Artificial Intelligence – An Introduction to Robotics

Tim Niemueller and Sumedha Widyadharma

July 8, 2003

Abstract

This document gives a short introduction to the basics of robotics in the context of artificial intelligence. It describes the very basics of robotics like sensors and effectors, gives an overview on robotic history, and introduces some basic problems encountered in modern robotics. It describes possible solutions to those problems without going deeply into theory. The problems introduced are perception, basic pose description, transition and sensor models, localization as a special case of perception (Monte Carlo Localization, Extended Kalman Filter), representation of environment (workspace and configuration space), path planning (cell decomposition, potential fields, skeletonization, Voronoi diagrams, and probabilistic roadmaps), movement of robots, and some real-life examples.

This document was created accompanying a talk in the context of the proseminar "Artificial Intelligence" in summer semester 2003 at the RWTH Aachen, Chair of Computer Science V, Knowledge-based Systems Group.

1 Introduction

1.1 Capek and his Robots

The term "Robot" can be traced back to Karel Capek's play "R.U.R. Rossum's universal robots" (in 1921) that comes from the Czech word for "corvee".

1.2 A brief History of Robots

Robotics are based on two *enabling technologies*: Telemanipulators and the ability of numerical control of machines.

Telemanipulators are remotely controlled machines which usually consist of an arm and a gripper. The movements of arm and gripper follow the instructions the human gives through his control device. First telemanipulators have been used to deal with radio-active material.

Numeric control allows to control machines very precisely in relation to a given coordinate system. It was first used in 1952 at the MIT and lead to the first programming language for machines (called APT: Automatic Programmed Tools).

The combination of both of these techniques lead to the first programmable telemanipulator. The first industrial robot using these principle was installed in 1961. These are the robots one knows from industrial facilities like car construction plants.

The development of *mobile robots* was driven by the desire to automate transportation in production processes and autonomous transport systems. The former lead to driver-less transport systems used on factory floors to move objects to different points in the production process in the late seventies. New forms of mobile robots have been constructed lately like insectoid robots with many legs modeled after examples nature gave us or autonomous robots for underwater usage.
Since a few years wheel-driven robots are commercially marketed and used for services like "Get and Bring" (for example in hospitals).

Humanoid robots are being developed since 1975 when Wabot-I was presented in Japan. The current Wabot-III already has some minor cognitive capabilities. Another humanoid robot is "Cog", developed in the MIT-AI-Lab since 1994. Honda's humanoid robot became well known in the public when presented back in 1999. Although it is remote controlled by humans it can walk autonomously (on the floor and stairs).

In science fiction robots are already human's best friend but in reality we will only see robots for specific jobs as universal programmable machine slave in the near future (which leads to interesting questions, see [17]).

1.3 Definition: What is a Robot?

Robots are physical agents that perform tasks by manipulating the physical world. They are equipped with sensors to perceive their environment and effectors to assert physical forces on it (covered in more detail in next section). As mentioned before Robots can be put into three main categories: manipulators, mobile robots and humanoid robots.

1.4 Robotics and AI

Artificial intelligence is a theory. The base object is the *agent* who is the "actor". It is realized in software. Robots are manufactured as hardware. The connection between those two is that the control of the robot is a software agent that reads data from the sensors, decides what to do next and then directs the effectors to act in the physical world.

2 Theory and Application

2.1 Robot Hardware

2.1.1 Sensors

Sensors are the perceptual interface between robots and their environment. [1] On the one hand we have *passive sensors* like cameras, which capture signals that are generated by other sources in the environment. On the other hand we have *active sensors* (for example sonar, radar, laser) which emit energy into the environment. This energy is reflected by objects in the environment. These reflections can then be used to gather the information needed.
Generally active sensors provide more information than passive sensors. But they also consume more power. This can lead to a problem on mobile robots which need to take their energy with them in batteries.
We have three types of sensors (no matter whether sensors are active or passive). These are sensors that either

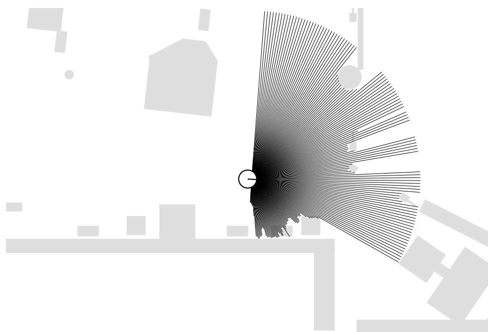


Figure 1: Laser Range Scanner

- record distances to objects or
- generate an entire image of the environment or
- measure a property of the robot itself.

Many mobile robots make use of *range finders*, which measure distance to nearby objects. A common type is the sonar sensor (see [6] for an example). Alternatives to sonar include radar and laser (see Figure 1). Some range sensors measure very short or very long distances. Close-range sensors are often *tactile sensors* such as whiskers, bump panels and touch-sensitive skin. The other extreme are long-range sensors like the Global Positioning System (GPS, see [8, 10]).

The second important class of sensors are *imaging sensors*. These are cameras that provide images of the environment that can then be analyzed using computer vision and image recognition techniques.

[1] For an overview of available sensor hardware especially for small robots see [4].

