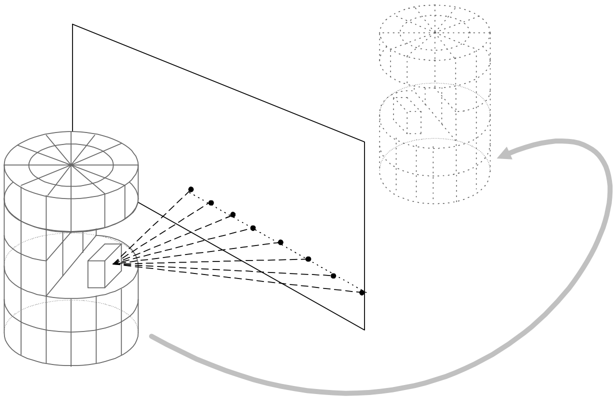
C4B — Mobile Robotics



Paul Michael Newman

October 2003, Version 1.00

Contents

1 Introduction and Motivation 6

2 Introduction to Path Planning and Obstacle Avoidance 8

- 2.1 Holonomicity 9
- 2.2 Configuration Space 11
- 2.3 The Minkowski-Sum 13
- 2.4 Voronoi Methods 14
- 2.5 Bug Methods 15
- 2.6 Potential Methods 15

3 Estimation - A Quick Revision 19

- 3.1 Introduction 19
- 3.2 What is Estimation? 19
  - 3.2.1 Defining the problem 20
- 3.3 Maximum Likelihood Estimation 21
- 3.4 Maximum A-Posteriori - Estimation 22

3.5 Minimum Mean Squared Error Estimation 24

3.6 Recursive Bayesian Estimation 25

4 Least Squares Estimation 28

4.1 Motivation 28

4.1.1 A Geometric Solution 29

4.1.2 LSQ Via Minimisation 29

4.2 Weighted Least Squares 30

4.2.1 Non-linear Least Squares 30

4.2.2 Long Baseline Navigation - an Example 31

5 Kalman Filtering -Theory, Motivation and Application 35

5.1 The Linear Kalman Filter 35

5.1.1 Incorporating Plant Models - Prediction 39

5.1.2 Joining Prediction to Updates 42

5.1.3 Discussion 43

5.2 Using Estimation Theory in Mobile Robotics 46

5.2.1 A Linear Navigation Problem - “Mars Lander” 46

5.2.2 Simulation Model 48

5.3 Incorporating Non-Linear Models - The Extended Kalman Filter 52

5.3.1 Non-linear Prediction 52

5.3.2 Non-linear Observation Model 54

5.3.3 The Extended Kalman Filter Equations 56

6 Vehicle Models and Odometry 59

6.1 Velocity Steer Model 59

6.2 Evolution of Uncertainty 61

6.3 Using Dead-Reckoned Odometry Measurements 63

6.3.1 Composition of Transformations 65

7 Feature Based Mapping and Localisation 69

7.1 Introduction 69

7.2 Features and Maps 70

7.3 Observations 70

7.4 A Probabilistic Framework 71

7.4.1 Probabilistic Localisation 71

7.4.2 Probabilistic Mapping 72

7.5 Feature Based Estimation for Mapping and Localising 73

7.5.1 Feature Based Localisation 73

7.5.2 Feature Based Mapping 74

7.6 Simultaneous Localisation and Mapping - SLAM 78

7.6.1 The role of Correlations 81

8 Multi-modal and other Methods 83

8.1 Montecarlo Methods - Particle Filters 83

8.2 Grid Based Mapping 85

9 In Conclusion 89

10 Miscellaneous Matters 90

10.1 Drawing Covariance Ellipses 90

10.2 Drawing High Dimensional Gaussians 93

11 Example Code 94

11.1 Matlab Code For Mars Lander Example 94

11.2 Matlab Code For Ackerman Model Example 98

11.3 Matlab Code For EKF Localisation Example 100

11.4 Matlab Code For EKF Mapping Example 103

11.5 Matlab Code For EKF SLAM Example 107

11.6 Matlab Code For Particle Filter Example 111

Topic 1

Introduction and Motivation

This set of lectures is about navigating mobile platforms or robots. This is a huge topic and in eight lectures we can only hope to undertake a brief survey. The course is an extension of the B4 estimation course covering topics such as linear and non-linear Kalman Filtering. The estimation part of the lectures is applicable to many areas of engineering not just mobile robotics. However I hope that couching the material in a robotics scenario will make the material compelling and interesting to you.

