Lab class: Autonomous robotics
Background material

Jean-Stephane Jokeit, Oliver Lomp,
Mathis Richter, Jan Tekülve, Prof. Gregor Schöner

Summer term 2016

Contents

"We will talk only about machines with very simple internal structures, too simple in fact to be interesting from the point of view of mechanical or electrical engineering. Interest arises, rather, when we look at these machines or "vehicles" as if they were animals, in a natural environment. We will be tempted, then, to use psychological language in describing their behavior. And yet we know very well that there is nothing in these vehicles that we have not put there ourselves."

— Valentino Braitenberg

Figure 1: The e-puck robot.

1 The e-puck robot

The e-puck, shown in Figure 1, is a small mobile robot that was developed primarily for research. Each e-puck has two wheels, one on the left side of its body and one on the right. Each wheel can be controlled individually via a servo motor. The wheels have a diameter of 40 mm and the distance between the two wheels is 53 mm. The robot is also equipped with eight infrared sensors, a VGA camera (resolution of 640x480 pixels), three microphones, a loudspeaker, and several LEDs. The power for the e-puck is provided by a rechargeable battery, which can be accessed and replaced at its bottom. The robot can be controlled wirelessly via a bluetooth connection. The e-puck has a small blue reset button. Pressing this button instantly stops the robot's wheels. Figure 1 shows a picture of an e-puck.

1.1 Drive mechanism

The e-puck is moved by its two wheels, each of which is driven by an individual servo motor. The smallest distance that the robot can move is achieved by applying a single electric pulse to the motors. This distance measures about 0.13 mm. On this low level, the unit of velocity for the e-puck can be measured in pulses per second. The maximal velocity is 1023 pulses/s. If that velocity is set for both wheels, the e-puck can therefore cover a distance of $0.13\,\mathrm{m\,s^{-1}}$. If the motor velocities of both wheels are set to negative values,

the robot moves backwards.

Each motor is equipped with an *encoder* that counts the pulses sent to it. Note, that this value overflows at $2^{15} = 32768$ (positive or negative). Reading out these encoder values thus enables us to compute the distance covered by each wheel. However, this method is not always accurate as the actual wheel velocity often deviates from the one specified due to various perturbing influences.

1.2 Infrared sensors

The robot is equipped with eight infrared sensors that can both emit and register infrared light, a type of light at a wavelength just below the spectrum visible to humans. The eight infrared sensors are attached to the e-puck robot at directions of $\pm 13°$, $\pm 45°$, $\pm 90°$ and $\pm 135°$ (relative to the forward direction of the robot). They are numbered clockwise, starting with the sensor to the right of the front at $-13°$ (see Figure 2). Each infrared sensor consists of an emitter (a light emitting diode; LED), and a receiver (a semiconductor transducer). The receiver can be used on its own to detect ambient light, or in conjunction with the emitter to measure distances to surrounding objects.

1.2.1 Measuring distance

The e-puck's infrared emitters and receivers can be used in conjunction in order to detect the distance from a sensor to the closest surface. For this, the emitter of each sensor sends a brief pulse of light, which is reflected from obstacles and is detected by the corresponding infrared receiver. The values of the infrared receivers, again in the range from 0 to 4095, indicate the difference of light intensity with and without light emission. If an object is closer to the sensor, more light is reflected, yielding a higher difference in light intensity and larger values from the sensor.

The response of the sensor does not depend linearly on the distance from the object and the function describing this dependency is also influenced by the color, surface structure, and material of the object. In addition, this function may be different for individual sensors. This means that by default, the sensors do not produce an accurate measure of the distance to the nearest object. The accuracy can be increased by calibrating the sensors, that is, approximating the function that maps sensor readings to distances for each sensor. Below, we show one possible method for such an approximation.

For the approximation, we first address the differences between ranges and accuracies of different sensors. To do this, we measure the mean responses of each sensor, once in the absence of obstacles, and once when they are

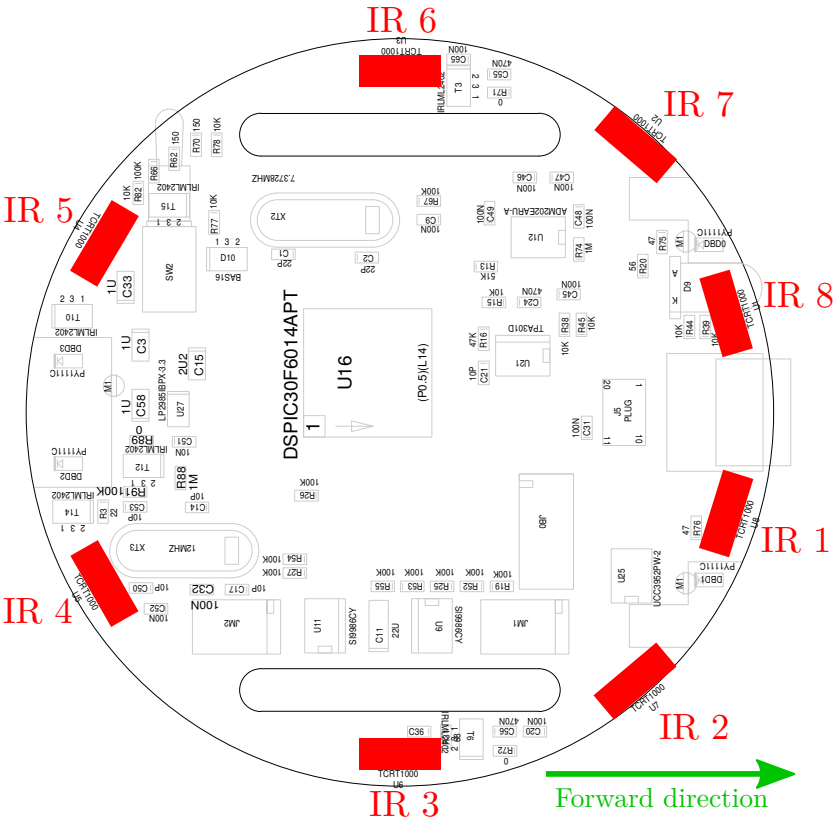


Figure 2: Top-view schematic of the e-puck robot showing the positions of infrared sensors.

as close as possible. We use the mean values values to normalize the sensor response so that the result lies (roughly) in the interval $[0, 1]$ (due to noise, the resulting values may exceed the boundaries of this interval, but our approach is resistant to small deviations like this).