The third important class are *proprioceptive sensors*. These inform the robot of its own state. To measure the exact configuration of a robotic joint motors are often equipped with shaft decoders that count the revolution of motors in small increments. Another way of measuring the state of the robot is to use force and torque sensors. These are especially needed when the robot handles fragile objects or objects whose exact shape and location is unknown. Imagine a ton robot manipulator screwing in a light bulb.

2.1.2 Effectors

Effectors are the means by which robots manipulate the environment, move and change the shape of their bodies.

To understand the ability of a robot to interact with the physical world we will use the abstract concept of a *degree of freedom (DOF)*. We count one degree of freedom for each independent direction in which a robot, or one of its effectors can move. As an example lets contemplate a rigid robot like an autonomous underwater vehicle (AUV). It has six degrees of freedom, three for its (x,y,z) location in space and three for its angular orientation (also known as yaw, roll and pitch). These DOFs define the kinematic state of the robot. This can be extended with another dimension that gives the rate of change of each kinematic dimension. This is called dynamic state.

Robots with nonrigid bodies may have additional DOFs. For example a human wrist has three degrees of freedom – it can move up and down, side to side and can also rotate. Robot joints have 1, 2, or 3 degrees of freedom each. Six degrees of freedom are required to place an object, such as a hand, at a particular point in a particular orientation.

The manipulator shown in Figure 2 has exactly six degrees of freedom, created by five revolute joints (R) and one prismatic joint (P). Revolute joints generate rotational motion while the prismatic joints generates sliding motion.

If you take your arm as an example you will notice, that it has more than six degrees of freedom. If you put your hand on the table you still have the freedom to rotate your elbow. Manipulators which have more degrees of freedom than required to place an end effector to a target location are easier to control than robots having only the minimum number of DOFs.

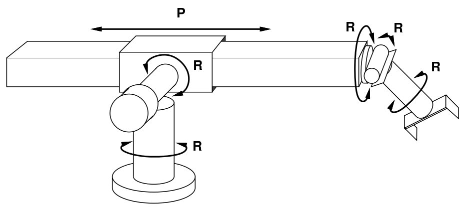


Figure 2: The Stanford Manipulator [12]

Mobile robots are somewhat special. The number of degrees of freedom do not need to have corresponding actuated elements. Think of a car. It can move forward or backward, and it can turn, giving it two DOFs. But if you describe the car's kinematic configuration you will notice that it is three-dimensional. On a flat surface like a parking site you can maneuver your car to any (x,y) point, in any orientation. You see that the car has 3 *effective DOFs* but only 2 *controllable DOFs*. We say a robot is *nonholonomic* if it has more effective DOFs than controllable DOFs and *holonomic* if the two numbers are the same.

Holonomic robots are easier to control than nonholonomic (think of parking a car: it would be much easier to be able to move the car sideways). But holonomic robots are mechanically more complex. Most manipulators and robot arms are holonomic and most mobile robots are nonholonomic.

2.1.3 Movement

For mobile robots a special group of effectors are the mechanisms the robot uses for locomotion, including wheels, tracks, and legs. The *differential* drive consists of two independently actuated wheels – one on each side. If both wheels move at the same velocity, the robot moves on a straight line[2]. If they move in opposite directions, the robot turns on the spot.

An alternative is the *synchro drive*[3], in which each wheel can move and turn around its own axis. This could easily lead to chaos. But if you assure the constraint that all wheels always point in the same direction and move with the same speed your robot is save.

[2]If you have ever tried to implement that (for example with a Lego Mindstorm) you know this can become hard especially with cheap hardware, see [7]

[3]Staying with the Mindstorm example you might want to have a look at [1].

Both differential and synchro drives are nonholonomic. Some more expensive robots use holonomic drives, which usually involve three or more wheels and can be oriented and moved independently.

2.1.4 Power Sources

Robots need a power source to drive their effectors. The most popular mechanism for both manipulator actuation and locomotion is the *electric motor*.
Other possible ways are *pneumatic actuation* using compressed gas and *hydraulic actuation* using pressurized fluids. They have their application niches but are not widely used.

2.1.5 Bits and Pieces

Most robots have some kind of digital communication like wireless networks. Especially today those modules get cheaper. They can be used for communication between robots or for some kind of back link to the robots home station.
Finally you need a body frame to hang all the bits and pieces.

2.2 Robotic Perception

A robot receives raw sensor data from its sensors. It has to map those measurements into an internal representation to formalize this data. This process is called *robotic perception*. This is a difficult process since in general the sensors are noisy and the environment is partially observable, unpredictable, and often dynamic.
Good representation should meet three criteria: They should

- contain enough information for the robot to make a right decision
- be structured in a way that it can be updated efficiently
- be natural, meaning that internal variables correspond to natural state variables in the physical world

Filtering and updating the belief state is not covered here as it was covered in earlier presentations. Some topics are Kalman filters and dynamic Bayes nets. Information can also be found in [12] chapter 15.

