Lab class: Autonomous robotics
Background material

Jean-Stephane Jokeit, Oliver Lomp,
Mathis Richter, Stephan Zibner, Prof. Gregor Schöner

Summer term 2015

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 is a small mobile robot that was developed primarily for research. Figure 1 shows a photo of an e-puck. On each side of its body, they have a wheel with a motor that can be individually controlled. The wheels have a diameter of 40 mm and the distance between the two wheels is 53 mm. Additionally, the robot is 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 rechargable battery, which can be attached to its bottom side. The robot can be controlled via a wireless bluetooth connection.

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 velocity is set to negative values, the robot moves backwards.

Each motor is equipped with an *encoder* that counts the pulses sent to

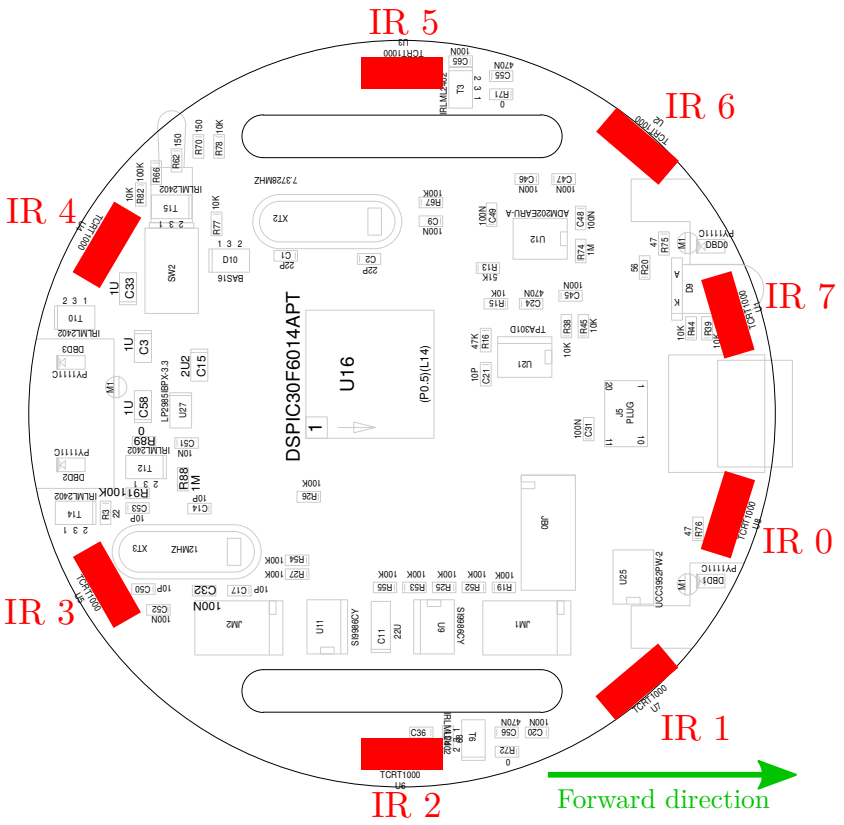


Figure 2: Top-view schematic of the e-puck robot showing the positions of infrared sensors.

the wheel. Accessing the encoder values 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 $\pm 135°$ (from the front 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 ambient light

By measuring the intensity of ambient light, it is possible to detect light sources (such as light bulbs), since the values from the sensors directed at the light source will differ from those in other directions. Measuring ambient light requires the use of the infrared receivers only, not the emitters.

Ambient light values returned by the sensors range from 0 to 4096, with values increasing for *decreasing* light intensity. They are influenced by the type and color of the light source as well as the distance from it. Please note that cold light, such as emitted from LEDs, will only evoke a small response from the infrared sensors.

1.2.2 Measuring distance

One way of detecting obstacles around the e-puck is by continuously measuring the distance of the robot to its surrounding environment. Distance can be measured using the infrared emitters and receivers in conjunction. 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 4096, 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. The response function varies depending both on the color, surface structure, and material of the object, as well as individual differences between sensors. This means that by default, the sensors do not produce an accurate measure of distance. The accuracy can be increased by calibrating the sensors, approximating the function that maps sensor readings to distances. The interval between two measuring points can be interpolated by a linear function. Since the calibration is a tedious effort, we will use an approximation of the function that we have determined heuristically.

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

1.3 Controlling the e-puck in Matlab

The following functions are available to control the e-puck robots in Matlab. h denotes the *handle* for the robot that is created when the connection is

established.