Lets begin by dispelling some myths. For the most parts when we talk about "mobile robots" we are not talking about gold coloured human-shaped walking machines [1]. Instead we are considering some-kind of platform or vehicle that moves through its environment carrying some kind of payload. Almost without exception it is the payload that is of interest - not the platform. However the vast majority of payloads require the host vehicle to navigate —to be able to parameterize its position and surroundings and plan and execute trajectories through it. Consider some typical autonomous vehicle and payloads:

- Humans in a airplane on autopilot (or car in the near-ish future. CMU NavLab project)
- Scientific equipment on a Mars lander
- Mine detectors on an autonomous underwater vehicle
- Cargo containers in a port transport vehicle
- Pathology media in a hospital deliver system

---

[1] although the Honda P5 humanoid is a good approximation

- Verbal descriptions in a museum tour guide
- A semi-submersible drill ship (oil recovery)
- Cameras on an aerial survey drone
- Obvious military uses (tend to be single mission only....)

All of the above require navigation. This course will hopefully give you some insight into how this can be achieved.

It is worth enumerating in general terms what makes autonomous navigation so hard. The primary reason is that the majority of mobile robots are required to work in un-engineered environments. Compare the work-spaces of a welding robot in automotive plant to one that delivers blood samples between labs in a hospital. The former operates in a highly controlled, known, time invariant (apart from the thing being built) scene. If computer vision is used as a sensor then the workspace can be lit arbitrarily well to mitigate against shadows and color ambiguity. Many industrial robots work in such well known engineered environments that very little external sensing is needed — they can do their job simply by controlling their own internal joint angles. Hospitals are a different ball-park altogether. The corridors are dynamic — filling and emptying (eventually) with people on stretchers. Even if the robot is endowed with a map of the hospital and fitted with an upward looking camera to navigate off markers on the ceiling it still has to avoid fast moving obstacles (humans) while moving purposely towards it's goal destination. The more generic case involves coping with substantial scene changes (accidental or malicious) — for example doors closing in corridors or furniture being moved. The thing then, that makes mobile robotics so challenging is uncertainty. Uncertainty is pervasive in this area and we must embrace it to make progress....

Topic 2

Introduction to Path Planning and Obstacle Avoidance

A basic requirement of a mobile autonomous vehicle is path planning. With the vehicle in an arbitrary initial position **A** we wish to issue a desired goal position **B** (including orientation) and have the vehicle execute a trajectory to reach **B** . This sounds pretty simple and we can think of several ways in which we could combine simple control laws that will get us from **A** to **B** [1] Unfortunately the waters quickly become muddied when we start talking about our other concerns:

- while executing its trajectory the vehicle must not smash into objects in the environment (especially true regarding squishy humans).
- we cannot guarantee that the vehicle in question can turn-on-the-spot and would like to be able to operate a vehicle of arbitrary shape. These are called "kinematic" constraints.
- we expect only uncertain estimates of the location of the robot and objects in the environment.

The combination of path-planning, obstacle avoidance, kinematic constraints and uncertainty makes for a very hard problem indeed — one which is still an active area of research. However we can do some interesting things if we decouple some of the issues and make some

---

[1] for example drive in a straight line from **A** to **B** until **B** (x,y) is reached then rotate to align with **B** (theta).

