

Chapter 1

Basic concepts

1.1 Introduction

This chapter introduces to the main concepts on which this thesis is based. First of all, we will concentrate ourselves on describing the concept of sensors and their importance in self-perception of robots. Sensors, indeed, are very useful to allow a robot to localize itself and have an idea of surrounding environment; anyway, to make all this possible, it is necessary to design them in such a way to satisfy a series of requirements. Sensors, although their immense utility, furnish noisy measurements with which localization algorithms must deal. In this chapter, we will describe one particular kind of sensors, the ones based on the *time of flight (TOF)* method.

The main technique used by robots to estimate their current position and orientation is *odometry*, which is based on data returned by robot's encoders. We will analyse the two types of errors it introduces and how they have been experimentally measured.

Data furnished by sensors is often huge, so it is important to extract only relevant parts of it, in order to reduce time in computation. We will describe how such parts can be extracted and which properties we expect them to have.

Finally, we will propose a process based on least squares minimization, that allows to estimate a robot's state (in terms of position and orientation), exploiting the error between measured and expected state value.

1.2 Robot sensors

Robots can be considered, among human artefacts, as the ones closest to the concept of a living being, with their capabilities to sense world and change their own choices on the basis of what they perceive in the surrounding environment. Without a sensing system, robots could be just able to repeat the same task and wouldn't be able to adapt their decisions to an environment continuously changing around them. Thanks to sensors, robots can operate in places that are too dangerous for humans and interact with each other.

When designing a sensor, there is a small set of requirements that should be kept into account:

- a good field of view, to satisfy the needs of the greatest number of applications
- a good minimum and maximum range of detection
- the largest accuracy and resolution
- a great ability to detect objects in the environment, dealing with phenomena such as reflection and noise due to ambient interference
- high update frequency, to have real time data at disposal as fast as possible
- generated data should be concise and easy to interpret
- low costs in designing, construction and maintenance
- low power consumption
- small size and weight

1.2.1 Range sensors

In many cases, a robot is required to build a map of its surrounding environment and for such goal a measurement of range between itself and obstacles is needed. There exists a class of range sensors widely used for such purpose based on the so called *time of flight (TOF)* method; an ultrasonic, RF or optical source generates an energy wave that propagates into ambient and the distance between robot and an object is computed as the product of the velocity of wave and the time required to travel the round-trip distance, according to relation

$$d = vt \quad (1.1)$$

where d is the round-trip distance, v is the speed of propagation of energy wave and t is the elapsed time. The time interval t in previous equations is the one that elapses while the energy wave reaches the obstacle and gets back to robot; thus, to retrieve the real range between robot and target, t must be divided by two.

The signal emitted by robot reaches an object and walks back on the same way to the robot, where it will be detected by a receiver, that can be located close to emitter or integrated with emitter itself. As it is evident in (), the range to an object can be easily retrieved and no further knowledge about object's nature is necessary.

Similarly to all kinds of sensors, even TOF systems are subject to measurements errors, which can be summarized in the following short list:

- **variations in the speed of propagation of emitted signal**

- **uncertainty in determining when reflected signal was detected**

Such uncertainties depend on the fact that different surfaces reflect signal differently and this may cause time for detection of returned signal to rise. When a threshold on detection is established, more reflective objects are perceived as closer.

- **uncertainty in determining the exact round-trip time t**

It is due to problems correlated to inner timing circuitry.

- **interaction between originated signal and surfaces**

When source signal hits a surface, only part of it is reflected and perceived by the robot. The remaining part is propagated into ambient or passes through the hit object. Thus, it may happen that no signal is received by robot, particularly if emitting angle exceeds a certain threshold. The part of signal that is not sent back to robot, may be spread to environment and perceived by another robot's sensors.

1.3 Odometry

Odometry is probably the most used method to estimate the position of a robot at any instant in time. The reasons of its success must be found in good accuracy, low cost and high sampling rates. Anyway, the way odometry computes robot's position (by integrating motion information across time) leads to errors; in particular, errors in estimating robot's orientation lead to large position errors, which grow proportionally with the distance afforded by robot.