2.2.1 Where am I? (Localization)

A very generic perception task is *localization*. It is the problem of determining where things are. Localization is one of the most pervasive perception problems in robotics. Knowing the location of objects in the environment that the robot has to deal with is the base for making any successful interaction with the physical world – manipulators have to know where the objects are they have to manipulate and mobile robots need to know where they are in order to find their way to a defined target location. There are three increasingly difficult flavors of localization problems:

- *Tracking* – If the initial state of the object to be localized is known you can just track this object.
- *Global localization* – In this case the initial location of the object is unknown you first have to find the object. After you found it this becomes a tracking problem.
- *Kidnapping* – This is the most difficult task. We take the object the robot is looking for and place it somewhere else. Now the robot again has to localize the object without knowing anything about the object's location. Kidnapping is often used to test the robustness of a localization technique under extreme conditions.

Basic Terms To keep things simple we assume that the robot moves slowly in a plane and that it has an exact map of the environment.

The pose of the robot can be described by its cartesian coordinates (x,y) and its heading θ (see Figure 3). We formalize this pose in a vector $X_t = (x_t, y_t, \theta_t)^\top$. We now define our transition in time as $t \mapsto t+1$ with the angular velocity ω_t and translational velocity v_t for small time intervals Δt. This gives us a *deterministic transition model*:

$$\hat{X}_{t+1} = f(X_t, \underbrace{v_t, \omega_t}_{a_t}) = X_t + \begin{pmatrix} v_t \Delta t \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta t \end{pmatrix}$$

Next, we need a *sensor model*. There are two different approaches for sensor models. The first assumes that the sensors detect stable and recognizable features of the environment called *landmarks*. The sensors give the distance and bearing to the landmarks. If the current state of the robot is $X_t = (x_t, y_t, \theta_t)^\top$ and the robot knows a landmark at location $(x_i, y_i)^\top$ (it may be on the map for instance). Again we assume that there is no noise. Then the range and bearing can be calculated by simple geometry[4].

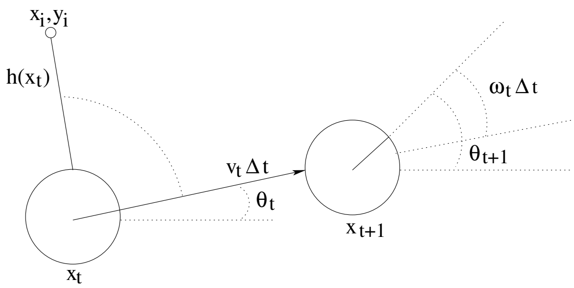


Figure 3: A simplified kinematic model of a robot

$$\hat{z}_t = h(x_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

A different sensor model is often appropriate for range scanners. Such a sensor may produce vectors of range values $z_t = (z_1, \ldots, z_M)^\top$. These values are called the beams that the sensor emits. The angle for each beam is fixed (for example z_1 could always have angle $\gamma = 0°$ meaning that is the "forward beam"). So if the robot is in pose x_t then $\hat{z}_j$ is the exact range along the jth beam direction from x_t to the nearest obstacle. You get $\hat{z}_j$ through some kind of convert function with $\hat{z} = f_c(z_j)$ that maps the sensor value to a real distance value.

Comparing the range scan model to the landmark model, we see that the range scan model is more independent since there is no need to identify a landmark before the sensor data can be interpreted. But if we can detect an identifiable landmark it can provide immediate localization. As we will see in section 3.2 (page 13) some robots implement both techniques at the same time.

Monte Carlo Localization (MCL) MCL is essentially a particle filter algorithm. The requirements are a map of the environment showing the regions the robot can move to and an appropriate motion model and sensor model. We assume that the robot uses range scanners for localization. First we want to talk about the theory. The algorithm takes the given map and creates a population of N samples by a given probabilistic distribution (if the robot has some knowledge about where it is – for example it knows that it is in a particular room – the algorithm could take care of this and we would see particles only in that room). Then we start a continuous update cycle. The localization starts at time $t = 0$. Then the update cycle is repeated for each time step:

- Each sample is propagated forward by sampling the next state value X_{t+1} given the current value X_t for the sample, and using the transition model given.
- Each sample is weighted by the likelihood it assigns to the new evidence, $P(e_{t+1} \mid x_{t+1})$.
- The population is resampled to generate a new population of N samples. Each new sample is selected from the current population; the probability that a particular sample is selected is proportional to its weight. The new samples are unweighted.

(a) Initialization (b) Unsure due to symmetry (c) Localized

Figure 4: An example for a Monte Carlo Localization

So as the robot gathers more knowledge about the environment by analyzing the range scanner data it resamples the population and the particles concentrate at one or more points. At some point in time all points are in one cluster point. That is the location of the robot in the field.

There are some problems you can encounter. For example if you have a completely symmetric map like the one in Figure 5 a robot can walk around those rooms and has always more cluster points that could be the location of the robot.

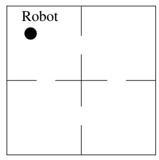


Figure 5: Symmetric map

Now we want to study the example shown in Figure 4. First the robot initializes the samples (graphically the particles in the map). Since it does not know anything the particles look uniformly distributed. The robot has created the population of N samples from a given probability distribution (since may also be randomly chosen particles in reachable areas).