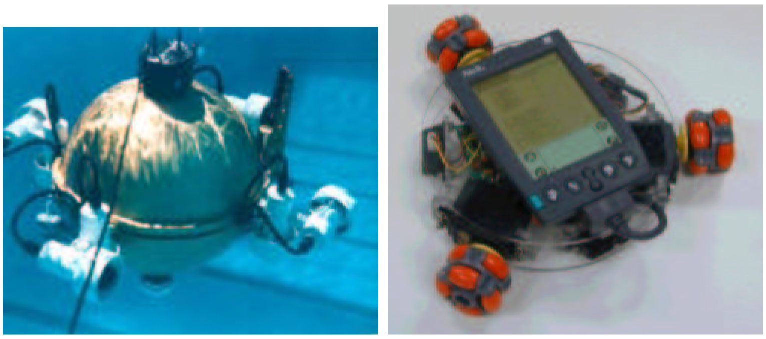


Figure 2.1: Two holonomic vehicles. The underwater vehicle (ODIN, University of Hawaii) can move in any direction irrespective of pose and the complex wheel robot PPRK (CMU) driven by a "Palm Pilot" uses wheels that allow slip parallel to their axis of rotation. A sum of translation and slip combine to achieve any motion irrespective of pose.

simplifying assumptions. We shall begin by discussing vehicle properties and categorizing them into two classes — holonomic and non-holonomic.

2.1 Holonomicity

Holonomicity is the term used to describe the locomotive properties of a vehicle with respect to its workspace. We will introduce a mathematical definition of the term shortly but we will begin by stating, in words, a definition:

A vehicle is holonomic if the number of local degrees of freedom of movement equals the number of global degrees of freedom.

We can make this a little more clear with a few examples:

- a car is non-holonomic : the global degrees of freedom are motion in x,y and heading however locally, a car can only move forward or turn. It cannot slide sideways.(Even the turning is coupled to motion).

Figure 2.2: A commercial non-holonomic robot vehicle (Roombavac from iRobot coorporation. This vehicle can be purchased for about 100 USD - but is it's utility great enough to warrant the prices? Discuss....

- The "spherical" underwater robot (Odin) and the rolling wheel vehicle in Figure 2.1 are holonomic they can turn on the spot and translate instantaneously in any direction **without** having to rotate first.
- A train is holonomic: it can move forwards or backwards along the track which is parameterised by a single global degree of freedom — the distance along the track.
- The robot "Roombavac" (iRobot corporation) vacuum cleaner in Figure 2.1 is also non-holonomic. It can rotate in place but cannot slide in any direction — it needs to use a turn-translate-turn or turn-while-drive (like a car) paradigm to move.

It should be obvious to you that motion control for a holonomic vehicle is much easier than for a non-holonomic vehicle. If this isn't obvious consider the relative complexity of parking a car in a tight space compared to driving a vehicle that can simply slide into the space sideways (a hovercraft).

Unfortunately for us automation engineers, the vast majority of vehicles in use today (i.e. used by humans) are non-holonomic. In fact intrinsically holonomic vehicles are so rare and complex (or so simple [2]) that we shall not discuss them further.

We can now place some formalism on our notion of holonomicity. We say a vehicle whose state is parameterised by a vector $\mathbf{x}$ is non-holonomic if there exists a constraint $\mathcal{Q}$ such that

$$\mathcal{Q}(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, \cdots) = 0 \tag{2.1}$$

where the state derivatives cannot be integrated out. To illustrate such a constraint we will take the case of a front wheel steered vehicle as shown in figure 2.1

---

[2] path planning for a train is quite uninteresting

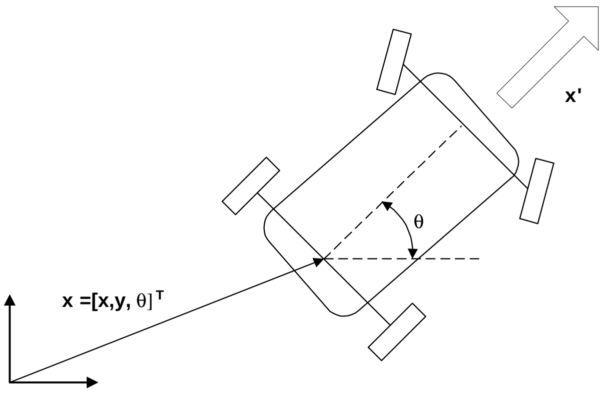


Figure 2.3: A steered non-holonomic vehicle

Immediately we can write a constraint expressing that the vehicle cannot slip sideways that is a function of $\mathbf{x}$ and its first derivative:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} . \begin{bmatrix}-\sin\theta\\ \cos\theta\\ 0\end{bmatrix} = 0 \tag{2.2}$$

$$\mathcal{Q}(\mathbf{x}, \dot{\mathbf{x}}) = 0 \tag{2.3}$$

The state and its derivatives are inseparable and by our definition the vehicle is non-holonomic.