We then find a function that inverts the sensor characteristic — that is, given a normalized IR value v, we want a function $d(v)$, whose result is a distance in mm or cm.

We record the mean normalized response for obstacles at different distances for a single sensor. Then, we perform a logarithmic regression on these points, by which we obtain

$$d(v) = -0.96 \cdot \ln(v) - 0.1, \tag{1}$$

which yields a distance in cm. Note, that this function is only valid for a single sensor on the exact e-puck we used for this. for other sensors and e-pucks, it may be necessary to further modify the parameters of this function to obtain more accurate readings.

Measuring distances is particularly problematic in the presence of interfering light sources or when measuring the distance to obstacles whose surface absorbs infrared light (e.g., black coarse fabrics) or is translucent (e.g., glass).

1.3 Controlling the e-puck in Matlab

A list of commands relevant for controlling the e-pucks during the lab class is shown in Table 2.

Note that there are also some practical considerations when working with the e-pucks. Each of the commands listed in Table 2 requires sending a command to the robot and receiving a reply. This is generally a slow process (how slow exactly depends on the robot, but each call typically takes at least 20 ms in faster cases), therefore it is advisable to minimize the amount of calls to these functions.

Another practical issue to consider concerns the calls to `kOpenPort` and `kClose`. We recommend storing the handle obtained from `kOpenPort` in a global variable that is passed to individual scripts and functions rather than including the `kOpenPort` and `kClose` commands within the script itself. This has three main reasons: first, your scripts will occasionally exit with an error. This may lead to `kClose` not being called. Restarting the script without a manual call to `kClose` (which is easily forgotten) then means that the handle to the already open communication channel is lost, and connecting to the robot becomes impossible without at least restarting Matlab. Second, there is a small chance of failure for each `kOpenPort` call that also leads to the

`h=kOpenPort()`	opens the serial port and returns a handle h to the robot
`kClose(h)`	closes the serial port
`kSetSpeed(h,l,r)`	sets the velocity for the left (`l`) and right (`r`) motor (both in pulses/s)
`kStop(h)`	stops the robot
`kGetSpeed(h)`	returns a vector `[left, right]` with the velocities set for both wheels (in pulses/s). Note, that this does not actually measure the velocities.
`kGetEncoders(h)`	returns a vector `[left, right]` with the values of the wheel encoders (in pulses)
`kSetEncoders(h,l,r)`	sets the encoders to the given values; `kSetEncoders(h)` sets both encoders to zero
`p = kProximity(h)`	returns a vector with distance measurements for the eight infrared sensors with `p = [p(1),...,p(8)]`; the values are in the range $[0, 4095]$

Table 2: Relevant commands for controlling the e-puck robot. Here, `h` always denotes the *handle* for the robot that is created when the connection is established with `kOpenPort`.

need to restart Matlab. The risk of this actually happening can be reduced by avoiding frequent calls to `kOpenPort`. Third, the first command sent to the e-pucks will always fail due to complications in the communication protocol. By keeping the channel open, you will encounter this problem less frequently. Follow up each `kOpenPort` with a `kSetSpeed(h, 0, 0)` command to work around this limitation.

2 Kinematics and odometry

The e-puck robot does not have any prior knowledge about the world. Without sensors or additional programming, it does not have any information about the location of objects or target positions; it does not even know where in the world it is, nor how it is oriented. However, this information is

required to navigate to targets and avoid obstacles on the way.

The present section focuses on how to infer the robot's position in the world based on the data from the wheel encoders. Analogously, we look at how to generate appropriate commands for the wheels to turn the robot to desired orientations. Both computations will build on our knowledge of the e-puck's physical measurements which can be found in Section 1.

2.1 Coordinate frames

Due to constraints in the geometry of its chassis, the e-puck is mostly restricted to moving on flat surfaces. Within the lab class, this means that we can describe the robot's position and orientation in a two-dimensional coordinate system aligned with the plane of the surface. On this plane, we use two types of coordinate frames in which we specify the positions of the robot, targets and obstacles. The first is an *ego-centric* coordinate frame, the second is the *allo-centric* one.

The origin of the *ego-centric* (or *local*) coordinate frame is always at the center of the robot. The x-axis is always facing straight ahead, that is, it points in the heading direction of the robot. The y-axis, by convention, always points (orthogonally) toward the left of the x axis when looking at the system from above. The gray, tilted coordinate frame in Figure 3 illustrates an ego-centric coordinate frame.

The origin of the *allo-centric* (or *global*) coordinate frame is positioned arbitrarily and is independent of the robot's movement. It can be placed, for instance, at a prominent landmark in the world or the corner of a piece of paper. The orientation of this coordinate frame is arbitrary but fixed, and the robot's heading direction is specified with respect to this orientation. The black coordinate frame in Figure 3 shows an allo-centric coordinate frame and the way it relates to the ego-centric coordinate frame (shown in gray).

2.2 Inverse kinematics

Directing the robot to given coordinates in the world, possibly even with a specific final orientation, can be a hard problem to solve. There are infinitely many routes the robot could drive. Some may seem like the routes a human would drive with a car, others may look choppy or completely random. Selecting one of these routes requires setting up constraints on what makes a 'good' route. This problem is analogous to what is usually referred to as *inverse kinematics* in problems involving movement of robotic arms: figuring out a good trajectory of the arm to a target given that there are many joints to move in various directions.

