Graph search
- Depth First Search (DFS)
- Breadth First Search
- Best First Search (A*)

Plattegrond

De robot (**rode vakje**) moet naar het doel (`groene vakje`) in omgeving met obstakels (zwarte vakjes)

Hij kan alleen horizontale of verticale stappen nemen



startpositie is $(0, 0)$
`doelpositie` is $(3, 0)$

Graph search

Voor de omgeving wordt een **boom** gebouwd met:

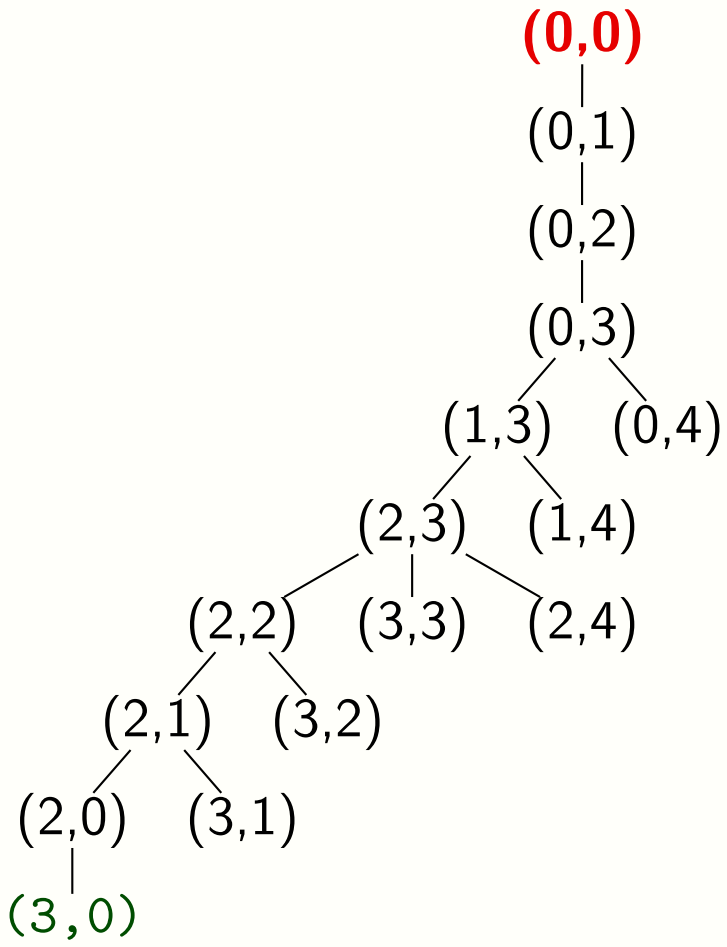
- `start positie` is wortel
- de `bereikbare posities` zijn knopen op volgende niveau

In algoritme worden verzamelingen **OPEN**, **CLOSED** en **M** gebruikt:

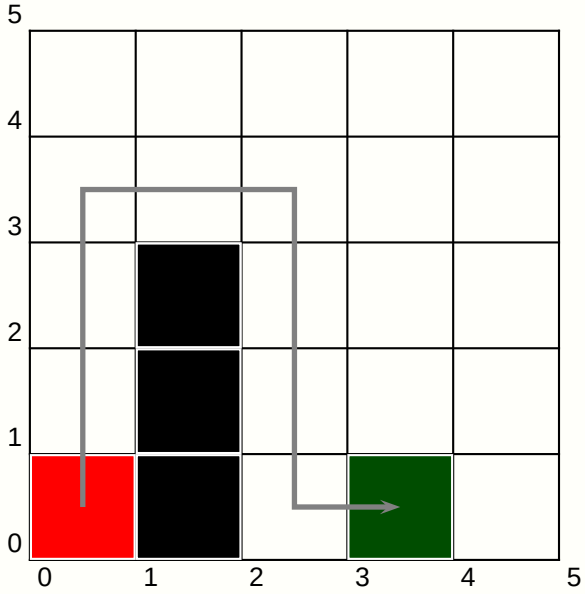
- **CLOSED** bevat de knopen die al bezocht zijn
- **OPEN** bevat de knopen die nog niet bezocht zijn
- **M** bevat de knopen die vanaf huidige knoop bereikbaar zijn