Despite limitations due to errors, odometry gained a great success in mobile robots field, for at the least four reasons:

- other kinds of measurements can be integrated to obtain a better accuracy
- in cases in which landmarks are used for helping to estimate robot's position, their number can be smaller, thanks to odometry's presence
- odometry allows to assume that robot is stuck in a certain pose for enough time to perceive all the landmarks in a given area and compute possible matchings with landmarks detected previously
- in some circumstances, because of the absence of external references (e.g. landmarks) or the impossibility to place any of them in a hostile environment, odometry can be the only way to estimate robot's position

Odometry exploits the measurements obtained from encoders mounted on robot's wheels to estimate its new pose. Let's suppose that, in a certain time interval, robot's wheels have pulses N_L and N_R . Let's denote with c_m the conversion factor that allows to convert pulses into a linear displacement. We can suppose that

$$c_m = \frac{\pi D_n}{nC_e} \quad (1.2)$$

where:

- D_m is the nominal wheel's diameter
- C_e is the encoder resolution (pulses per revolution)

- n is the reduction gear ratio between motor and drive wheel

The incremental linear displacements $\Delta U_{L,i}$ and $\Delta U_{R,i}$ for the two wheels can be computed as:

$$\begin{aligned}\Delta U_{L,i} &= c_m N_L \\ \Delta U_{R,i} &= c_m N_R\end{aligned}\tag{1.3}$$

We can now compute the displacement for robot's centerpoint ΔU_i (that will be taken as a reference for robot's position) and change in orientation $\Delta \theta_i$:

$$\begin{aligned}\Delta U_i &= \frac{\Delta U_R + \Delta U_L}{2} \\ \Delta \theta_i &= \frac{\Delta U_R - \Delta U_L}{b}\end{aligned}\tag{1.4}$$

where b is the distance between the two points where wheels touch the ground.

Thus, the new orientation of robot at time i is:

$$\theta_i = \theta_{i-1} + \Delta \theta_i\tag{1.5}$$

The new position, at the same instant of time, of its centerpoint will be:

$$\begin{aligned}x_i &= x_{i-1} + \Delta U_i \cos \theta_i \\ y_i &= y_{i-1} + \Delta U_i \sin \theta_i\end{aligned}\tag{1.6}$$

1.3.1 Systematic and Non-Systematic errors in odometry

As explained previously, the odometry exploits three simple equations to estimate current position of robot. All of them, are based on the assumption that any encoder's measurement can be translated into a linear displacement. Anyway, such assumption may lose its validity in certain circumstances; for instance, if a wheel slips, its encoder will register a measurement that is converted in a linear displacement, even if robot didn't move actually.

Slippage is only one of the possible causes of errors in odometry; they can be summarized and divided into the following two categories:

Systematic errors

- wheels with different diameters
- wheelbase different from nominal one
- wheels misalignment
- finite encoder resolution
- finite encoder sampling rate

Non-Systematic errors

- travelling on uneven floors
- travelling over objects
- slippage

Both errors categories are very difficult to manage; the first ones, in fact, are constantly present and accumulate over time; the latter, are unpredictable and robot must be able to react promptly when they manifest.

Across time, algorithms managing robot's position have thought it as surrounded by an error ellipse, denoting an area where robot may be in a certain moment. Ellipses grow with distance travelled, until an absolute measurement of position reduces ellipse's size. This method based on ellipses can be used to contain systematic errors only, since non systematic ones are unpredictable.

Estimating correctly the extent of odometric errors avoids further problems such as a wrong calibration of mobile platforms. In 1995, Borenstein and Feng developed a method based on a model which considers two errors to be dominant:

- the error due to different wheels' diameters $E_d = \frac{D_R}{D_L}$, with D_R and D_L , respectively, the actual diameters of right and left wheel
- the error due to uncertainty about real wheelbase $E_b = \frac{b_{actual}}{b_{nominal}}$, with b the wheelbase of robot

1.3.2 Measuring Systematic errors

Before describing Borenstein and Feng's method to estimate systematic errors, it is useful to analyse Borenstein and Koren's method (1987).