`h=kOpenPort`	opens the serial port and returns a handle h to the robot
`kClose(h)`	stops the robot and 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 of both wheels (in pulses/s)
`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
`kProximity(h)`	returns a vector with distance measurements for the eight infrared sensors $[d(0), \ldots, d(7)]$
`kAmbient(h)`	returns a vector with ambient light measurements for the eight infrared sensors $[a(0), \ldots, a(7)]$
`kSetCameraParameters(h, mode,width,height,z)`	sets parameters for the VGA camera; `mode` is either 0 (gray level images) or 1 (color images); the product of `width` and `height` should not exceed 1600, the zoom factor `z` may be 1, 4, or 8 (the default is 8; zoom increases with smaller values)
`kGetImage(h)`	returns a single camera image; requires that `kSetCameraParameters` has been called with valid arguments
`kGetMicrophones(h)`	returns the sound amplitudes from the three microphones

2 Kinematics and Odometry

The e-puck robot does not have any prior knowledge about the world. Without sensors and 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 situated or how it is oriented. However, this information is required to navigate to targets and avoid obstacles on the way.

This section will focus on how to infer the robot's position in the world based on the data from the wheel encoders. Analogously, we will look at how to generate appropriate velocities for the wheels to turn the robot to certain orientations. Both computations will build on our knowledge of the e-puck's physical measurements.

2.1 Coordinate frames

The position and orientation of the robot in the world can be uniquely described by only three parameters: its x- and y-position and its orientation ϕ (assuming that the robot is standing right-side-up on a table). Such a description is relative to a coordinate frame and it is important that you know for every parameter which coordinate frame it is relative to.

In this context, we distinguish between two types of coordinate frames. They differ in the position of the origin and the way they behave when the robot moves. The *ego-centric* (or *local*) coordinate frame has its origin in the center of the robot. The x-axis is always facing straight ahead into the heading direction of the robot, while the y-axis is always facing orthogonally to that toward the left. See Figure 3 for an example of an ego-centric coordinate frame (the gray, tilted coordinate system). This means that the coordinate frame moves around with the robot. It can for instance be used to express the position of the target or of obstacles relative to the robot.

The *allo-centric* (or *global*) coordinate frame has its origin at an arbitrary but fixed position in the world. This can for instance be a prominent landmark in the world or the corner of a piece of paper. The orientation of the coordinate frame is also arbitrary but can be aligned with whatever feels natural. See Figure 3 for a diagram that includes both an allo-centric and an ego-centric coordinate frame. The important thing is that the allo-centric coordinate frame does not move with the robot but stays fixed. It can for instance be used to express the position of the robot or the position of a target position. Please note that you can define the allo-centric coordinate frame to be exactly the same as the ego-centric coordinate frame before the robot begins to move. Once it moves, the allo-centric frame will stay at the starting position while the ego-centric frame will move away with the robot.

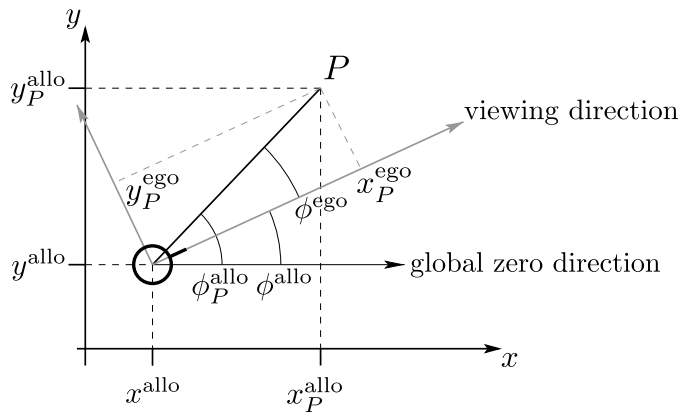


Figure 3: With an allo-centric coordinate frame (black coordinate system) it is possible to describe the global position $(x^{\text{allo}}, y^{\text{allo}})$ and orientation ϕ^{allo} of the robot. In the egocentric coordinate frame (tilted, gray coordinate system), the robot is located in the center 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}.

2.2 Inverse kinematics

Directing the robot to given coordinates in the world, possibly even with a given 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 arm movements: figuring out a good trajectory of the arm to a target given that there are many joints to move in various directions. Since this is such a hard problem to solve, we will sidestep it in this lab class.

We will focus instead on computing how much to turn the wheels to rotate *on the spot* around a given angle. For instance, having the robot face toward 0° in global coordinates, we want to know how to move the wheels so that the robot faces toward 90°. To direct the robot toward a certain position in the world, we can thus first rotate on the spot and then drive a straight line to the target. This may not produce the most natural movements for the robot but it is easy enough to compute and will produce short routes.