Graph Search algoritme

```
OPEN = {s}
CLOSED = {}
found = false
while (OPEN != 0 and not found) do
    kies knoop n uit OPEN
    haal n uit OPEN en zet in CLOSED
    als n doelknoop
        found = true
    anders
        bepaal aangrenzende knopen M niet CLOSED en OPEN
        voeg M aan OPEN toe
end while
```



zuid, oost, noord, west



Graph search algoritme

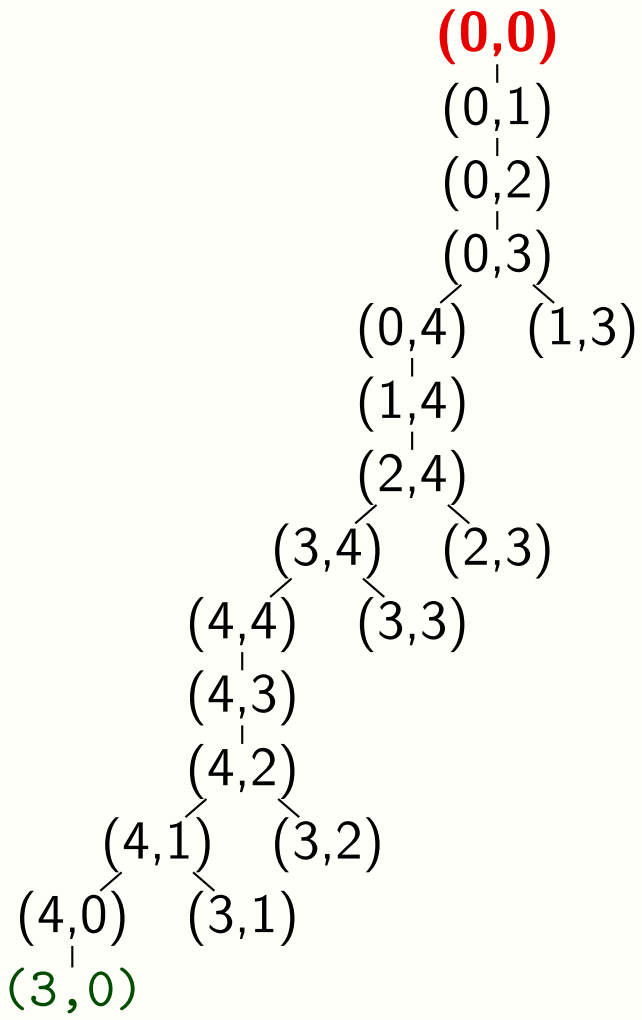
- Het **graph search** algoritme bepaalt een `pad` van beginpositie tot eindpositie, maar niet altijd **kortste**
- Op elke positie wordt een **aangrenzende** nog niet bezochte positie gekozen
- Als er geen aangrenzende nog niet bezochte positie gevonden kan worden, moet `teruggelopen` worden door de boom (**backtracking**)
- Er wordt net zo lang gezocht tot de eindpositie gevonden is, of tot alle posities doorzocht zijn
- `backtracking` gebeurt op elk einde van een tak

Keuze van aangrenzende positie

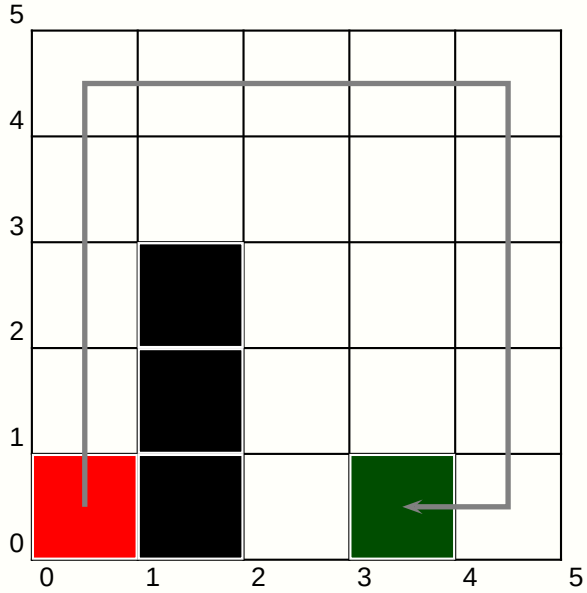
Keuze van aangrenzende positie bepaalt hoe goed het algoritme werkt