2.2 Configuration Space

We are describing the robot by its state $\mathbf{x} = [x_1, x_2, \cdots x_n]^T$ (for a 2D plane vehicle commonly $x_1 = x \quad x_2 = y \quad x_3 = \theta$)which is a n-state parameterisation. We call the space within which $\mathbf{x}$ resides the **configuration space** $\mathcal{C} - space$ of the vehicle:

$$\mathcal{C} = \bigcup_{\forall x_1, x_2 \cdots x_n} \mathbf{x} \tag{2.4}$$

The configuration space (or $\mathcal{C} - space$) is the set of all allowable configurations of the robot. For a simple vehicle moving on a plane the configuration space has the same dimension as the work space but for more complicated robots the dimension of the configuration space is much higher. Consider the case of the bomb disposal vehicle in figure 2.2. The configuration space for such a vehicle would be 11 — 3 for the base and another 8 for the pose of the arm and gripper. We can view obstacles as defining regions of $\mathcal{C} - space$ that are forbidden. We

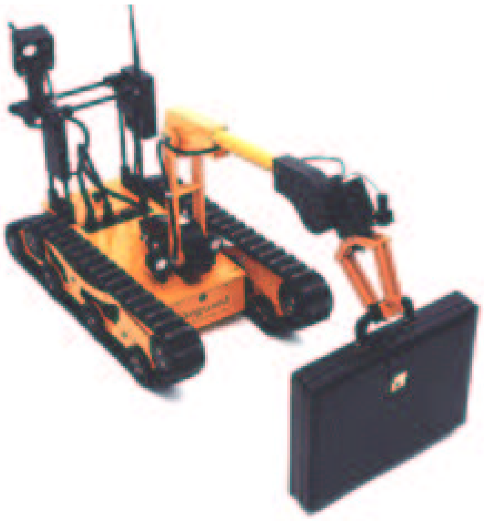


Figure 2.4: A vehicle with a high dimensional configuration space - a commercial bomb-disposal platform (picture courtesy of Roboprobe Ltd. The configuration space for a human is immensely high dimensional —around 230.

can label this space as $\mathcal{C}_{\otimes}$ and the remaining accessible/permissable space as $\mathcal{C}_{\odot}$ such that $\mathcal{C} = \mathcal{C}_{\otimes} \cup \mathcal{C}_{\odot}$. It is often possible to define and describe the boundaries of $\mathcal{C}_{\otimes}$(and hence the boundaries of $\mathcal{C}_{\odot}$in which the vehicle must move) as a constraint equation. For example if the workspace of a point robot in 2D $\mathbf{x} = [xy]^T$ is bounded by a wall $ax + by + c = 0$ then

$$\mathcal{C}_{\odot} = \underbrace{\bigcup \{\mathbf{x} \mid ax + by + c > 0\}}_{\text{union of all } \mathbf{x} \text{ for which ax+by+c} > 0} \tag{2.5}$$

If each of the constraints imposed on $\mathcal{C} - space$by each obstacle k is represented by an equation of the form $\mathcal{C}_k(x) = 0$ then the open free space $\mathcal{C}_{\odot}$admitted by n obstacles can be written as the following intersection:

$$\mathcal{C}_{\odot} = \bigcap_{k=1}^{k=n} \left(\bigcup \{\mathbf{x} \mid \mathcal{C}_k(x) = 0\} \right) \tag{2.6}$$

Equation 2.6 simple states what configurations are open to the vehicle —nothing more. Any path planning algorithm must guide the vehicle along a trajectory within this space while satisfying any non-holonomic vehicle constraints and finally deliver the vehicle to the goal pose. This clearly is a hard problem. Two poses that may be adjacent to each other in state space may require an arbitrarily long path to be executed to transition between them. Take, for example, the seemingly simple task of turning a vehicle through 180^o when it is near a wall. One solution trajectory to the problem is shown in figure 2.2. The non-holonomic vehicle constraints conspire with the holonomic constraint imposed by the wall to require a complicated solution trajectory.