The robot starts to move in north-east direction and gets into the door. This means that the robot first pipes all of its particles through its transition model. So all particles move about the same way on the map as the robot moves (that is data the robot can easily gather since it has advised the effectors which actions to take – like moving the wheels at a given speed v for time Δt). Then it weighs all particles. Some particles may have moved into a wall so they will get a very small weight (zero if that is the smallest possible value). Now the robot again does a range scan. It detects that it is in a door with a slim wall on the one side and a bulky one on the other. This configuration appears twice in the map so the particles in this positions get very high weights (but on both doors almost the same). Since the probability that a specific sample is taken into the new population is proportional to its weight almost all new samples are from those two areas shown in Figure (b). But there is still uncertainty about the location. The robot yet has to move on and gather more data to be sure about its location. The robot moves further and creates a new population. Now it has enough data so that only the particles moved in the west of the map (after moved with the transition model) get high weights, the particles in the east of the map get very low weights. So now all particles are in one cluster point. The robot has localized its position and can be pretty sure that its estimation is correct. A more in-depth description of MCL may be found at [2].

Extended Kalman Filter (EKF) The Kalman filter is another major way to localize. The Kalman filter was described in 1960 by R. E. Kalman as a solution to the discrete linear filtering problem.

It is a continuous update cycle with alternating *prediction* (time update) phase and *correction* (measurement update) phase. This can be used in robotics to estimate the current position of the robot. The robot's state X_t is calculated by its posterior probability with $P(X_t \mid z_{1:t}, a_{1:t-1})$ by a Gaussian. The problem with Gaussian beliefs is, that they are only closed on linear motion models f and linear measurement models h. Usually motion models and measurement models are not linear resulting in a non-closed Gaussian probability distribution. Thus, localization algorithms *linearize* the motion and sensor models. Linearization is a local approximation of a nonlinear function by a linear function. The linearization method used for EKF

[4] In most cases you would have to care about noise and use a Gaussian noise for the sensors measurements.

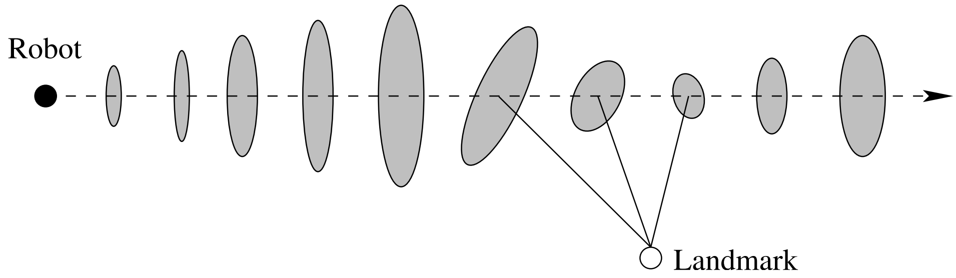


Figure 6: Localization using the extended Kalman filter

is called Taylor expansion. An introduction to the generic Kalman filter can be found in [19].

So how is this used for localizing a robot? We assume that the robot has an exact map of the environment (a priori known or learned), knows one or more special features (like landmarks) on this map, and has a transition and sensor model that can be linearized with a first degree Taylor expansion.

In Figure 6 you see that the robot starts at a given position on the left of the picture. Now as it moves on the dashed line the uncertainty about its location increases (illustrated as ellipse). This is because of the Gaussian noise of the sensors and effectors. Imagine a robot with a differential drive. The robot could easily drift a little bit in one or more direction or be a little bit faster or slower as expected (depending on the grip its wheels have on the ground). When the robot detects one of the known landmarks the uncertainty about the location decreases.

2.2.2 Multi-Object Localization – Mapping

Until now we only discussed the localization of a single object, but often one seeks to localize many objects. The classical example of this problem is robotic mapping.

In the localization algorithms before we assumed that the robot knew the map of the environment a priori. But what if it does not? Then it has to generate such a map itself. Humans have already proven their mapping skills with maps of the whole planet. Now we will give a short introduction how robots can do the same.

This problem is often referred to as *simultaneous localization and mapping (SLAM)*. The robot does not only construct a map, but it must do this without knowing where it is. A problem is that the robot may not know in advance how large the map is going to be.

The most widely used method for the SLAM problem is EKF. It is usually combined with landmark sensing models and requires that the landmarks are distinguishable. Think of a map where you have several distinguishable landmarks of unknown location. The robot now starts to move and discovers more and more landmarks. The uncertainty about the location of the landmarks and itself increases with time. When the robot finds one landmark he already discovered earlier again the uncertainty of its position and of *all* landmarks decreases. For more details see [12] and [16].

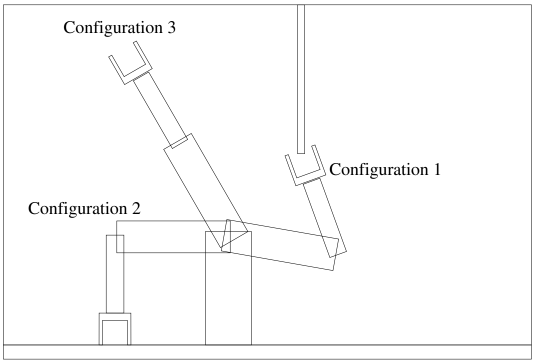
2.3 Planning to Move

After landing safely on the surface of a new planet, our autonomous ScoutBot somehow decides, that it should analyze a strange stone formation a couple hundred meters south. Wheels are set in motion and the robot finally reaches the new location.

