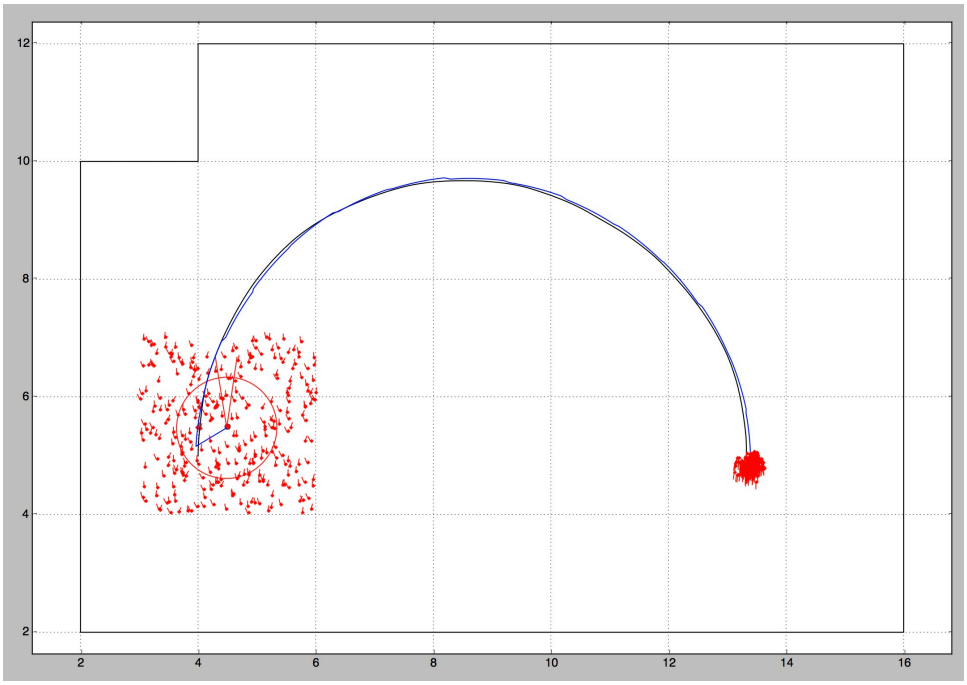
Monte-Carlo-Lokalisierung

Der Roboter soll sich mit Hilfe eines Partikelfilters lokalisieren. Der Roboter kennt dabei seine ungefähre Anfangsposition. Es ist also nur die relative Selbstlokalisierung zu lösen.



Gehen Sie wie folgt vor.

1) Starten Sie mit einem kleinen Programm, das einen Roboter auf einem Halbkreis in der Welt `simpleWorld` bewegt.
2) Erzeugen Sie aus der Umgebungskarte `simpleWorld` ein Likelihoodfield (distant map). (Hinweis: demo_simulator_4.py)
3) Schreiben Sie eine Klasse `ParticleFilterPoseEstimator` mit folgenden Methoden:
   a) `initialize(self, poseFrom, poseTo, n = 200)`
      Erzeugt n zufällige Partikel in dem vorgegebenen Posen-Bereich.
   b) `integrateMovement(self, motion)`
      Wendet auf alle Partikel den Bewegungsbefehl `motion` mit einem zufälligen Rauschen an.
   c) `integrateMeasurement (self, dist_list, alpha_list, distantMap)`
      Gewichtet alle Partikel mit dem Likelihoodfield-Algorithmus und führt ein Resampling durch. `dist_list, alpha_list` sind vom Roboter aufgenommene Laserdaten in Polarkoordinaten.
   d) `getPose(self)`
      Berechnet aus der Partikelmenge eine Durchschnittspose.
   e) `getCovariance(self)`
      Berechnet die Kovarianz der Partikelmenge.

4) Testen Sie Ihre Klasse `ParticleFilterPoseEstimator`

   a) Integrieren Sie mehrere Bewegungsschritte mit `integrateMovement.` Stellen Sie die Partikelmenge davor und danach dar. In der Klasse `PlotUtilities` gibt es geeignete plot-Funktionen. Überprüfen Sie, dass die Streuung der Partikelmenge zunimmt.

   b) Erzeugen Sie eine Partikelmenge und integrieren Sie genau eine Sensormessung. Stellen Sie die Partikelmenge davor und danach dar. Überprüfen Sie, dass die Streuung der Partikelmenge abnimmt und dass die geschätzte Pose `getPose()`näher an der tatsächlichen liegt.

   c) Führen Sie nun die Halbkreisfahrt aus 1) durch und stellen Sie sowohl die geschätzten als auch die tatsächlichen Posen dar.