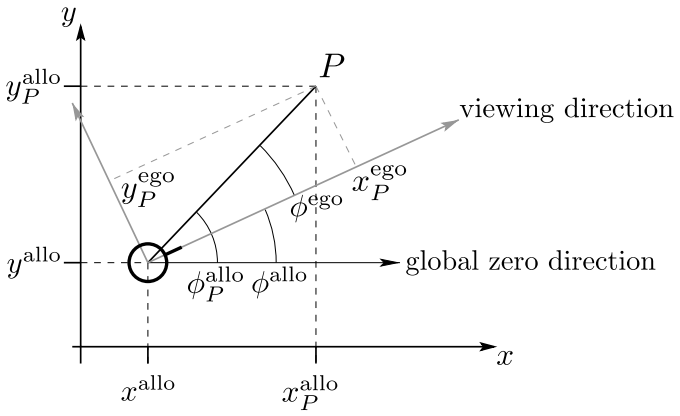


Figure 3: With an allo-centric coordinate frame (black coordinate system) it is possible to describe the global position $(x^{\text{allo}}, y^{\text{allo}})$ and orientation ϕ^{allo} of the robot. In the egocentric coordinate frame (tilted, gray coordinate system), the robot is located in the center and its forward direction is aligned with the local zero orientation. The position of a point P can be given either in global coordinates $(x_P^{\text{allo}}, y_P^{\text{allo}})$ or in local coordinates of the robot as $(x_P^{\text{ego}}, y_P^{\text{ego}})$. The direction of the point from the robot is ϕ_P^{ego}.

Assume that our robot has a starting position $(x_{\text{start}}, y_{\text{start}})$ and starting orientation ϕ_{start} in the allocentric reference frame. We want the robot to drive to the target position, $(x_{\text{target}}, y_{\text{target}})$, with a final orientation ϕ_{target}. How can we get the robot move to these coordinates?

As a first step, we can simplify the problem by decomposing it into two separate movements. The first one rotates the robot on the spot so that it faces the target point. The second moves the robot forward in a straight line until it has traversed the distance between the start and target point.

To realize the first part of the simplified movement, we need a way to turn the robot on the spot by an arbitrary angle, $\Delta\phi$. To do so, we turn one wheel in one direction and the other wheel in the opposite direction with the same speed. This makes the wheels drive along circular arcs whose radius, r, is half the distance between the robot's wheels. The length b of this circular arc can be computed by taking a fraction, $\frac{\Delta\phi}{2\pi}$, of the full circumference, $2\pi r$, of the circle:

$$b = \frac{\Delta\phi}{2\pi}\, 2\pi r = \Delta\phi\; r, \tag{2}$$

Inserting the wheel distance, l, into the equation, we get

$$b = \Delta\phi \cdot \frac{l}{2}. \tag{3}$$

In order for a wheel to traverse this distance, it has to travel with a

speed v for the time interval Δt. This follows from the equation describing a straight-line movement:

$$b = v \cdot \Delta t \tag{4}$$

$$\Leftrightarrow v = \frac{b}{\Delta t} \tag{5}$$

Note that either the time or the speed of the movement can be chosen freely (for the lab class, specify the velocity).

This equation also allows us to realize the second part of our movement towards the target. To traverse the distance between start and target, we can again use Equation 5 to determine the speed or duration of a movement for the given distance.

Note that in both cases, the speed v has the unit $\mathrm{mm\,s^{-1}}$. Because the robot expects commands to be in pulse/s, you first have to convert it using the physical measurements of the robot (see Section 1).

Please make sure that your code follows the mathematical convention that rotations around a positive angle are counter-clockwise.

2.3 Forward kinematics

The robot does not necessarily execute commands perfectly. Thus, to keep track of the robot's position and orientation, we need to rely on readings from its sensors. The problem of calculating the robot's position from these sensor readings is called *forward kinematics*.

For the e-pucks, the main sensor for tracking its location are the wheel encoders. They provide an estimate of how much distance each wheel has covered in a given time interval. To calculate the change in position from this, we separate the movement of the robot into segments so that during each segment the speed of both wheels remains, ideally, constant.

Let us now consider a single such segment. Given the robot's position x, y and orientation ϕ before the movement, we now want calculate its position x', y' and orientation ϕ' after the movement. Using the encoders, we can determine the distance traversed by the left (b_L) and right (b_R) wheel. Because we assumed that the wheel speeds were constant, only four cases can have occurred: The first case is that the robot simply did not move ($b_L = b_R = 0$). The second is that it moved in a straight line (i.e., $b_L = b_R$). The third is that it moved on the spot (i.e., $b_L = -b_R$). In these cases, determining the new position and orientation can be computed trivially by applying basic trigonometric functions (the same equations that were used in Section 2.2). In the fourth case ($|b_L| \neq |b_R|$), the robot has moved along a circular arc, as