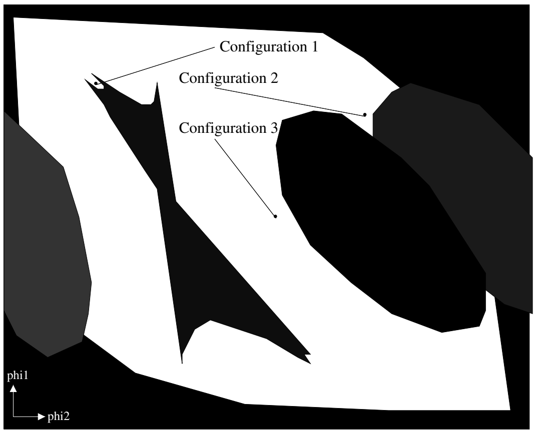
This is the most basic movement problem called *point-to-point motion problem*, which is to determine the right motions to change the position of the robot or its effectors to a new configuration without collision with known obstacles.

Furthermore it needs to take several examples from the stones. It uses a built-in drill to extract some stone fragments. This is a more complicated problem called the *compliant motion problem*. It comprises the motion of the robot while in contact with an obstacle.

Before an algorithm for one of the above problems can be developed, we need to find a representation which enables us to properly describe our problems and find solutions for them. The intuitive attempt



(a) The workspace



(b) the configuration space

Figure 7: A configuration space example

would be to describe each movable piece of a robot by its cartesian coordinates. This method is known as *workspace representation* because the robot and its environment share the same coordinate system. The problem with this representation is, that even in an obstacle free environment, a robot can never reach all possible workspace coordinates because some parts are fixed to each other and cannot move independently but have to obey certain *linkage constraints*. A solution to solve motion problems defined over workspace coordinates would have to generate paths that take these constraints into account which is not easy, as the state space is continuous and the constraints are nonlinear.

Another, more practical approach is called *configuration space representation* where the state of the robot is represented by the configuration of its joints instead of cartesian coordinates.

The example robot has two joints. We can represent the state of the robot by just the two angles φ_1 and φ_2. If there are no obstacles present, one could simply connect start and end configuration with a straight line and let the robot rotate every joint to its new configuration with a constant speed. A configuration space can always be decomposed into two parts. The *free space* which comprises all attainable configurations and its complement called *occupied space*, which covers all configurations blocked by obstacles or restricted joint capabilities. This leads to a problem of configuration spaces. As it is in general easy to convert configuration space to workspace coordinates (which is called *kinematics*) the inverse operation (*inverse kinematics*), to generate a configuration from working space coordinates, is a difficult task as for certain configurations the solution is seldom unique therefore the practical approach is to generate a configuration and apply it to the robot, to see if it is in free space.

2.3.1 Cell Decomposition

One approach towards planning a path in configuration space is to simply map the free space onto a discrete raster which reduces the path planning problem within each of the *cells* to a trivial problem (move on a straight line) as it is completely traversable. By this means, we end up with a discrete graph search problem, which are relatively easy to solve.The simplest *cell decomposition* matches the free space onto a regularly spaced grid but more complex methods are possible.

Cell decomposition has the advantage of an easy implementation, but also suffers some deficiencies:

- The amount of grid cells increases exponentially with the dimension of the configuration space.
- *Polluted* cells (cells which contain free *and* occupied space) are problematic. If included, they might lead to solutions, which are in reality blocked, if omitted, some valid solutions would be missing.

The latter problem could be solved by further subdivision of the affected cells (Where recursive splitting of such cells could again increase the amount exponentially) or by requiring *exact* cell decomposition by allowing irregularly shaped cells. Those new cells would need to have an 'easy' way to compute a traversal through them, which would lead to complex geometric problems.

Another problem is the fact that a path through such a cell grid can contain arbitrary sharp corners, which would be impossible to perform by real life robots. Such a path can lead very close to an obstacle, rendering even the slightest motion errors fatal.
To maximize movement tolerance and at the same time minimizing the path length one can employ a so called *potential field*. A *potential field* is a function over the configuration space itself whose value grows with the proximity to an obstacle. If integrated into the path finding algorithm as an additional cost and weighed accurately, it can lead to relatively short *and* secure paths.

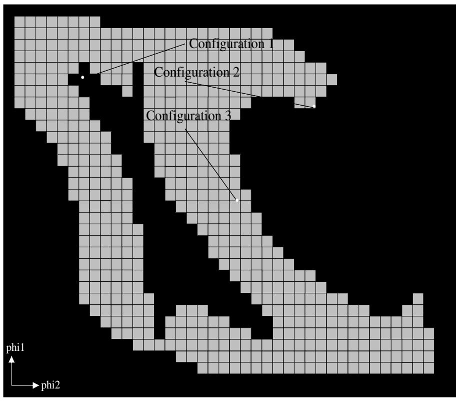


Figure 9: Decomposed free space

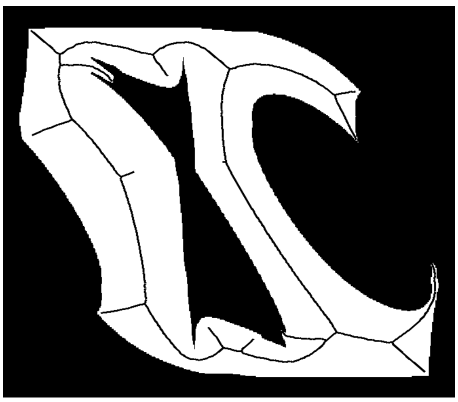
2.3.2 Skeletonization Methods

The second major path planning algorithm family attempts to reduce the complexity of the free space to a one-dimensional representation. The so called *skeleton* of the configuration space.