To turn on the spot, we turn one wheel in one direction and the other wheel into the opposite direction by the same amount. In doing so, the

wheels will drive along circular arcs. The length b of a circular arc can be computed with the formula

$$b = \Delta\phi \cdot r, \tag{1}$$

where $\Delta\phi$ is the change in orientation of the robot and r is the radius of the circle.[1] Since the arcs of the wheels are forming a circle around the center of the robot, the radius of the circle is half of the distance l between the wheels, giving us

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

To cover this distance b, the wheels will have to travel with a speed v for the time interval Δt:

$$b = v \cdot \Delta t \tag{3}$$

$$\Leftrightarrow v = \frac{b}{\Delta t} \tag{4}$$

If we were to write units on the equations above, we would recognize that the speed v has the unit $\mathrm{mm\,s^{-1}}$. We now have to use our knowledge of the physical measurements of the robot to convert that speed into units that the robot can use for its motors (pulse/s). Refer to section 1 for this information. Please remember that, for rotations on the spot, the two wheels have to turn into opposite directions. You will have to define that the velocity for the left wheel is multiplied by -1. This conforms with the mathematical convention that rotations around a positive angle are counter-clockwise.

2.3 Forward kinematics

As the robot moves around in the world, it needs to know its own location and orientation in order to be reactive to the environment and replan afterward. We can compute the new position of the robot in local coordinates after it has moved based on the readings of the encoders. This is commonly referred to as *forward kinematics*, the inverse computation of the *inverse kinematics*.

The robot moves by rotation of its two wheels. If both wheels turn at different speeds, the robot will drive along a circular arc. Over longer periods of time, the movement of the robot can thus be described as a sequence of circular arcs. Straight movements and rotations on the spot can be considered as special cases of a circular arc (with infinite radius or radius of zero, respectively). While the following equations only hold for arcs of a true circle, the forward kinematics for the two special cases can be derived easily.

[1] This is the same equation that gives the well-known formula $b = 2\pi \cdot r$ for computing the circumference b of a circle with radius r. The angle of a full circle is 2π.

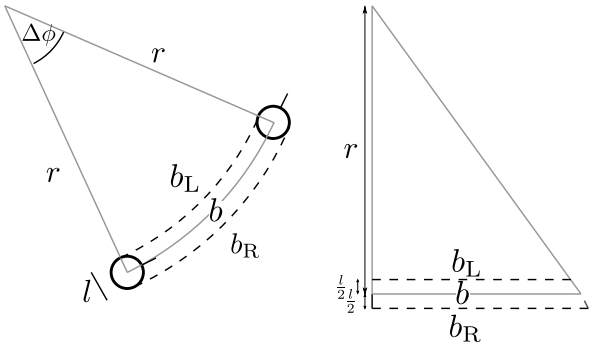


Figure 4: Locally the robot moves along the arc of a circle (left). Its two wheels describe arcs with different radii. The arcs that the wheels drive have different lengths (right), the difference of which can be used to compute the robot's change in orientation.

Let us assume that we initialize the robot in a certain position and orientation and we remember its encoder values. It then moves for a short amount of time along the arc of a circle and stops again. We can now determine its new position as follows: First, note that as the robot moves along the arc of the circle, its two wheels drive along circular arcs as well, with different radii (see Figure 4 for a diagram). The left wheels describes a circle of radius $r - \frac{l}{2}$ and the right wheel a circle of radius $r + \frac{l}{2}$, where l is the distance between the wheels. We can determine the length $b_{\mathrm{L}}, b_{\mathrm{R}}$ of these two arcs based on the differences $e_{\mathrm{L}}, e_{\mathrm{R}}$ between the new and old encoder values (for the left and right wheel, respectively) and our knowledge of how much distance each motor pulse moves the robot forward (see section 1). Computing the mean of the lengths b_{L} and b_{R} gives us the length b of the arc that the center of the robot is moving on

$$b = \frac{1}{2} \cdot (b_{\mathrm{R}} + b_{\mathrm{L}}). \tag{5}$$

Refer to Figure 4 if this is unclear. Furthermore, we can figure out the radius r of the circle by establishing that the following equation holds

$$\frac{r}{b} = \frac{l}{b_{\mathrm{R}} - b_{\mathrm{L}}}, \tag{6}$$

that is, the length of the radius r is to the length b of the arc what the distance l between the wheels is to the difference $b_{\mathrm{R}} - b_{\mathrm{L}}$. Refer to Figure 4 (right) and compare the large and the small triangle to verify this.

