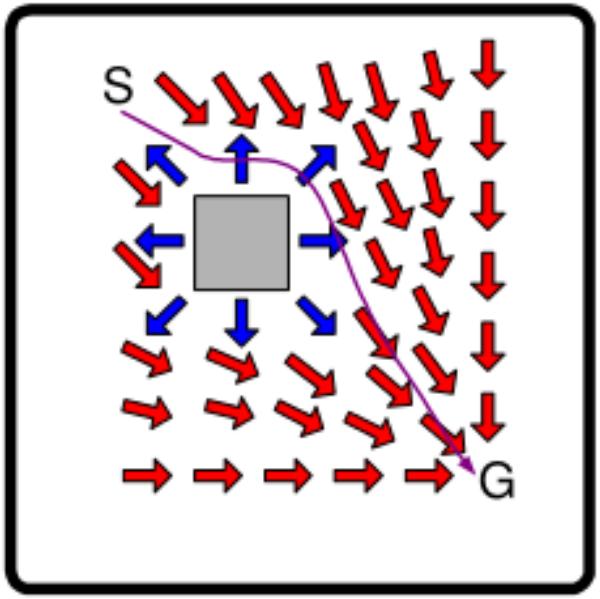
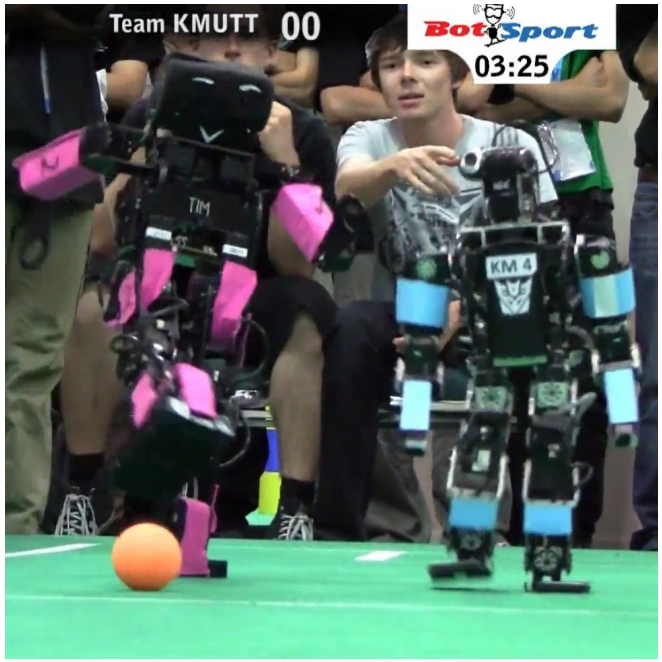
Freie Universität Berlin

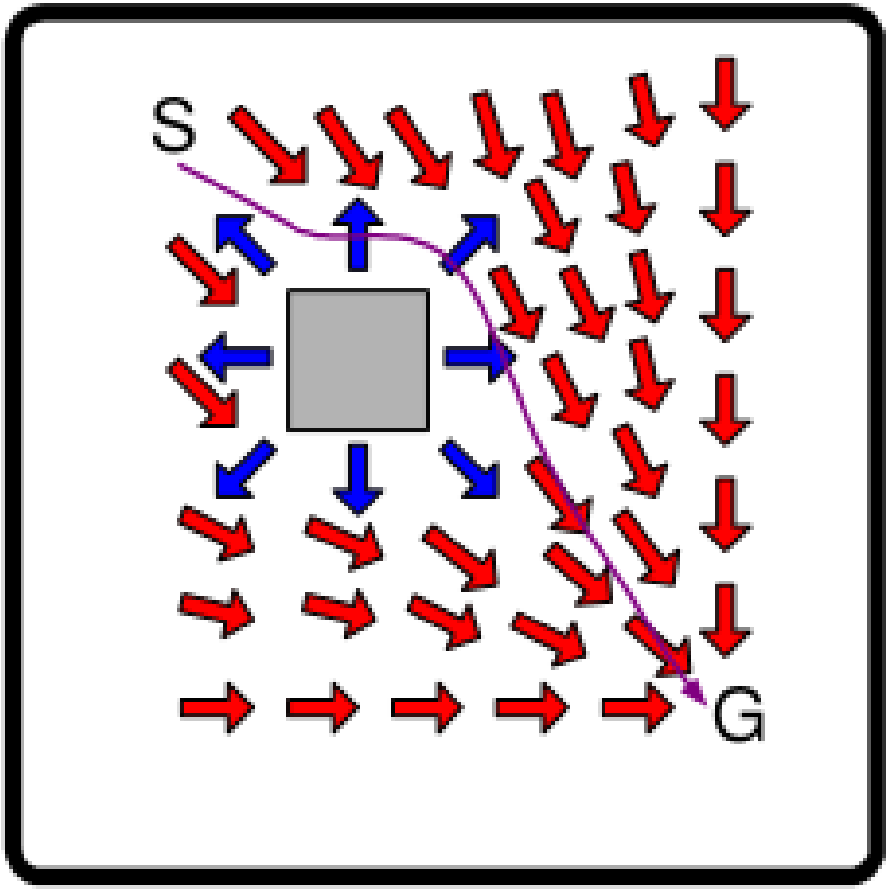
# Anwendung von künstlichen Potentialfeldern für humanoide Fußballroboter