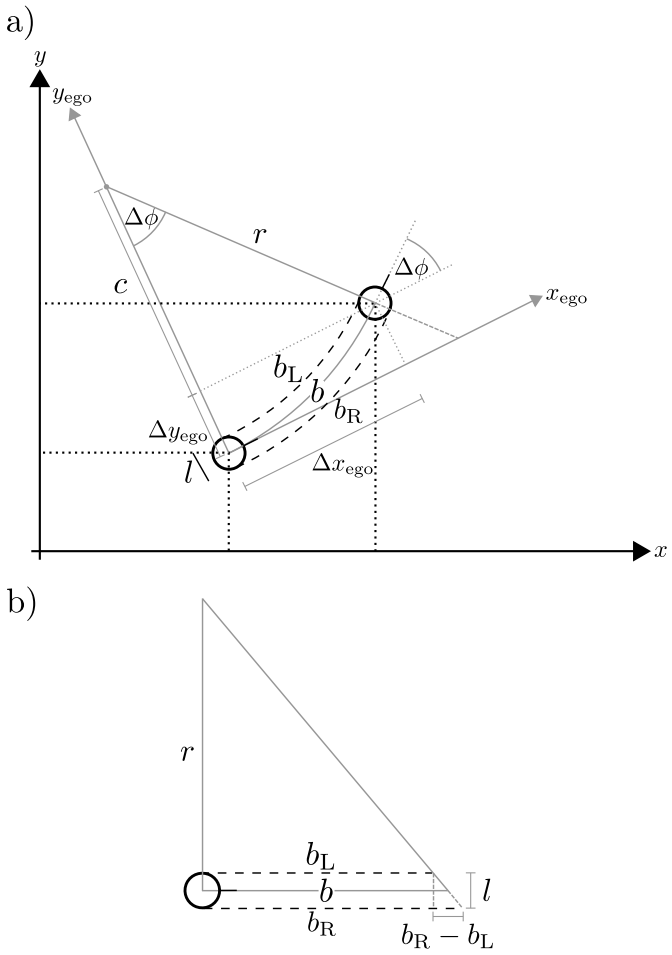


Figure 4: An illustration of the derivation of the change in position when the robot has traversed a circular arc.

shown in Figure 4. To find out the coordinate changes in this case, we need to find out the radius r of the arc as well as the angle $\Delta\phi$ covered by the arc.

To begin, we determine the radius by constructing a new triangle as shown in Figure 4b. We know the involved quantities of the smaller triangle defined by the sides of length l and $b_R - b_L$. By construction, this triangle has the same angles as to the larger triangle with sides r, and b. Thus, the following must hold:

$$\frac{r}{b} = \frac{l}{b_R - b_L} \Leftrightarrow r = \frac{b\,l}{b_R - b_L} \tag{6}$$

Using Equation 2, we find the change of the angle to be

$$\Delta\phi = \frac{b}{r} = \frac{b_R - b_L}{l}. \tag{7}$$

To compute the change of the robot's position, Δx_{ego} and Δy_{ego}, we use the triangle with sides r and c as shown in Figure 4a. The length c is given by

$$c = r\cos(\Delta\phi). \tag{8}$$

Using trigonometry, we get

$$\Delta x_{\text{ego}} = r\sin(\Delta\phi) \tag{9}$$

Because by construction, $r = \Delta y_{\text{ego}} + c$, it follows that

$$\Delta y_{\text{ego}} = r - c = r(1 - \cos(\Delta\phi)). \tag{10}$$

To transform these changes from the local (ego-centric) coordinate system to the global (allo-centric) one, we must rotate and translate them accordingly.

The change of the heading direction is not affected by translation, and rotation amounts to an addition. The new heading direction is therefore

$$\phi' = \phi + \Delta\phi. \tag{11}$$

The change in position is first rotated according to

$$\Delta x_{\text{allo}} = \Delta x_{\text{ego}}\cos(\phi) - \Delta y_{\text{ego}}\sin(\phi) \tag{12}$$

$$\Delta y_{\text{allo}} = \Delta x_{\text{ego}}\sin(\phi) + \Delta y_{\text{ego}}\cos(\phi). \tag{13}$$

Then, we translate it by adding the origin of the ego-centric coordinate system. The new position is thus

$$x' = x + \Delta x_{\text{allo}} \tag{14}$$

$$y' = y + \Delta y_{\text{allo}}. \tag{15}$$

Note that the method we present here for estimating the position of a robot via odometry is affected both by systematic and random errors, for example, insufficient wheel traction, effects of the gear system, deviations of the e-puck chassis from the specifications and so on.

3 The dynamical systems approach to navigation in autonomous robotics

The dynamical systems approach is geared toward behavioral control of a robot. In particular, this approach can be applied for navigation.

In the dynamical systems approach, control is local. This means that we restrict ourselves to information that the robot can obtain directly with its on-board sensors. Global information, such as the layout of obstacles in the world is not used. As a consequence, when faced with the task to traverse a maze of obstacles, we cannot determine the path we want to take beforehand. Instead, we use the available local information to instantaneously control basic behavioral variables such as the heading direction and forward speed of the robot. These variables are controlled by specifying a law for their rates of change that depends on the current sensory information. This law is formulated as a dynamical system.

In this section, we therefore start by briefly introducing the basics of dynamical systems. As an example, we then use a dynamical system to control the heading direction of a robot in order for it to approach a given target. Finally, we extend this dynamical system to make it avoid obstacles it encounters along its way to the target.

3.1 Dynamical systems and differential equations

A dynamical system is described by one or more dynamic variables, $\vec{x}(t)$, whose values change over time t. Here, we only look at systems with a single such variable, $x(t)$. The change of this variable may be described by a differential equation,

$$\dot{x} = \frac{dx}{dt} = f(x,t), \tag{16}$$

where $f(x,t)$ is a function that specifies how the variable changes depending on its current value. To make this concrete, let us look at the linear differential equation

$$\dot{x} = f(x) = -\alpha x.$$

How can we find the actual value, $x(t)$, of the system at a given point in time? This is exactly the problem of finding the solution of the equation; for the system above, it is known[1] to be an exponential decay of the form

$$x(t) = x_0 \exp(-\alpha t).$$

[1] You can look this up in any textbook on dynamical systems.

To see why this solves the system, look at the derivative of the equation at some arbitrary point in time:

$$\dot{x}(t) = \frac{dx}{dt} = -\alpha x(0) \exp(-\alpha t) = -\alpha x(t).$$

Note that we need to specify the initial value of the system, $x(t = 0)$. Solving this dynamics is therefore also called an *initial value problem*. Also note that, for $\alpha > 0$ and $t \to \infty$, the system will approach ± 0 regardless of the choice of initial value. This is the attractor of the system; we discuss this in detail below.

