Mobile Robotics

- We will consider wheeled mobile robots, the most used in industrial environments.
- We will focus on the two most commonly adopted kinematic structures:
  - the unicycle (and its kinematically equivalent implementations)
  - the bicycle (and its kinematically equivalent implementations)
- Mobile robots are nonlinear systems and the techniques developed so far can no more be applied directly.
The configuration of a mobile robot is represented by a set of configuration parameters.

All the possible configurations that a robot can assume constitute its configuration space.

The configuration space depends on the mechanical structure of the robot.

Controlling a wheeled mobile robot means to control its evolution in the configuration space.

A constraint is a condition that is imposed to a mechanical system and that prevents it from assuming a generic position and/or velocity.

A mechanical system is subject to holonomic constraints if there are algebraic relation among the coordinates of the system (or if there are integrable differential relation).

A mechanical system is subject to nonholonomic constraints if there are non integrable differential relations among the coordinates of the system.
Nonholonomic Constraints

- **Simplifying assumption:** Each wheel rolls without sliding (i.e., it does not slide neither longitudinally nor laterally).

- Each wheel introduces a nonholonomic constraint in the mechanical system since it does not allow a translation orthogonal to the rolling direction.

- Wheels limit the instantaneous mobility of the robot without limiting the possibility to reach any desired configuration (e.g., parallel car parking).

\[
\begin{align*}
\dot{x} &= v_r \cos \theta + v_n \cos \left(\frac{\pi}{2} - \theta\right) \\
\dot{y} &= v_r \sin \theta + v_n \sin \left(\frac{\pi}{2} - \theta\right)
\end{align*}
\]

The wheel rolls without sliding on the plane. The velocity orthogonal to the rolling direction is zero: \(v_n = 0\).

\[
\begin{align*}
\dot{x} &= v_r \cos \theta \\
\dot{y} &= v_r \sin \theta \quad \Rightarrow \quad \tan \theta = \frac{dy}{dx} \\
\dot{x} \sin \theta - \dot{y} \cos \theta &= 0
\end{align*}
\]

Constraint on the mobility
Nonholonomic Constraints

- For each wheel it is possible to write a nonholonomic constraint in terms of the configuration variables $q$ as:
  \[ a(q)\dot{q} = 0 \]
- For $N$ wheels the constraints can be rewritten as:
  \[ A(q)\dot{q} = 0 \]  
  Pfaffian Constraint

- Nonholonomic constraints do not reduce the reachable part of the configuration space but they reduce the instantaneous mobility of the robot.

Nonholonomic Constraints

\[ A(q)\dot{q} = 0 \]
Constraint in the velocity
\[ \dot{q} \in \ker A(q) \]

The admissible velocities can be generated by a matrix $G(q)$ such that:
\[ \text{Im}(G(q)) = \text{Ker}(A(q)) \forall q \in C \]

Kinematic model of a mobile robot subject to Pfaffian constraints
\[ \dot{q} = G(q)v \]
Kinematic input
Kinematic model of a WMR

\[ \dot{q} = G(q)v \]

- It represents the admissible direction of motion in the configuration space.
- It relates the velocity inputs with the configuration derivative.
- It is the model used for facing the main problems of mobile robotics:
  - Trajectory planning
  - Motion Control
  - Localization
  - ...

Kinematic model of the Unicycle

A unicycle is a vehicle with one wheel that can be oriented.

![Unicycle diagram]

- Configuration: \[ q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \]
- Constraint: \[ \dot{x} \sin \theta - \dot{y} \cos \theta = [\sin \theta, -\cos \theta, 0]q = 0 \]

- In Pfaffian form: \[ A(q)\dot{q} = 0 \quad A(q) = [\sin \theta, -\cos \theta, 0] \]

- Kinematic model: \[ \text{Ker}(A(q)) = \text{span} \begin{bmatrix} \cos \theta \\ \sin \theta \\ 0 \end{bmatrix} = \text{Im}(G(q)) \quad G(q) = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \]
Kinematic model of the Unicycle

\[
\dot{q} = \begin{bmatrix}
\cos \theta \\
\sin \theta \\
0
\end{bmatrix} v + \begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix} \omega = \begin{bmatrix}
\cos \theta & 0 \\
\sin \theta & 0 \\
0 & 1
\end{bmatrix} \begin{bmatrix}
v \\
\omega
\end{bmatrix}
\]

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \omega
\end{align*}
\]

The inputs \(v\) and \(\omega\) have a clear physical meaning:

1. \(v\) is the linear velocity of the contact point between the wheel and the ground and it is given by the product between the rotation velocity of the wheel and the radius of the wheel itself.