- In vorige voorbeeld gekozen: `zuid, oost, noord, west`
- Beter is keuze te laten afhangen van **doelpositie**

Wat als we in het voorbeeld gekozen hadden: `west, noord, zuid, oost`?



west, noord, zuid, oost



Verschillende graph search algoritmen

- Graph search: **OPEN** is verzameling
- Depth First Search (DFS): **OPEN** is stack
- Breadth First Search: **OPEN** is queue
- Best First Search: Gebruik van `heuristische` regels
  A* valt hieronder en berekent kosten functie

Het gaat erom het juiste `oost`, `zuid`, `noord` dan wel `west` vakje te kiezen

A* algoritme

A* gebruikt de `afstand` tussen de huidige positie en de doelpositie en kiest het vakje met de kortste afstand

Het berekent voor elke **node n**:

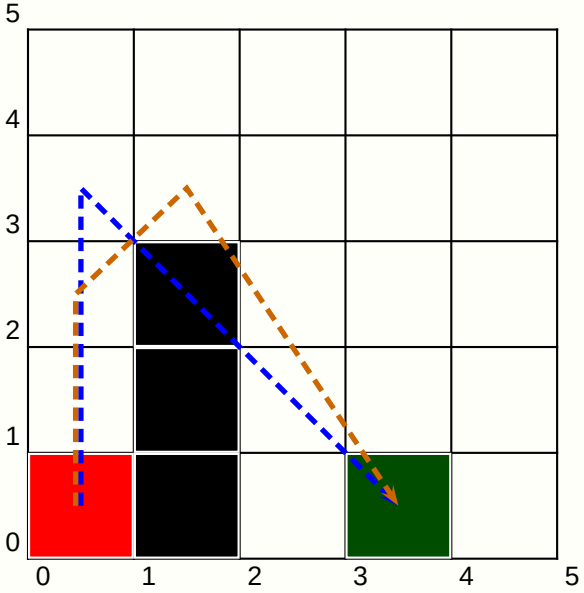
- `g(n)`: de kosten van de start node tot de huidige node **n**
- `h(n)`: de kosten van de huidige node **n** tot de doel node
- `f(n)` = `g(n)` + `h(n)` wordt berekend voor **elke aangrenzende** node
- de node met de **laagste kosten** wordt gekozen
- **afstand** tussen twee nodes is de afstand tussen het centrum van vakjes

A* algoritme

```
OPEN = {s}
CLOSED = {}
found = false
while (OPEN != 0 and not found) do
    kies knoop n met laagste kosten functie uit OPEN  <======
    haal n uit OPEN en zet in CLOSED
    als n doelknoop
        found = true
    anders
        bepaal aangrenzende knopen M niet CLOSED en OPEN
        voeg M aan OPEN toe
end while
```

Voorbeeld A* met diagonale stappen ook mogelijk

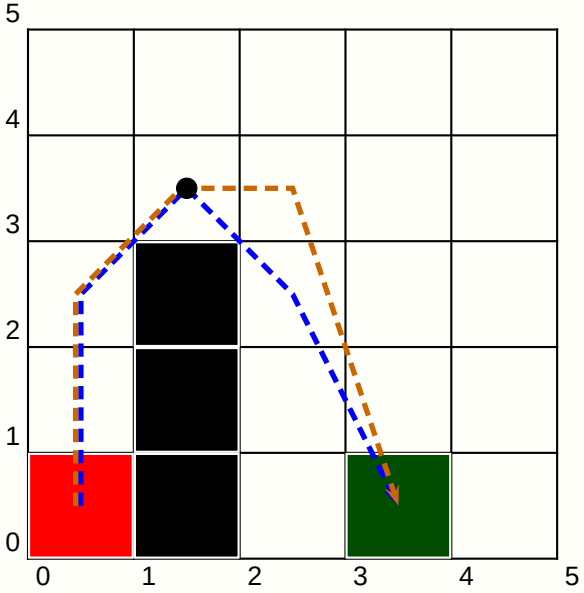
laagste kosten



start positie **(0,0)**
aangrenzend positie (0,1)
aangrenzend positie (0,2)
`kiezen tussen` (0,3) en (1,3)
f(0,3)$= 3 + \sqrt{3^2 + 3^2} = 7.2$
f(1,3)$= 2 + \sqrt{2} + \sqrt{2^2 + 3^2} = 7.0$
`kies` (1,3)