3.2 Numerical approximation

Analytically finding the solution for a dynamical system is generally a complex problem. An alternative approach is to approximate the solution using numerical methods. One such method is the Euler method, which involves transforming the dynamics into a difference equation. Since a derivative is a limit case of the inclination of a secant, it can be approximated by the inclination of a secant with a fixed size Δt

$$\dot{x}(t_0) = \lim_{t \to t_0} \frac{x(t) - x(t_0)}{t - t_0} \approx \frac{x(t_0 + \Delta t) - x(t_0)}{\Delta t}$$

We can use this relationship to derive an approximation, $\hat{x}(t_0 + \Delta t)$, of the value of x at time $t_0 + \Delta t$ if we already know the approximation, $\hat{x}(t_0)$, at a previous time, t_0:

$$\hat{x}(t_0 + \Delta t) = \hat{x}(t_0) + \Delta t \cdot \dot{x}(t_0)$$

Here we have approximated the real function by a linear one for a small interval of time. This means that we disregard higher-order terms and thus introduce an approximation error. This error grows larger the more important the higher order terms become; generally, larger step sizes, Δt, increase the error. Figure 5 illustrates this graphically for a large time step (note, that $t_0 = 0$ was chosen for clarity). Thus, the approximation improves in accuracy the smaller the time step Δt. However, smaller time steps also lead to a higher computational burden.

3.3 Attractors and repellors

To control a robot using a dynamical system, we have to construct a differential equation that produces the desired behavior, for instance turning the robot toward a target. For the dynamical systems approach, this means

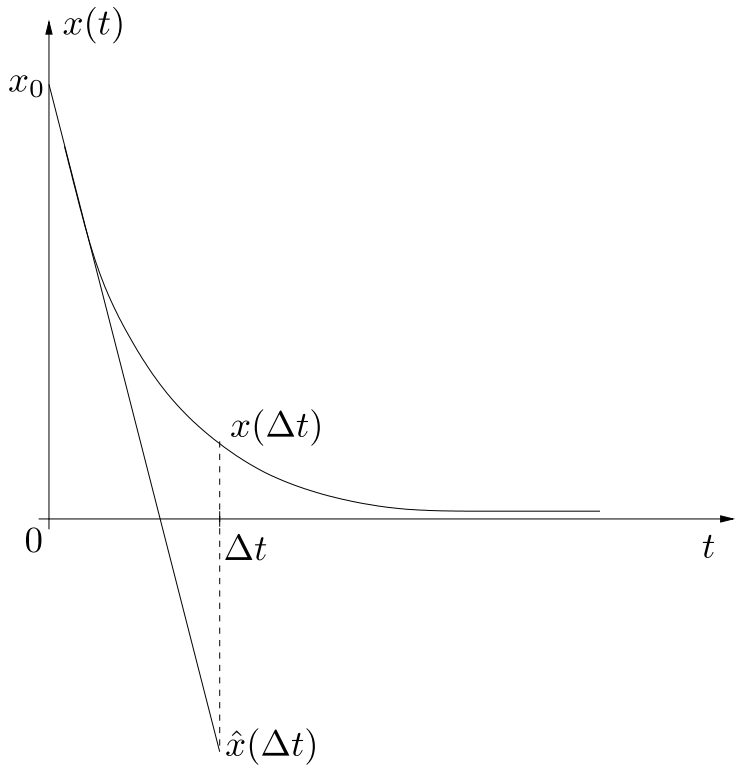


Figure 5: For values of Δt that are large, the approximation error may become large as well. Here, $\hat{x}(\Delta t)$ strongly deviates from the analytically calculated $x(\Delta t)$.

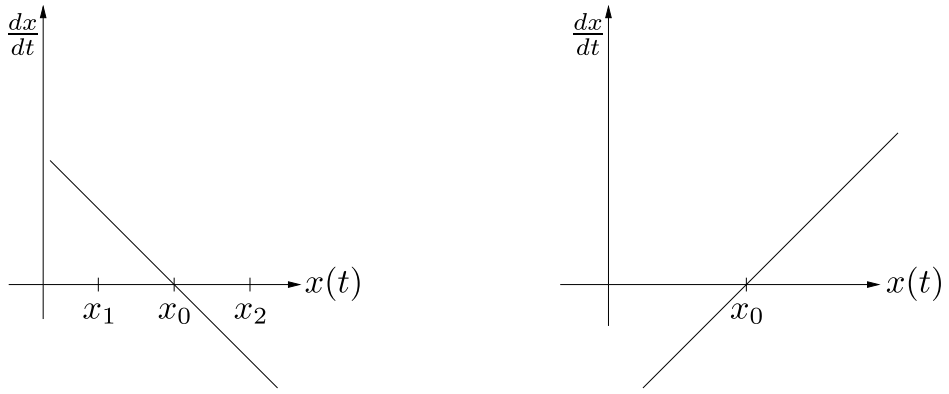


Figure 6: Attractor (left) and repellor (right) of a dynamical system

constructing a dynamics that has attractors at target states that "pull" the system's state towards them and repellors that "push" the system's state away from undesired states.

In the simplest case, attractors and repellors are fixed points of the system. A fixed point is a state of the dynamical system in which the rate of change, $\dot{x}$, is equal to zero. Once the system has reached such a state, it will remain there unless it is driven out of the state by some external influence. If we plot the rate of change $\dot{x}$ of a system against the state variable x, as we have done in Figure 6 for a linear system, the fixed points are the zero crossings in the graph. This kind of plot is commonly referred to as a *phase plot*.

Let us first focus on the dynamical system shown in the left plot of Figure 6. At position x_0 it has a zero crossing with a negative slope. If the system were in a state x_1, where $x_1 < x_0$, the given change $\dot{x}$ of the system would be positive. The system would thus go to a state larger than x_1, moving closer toward x_0. However, if the system were in a state $x_2 > x_0$, the change would be negative. The system would go toward a state smaller than x_3, also approaching x_0. Were the system in state x_0, the change would be zero and it would remain in this state. Such a fixed point is called an *attractor*.