Let's suppose that robot starts its path from an initial position $\langle x_0, y_0, \theta_0 \rangle$ and has to move on a 4x4 meter square path. The robot is programmed to return to its initial position at the end of the path, but because of systematic errors, this will not happen accurately and a series of errors will be accumulated (*return position errors*):

$$\begin{aligned}\epsilon_x &= x_{abs} - x_{calc} \\ \epsilon_y &= y_{abs} - y_{calc} \\ \epsilon_\theta &= \theta_{abs} - \theta_{calc}\end{aligned}\tag{1.7}$$

We have denoted with the *abs* subscript the absolute position and orientation of the robot and with *calc* subscript the position and orientation of robot according to odometry.

This kind of experiment is not suitable for testing odometry for differential drive platforms. That is why Borenstein and Feng introduced their *bidirectional square-path* experiment. This time, the robot is required to walk the path both in clockwise and counterclockwise direction. The errors E_d and E_b could compensate each other when robot travelled on one direction only; now, travelling on both direction, the two errors sum up.

If we let the robot travel on both directions n times (usually, $n = 5$), at the end of each run robot will accumulate a pose error as explained in (). We can compute the center of gravity of these errors according to the following relations:

$$\begin{aligned}x_{cg}^{cw} &= \frac{1}{n} \sum_{i=1}^n \epsilon_{x,i}^{cw} \\ y_{cg}^{cw} &= \frac{1}{n} \sum_{i=1}^n \epsilon_{y,i}^{cw} \\ x_{cg}^{ccw} &= \frac{1}{n} \sum_{i=1}^n \epsilon_{x,i}^{ccw} \\ y_{cg}^{ccw} &= \frac{1}{n} \sum_{i=1}^n \epsilon_{y,i}^{ccw}\end{aligned}\tag{1.8}$$

The two absolute offsets of centers of gravity from origin are:

$$\begin{aligned}r_{cg}^{cw} &= \sqrt{(x_{cg}^{cw})^2 + (y_{cg}^{cw})^2} \\ r_{cg}^{ccw} &= \sqrt{(x_{cg}^{ccw})^2 + (y_{cg}^{ccw})^2}\end{aligned}\tag{1.9}$$

Finally, a measure of systematic odometric error can be obtained as

$$E_{sys} = \max(r_{cg}^{cw}, r_{cg}^{ccw}).\tag{1.10}$$

The orientation error ϵ_θ is not considered in E_{sys} , since each orientation error translates into a position error.

1.3.3 Measuring non-Systematic errors

The square path test that was used to estimate systematic error, can be implemented again for non-systematic case, adding to the path some artificial bumps. Since return errors are sensible to the positions of bumps, this time error ϵ_θ will be used. We can, then, compute an *average absolute orientation error*:

$$\epsilon_{\theta,avg}^{nonsys} = \frac{1}{n} \sum_{i=1}^n |\epsilon_{\theta,i,cw}^{nonsys} - \epsilon_{\theta,i,cw}^{sys}| + \frac{1}{n} \sum_{i=1}^n |\epsilon_{\theta,i,ccw}^{nonsys} - \epsilon_{\theta,i,ccw}^{sys}|\tag{1.11}$$

The use of absolute values in previous equations is needed to avoid that two return orientation errors with opposite signs compensate each other. Thus, if after first run we have $\epsilon_\theta = 1^\circ$ and after second one we have $\epsilon_\theta = -1^\circ$, we will not erroneously derive that $\epsilon_{avg}^{nonsys} = 0^\circ$.

1.4 Extracting features

In many different fields (machine learning and pattern recognition, just to mention a pair) the amount of data to manage (for example, the measurements returned by robot sensor) is too huge to be used in toto, so it becomes of great importance to extract only the necessary information, removing, moreover, all the redundancies that could make computation unfeasible. Thus., it is convenient to design processes that are able to extract information that is representative

of data and allows to overcome the problem of using data in its totality. Such processes perform what is called *features extraction*, i.e. they retrieve (and sometimes compute) elements (*features*) that are distinctive of data, with which it is easier to operate; features are required to be informative, discriminative and independent. For some uses, features are even required to maintain some properties across space and time.