Voorbeeld A* met diagonale stappen ook mogelijk

laagste kosten



gebleven op positie (1,3)

`kiezen tussen` (2,2), (2,3) en ...

f(2,2)$= 2 + \sqrt{2} + \sqrt{2} + \sqrt{1^2 + 2^2} = 7.1$

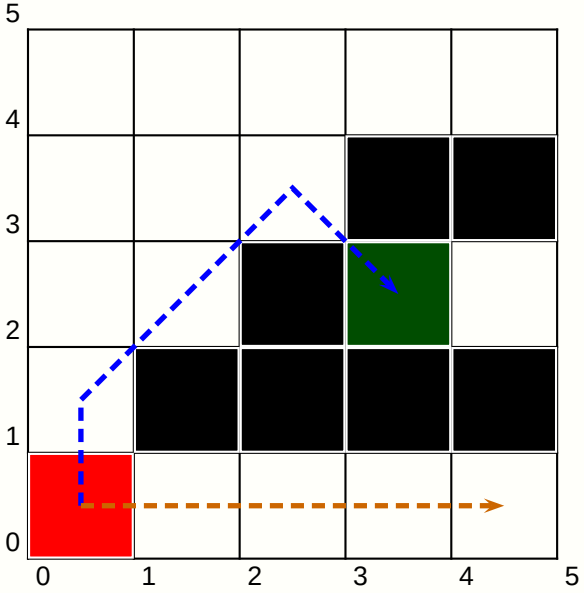
f(2,3)$= 2 + \sqrt{2} + 1 + \sqrt{1^2 + 3^2} = 7.6$

`kies` (2,2)

Dead end

Wat gebeurt er met **A*** als we doodlopen?

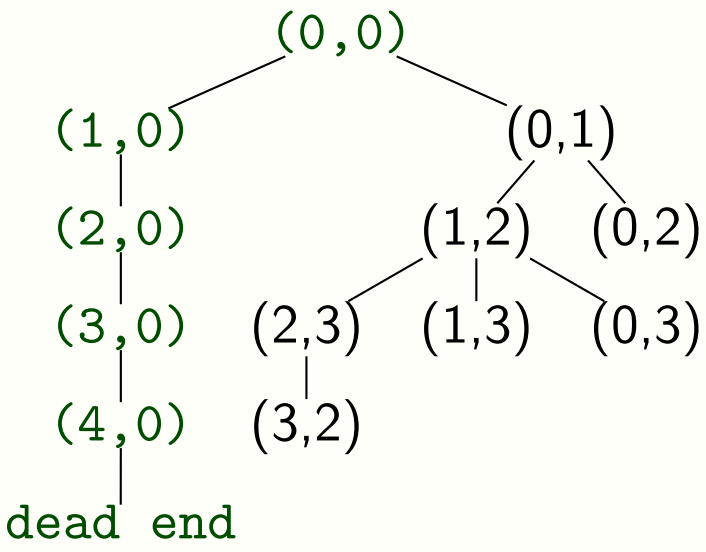
kortste pad loopt dood



We hebben dus mechanisme nodig zodat:

- ander pad gezocht wordt als gekozen pad doodloopt
- doodlopende paden gemeden worden