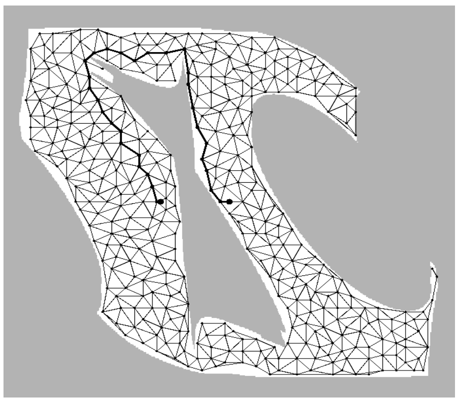
Voronoi Diagrams The graph shown in Figure 8 is a Voronoi graph of the free space (see [18] for more information). A robot using this method would first change its configuration to a point on the skeleton, which is due to the nature of a Voronoi diagram always possible with a straight line movement, then follow the skeleton to the point closest to its destination and make another straight line move to the final position. This method again reduces the original path planning problem to a discrete graph search along the Voronoi diagram. In general the solution is not the shortest possible, but on the other hand it maximizes clearance.

Disadvantages of Voronoi graph techniques are:

- They are difficult to apply to higher dimensional configuration spaces
- They tend to induce large detours if the free space is wide open



(a) Voronoi diagram



(b) Probabilistic roadmap

Figure 8: Skeletonization methods

- They can be hard to compute, especially in configuration space, where obstacles can take strange shapes

Probabilistic Roadmaps To avoid detours in wide open free spaces one needs more routes. *Probabilistic roadmaps* (see Figure 8) are generated by simple picking random configurations and discarding them, if they do not fall into free space. If enough valid configurations have been found, any two nodes which have an 'easy' connection (a straight line) are connected with each other. Thus we get a random graph in the robot's free space. If the start and end point of the desired configuration are added to the graph, we only need to solve the resulting discrete graph search problem.

A disadvantage of probabilistic roadmaps is that they are theoretically incomplete (a bad choice of random points might make some paths not solvable) The probability of an error can be reduced by introducing more points, taking into account geometric properties of of the configuration space, concentrating point generation in areas that are likely to contain a good path and bidirectional search. With these improvements probabilistic roadmaps tend to scale better to high dimensional configuration spaces than most other path planning techniques.

2.4 Moving

Now, as we may have found a path to reach a certain target, we need to actually move the robot or its effectors there. This can be more tricky than one would expect, because robots are physical agents. They cannot follow arbitrary paths, except at arbitrary slow speeds. In general, a robot cannot simply set his effectors to a certain position, but needs to exert forces to put them to the desired new position.

2.4.1 Dynamics and Control

Better motion plans could be generated if, instead of basing them on kinematic models of the robot, *dynamic state models* would be used. Such models are typically expressed through *differential equations*. Unfortunately the complexity of dynamic space is greater than that of kinematic space rendering motion planning based on them for more than simplest robots impossible.

Thus in practice simpler kinematic path planners are common. To overcome their limitations however a common technique is to use a separate *controller* to keep the robot on track. A controller which tries to keep a robot on it is preplanned *reference path* is referred to as reference controller.
The most basic controller would be a *proportional* Controller (P controller) which apply a force proportional to the deviations amount to try to compensate it. This could still lead to an infinite oscillating behavior around the planned path. The control a_t generated by a P controller might look like this:

$$a_t = K_P(y(t) - x_t)$$

Where x_t is the state of the robot at time t and K_P the *gain factor* which determines, how strong the controller tries to correct the discrepancy between $y(t)$ *(the desired state)* and x_t *(the actual state)* Lower K_P values only slow down the oscillating behavior, but do not solve the issue.

Reference controllers can be

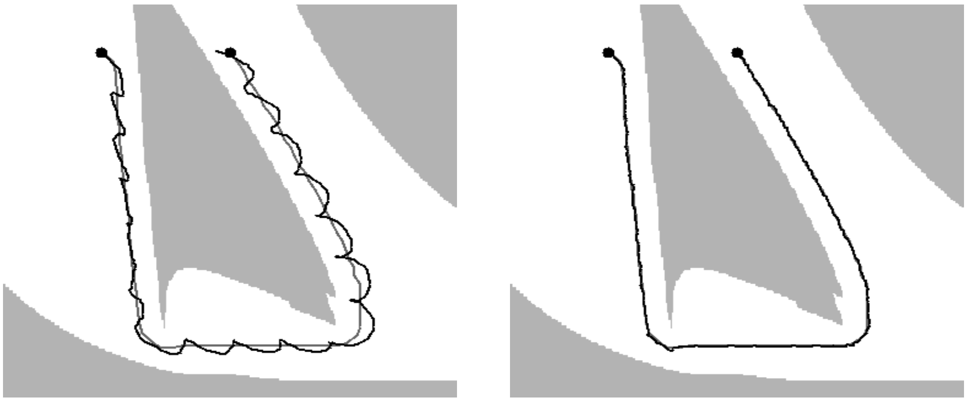
- stable, if small divergences lead to a bounded error between the robot and the reference signal.
- strictly stable, if it is able to return to the reference path after a disturbance.