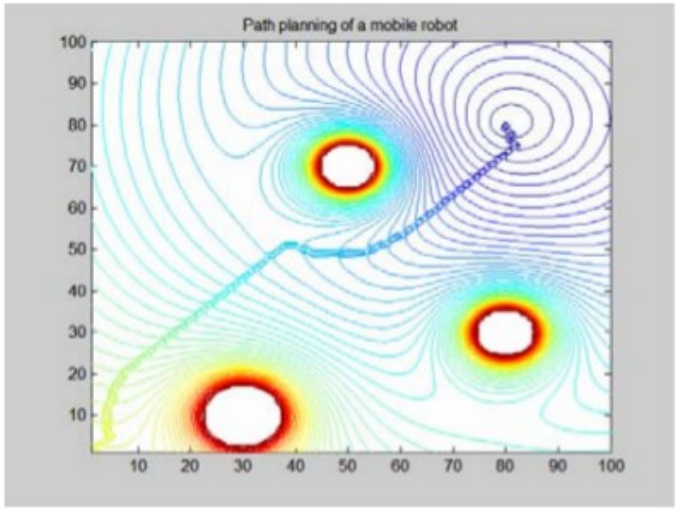



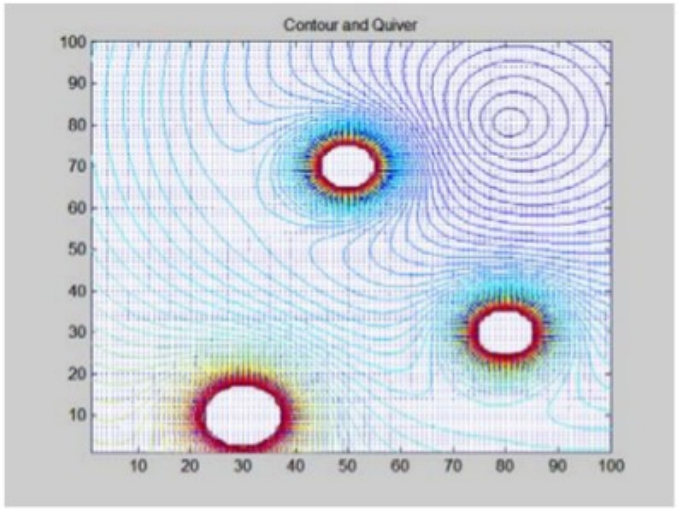

Marcus ScheunemannJonschkowski
FUmanoids
Institut für Informatik, FU Berlin
Sonntag, 22. Mai 2011