Usually, features have a numerical nature, but sometimes they can present a different aspect (in some cases they can be strings of characters or histograms).

Before designing an extraction process, features' structure needs to be planned, in order to respect the goal of good representation of data and to obtain elements with which working will be easy and effective. The choice about structure and computation of features depends on their final use and algorithms' purposes; here is a short list of common choices made in different fields:

- histograms of the distribution of black pixels in characters recognition
- phonemes in speech recognition
- repetitive words or text structures in spam detection, inside email inboxes
- edges and corners in computer vision

As explained in previous section, data retrieved from odometry are often inaccurate and they can lead to errors in robot's position estimation; features, after their extraction from sensors' data, can add a precious information allowing to reduce the errors and gain a better estimate of robot position in time. Indeed, each feature carries information about its reciprocal position with respect to robot and it can be used to improve localization methods

It is necessary to underline that even a well designed process of extraction can return a number of features that is too huge to be handled; so, it becomes crucial, in such cases, to be aware about which features are really necessary and which can be ignored. This mainly depends on the context in which robot operates, the perception system and the way features are represented.

1.4.1 Extracting lines as features

When receiving data from a ranger sensor (for instance, a laser range), one needs to operate with pairs (ρ_i, θ_i) , which denote the position of i -th point in space, in polar coordinates, in robot's reference frame. Retrieving features from such information can be performed in two different ways:

- use a subset of points as features
- extract lines from the sequence of points, representing the part of environment seen by sensor during last scan

For the purpose of this thesis, we will consider the second option.

Across the years, many algorithms have been developed for extracting lines, starting from a laser's sequence of points, each with its pros and cons. Here, we will describe the so-called *split and merge algorithm*, which operates using a divide et impera logic, starting with an initial set of points, progressively divided into subsets (on the basis of a check condition), until satisfying the conditions for determining the existence of a line (the subset under inspection has not enough points to be divided into two subsets and/or no point satisfies check condition to proceed to another division). The main steps of procedure are described by Algorithm 1. The second step of algorithm can be performed using any fitting process, even

Algorithm 1 Split and merge algorithm

- 1: Initial: put all points in a set s_1 and insert s_1 into an initially empty list \mathcal{L}
 - 2: Get a set s_i from \mathcal{L} and fit a line through its points
 - 3: Find point P having the greatest distance d_P from line
 - 4: If d_P is less than a threshold, go back to step 2
 - 5: Otherwise, split s_i in P into two subsets s_{i1} and s_{i2} . Replace s_i with s_{i1} and s_{i2} in \mathcal{L} . Go back to step 2
 - 6: If all sets in \mathcal{L} have been checked, merge collinear lines
-

by connecting the first and last point in the sequence s_i . Each computed line is represented by its two extremes.

1.5 Least squares estimation

Let \mathbf{x} be a vector variable indicating the state of the system and let's suppose to have a series of functions

$$\{f_i(\mathbf{x})\}_{i=1:n}$$

which, given \mathbf{x} , predict its measurement.

Let's suppose, moreover, to have a series of real measurements \mathbf{z}_i . The goal is to estimate the state \mathbf{x} that best fits the

measurements $\mathbf{z}_{i:n}$.

We can, then, determine an error between each real measurement and the relative predicted one:

$$\mathbf{e}_i = \mathbf{z}_i - f_i(\mathbf{x}) \quad (1.12)$$

This error is assumed to be zero mean and normally distributed with information matrix Ω_i .