Rearranging the equation and using Equation 5 we get

$$r = l \cdot b \cdot \frac{1}{b_\mathrm{R} - b_\mathrm{L}}, \tag{7}$$

$$r = \frac{1}{2} \cdot l \cdot \left(\frac{b_\mathrm{R} + b_\mathrm{L}}{b_\mathrm{R} - b_\mathrm{L}} \right). \tag{8}$$

Now that we can express both the radius r and the length of the arc b purely in terms we already know, we can determine the robot's change in orientation $\Delta\phi$ by using the equation $b = \Delta\phi \cdot r$ that we have already used in the inverse kinematics (see Section 2.2)

$$\Delta\phi = \frac{b}{r}. \tag{9}$$

Knowing now the change in orientation $\Delta\phi$, we can also determine the change of the robot position $(\Delta x, \Delta y)$ in *local* coordinates. Since the local coordinate system is always centered on the robot, the x-axis pointing in its heading direction, it is always a tangent to the traversed circular arc. Analogously, the y-axis is aways aligned with the radius of the arc. This gives us the following equations for the change in position

$$\Delta x = r \cdot \sin(\Delta\phi) \tag{10}$$

$$\Delta y = r \cdot (1 - \cos(\Delta\phi)). \tag{11}$$

Please note that we will only obtain a good approximation of the robot's position if its trajectory really describes the arc of a circle with a fixed radius. For longer periods of time this is obviously not the case since the trajectory can take all kinds of forms. However, any smooth trajectory can be approximated in every point by the arc of a circle. We will thus use this approach to approximate complex trajectories as sequences of circular arcs. This approximation increases in accuracy the more we subdivide the circular arc. This means that we have to strive for updating the estimation of the position as fast as possible.

2.4 Odometry—Integration of the traversed path

We would now like to track the position and orientation of the robot in global coordinates. This requires that we know the starting position and orientation in global coordinates. We can then continuously update the robot's position and orientation with the information we get from the forward kinematics.

However, since the forward kinematics yields change values in local coordinates, we have to transform them into a global position change. This corresponds to rotating the vector that describes the robots position, depending

on the robot's current orientation. First we update the global orientation of the robot

$$\phi' = \phi + \Delta\phi, \tag{12}$$

where ϕ and ϕ' are the old and new orientations of the robot in global coordinates, respectively. We then compute the change of global position of the robot by

$$\begin{aligned} \Delta x^{\text{allo}} &= \Delta x^{\text{ego}} \cdot \cos(\phi') - \Delta y^{\text{ego}} \cdot \sin(\phi'), \\ \Delta y^{\text{allo}} &= \Delta x^{\text{ego}} \cdot \sin(\phi') + \Delta y^{\text{ego}} \cdot \cos(\phi'). \end{aligned} \tag{13}$$

The method we present here for estimating the position of a robot via odometry is affected both by systematic and random errors, such as the following.

- systematic errors
  - unequal wheel diameters/gear transmission between the wheels
  - deviation of the actual wheel diameter from the assumed values
  - deviation of the actual wheel base from the assumed values
  - deformation of wheels
  - discretization errors due to finite encoder resolution
- random errors
  - driving on uneven ground or over obstacles
  - backlash in the gear system
  - sliding on slick ground or at high acceleration
  - wheelspin upon collision with obstacles

3 Dynamical systems approach to navigation in autonomous robotics

The dynamical systems approach is geared toward local behavioral control of a robot that can be applied for path planning in particular. Instead of determining the entire path from an initial position to a target using a map of obstacles and possible paths, it controls basic behavioral variables (e.g., heading direction and forward speed) on the basis of local sensory information. The necessary change of these behavioral variables depending on its current value and the current sensory information is expressed as a dynamical system.

In this section, we will briefly introduce the basics of dynamical systems. As a simple example, we will then use a dynamical system to control the heading direction of a robot in order for it to approach a given target. The following section will extend this system with a behavior for obstacle avoidance based on distance sensors.

3.1 Dynamical systems and differential equations

Dynamical systems describe the change of variables over time. This is commonly done in the form of differential equations. The notation

$$\frac{dx}{dt} = f(x) \quad (\text{also written } \dot{x} = f(x)) \tag{14}$$

expresses the change of variable x over time t as a function $f(x)$ of its own current value. The solution of a differential equation is a function $x(t)$, for which $\dot{x}(t) = f(x(t))$ holds for every point of time. Please note that $x(t)$ is a fuction of time t, while $f(x)$ defines the relationship between the value of x and its derivative over time. It is generally hard to find an analytical solution for a given differential equation. However, solutions exist for some basic differential equations. For instance, the linear differential equation

$$\dot{x} = -\alpha x,$$

that formalizes an exponential decay has the solution

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

We can verify this by looking at the derivative of the equation at some arbitrary point of time

$$\dot{x}(t) = -\alpha \cdot \exp(-\alpha t) = -\alpha x(t).$$

Please note that any function of the form $x(t) = \exp(-\alpha t) + b$ will be a solution for the differential equation. A unique solution can be found by additionally specifying a starting value $x_0 = x(t_0)$. Solving such a differential problem is called an *initial value problem*.