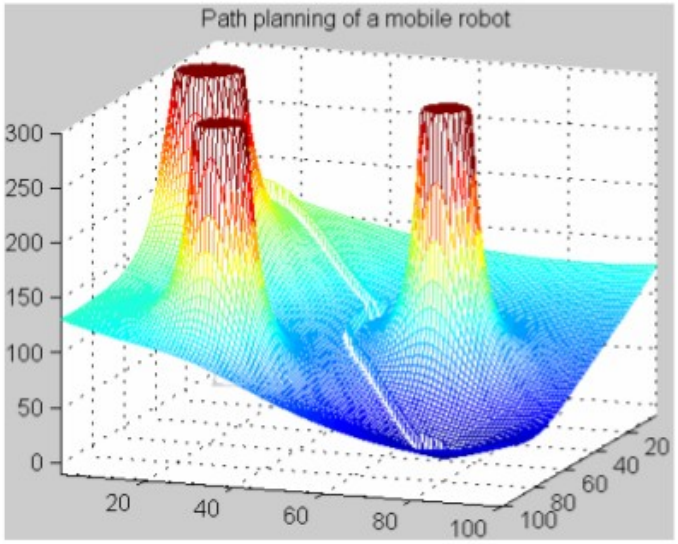
# Idee

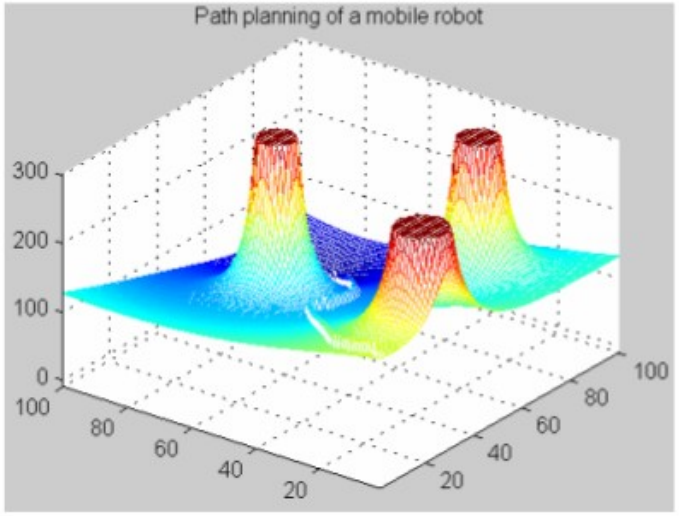


# Idee

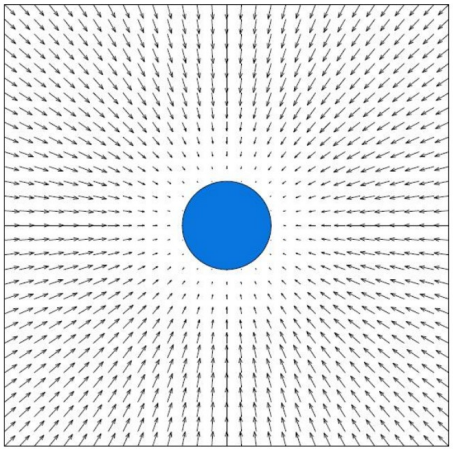

Path planning of a mobile robot


Contour and Quiver

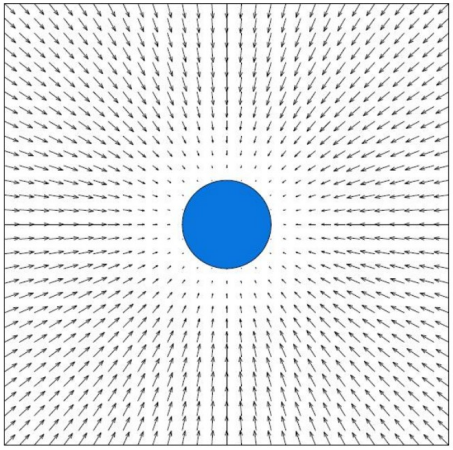

Path planning of a mobile robot

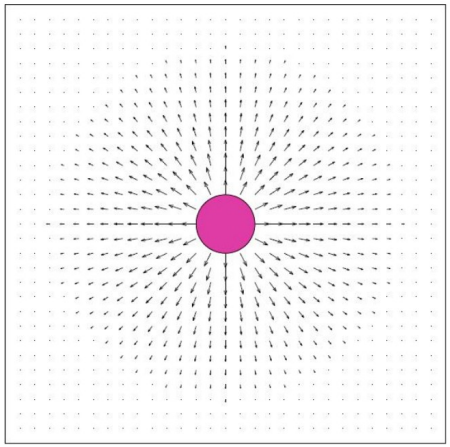

Path planning of a mobile robot

# Typen von Potentialen

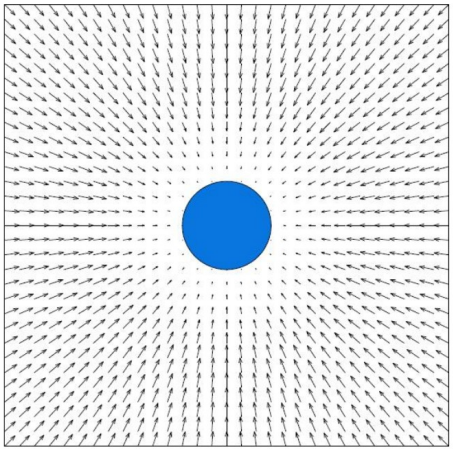


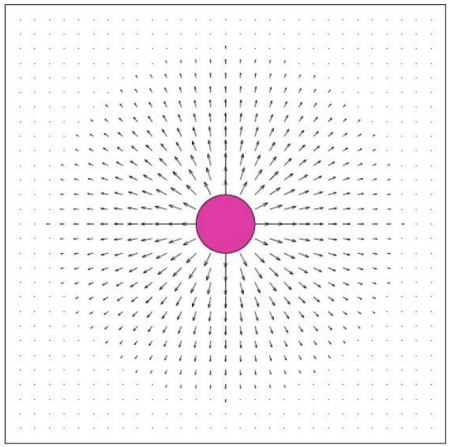
# Typen von Potentialen

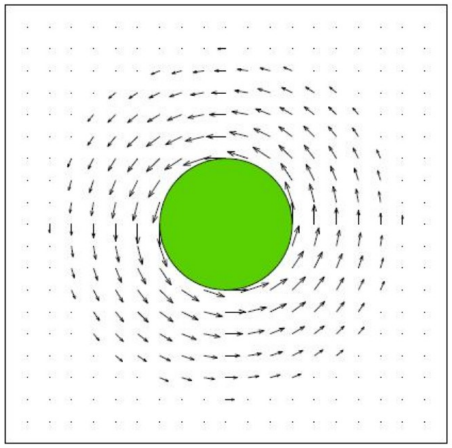




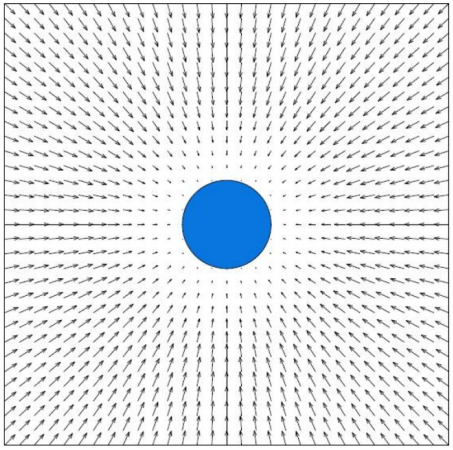
# Typen von Potentialen

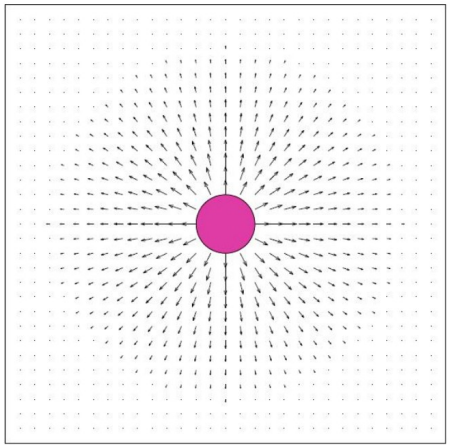


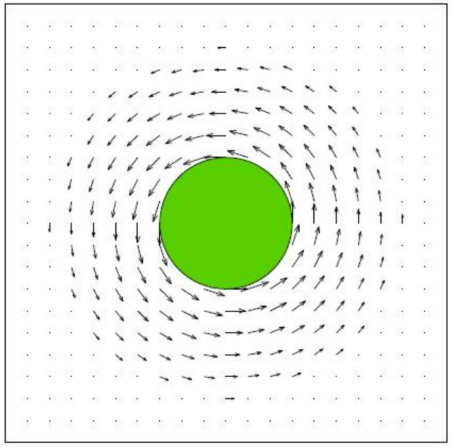


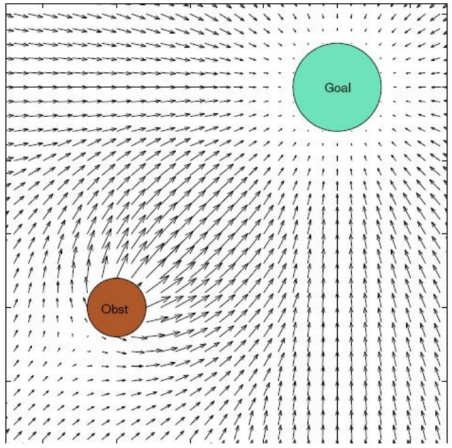


# Typen von Potentialen



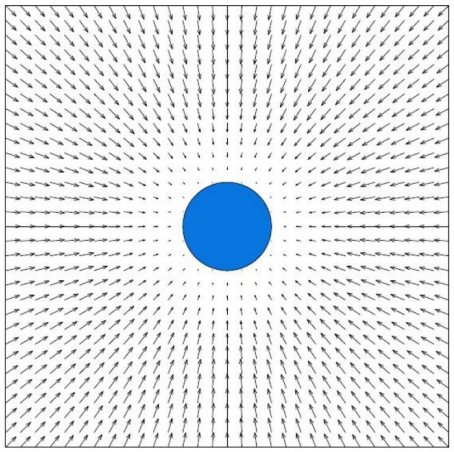






# Implementierung in Matlab

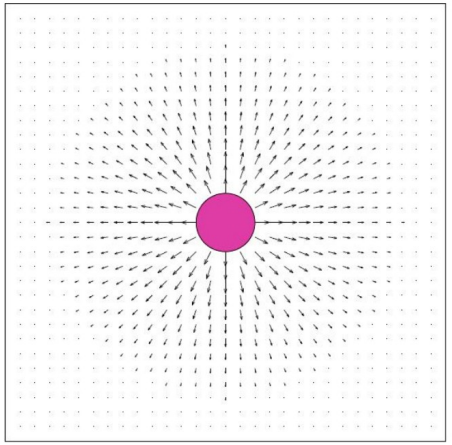