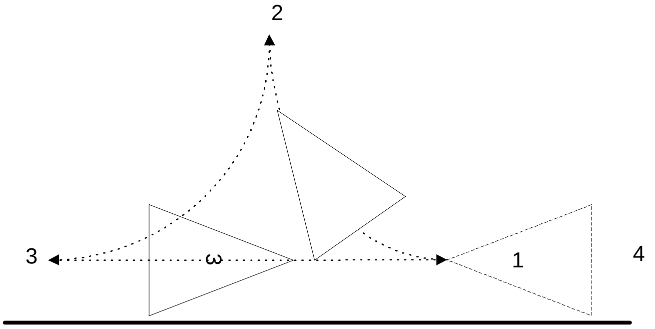


Figure 2.5: A simple task - turning through 180^o quickly becomes complicated by $\mathcal{C}-space$constraints.

2.3 The Minkowski-Sum

Real robots have arbitrary shapes and these shapes make for complicated interactions with obstacles which we would like to simplify. One way to do this is to transform the problem to one in which the robot can be considered as a point-object and a technique called the "Minkowski-Sum" does just this. The basic idea is to artificially inflate the extent of obstacles to accommodate the worst-case pose of the robot in close proximity. This is easiest to understand with a diagram shown in Figure 2.3. The idea is to replace each object with a virtual object that is the union of all poses of the vehicle that touch the obstacle. Figure 2.3 has taken a conservative approach and "replaced" a triangular vehicle with a surrounding circle. The minimal Minkowski-Sum would be the union of the obstacle and all vehicle poses with the vehicle nose just touching it boundary. With the obstacles suitably inflated the vehicle can be thought of a point-object and we have a guarantee that as long as it keeps to the new, shrunken, free space it cannot hit an object [3]. Note it is usual to fit a polygonal hull around the results of the Minkowski-Sum calculation to make ensuing path planning calculations easier.

So now we have a method by which we can calculate $\mathcal{C}_{\odot}$. The next big question is how exactly do we plan a path through it? How do we get from an initial pose to a goal pose? We will consider three methods : Voronoi, "Bug" and Potential methods.

---

[3] This doesn't mean that awkward things won't happen — the situation shown in figure 2.2 is still possible. However progress can be made by planning motion such that at object boundaries the vehicle is always capable of moving tangentially to the boundaries

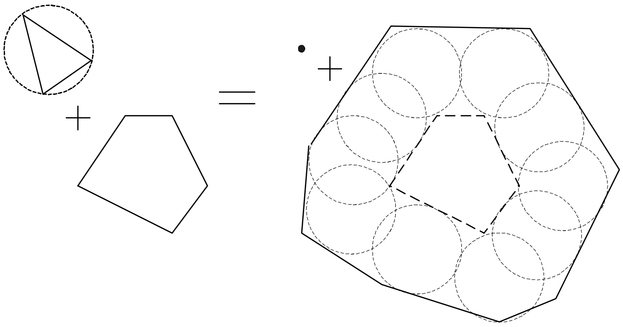


Figure 2.6: The Minkowski-Sum transforms a arbitrarily shaped vehicle to a point while inflating the obstacle. The result is guaranteed free space outside the inflated object boundary.

2.4 Voronoi Methods

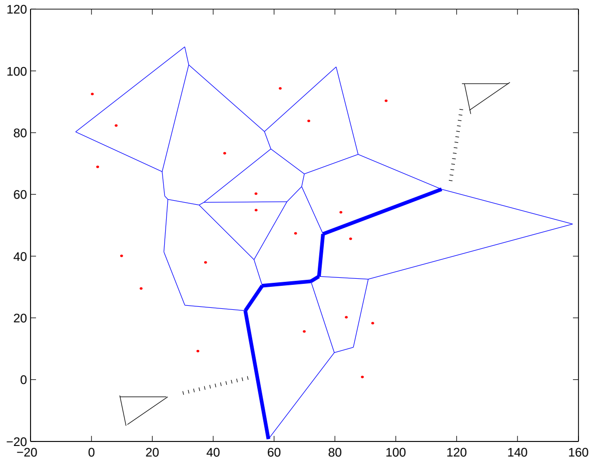


Figure 2.7: A Voronoi diagram for obstacle avoidance in the presence of point objects.