A differential equation can be solved numerically by transforming it 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

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

If the system is at x_0 at time t_0, the next value can be computed by

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

The numerical solution of a differential equation is a good approximation as long as the inclination $\dot{x}$ remains close to constant during a time step Δt. This may not be the case if Δt is too large, possibly leading to large deviations between the actual value $x(\Delta t)$ and the approximation $\hat{x}(\Delta t) = x(0) + \Delta t \cdot \dot{x}(0)$ (see Figure 5 for an illustration). The approximation improves in accuracy the smaller the time step Δt. However, smaller time steps also lead to a higher computational burden.

3.2 Attractors and repellors

To control a robot using a dynamical system, one has to construct a differential equation that produces the desired behavior, for instance to turn the robot toward a target. Instead of specifying the desired change for every possible state of the numerical value, we would like to specify more generic properties of the system. Dynamical systems can be characterized by the configuration of their attractors and repellors in particular. In the simplest case, these 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 you plot the rate of change $\dot{x}$ of the system against the state variable x, as we have done in Figure 6, 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 the 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

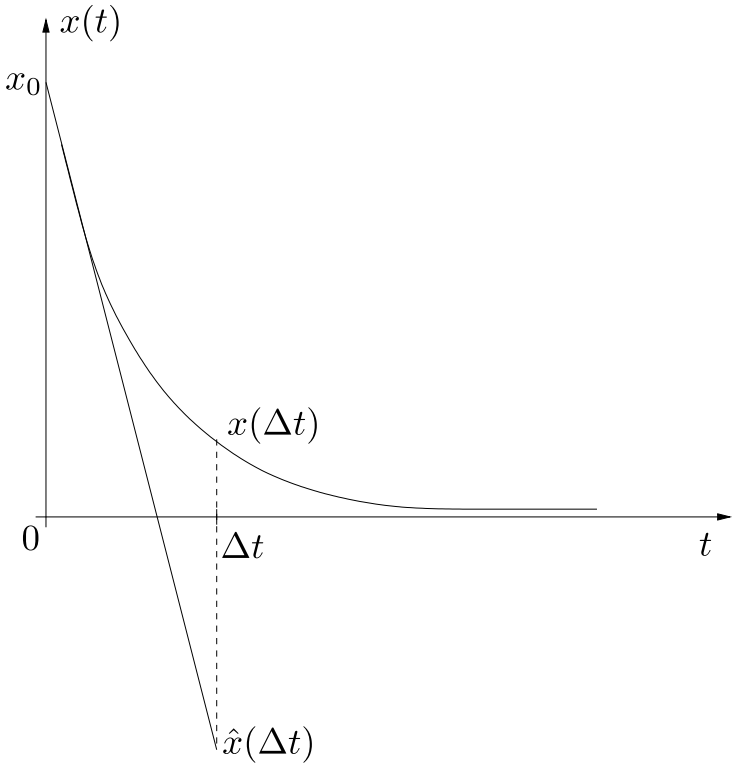


Figure 5: For values of Δt that are too large, the approximation error can become large. In the figure, $\hat{x}(\Delta t)$ strongly deviates from the analytically calculated $x(\Delta t)$.