```
1  function dp = goal(p, g, r, s)
2
3          d = norm(g-p);
4          t = atan2(g(1)-p(1),g(2)-p(2));
5
6          if d < r
7                  dp = [0 0];
8          elseif d <= s + r
9                  dp = [(d-r)/s*cos(t) (d-r)/s*sin(t)];
10         else
11                 dp = [cos(t) sin(t)];
12         end
```



# Implementierung in Matlab

Freie Universität Berlin

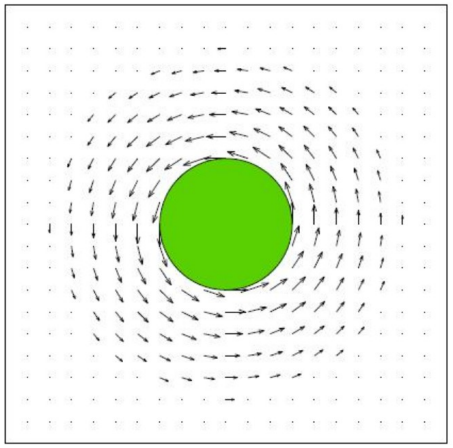
```
1  function dp = obst(p, o, r, s)
2
3              d = norm(o-p);
4              t = atan2(o(1)-p(1),o(2)-p(2));
5
6              if d < r
7                      dp = [cos(t)   sin(t)];
8              elseif d <= s + r
9                      dp = [(1+(r-d)/s)*cos(t)   (1+(r-d)/s)*sin(t)];
10             else
11                     dp = [0 0];
12             end
```