Voronoi diagrams are elegant geometric constructions [4] that find applications throughout computer science — one is shown in figure 2.4. Points on a 2D-Voronoi diagram are equidistant from the nearest two objects in the real world. So the Voronoi diagram of two points a, b is the line bisecting them — all points on that line are equidistant from a, b. The efficient computation of Voronoi diagrams can be quite a complex matter but what is important here is that algorithms exist that when given a set of polygonal objects can generate the appropriate equi-distant loci. So how does this help us with path planning? Well, if we follow the paths defined by a Voronoi diagram we are guaranteed to stay maximally far away from nearby objects. We can search the set of points on the diagram to find points that are closest to and visible from the start and goal positions. We initially drive to the "highway entry" point, follow the "Voronoi - highways" until we reach the "exit point" where we leave the highway and drive directly towards the goal point.

---

[4] *http://www.voronoi.com/*

2.5 Bug Methods

The generation of a global Voronoi diagram requires upfront knowledge of the environment. In many cases this is unrealistic. Also Voronoi planners by definition keep the vehicle as far away from objects as possible - this can have two side effects. Firstly the robot may be using the objects to localise and moving away from them makes them harder to sense. Secondly the paths generated from a Voronoi planner can be extremely long and far from the shortest path (try playing with the Matlab Voronoi function). An alternative family of approaches go under the name of " bug algorithms". The basic algorithm is simple:

1. starting from **A** and given the coordinates of a goal pose **B** draw a line **AB** (it may pass through obstacles that are known or as yet unknown)
2. move along this line until either the goal is reached or an obstacle is hit.
3. on hitting an obstacle circumnavigate its perimeter until **AB** is met
4. goto 2

In contrast to the Voronoi approach this method keeps the vehicle as close to the obstacles as possible (but we won't hit them as the obstacles have been modified by the Minkowski sum!). However the path length could be stupidly long. A smarter modification would be to replace step 3 in the original algorithm with *"on hitting and obstacle circumnavigate it's perimeter until* **AB** *is met or the line* **AB** *becomes visible in which case head for a point on* **AB** *closer to* **B** *"* Figure 2.5 shows the kind of trajectory this modified "VisBug" algorithm would execute. Clearly the hugging the object boundary is not always a good plan but interestingly it is guaranteed to get the robot to the goal location if it is indeed reachable.

2.6 Potential Methods

A third very popular method of path planning is a so called "Potential Method". Once again the idea is very simple. We view the goal pose as a point of low potential energy and the

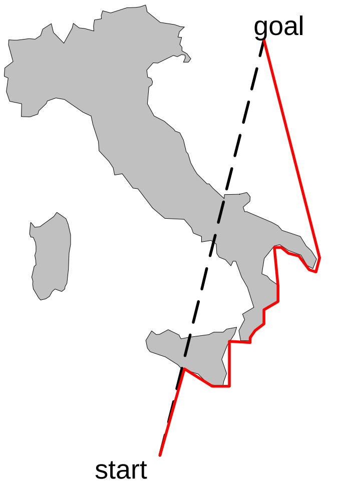


Figure 2.8: The visbug algorithm executing a goal-seek trajectory around Southern Italy.

obstacles as areas of high potential energy. If we think of the vehicle as a ball-bearing free to roll around on a “potential terrain“ then it will naturally roll around obstacles and down towards the goal point. The local curvature and magnitude of the potential field can be used to deduce a locally preferred motion. We now firm up this intuitive description with some mathematics.