P controllers are obviously stable, but not strictly stable.

The simplest strictly stable controller is the *proportional derivative* controller (PD controller) which is described by the following equation:

$$a_t = K_P(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t}$$

The addition of the differential component (with the gain factor K_D) counteracts the proportional term reducing the overall response to perturbations in case of the error changing quickly over time. If an error



(a) Proportional controller (b) Proportional derivative controller

Figure 10: Types of controllers

persists and does not change, the derivative will vanish and the proportional part will dominate the choice of control. Nevertheless PD controllers may fail to regulate an error down to zero because sometimes over-proportional feedback is necessary to do so. The solution to this problem lies in adding the integrated error over time to the equation:

$$a_t = K_P(y(t) - x_t) + K_I \int (y(t) - x_t)dt + K_D \frac{\partial(y(t) - x_t)}{\partial t}$$

Again, K_I is the components gain factor. The integral ensures, that long-lasting deviation from the reference path gets accumulated resulting in a growth of a_t which will then make the controller react stronger, thus correcting the path.

This kind of controller is called a *PID controller*. They are widely used in industry, for a variety of control problems.

2.4.2 Potential Field Control

Potential fields, which we already used as an additional cost function in motion planning, can also be used to directly generate robot motion. To do this, we need to define an attractive potential field, which pushes the robot towards its destination, and another field, which pushes it away from obstacles. The goal becomes the global minimum of the resulting potential, all other points have values based on their distance to the target configuration and their proximity to obstacles. Potential fields can be efficiently calculated for any configuration and scale very good with the dimensionality of the configuration space. Unfortunately the path can eventually be trapped in a local minimum. Nevertheless this method is great for local robot control but still requires global planning. Another point is, that it is a kinematic method and might fail if the robot is moving quickly.

2.4.3 Reactive Control

In cases, where we cannot get accurate models of our environment or where noise and lack of computational power might prevent application of the methods known so far another approach must be made. In some cases a *reflex agent* (reactive control) is more appropriate. The controller bases his behavior not on a model of the environment, but on immediate feedback (like short range proximity scanners or even reacting on being stuck). This interplay of a (simple) controller and a (complex) environment is often referred to as *emergent behavior*. As no model is perfect, this has to be seen in a wider sense. Other methods rely upon training data, rather than a model of their environment.

3 Reality

3.1 Types of Robots

Now we will see, what robots are mainly used for nowadays.

3.1.1 Hard working Robots

Traditionally robots have been used to replace human workers in areas of difficult labor, which is structured enough for automation, like assembly line work in the automobile industry (the classical example) or harvesting machines in the agricultural sector. Some existing examples apart from the assembly robot are:

- Melon harvester robot
- Ore transport robot for mines
- A robot that removes paint from large ships
- A robot that generates high precision sewer maps

If employed in a suitable environment robots can work faster, cheaper and more precise than humans.

3.1.2 Transporters

Although most autonomous transport robots still need environmental modifications to find their way they are already widely in use. But building a robot which can navigate using natural landmarks is probably no more science fiction. Examples of currently available transporters are:

- Container transporters used to load and unload cargo ships
- Medication and food transport systems in hospitals
- autonomous helicopters, to deliver goods to remote areas

3.1.3 Insensible Steel Giants

As robots can be easily shielded against hazardous environments and are somewhat replaceable, they are used in dangerous, toxic or nuclear environments. Some places robots have helped cleaning up a mess:

- In Chernobyl robots have helped to clean up nuclear waste
- Robots have entered dangerous areas in the remains of the WTC
- Robots are used to clean ammunition and mines all around the world

For the same reasons robots are sent to Mars and into the depth of the oceans. They explore sunken ships or walk the craters of active volcanoes.

3.1.4 Servants and Toys

Robots may not yet be a common sight in our world, but we already encounter them in many places. Many modern toys like the Sony Aibo are conquering todays children's life. Robots are developed that will help older people to have a better and more secure life (ball-robot Rollo, see [9]). Nowadays, they start to come to us as toys or household helpers. Their time has just begun.

3.2 Showcase: AllemaniAC Robocup Robots

As a real life example we will take a look at the robotic football team of the RWTH Aachen which participates in the Robocup tournament (see [13]). You see a schematic view in Figure 11.