# Implementierung in Matlab

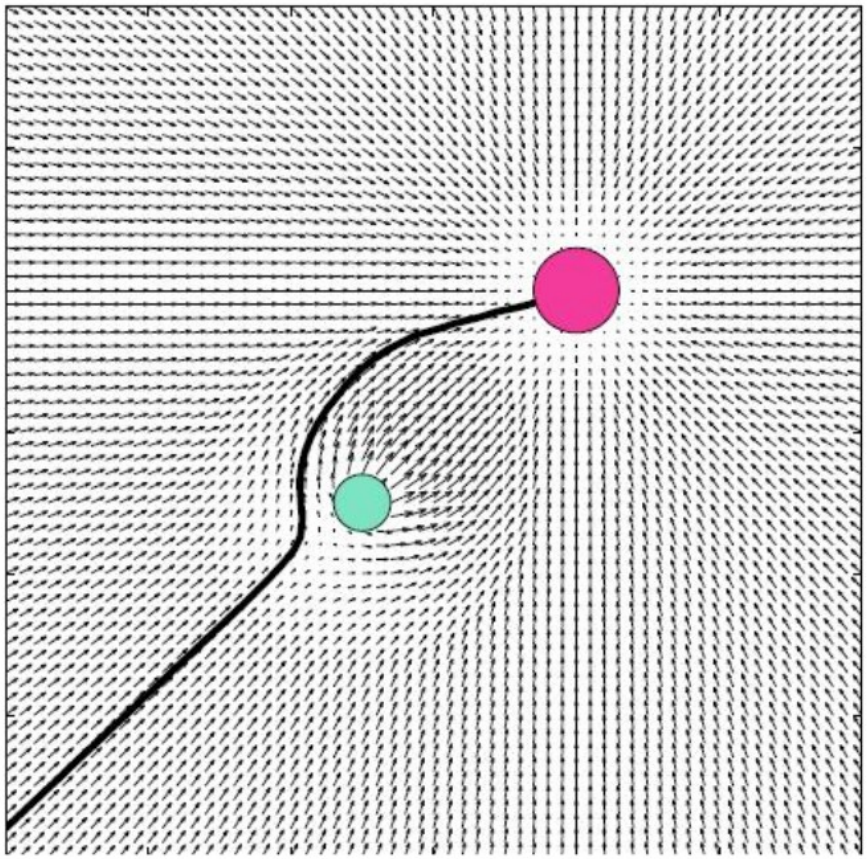
Freie Universität Berlin

```
1  function dp = apftan(p, o, r, s)
2
3          d = norm(o-p);
4          t = atan2(o(1)-p(1),o(2)-p(2)) + pi/2;
5
6          if d < r
7                  dp = [cos(t)   sin(t)];
8          elseif d <= s + r
9                  dp = [(1+(r-d)/s)*cos(t)   (1+(r-d)/s)*sin(t)];
10         else
11                 dp = [0 0];
12         end
```



# Implementierung in Matlab

```
6  for i = 1:n
7          for j = 1:n
8                  m(i,j,:) = 1*goal([i j], [20 20], 0, 1) - 1*obst([i j], [15 15], 5, 10);
9          end
10 end
```



# Implementierung für den Roboter

Freie Universität Berlin

- Basisfunktionen – in XABSL verwendbar
  - float apf.reset(myself.x, myself.y);
  - float input apf.addGoal(x, y, a, r, s);
  - float input apf.addObstacle(x, y, a, r, s);
  - float input apf.addTangent(x, y, a, r, s);
  - float input apf.scaleVector(length);
  - float input apf.vector.x;
  - float input apf.vector.y;

# Implementierung für den Roboter

Freie Universität Berlin

```
dummy = apf.reset(myself.x = 0, myself.y = 0);
dummy = apf.addGoal(
  x = ball.posRel.x - striker.kickPos.x,
  y = ball.posRel.y - striker.kickPos.y,
  a = 100, r = 0, s = 1 // 100, 0, 1
);

dummy = apf.addObstacle(
  x = ball.posRel.x - striker.kickPos.x,
  y = ball.posRel.y - striker.kickPos.y,
  a = 250, r = 0, s = 20 // 250, 0, 20
);

dummy = apf.addObstacle(
  x = ball.posRel.x + 10, // +10
  y = ball.posRel.y,
  a = -250, r = 10, s = 30 // -250, 10, 30
);

dummy = apf.scaleVector(length = vector.abs(x = ball.posRel.x - striker.kickPos.x,
      y = ball.posRel.y - striker.kickPos.y));

// approach ball
WalkOmniFacingOppGoal(
    target.x = apf.vector.x,
    target.y = apf.vector.y,
    target.eps = 0,
    parent.state_time = state_time);
```

# Implementierung für den Roboter

Freie Universität Berlin

- Weitere Funktionen
  - float input apf.AddTeammateObstacles(a, r, s);
  - float input apf.AddVisionObstaclesRel(a, r, s);
  - float input apf.AddVisionObstaclesAbs(a, r, s);

# Anwendungen

Freie Universität Berlin

- Positionierung des Teams
- Hindernisvermeidung
- (Zum Ball laufen)
- (Dribbeln)
- Hinter den Ball laufen