The right plot of Figure 6, on the other hand, shows a repellor. The inclination at the zero crossing is positive. Using analogous arguments as above, we can see that the system moves away from that point. Note, however, that the system would remain at x_0 in the absence of external influences if it started exactly in this state.

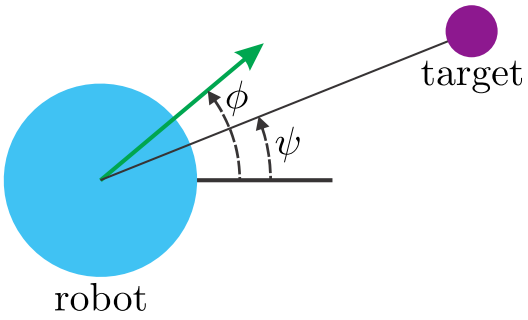


Figure 7: Control of the heading direction for approaching a target. The current heading direction of the robot is ϕ, the direction of the target is ψ. Both angles are defined relative to the zero-direction of a global coordinate system.

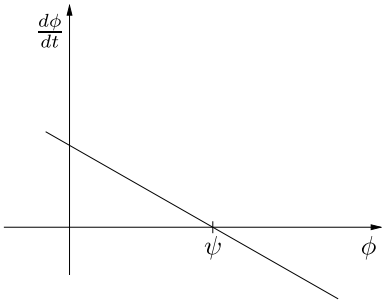


Figure 8: The linear dynamical system has an attractor at ψ, the direction of the target.

3.4 Controlling heading direction with a dynamical system

To turn a robot toward a target that lies in the direction ψ, we now construct a dynamical system that controls the robot's heading direction ϕ (Figure 7 shows an illustration of the involved angles). We construct the system so that it has an attractor at the orientation of the target:

$$\dot{\phi} = -\lambda \cdot (\phi - \psi), \quad \lambda > 0. \tag{17}$$

The parameter $\lambda > 0$ has the unit radians/s and controls the turning speed of the robot. Figure 8 shows a phase plot of this dynamical system. Note, that we assume here that the robot rotates on the spot, meaning that relative to the fixed reference axis, the angle ψ is constant over the course of the rotation.

We can see that the system makes the robot turn towards the target by looking at how the heading direction changes: For $\phi < \psi$ the system has a

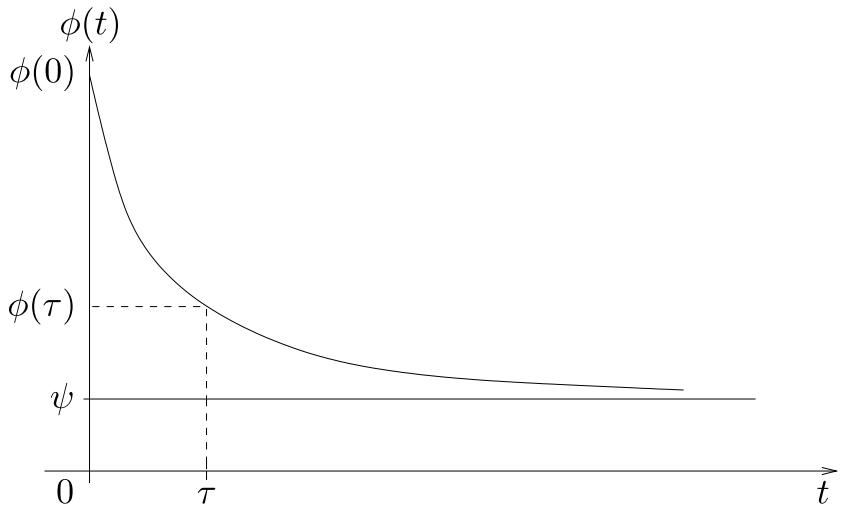


Figure 9: The time constant τ determines how fast a dynamical system relaxes to an attractor.

positive change, turning the robot counterclockwise toward the target; for $\phi > \psi$, the change is negative, turning the robot clockwise toward the target. If $\phi = \psi$, the robot will not turn. This is the same argument we made before: ψ is an attractor of the system.

3.5 Relaxation time

It is possible to solve the differential equation (Equation 17) analytically. The solution is

$$\phi(t) = \psi + (\phi(0) - \psi) \exp(-\lambda t),$$

where $\phi(0)$ is the initial heading direction of the robot. The parameter λ determines how fast the robot will turn toward the direction of the target. After $\tau = \frac{1}{\lambda}$ much time, the angle between the heading direction of the robot and the target direction will have dropped to $\frac{1}{e}$ of its initial value.

$$\phi(\tau) = \psi + (\phi(0) - \psi) \exp(-\lambda\tau) = \psi + (\phi(0) - \psi) \exp(-1) \tag{18}$$

See Figure 9 for an illustration. The value of τ is called *time constant* or *relaxation time* of a dynamical system.

3.6 Implementation

As we argued above, analytical solutions quickly becomes impractical for more complex dynamical systems that deal with current sensor values and

a changing position of the target. To implement a dynamical system, we therefore compute the change in heading direction in discrete time steps, t_i, according to

$$\Delta\phi(t_i) = -\lambda \cdot (\phi(t_i) - \psi(t_i)). \tag{19}$$

When following the Euler approach, we would normally keep track of the approximation, $\hat{\phi}$, of the heading direction by *integrating* as described in Section 3.2. This would give us

$$\hat{\phi}(t_{i+1}) = \hat{\phi}(t_i) + \Delta t \Delta\phi(t_i). \tag{20}$$

However, when working with e-pucks, this is not necessary. Instead, we send wheel velocities derived from $\Delta\phi$ directly to the robot and the physical robot itself (measured via odometry) gives us the current heading direction. This replaces the estimate $\hat{\phi}(t_i)$ in Equation 20.

The implementation of such a system thus consists of a control loop, in which the current heading direction of the robot ϕ is determined by odometry, and the desired change, $\Delta\phi$, is computed and applied via the wheels.