3.2.1 Sensors

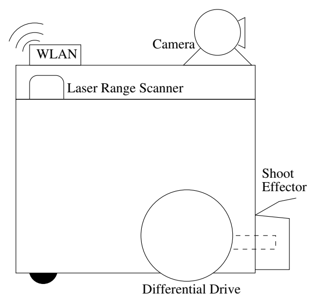


Figure 11: Schematic side view of an AllemaniAC

The part that is currently worked on is localization: The robot has to know where it is, where its team mates and opponents are and where the ball and goal are. The robot uses a laser range scanner to detect obstacles like its opponents. It is mounted about 30 cm above the floor. This way the ball is not detected by the range scanner. For this purpose the robot has a camera. It is also used to track the ball after the robot detected it and find to the goal and landmarks that are placed in the corners of the field for supporting the localization. There are plans to remove those landmarks as well in the future. The robot uses a form of MCL for localization.

The WLAN module is used to tell team mates if the robot can see them or if they see the ball. This is like real players shouting where other players should move or where the ball is. Right now the robots do not communicate directly to each other but they have a central controlling computer besides the field that acts like a message preprocessor and command repeater.

3.2.2 Effectors

The AllemaniAC uses a differential drive with shaft decoding for movement. The motor is very powerful with 2.5 kW moving the robot with a speed of 12 km/h.

Of course a football player has to interact with the ball. Therefore it has a "shoot effector" that consists of two electro magnetic spools. These spools enable the robot to control the strength that it kicks the ball with. This is currently a unique feature of the AllemaniAC (as far as we know).

3.2.3 CPU and Communication

The robot has two computers on board with a Pentium III (933MHz) each. One is used for image recognition and the other for higher functions like localization and communication with base station. The robots cannot communicate with each other yet. On tournaments there is usually too much traffic on all WLAN channels from other teams and the folks around. Since the robots still use 802.11b there is not enough bandwidth available to work properly and reliable under such conditions.

3.2.4 Software

Although we did not cover software architectures used for robot construction in this talk we want to present some principles of the underlying software of the AllemaniAC robot.

The AllemaniAC is based on *ReadyLog*, an enhanced version of *IndiGolog* (see [3]). ReadyLog extends IndiGolog with continuous actions (tracking the ball must be done all the time while the robot is moving) and actions with uncertain outcome (if the robots plays the ball towards the goal he cannot be sure that the ball hits the goal. An opponent might intercept it).

3.2.5 Current State

As seen in a demonstration [5] there is still a lot of work that has to be done before the robots really play as a team and do not interfere with each other. But that is why this is science and not a commercial product. We will see how well the AllemaniACs does perform in this years Robocup in Padua (see [14]). For more information about the AllemaniACs see [15].

[5] You may find several videos and more information on [15] in the documents section

References

[1] Doug Carlson. Synchro drive robot platform. *http://www.visi.com/~dc/synchro/index.htm*, 1998.

[2] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[3] G. De Giacomo, Y. Lesperance, H. Levesque, and R. Reiter. Indigolog overview. *http://www.cs.yorku.ca/~alexei/ig-oaa/indigolog.htm*, 2001.

[4] Bob Grabowski. Small Robot Sensors. *http://www.contrib.andrew.cmu.edu/~rjg/websensors/robot_sensors2.html*, 2003.

[5] G. Görz, C.-R. Rollinger, and J. Schneeberger (Hrsg.). *Handbuch der künstlichen Intelligenz, 3. Auflage*. Oldenbourg Verlag München Wien, 2000.

[6] Kam Leang. Minibot Sonar Sensor Howto. *http://www.atsemi.com/article/Howto.htm*, 1999.

[7] Lego. Lego mindstorms tutorial on correcting course. *http://mindstorms.lego.com/eng/community/tutorials/tutorial.asp?tutorialid=project6*, 2003.

[8] Trimble Navigation Limited. Trimble - What is GPS? *http://www.trimble.com/gps/what.html*, 2003.

[9] Helsinki University of Technology. Moving Eye - Virtual Laboratory Excercise on Telepresence, Augmented Reality, and Ball-Shaped Robotics. *http://www.automation.hut.fi/iecat/moving_eye/home.html*, 2001.

[10] Peter Röbke-Doerr. Navigation mit Sateliten. *c't*, 01/2003:150–151, Jan 2003.

[11] Stuart Russel and Peter Norvig. *Artificial Intelligence. A Modern Approach*. Prentice Hall, 1995.

[12] Stuart Russel and Peter Norvig. *Artificial Intelligence. A Modern Approach, 2nd Edition*. Prentice Hall, 2003.

[13] Robocup Team. Robocup. *http://www.robocup.org*, 2003.

[14] Robocup Team. Robocup 2003. *http://www.robocup2003.org*, 2003.

[15] RWTH Aachen Robocup Team. AllemaniACs. *http://robocup.rwth-aachen.de*, 2003.

[16] Sebastian Thrun. Robotic Mapping: A Survey. Technical report, School of Computer Science – Carnegie Mellon University, 2003.

[17] Andrew Tong. Star Trek TNG Episode: The Measure Of A Man. *http://www.ugcs.caltech.edu/st-tng/episodes/135.html*, 1995.

[18] Eric W. Weisstein. Voronoi diagram. *http://mathworld.wolfram.com/VoronoiDiagram.html*, 1999.

[19] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical report, Department of Computer Science – University of North Carolina at Chapel Hill, 2003.