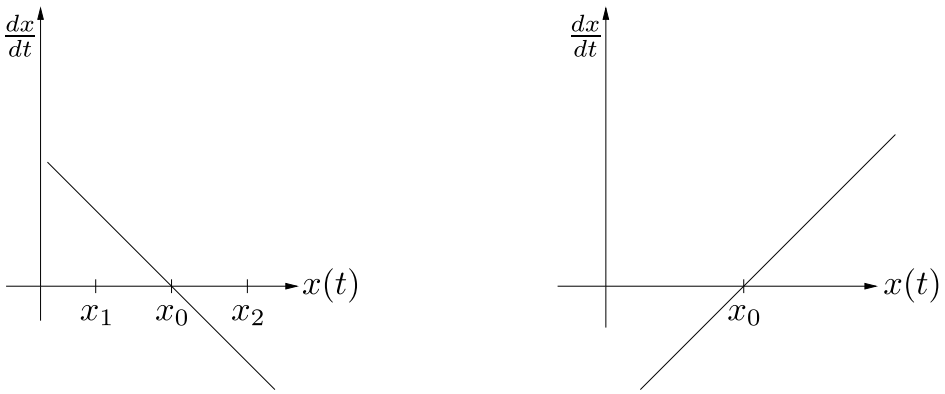


Figure 6: Attractor (left) and repellor (right) of a dynamical system

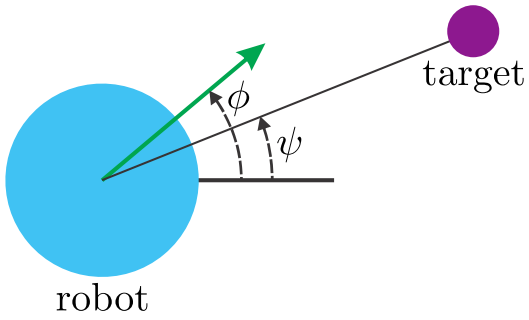


Figure 7: Control of the heading direction for approaching a target. The current heading direction of the robot is ϕ, the direction of the target is ψ. Both angles are defined relative to the zero-direction of a global coordinate system.

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 that attracts the system when it is in its proximity is called an *attractor*.

The right plot of Figure 6, on the other hand, shows a repellor. The inclination in the zero crossing is positive, such that the system will move away from that point if it is close to it. However, please note that the system would still remain at x_0 if it ever exactly reached that position.

3.3 Controlling heading direction with a dynamical system

To turn a robot toward a target, we now construct a dynamical system that controls its heading direction ϕ. We construct the system in such a way that it has an attractor at the orientation of the target. To do so, we represent the position and orientation of the e-puck in a global coordinate system (see Figure 7). To change the heading direction of the robot, we use the linear dynamical system

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

where ψ is the angle between the zero-orientation of the coordinate frame and the target. The parameter λ has the unit s^{-1}. Figure 8 shows a phase plot of this dynamical system. The e-puck should turn on the spot and the target should not move, making the angle ψ constant.

Since the system has an attractor in the direction of the target, it will turn toward it. For $\phi < \psi$ the system has a positive change, turning the robot counterclockwise toward the target; for $\phi > \psi$, the change is negative,

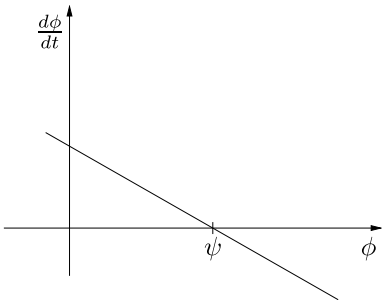


Figure 8: The linear dynamical system has an attractor at ψ, the direction of the detected object.

turning the robot clockwise toward the target. If $\phi = \psi$, the robot will not turn.

3.4 Relaxation time

It is possible to solve the differential equation (Figure 3.3) 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.

$$\phi(\tau) = \psi + (\phi(0) - \psi) \exp(-\lambda\tau) = \psi + (\phi(0) - \psi) \exp(-1) \tag{15}$$

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

3.5 Implementation

To implement a dynamical system, we implement a numerical solution that computes the change in heading direction in discrete time steps. At this point, an analytical solution would still be possible but it quickly becomes impractical for more complex dynamical systems that deal with current sensor values and a changing position of the target. The discretized equation is

$$\Delta\phi = -\Delta t \cdot \lambda \cdot (\phi - \psi). \tag{16}$$

The implementation of such a system consists of a continuous loop, in which the current heading direction of the robot ϕ is determined and the

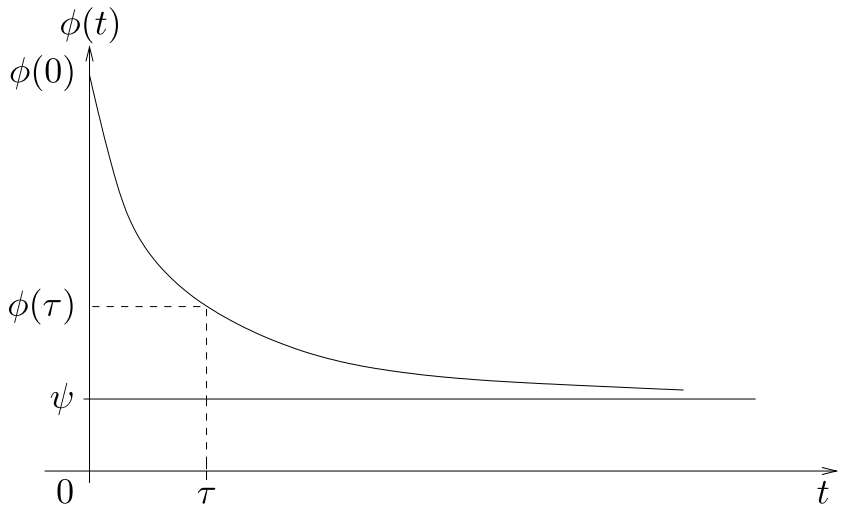


Figure 9: The time constant τ determines how fast a dynamical system relaxes to an attractor.

desired change $\Delta\phi$ is computed. The speed of the robot's wheels are set to reflect that desired change. Since the system runs on the robot in real time, the length of the time step Δt is determined by the duration of the loop. However, that duration is not known before the loop ends. In practice, a good approximation for the duration of the loop is the average of the last ten iterations of the loop.