  → verwendet fürs zum Ball laufen und Dribbeln
- Um Hindernisse herum dribbeln

# Anwendungen

Freie Universität Berlin

- Positionierung des Teams
  - Absolute Koordinaten
  - Goal: Zielposition
  - Obstacles: Kommunizierte Positionen der Mitspieler

# Anwendungen



- Hindernisvermeidung
  - Relative Koordinaten
  - Goal: Zielposition
  - Obstacles: Erkannte Gegner

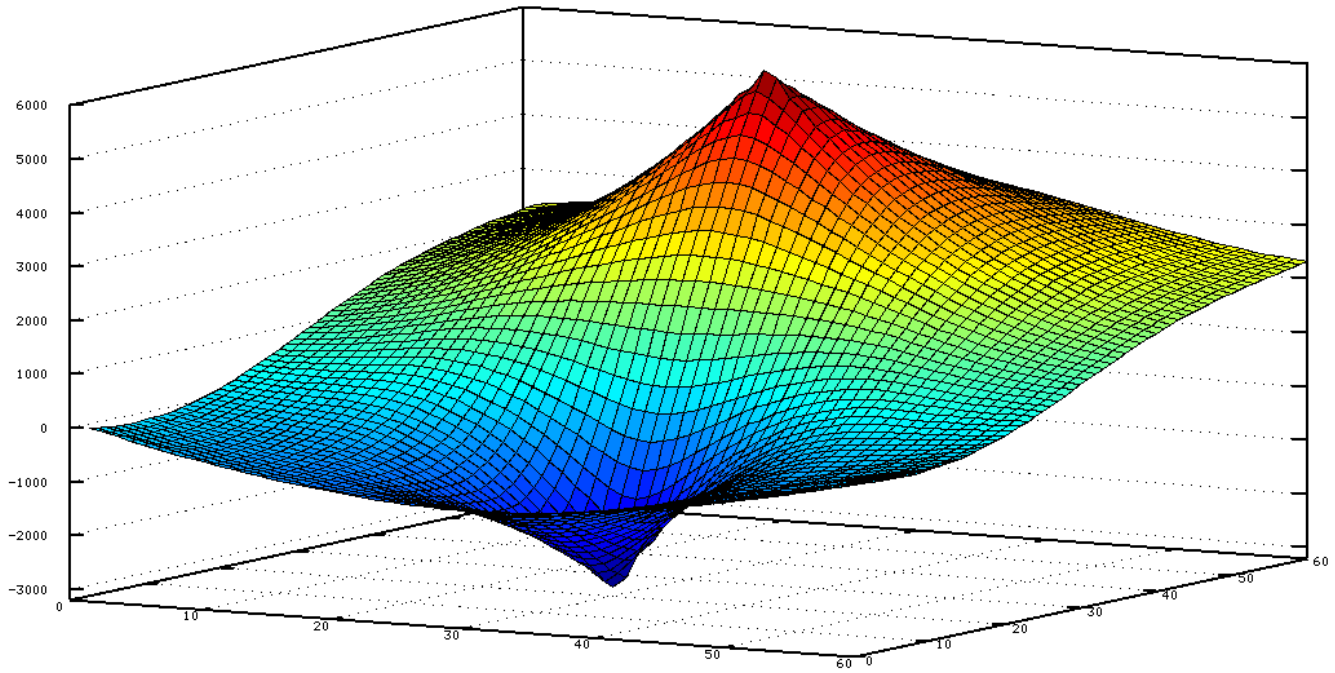
# Anwendungen

Freie Universität Berlin

Simulator - Demo

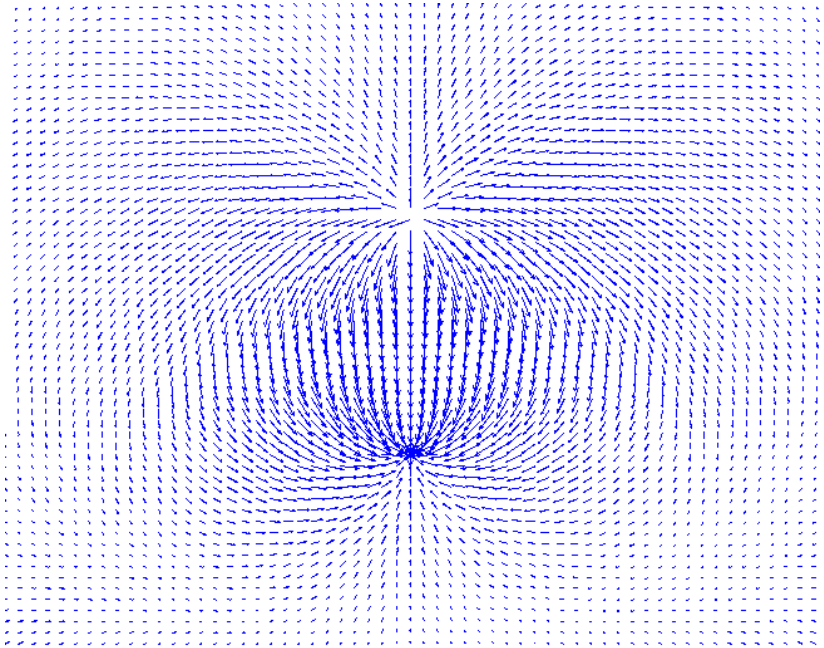
# Anwendungen

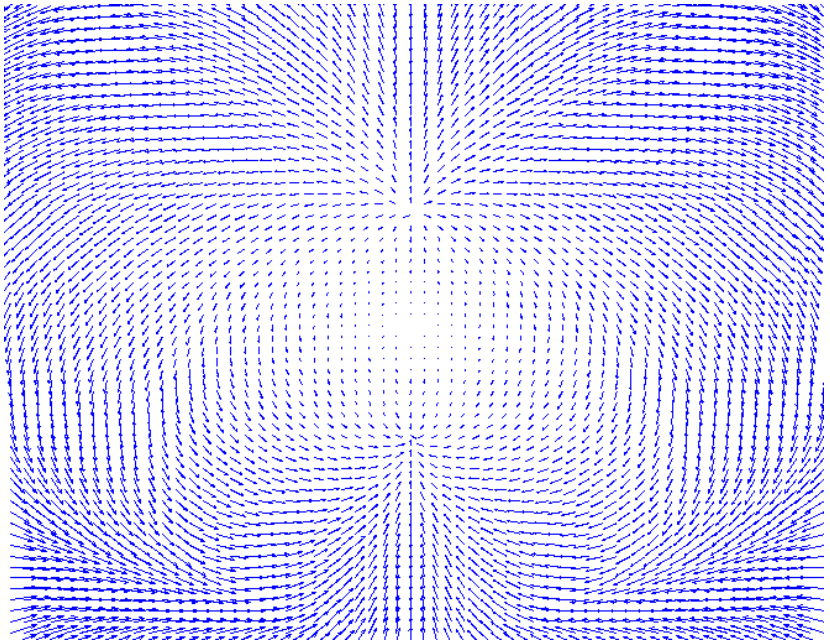
- (Zum Ball laufen)
  - Relative Koordinaten