The equivalent of the time step in the Euler approach in this implementation is given by the time between two iterations of the control loop. Because the wheel speeds remain constant between two such iterations, this can lead to problems depending on the value of λ. For small values of λ, the resulting change in the heading direction is very small and the robot thus turns slowly. More importantly though, the (discrete) wheel speeds may be so small that they cannot be properly realized by the stepper motors that drive the e-puck's wheels. This may manifest in deviations of the heading direction from the attractor, that is, the target angle. For large values of λ, the amount that the robot turns between two iterations of the loop can be so large that it turns beyond the given orientation of the target.

3.7 Nonlinear dynamics

Instead of a linear system, it is more practical to use a sine curve (see Figure 10) for the dynamical system because it fits the periodic structure of the behavioral variable and does not produce increasingly large values for larger angles. This means that the robot will always turn toward the target using the shortest path.

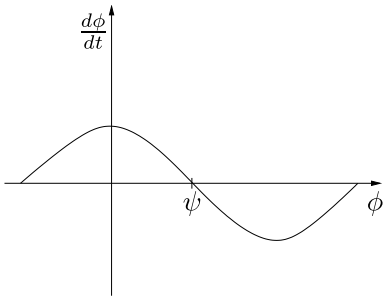


Figure 10: Controlling the heading direction with a sine dynamics.

4 Force-lets for obstacle avoidance in the dynamical systems approach to navigation

In the present section, we extend the dynamical systems approach presented above to avoid obstacles while still driving towards the target. The obstacle avoidance we develop here is a simplified version of the one in Bicho et al. (2000). We will use the infrared sensors of our e-puck robot to detect obstacles. Despite the limited number and accuracy of the sensors, they provide sufficient information for successfully avoiding obstacles.

4.1 Combining multiple influences

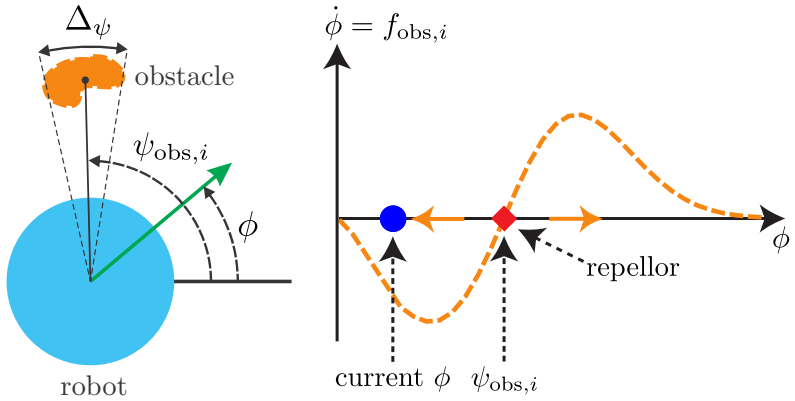
The key idea for building a dynamical system that reaches a target and avoids obstacles on the way is to consider multiple influences on the robot as contributions that "pull" the robot towards the target direction and "push" it away from the directions of obstacles.

We already know how to realize a single such influence from the target dynamics in Section 3.7. Here, the dynamical system

$$\dot{\phi} = f_{\text{tar}}(\phi) = -\lambda_{\text{tar}} \cdot \sin(\phi - \psi_{\text{tar}}) \tag{21}$$

creates an attractor that orients the robot towards the target at angle ψ_{tar}. We combine this dynamical system with a set of repelling influences, $f_{\text{obs},i}$, which we will design to create repellors at the locations of obstacles in the following sections. The full system then has the form

$$\dot{\phi} = f_{\text{tar}}(\phi) + \sum_i f_{\text{obs},i}(\phi). \tag{22}$$



(a) robot and obstacle (b) obstacle force-let

Figure 11: (a) shows a diagram of the robot and an obstacle with the relevant angles marked. Here, ϕ is the heading direction of the robot and $\psi_{\text{obs},i}$ is the direction of an obstacle, both relative to the global coordinate system. Δ_ψ represents the angular width of the obstacle. (b) shows a single obstacle force-let centered around the direction of the obstacle. The repellor generated by the force-let turns the robot away from this direction, as indicated by the orange arrows on the ϕ axis.

4.2 Obstacle contributions

Let us first consider the contribution of an individual obstacle term in the direction $\psi_{\text{obs},i}$ without the other influences (that is, without target and other obstacles). We want the robot to be repelled from the obstacle. Naively, we can solve this by placing a single repellor at the direction of the obstacle:

$$f_{\text{obs},i}(\phi) = \phi - \psi_{\text{obs},i}. \tag{23}$$

However, using a linear function has two undesired consequences. First, since the repellor acts across the entire angular space, the robot will always turn away from the obstacle, even if the robot is facing away from the obstacle, that is, the obstacle no longer lies in the robot's path. This is unnecessary and may even prevent the robot from reaching its target. Second, combining multiple linear functions additively for multiple obstacles would result in another linear function with a single fixed point that generally lies somewhere between the different obstacle directions.

We can address both these issues by restricting the angular range of the obstacle contribution. We use the idea of the force-let from Bicho et al. (2000), that is, we weight the contribution of individual linear functions by

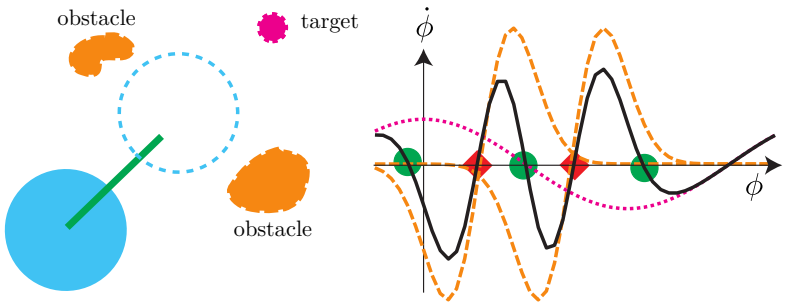


Figure 12: The path of the robot is blocked by two obstacles that are situated far apart from each other (left). The phase plot (right) shows the contributions from the two obstacles (dashed orange line), the target (dotted magenta line), and the resulting overall dynamics (solid black line). Green dots represent attractors, while red ones represent repellors.

a Gaussian with width σ, centered around the obstacle direction:

$$f_{\text{obs},i}(\phi) = (\phi - \psi_{\text{obs},i}) \cdot \exp\left(-\frac{(\phi - \psi_{\text{obs},i})^2}{2\sigma^2}\right). \tag{24}$$

Figure 11 shows such a force-let for a single obstacle.