Since the time step cannot be chosen freely here, the discrete computation of the heading direction can lead to problems, depending on the chosen value of λ. For small values of λ, the resulting change in the heading direction is very small and the robot will take a lot of time to turn. More importantly though, the (discrete) wheel speeds may be rounded to zero near the end of the turn, before the robot faces toward the target. For large values of λ, the turn within a single time step can be so large that the robot turns beyond the given orientation of the target.

3.6 Nonlinear dynamics

Instead of a linear system, it is more practical to use a sine curve (see Figure 10) for the dynamical system. On the one hand, it does not produce increasingly large values for larger angles. On the other hand, due to the periodic structure of the sine wave, the robot will always turn toward the target using the shortest path.

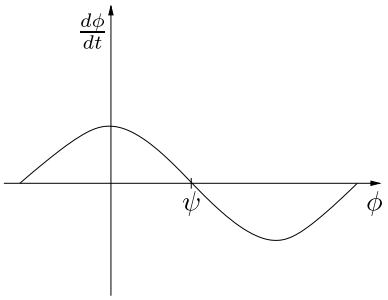


Figure 10: Controlling the heading direction with a sine dynamics.

4 Obstacle avoidance with infrared sensors

In the previous section, we investigated how a robot can reach a target using dynamic systems. In the present section, we will extend this approach to avoid obstacles while still driving towards the target. The obstacle detection we develop here is a simplified version of that in Bicho et al. (2000). We will use the infrared sensors of our E-puck robot. Despite their limited number and accuracy, 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 3.6. Here, the dynamical system

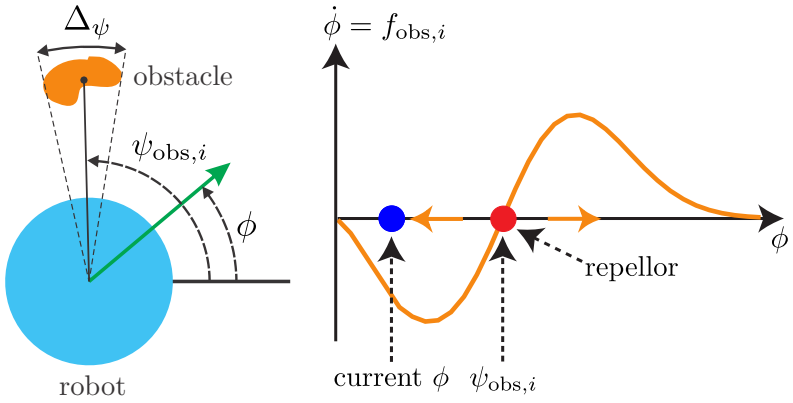
$$\dot{\phi} = f_{\text{tar}}(\phi) = -\lambda_{\text{tar}} \cdot \sin(\phi - \psi_{\text{tar}}) \tag{17}$$

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

$$\dot{\phi} = f_{\text{tar}}(\phi) + \sum_i f_{\text{obs},i}(\phi). \tag{18}$$

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



(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.

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:

$$f_{\text{obs},i}(\phi) = \phi - \psi_{\text{obs},i}. \tag{19}$$

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

$$f_{\text{obs},i}(\phi) = (\phi - \psi_{\text{obs},i}) \cdot e^{-\frac{(\phi-\psi_{\text{obs},i})^2}{2\sigma^2}}. \tag{20}$$

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

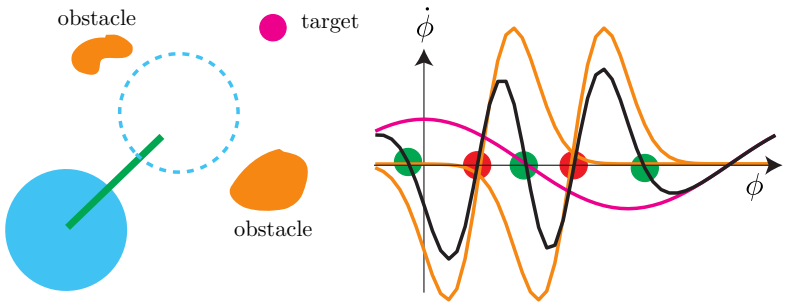


Figure 12: The path of the robot is blocked by two obstacles that are situated far apart from each other (left). The phase plot (right) shows the contributions from the two obstacles (orange line), the target (magenta line), and the resulting overall dynamics (black line). Green dots represent attractors,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