Dead end opgevangen met back tracking in boom

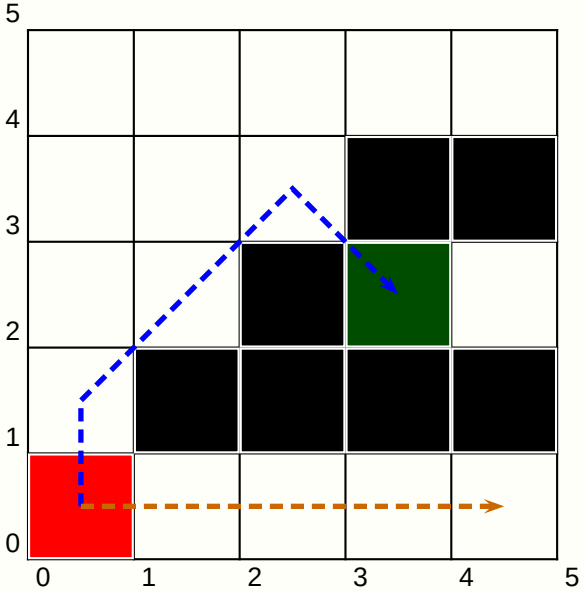


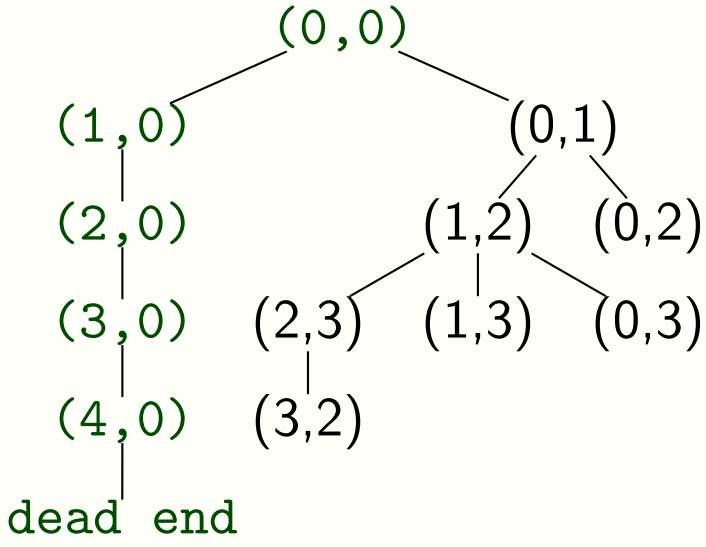
Groene knopen zitten in CLOSED
Zwarte knopen zitten in OPEN

Als **dead end** bereikt, ga terug naar de start node en kies volgende aangrenzende knoop
Als **doel positie** bereikt, ga via de parent knopen terug naar de start knoop op complete pad te krijgen

Pad gevonden bij dead end

kortste pad loopt dood





Padplanning met potentiaal velden

Keuze van pad bepaald door **potentiaal veld** die bestaat uit de som van:

- `aantrekkende kracht` die robot naar **doel** trekt
- `afstotende kracht` die robot van **obstakels** afduwt

$$U_{\texttt{som}}(x, y) = U_{\textbf{doel}}(x, y) + U_{\textbf{obstakel}}(x, y)$$

met (x, y) de huidige positie

De robot beweegt `naar beneden` wat bepaald wordt door de **gradient**

De **gradient** van de functie $U_{\texttt{som}}(x, y)$ op positie (x, y) is de richting waarin de functie het meest varieert. De grootte van de gradient zegt iets over hoe stijl de functie verloopt

Voorbeeld van gradient

De **gradient** van de tweedimensionale functie $U_{\texttt{som}}(x, y)$ is de vector bestaande uit de `partiële afgeleiden` van U in (x, y):

grad $U(x, y) = \nabla U(x, y) = (\frac{\delta U}{\delta x}, \frac{\delta U}{\delta y})$