Another consideration is the distance of obstacles. We only want obstacles that are close to the robot to have an influence on the robot's trajectory, whereas far obstacles should be ignored. We do this by weighting the force-let by another term, $\lambda_{\text{obs},i}$, leading to

$$f_{\text{obs},i}(\phi) = \lambda_{\text{obs},i} \cdot (\phi - \psi_{\text{obs},i}) \cdot e^{-\frac{(\phi - \psi_{\text{obs},i})^2}{2\sigma^2}}. \tag{25}$$

The weight function is defined as

$$\lambda_{\text{obs},i}(t) = \beta_1 \cdot \exp\left(-\frac{d_i(t)}{\beta_2}\right), \tag{26}$$

where β_1, β_2 are positive constants, and $d_i(t)$ is the distance of obstacle i at the current time t.

4.3 Bifurcations and decisions

So far, we have only looked at an individual force-let. In the full approach, we combine multiple such force-lets and a target contribution (see Equation 22). It is this combination that leads to emergent behaviors in our vehicle. Here, we show how even this fairly simple combination of functions endows our vehicle with the capacity to make decisions.

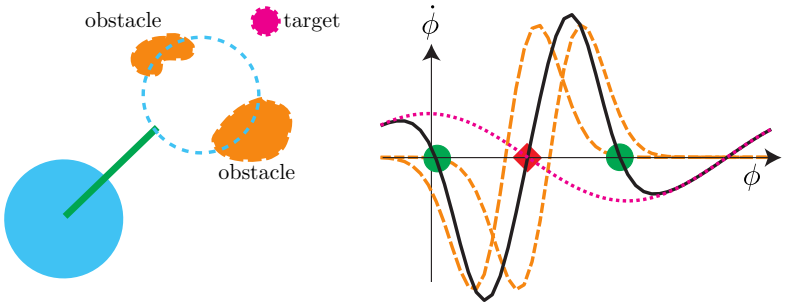


Figure 13: The path of the robot is blocked by two obstacles that are situated close to each other (left). The phase plot (right) is analogous to the one in Figure 12.

Let us first consider the case shown in Figure 12. Two obstacles are far enough apart for the robot to pass between them. This is reflected in the dynamics: the individual force-lets (dashed orange lines) have relatively little overlap. In the overall dynamics (solid black line), this leads to the emergence of two repellors (red diamonds), each close to one of the obstacles. The target contribution (dotted, magenta) creates an attractor (green dot) between the obstacles. If the robot is within the range of influence of this attractor, it will pass between the obstacles. Two more attractors are created on the outskirts of the force-lets due to the combination with the target contribution. These correspond to the robot going around the obstacles on the left or right hand side.

As the obstacles move closer together, the situation changes. As we can see in Figure 13, the obstacle force-lets overlap in such a manner that a single repellor emerges, centered between the two obstacles. The attractor in the target direction is canceled out by the stronger influences from the repelling force-lets. However, the two attractors that correspond to the robot circumnavigating the obstacles on the left or right hand side are still present.

There is a critical distance between the obstacles at which the attractor between the obstacles vanishes. Such a point where the number of fixed points or their stability changes is called a bifurcation. We can visualize how and when this happens by drawing a *bifurcation diagram*, where we plot the fixed points and their stability over the bifurcation parameter which, in our case, is the distance between obstacles. Figure 14 shows such a plot for our scenario.

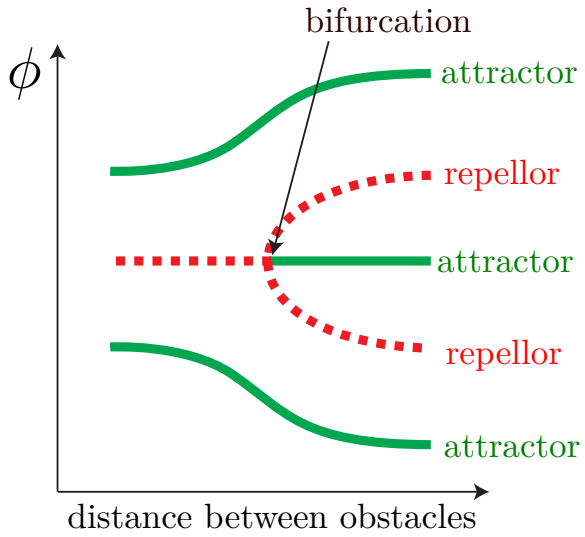


Figure 14: Bifurcation diagram for the target approach with obstacle avoidance dynamics.

4.4 Implementation

The robot's sensors do not deliver a discrete set of obstacles, but rather measurements related to the distance of the closest surface in the direction of the sensor. To generate a discrete set of obstacle force-lets, each sensor is thus said to point in the direction of an obstacle. Since the sensors are fixed on the robot, the directions of these *virtual obstacles* are given by

$$\psi_{\text{obs},i} = \phi_{\text{cur}} + \theta_i, \tag{27}$$

where ϕ_{cur} is the current heading direction of the robot, and θ_i is the angle at which sensor i is mounted, relative to the robot's forward direction.

References

Bicho, E., Mallet, P., and Schöner, G. (2000). Target representation on an autonomous vehicle with low-level sensors. *International Journal of Robotics Research*, 19(5):424–447.