  - Goal: Schussposition hinter dem Ball
  - Obstacle: Position vor dem Ball
  - Anziehendes Obstacle: Schussposition hinter dem Ball



# Anwendungen

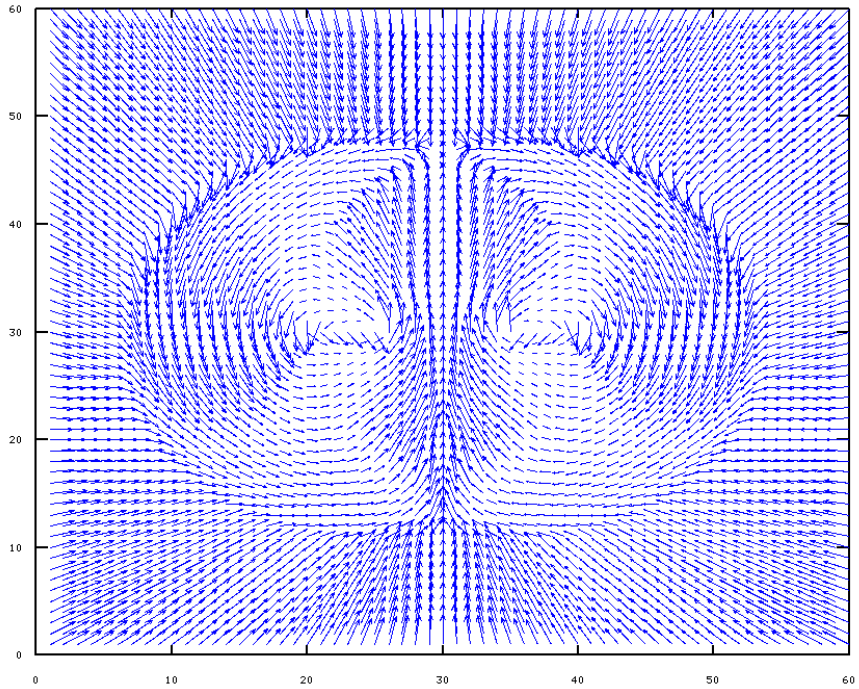
- (Zum Ball laufen)
  - Relative Koordinaten
  - Goal: Schussposition hinter dem Ball
  - Obstacle: Position vor dem Ball
  - Anziehendes Obstacle: Schussposition hinter dem Ball





# Anwendungen

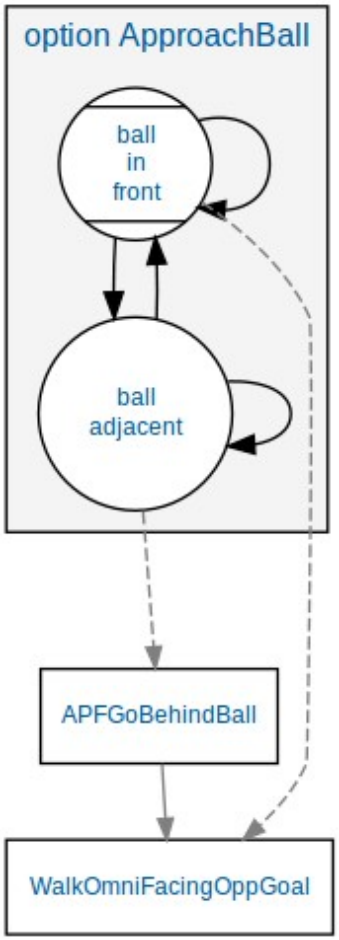
- (Dribbeln)
  - Relative Koordinaten
  - Goal: Position hinter dem Ball
  - Tangentialfelder: ein rechtsdrehendes rechts vom Ball, ein linksdrehendes links vom Ball