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 realize this by weighting the force-let by another term, $\lambda_{\text{obs},i}$, leading to

$$f_{\text{obs},i}(\phi) = \lambda_{\text{obs},i} \cdot (\phi - \psi_{\text{obs},i}) \cdot e^{-\frac{(\phi-\psi_{\text{obs},i})^2}{2\sigma^2}}. \tag{21}$$

The weight function is defined as

$$\lambda_{\text{obs},i}(t) = \beta_1 \cdot e^{-\frac{d_i(t)}{\beta_2}}, \tag{22}$$

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 18). 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.

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 (orange lines) have relatively little overlap. In the overall dynamics (black line), this leads to the emergence of two repellors (red dot), each close to one of the obstacles. The target contribution (magenta) creates an attractor (green dot) between the obstacles. If the

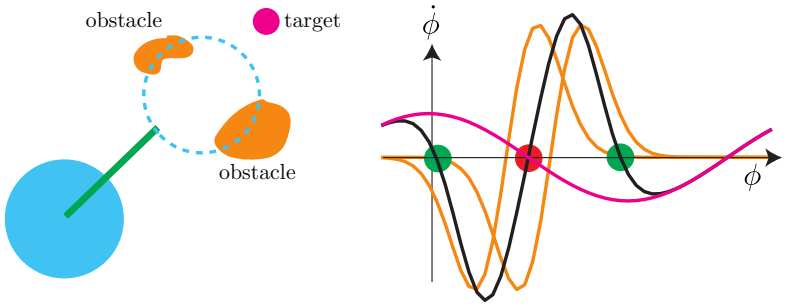


Figure 13: The path of the robot is blocked by two obstacles that are situated close to each other (left). The phase plot (right) is analogous to the one in Figure 12.

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 repellor. 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.

4.4 Implementation

When implementing the target approach with obstacle avoidance, the nature of the robot's sensors poses some issues. First, they do not deliver a discrete set of obstacles, but rather distance measurements to the closest surface in the direction of the sensor. Second, the values delivered from the sensors do not correspond to actual distances, but are an measure that grows inversely proportional with the distance, following a nonlinear function.

We address the first issue by making a simplifying assumption. Each sensor is said to point in the direction of an obstacle. Since the sensors are

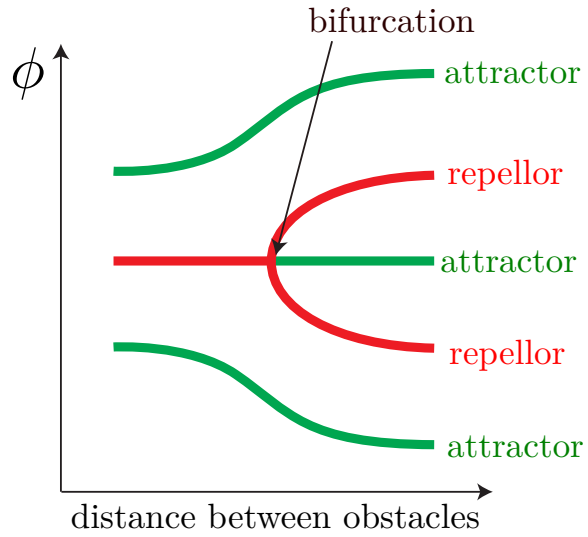


Figure 14: Bifurcation diagram for the target approach with obstacle avoidance dynamics.

fixed on the robot, the directions of these *virtual obstacles* are given by

$$\psi_{\text{obs},i} = \phi_{\text{cur}} + \theta_i, \tag{23}$$

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.

For measuring the distance, we must first find a function that determines the distance from an infrared value obtained from one of the robot's sensors. Formally, for an object placed at a distance $d(t)$, the sensors deliver an IR value

$$\text{ir}(t) = f(d(t)), \tag{24}$$

with an unknown function f that depends on factors such as the material of the object, but also the sensor itself.

Our goal, then, is to find the inverse relationship. That is, given an infrared value $\text{ir}(t)$, we want to determine the corresponding distance

$$d(t) = f^{-1}(\text{ir}(t)). \tag{25}$$

Since our dynamics are robust to noise and other minor errors, we can use an approximation for this inverse. One possible approach is measuring mean infrared values for obstacles placed at certain distances and linearly interpolating between them to obtain a distance. This approach is fairly accurate, but measuring is also time-consuming, tedious work. An alternative is to very roughly approximate the function f, and to then invert this approximation, or to simply determine a good function for the inverse empirically.

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.