At any point $\mathbf{x}$ we can write the total potential $\mathbf{U}_\Sigma$ as a sum of the potential induced $\mathbf{U}_o$ by k obstacles and the potential induced by the goal $\mathbf{U}_g$:

$$\mathbf{U}_\Sigma(\mathbf{x}) = \sum_{i=1:k} \mathbf{U}_{o,i}(\mathbf{x}) + \mathbf{U}_g(\mathbf{x}) \tag{2.7}$$

Now we know that the force $\mathbf{F}(\mathbf{x})$ exerted on a particle in a potential field $\mathbf{U}_\Sigma(\mathbf{x})$ can be written as :

$$\begin{aligned} \mathbf{F}(\mathbf{x}) &= -\nabla \mathbf{U}_\Sigma(\mathbf{x}) && (2.8) \\ &= -\sum_{i=1:k} \nabla \mathbf{U}_{o,i}(\mathbf{x}) - \nabla \mathbf{U}_g(\mathbf{x}) && (2.9) \end{aligned}$$

where $\nabla$is the `grad` operator $\nabla \mathbf{V} = \mathbf{i}\frac{\partial \mathbf{V}}{\partial x} + \mathbf{j}\frac{\partial \mathbf{V}}{\partial y} + \mathbf{k}\frac{\partial \mathbf{V}}{\partial x}$ [5] This is a powerful tool - we can simply move the vehicle in the manner in which a particle would move in a location-dependent force-field. Equation 2.8 tells us the direction and magnitude of the force for any vehicle position $\mathbf{x}$.

---

[5] don't get confused here with the ∇ notation used for jacobians in the material covering non-linear estimation in coming pages!

The next question is what exactly do the potential functions look like. Well, there is no single answer to that — you can "roll your own". A good choice would be to make the potential of an obstacle be an inverse square function of the distance between vehicle and obstacle. We may also choose an everywhere-convex potential for the goal so that where ever we start, in the absence of obstacles, we will "fall" towards the goal point. Figure 2.6 shows two useful potential candidates (forgive the pun). Defining $\rho(\mathbf{x})$ as the shortest distance between the obstacle and the vehicle (at $\mathbf{x}$) and $\mathbf{x}_g$ as the goal point, the algebraic functions for these potentials are:

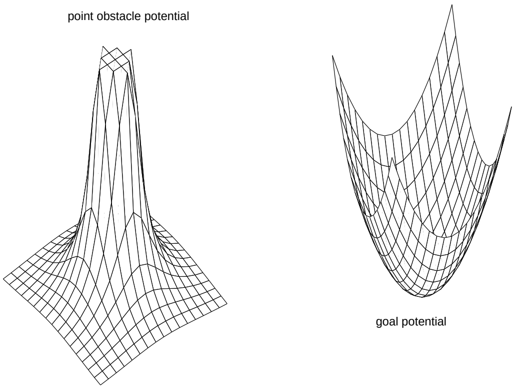


Figure 2.9: Two typical potential functions - inverse quadratic for obstacle and quadratic for the goal.

$$\mathbf{U}_{o,i}(\mathbf{x}) = \eta \begin{cases} \frac{1}{2}\left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0}\right)^2 & \forall \quad \rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

$$\mathbf{U}_g(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_g)^2 \tag{2.11}$$

The term ρ_0 places a limit on the region of space affected by the potential field — the virtual vehicle is only affected when it comes with ρ_0 of an obstacle. η is just a scale factor. Simple differentiation allows us to write the force vector exerted on a virtual point vehicle as:

$$\mathbf{F}_{o,i} = \begin{cases} \eta\left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0}\right)\frac{1}{\rho(\mathbf{x})^2}\frac{\partial\rho(\mathbf{x})}{\partial\mathbf{x}} & \forall \rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

where $\frac{\partial\rho(\mathbf{x})}{\partial\mathbf{x}}$ is the vector of derivatives of the distance function $\rho(\mathbf{x})$ with respect to x, y, z. We proceed similarly for the goal potential. So to figure out the force acting on a virtual vehicle and hence the direction the real vehicle should move in, we take the sum of all obstacle

Thank You for previewing this eBook

You can read the full version of this eBook in different formats:

- HTML (Free /Available to everyone)
- PDF / TXT (Available to V.I.P. members. Free Standard members can access up to 5 PDF/TXT eBooks per month each month)
- Epub & Mobipocket (Exclusive to V.I.P. members)

To download this full book, simply select the format you desire below

Free-eBooks.net