If we compute the squared form of error, we notice that it is a scalar and depends only on the state of the system:

$$e_i(\mathbf{x}) = \mathbf{e}_i(\mathbf{x})^T \Omega_i \mathbf{e}_i(\mathbf{x}) \quad (1.13)$$

Then, the problem of finding the state that best fits the real measurements can be reduced to find the state \mathbf{x}^* such that

$$\begin{aligned} \mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}} F(\mathbf{x}) \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i e_i(\mathbf{x}) \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \mathbf{e}_i^T(\mathbf{x}) \Omega_i \mathbf{e}_i(\mathbf{x}) \end{aligned} \quad (1.14)$$

A general solution to the problem is to derive the global error function and find its nulls, but this would result in a complex solution which doesn't admit closed forms. So, proceeding with numerical approaches is a more practical path.

If a good initial guess is at disposal and the measurement functions are smooth in the neighborhood of minima, we can solve problem using iterative (local) linearizations. The iterative process is composed of three steps:

- linearize the problem around current initial guess
- solve a linear system
- compute a set of increments to be applied to current estimate in order to get closer to minima

The first step is linearizing the problem and this can be done using a Taylor expansion:

$$\begin{aligned} \mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) &\simeq \mathbf{e}_i(\mathbf{x}) + \frac{\partial \mathbf{e}_i}{\partial \mathbf{x}} \Delta\mathbf{x} \\ &= \mathbf{e}_i + \mathbf{J}_i \Delta\mathbf{x} \end{aligned} \quad (1.15)$$

We can now place the Taylor expansion in the squared error:

$$\begin{aligned} e_i(\mathbf{x} + \Delta\mathbf{x}) &\simeq \\ &\mathbf{e}_i^T \Omega_i \mathbf{e}_i + \mathbf{e}_i^T \Omega_i \mathbf{J}_i \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{J}_i^T \Omega_i \mathbf{e}_i + \Delta\mathbf{x}^T \mathbf{J}_i^T \Omega_i \mathbf{J}_i \Delta\mathbf{x} \\ &= \underbrace{\mathbf{e}_i^T \Omega_i \mathbf{e}_i}_{c_i} + 2 \underbrace{\mathbf{e}_i^T \Omega_i \mathbf{J}_i}_{\mathbf{b}_i^T} \Delta\mathbf{x} + \underbrace{\Delta\mathbf{x}^T \mathbf{J}_i^T \Omega_i \mathbf{J}_i \Delta\mathbf{x}}_{\mathbf{H}_i} \\ &= c_i + 2\mathbf{b}_i^T \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{H}_i \Delta\mathbf{x} \end{aligned} \quad (1.16)$$

The global error is the sum of the squared errors, so, from (3) and (5), we have:

$$\begin{aligned} F(\mathbf{x} + \Delta\mathbf{x}) &\simeq \sum_i (c_i + 2\mathbf{b}_i^T \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{H}_i \Delta\mathbf{x}) \\ &= \sum_i c_i + 2(\sum_i \mathbf{b}_i^T) \Delta\mathbf{x} + \Delta\mathbf{x}^T (\sum_i \mathbf{H}_i) \Delta\mathbf{x} \end{aligned} \quad (1.17)$$

where

$$\mathbf{b}^T = \sum_i \mathbf{e}_i^T \Omega_i \mathbf{J}_i \quad (1.18)$$

$$\mathbf{H} = \sum_i \mathbf{J}_i^T \Omega_i \mathbf{J}_i \quad (1.19)$$

The expression of global error we have just obtained is quadratic in $\Delta\mathbf{x}$ and can be minimized; for this purpose, we can compute its derivative with respect to $\Delta\mathbf{x}$ in the neighborhood of current solution \mathbf{x} :

$$\frac{\partial F(\mathbf{x} + \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} \simeq 2\mathbf{b} + 2\mathbf{H}\Delta\mathbf{x} \quad (1.20)$$

with the optimal solution reached for

$$\Delta\mathbf{x}^* = -\mathbf{H}^{-1} \mathbf{b} \quad (1.21)$$

Thus, an iterative algorithm can be designed; it will execute the following steps:

- linearize the system around current guess \mathbf{x} and compute $\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x})$ as described in () and ()
- compute \mathbf{b}^T and \mathbf{H} as described in () and ()
- solve the system to get a new optimal increment $\Delta\mathbf{x}^*$, as indicated in ()
- update the previous estimate: $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^*$