`Rekenvoorbeeld`

$U(x, y) = 0.5(x - 10)^2 + 0.5(y - 10)^2$

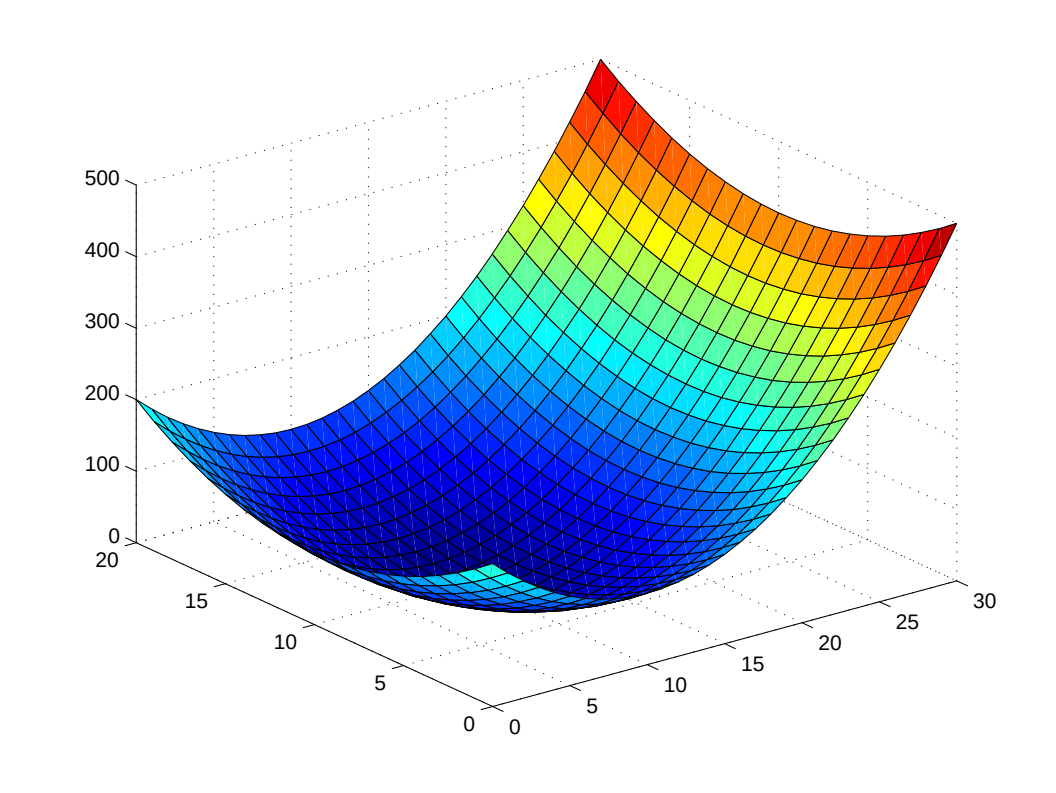
$\nabla U(x, y) = (x - 10, y - 10)$

Gradient in $(20, 5)$: $\nabla U(20, 5) = (20 - 10, 5 - 10) = (10, -5) \Rightarrow$

Partiële afgeleide naar x is groter dan nul: `stijgend` $\rightarrow$ **terug**

Partiële afgeleide naar y is kleiner dan nul: `dalend` $\rightarrow$ **verder**

$U(x, y)$ **met** $\nabla U(20, 5) = (10, -5)$



Langs **gradient**: van punt $(20, 5)$ naar $(19, 6)$

Aantrekkende kracht bepaald door doel

`Aantrekkende` potentiaal veld

$$U_{\textbf{doel}}(x, y) = \frac{1}{2} k_p \Big((x - x_{\textbf{d}})^2 + (y - y_{\textbf{d}}) \Big)^2$$

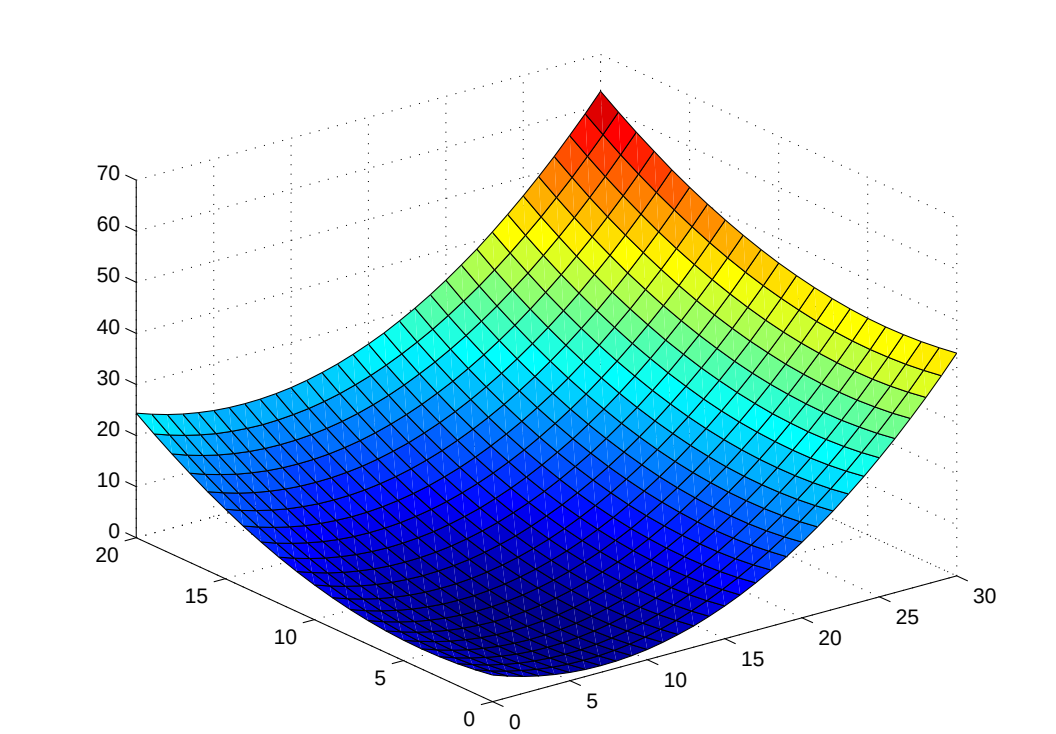
met $(x_{\textbf{d}}, y_{\textbf{d}})$ de **doel**positie