# Anwendungen



- Hinter den Ball laufen
  - Relative Koordinaten
  - Goal: Schussposition hinter dem Ball
  - Obstacle: Position vor dem Ball
  - Anziehendes Obstacle: Schussposition hinter dem Ball
- Verwendet fürs zum Ball laufen und Dribbeln
  - Simple Lösung für ball_in_front
  - Hysterese

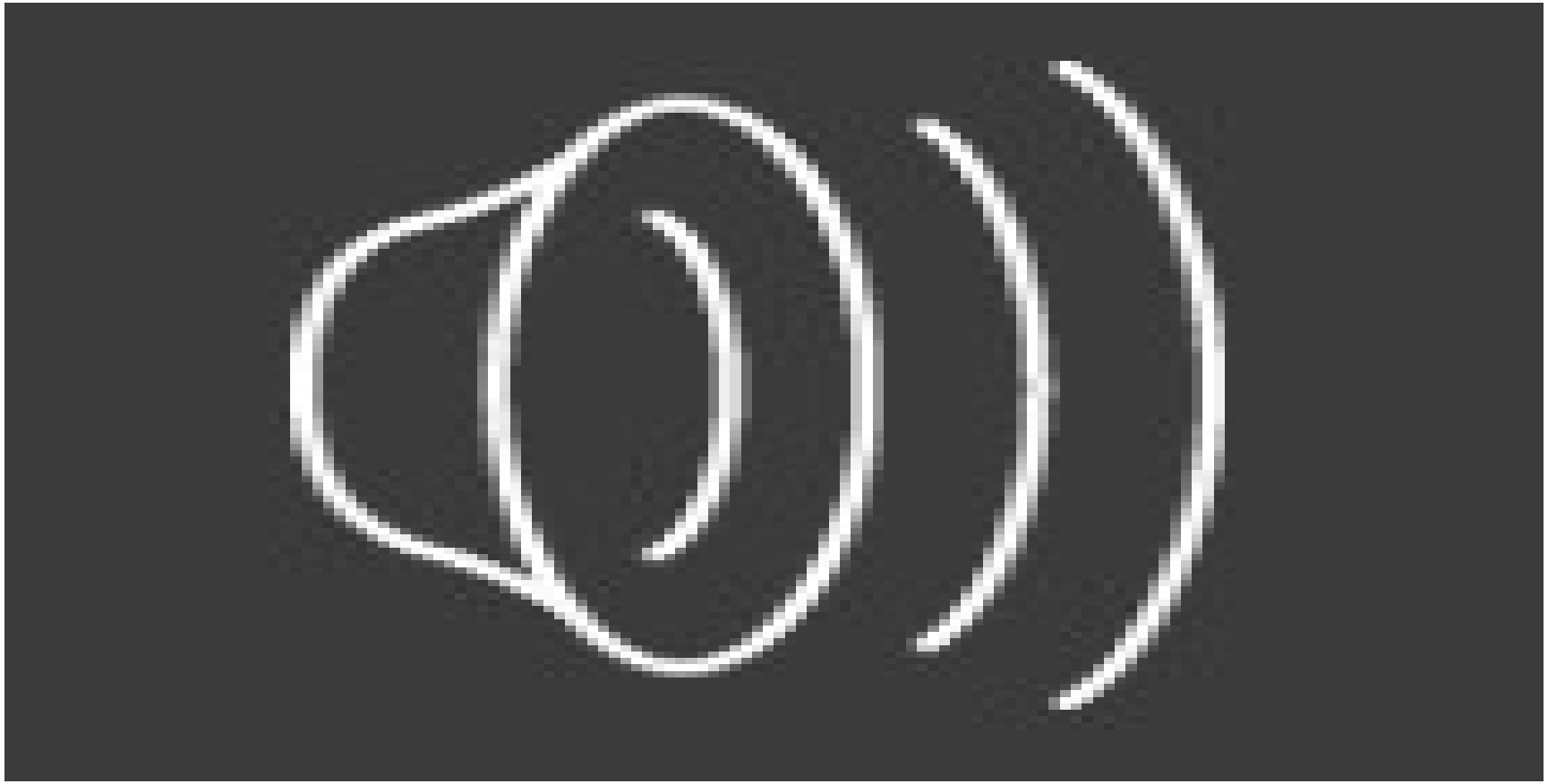


# Anwendungen

Freie Universität Berlin

- Um Hindernisse herum dribbeln
  - Bestimmung der Dribble-Richtung
    - Absolute Koordinaten
    - Position: Ball
    - Goal: Tor
    - Obstacles: Erkannte Gegner
  - In diese Richtung dribbeln
    - Relative Koordinaten
    - Goal: Punkt in Dribble-Richtung
    - Obstacles: Erkannte Gegner

# Anwendungen



# Danke!

# Fragen?