Voor elke waarde van (x, y) wordt $U_{\textbf{doel}}(x, y)$ berekend

Hoe verder (x, y) van **doel** verwijderd, hoe groter $U_{\textbf{doel}}(x, y)$

Langs **gradient** naar doel lopen

Aantrekkende kracht $U_{\textbf{doel}}(x, y)$ bepaald door doel



$\textbf{doel} = (x_{\textbf{d}}, y_{\textbf{d}}) = (7, 4)$

Afstotende kracht bepaald door obstakels

`Afstotend` potentiaal veld

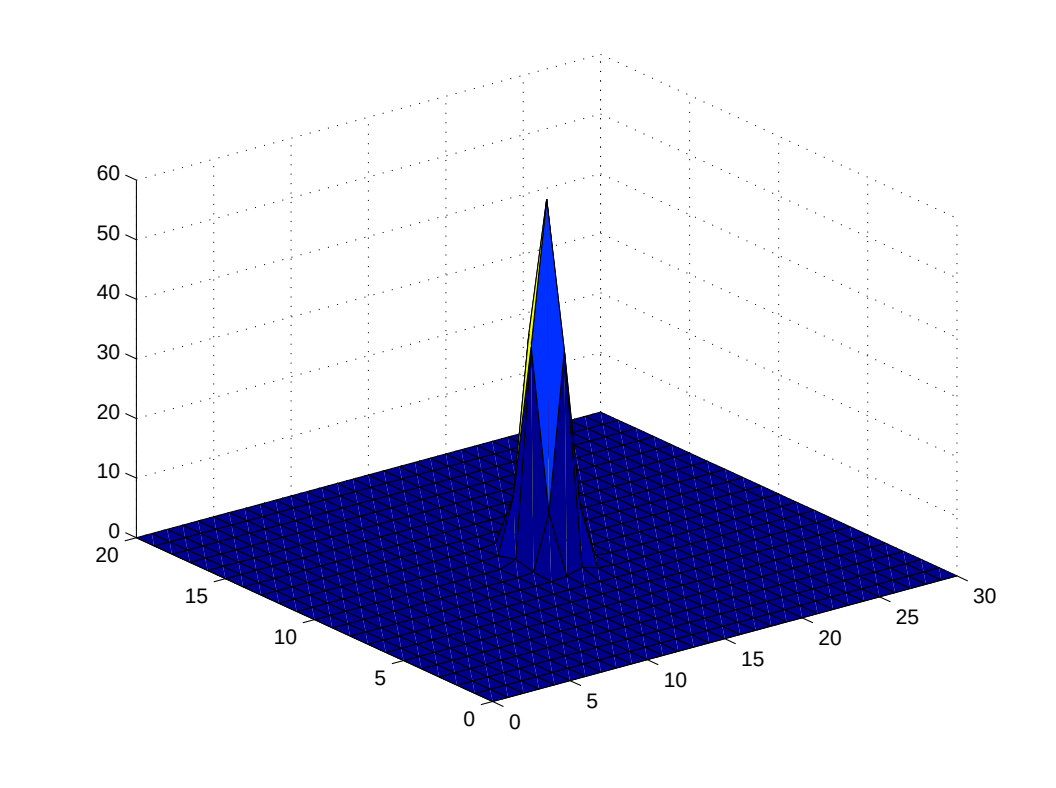
$$U_{\textbf{obstakel}}(x,y) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 & \text{als } \rho \leq \rho_0 \\ 0 & \text{als } \rho > \rho_0 \end{cases}$$

met ρ de afstand tot **obstakel** en ρ_0 `limiet afstand` waarbinnen potentiaal veld invloed heeft

Obstakel heeft alleen in vloed als (x, y) vlak in de buurt

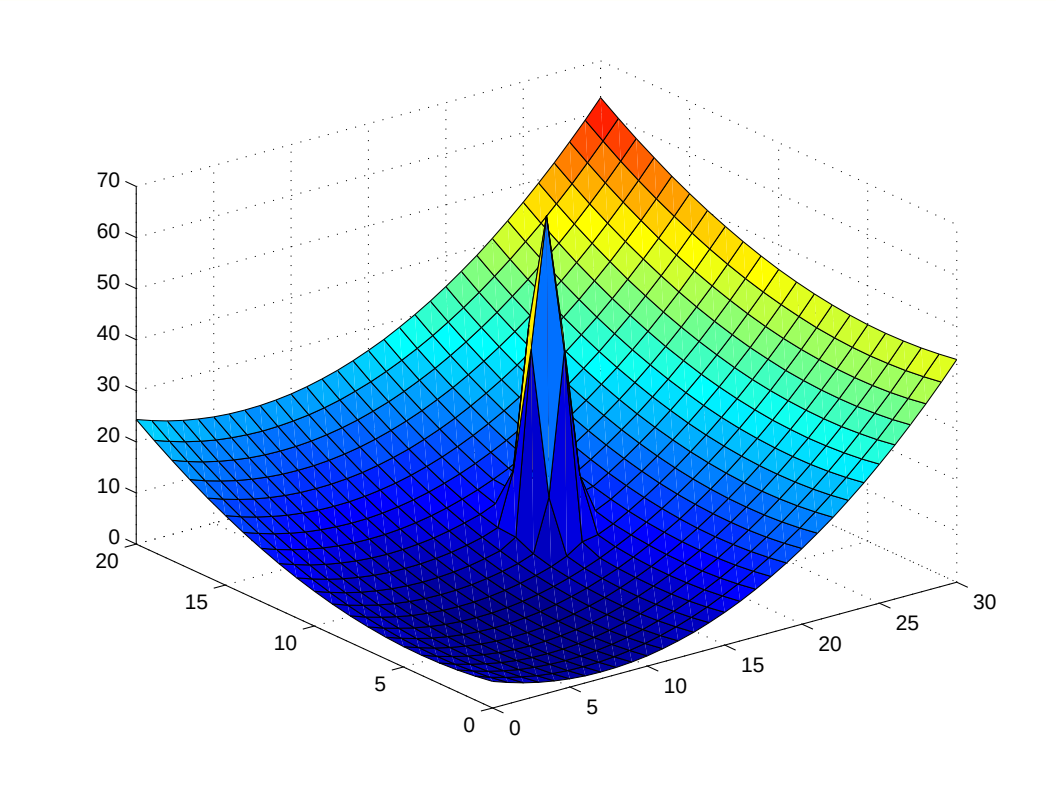
Als (x, y) ver van **obstakel** verwijderd, dan geen invloed

Afstotende kracht $U_{\textbf{obstakel}}(x, y)$ bepaald door obstakel



obstakel $= (x_{\textbf{o}}, y_{\textbf{o}}) = (15, 10)$

Som van aantrekkende en afstotende kracht



$$U_{\texttt{som}}(x, y) = U_{\textbf{doel}}(x, y) + U_{\textbf{obstakel}}(x, y)$$

Door potentiaal veld heen lopen langs gradient

Beweging langs gradient van **potentiaal veld** om `obstakels` heen naar `doel`

Omgeving moet gemodelleerd worden door middel van functie $U_{\texttt{som}}(x, y)$ waarin niet alleen **doel**positie maar ook meerdere **obstakels**:

$$U_{\texttt{som}}(x, y) = U_{\textbf{doel}}(x, y) + \sum U_{\textbf{obstakel}}(x, y)$$

In de praktijk wordt berekening alleen uitgevoerd voor **obstakels** dichtbij, waarbij dus aanname bestaat dat **obstakels** ver weg geen invloed hebben

De robot beweegt **down hill** in de richting van de `locale gradient`

Een doelpositie met twee obstakels