2. \(\omega\) is the angular velocity of the robot and it equal to the rotation velocity of the wheel around its vertical axis.

Acting on \(v\) e \(\omega\) it is possible to change the configuration of the robot.

---

Kinematic model of the Unicycle

- The unicycle has serious static stability problems!

- In practice, mechanical structures characterized by a higher static stability but kinematically equivalent to the unicycle are used.

- The most used unicycle-like robotic structures are:
  - The differential drive robot
  - The synchro drive robot
**Differential drive unicycle**

- It has two fixed wheels, co-axial and independently actuated, and a castor wheel (usually smaller) whose role is to keep the robot in static equilibrium.

- $q=[x,y,\theta]$, $(x,y)$ are the coordinates of the midpoint of the wheels axis and $\theta$ is the orientation of the robot.

- The castor wheel is passive and the two wheels are actuated independently. The robot translates if $\omega_R=\omega_L$ and it turns on itself if $\omega_R=-\omega_L$.

- It is the most common realization of the unicycle.

---

**Differential drive unicycle**

The velocity of the midpoint of the wheels axis $v$ and the angular velocity of the robot $\omega$ are biunivocally related to the wheels rotation velocity:

\[
\begin{align*}
\frac{v}{2} &= \frac{(\omega_R + \omega_L)r}{2} \\
\omega &= \frac{(\omega_R - \omega_L)r}{d}
\end{align*}
\]

- $\omega_R$ = rotation velocity of the right wheel
- $\omega_L$ = rotation velocity of the left wheel
- $r$ = radius of the wheels
- $d$ = distance between the centers of the wheels

Given a kinematic input $[v, \omega]^T$ to the unicycle, it is always possible to find a pair $[\omega_R, \omega_L]^T$ that produce it.
**Differential drive unicycle**

It is possible to find a kinematic model of the differential drive unicycle where the kinematic inputs are the rotation velocities of the wheels:

\[
\begin{bmatrix}
  v \\
  \omega
\end{bmatrix}
=\begin{bmatrix}
  r/2 & r/2 \\
  r/d & -r/d
\end{bmatrix}
\begin{bmatrix}
  \omega_R \\
  \omega_L
\end{bmatrix}
\]

\[
\dot{q} = \begin{bmatrix}
  \cos \theta & 0 & r/2 & r/2 & \omega_R \\
  \sin \theta & 0 & r/2 & r/2 & \omega_L
\end{bmatrix}
= \begin{bmatrix}
  r \cos \theta & r \cos \theta \\
  r \sin \theta & r \sin \theta
\end{bmatrix}
\begin{bmatrix}
  \omega_R \\
  \omega_L
\end{bmatrix}
\]

**Synchro drive unicycle**

It has three wheels that are aligned and that can be rotated around their axis. They are controlled by two motors by a mechanical coupling (e.g. belt).

One motor sets the rotation of the wheels around their rotation axis and, therefore, it determines the traction of the robot. The other motor sets the rotation of the wheels around their vertical axis and, therefore, it determines the orientation of the robot.

This robot is kinematically equivalent to the unicycle. Coordinates (x,y) represent a point of the robot and \( \theta \) represents the orientation of the vehicle. The kinematic inputs of the ideal unicycle are the same as those of the synchro drive unicycle.
A bicycle is a vehicle with a steerable wheel and a fixed wheel disposed as in the picture.

Configuration:
\[
q = \begin{bmatrix}
   x \\
   y \\
   \theta \\
   \varphi
\end{bmatrix}
\]

\[(x, y) = (x_r, y_r)\]

C = Center of instantaneous rotation

We will consider the case of a bicycle with traction in the front wheel.

The system is subject to two pure rolling constraints, one per each wheel.

\[
\begin{align*}
\dot{x}_f \sin(\theta + \varphi) - \dot{y}_f \cos(\theta + \varphi) &= 0 \\
\dot{x} \sin(\theta) - \dot{y} \cos(\theta) &= 0
\end{align*}
\]

\{(x_r, y_r)\} is the cartesian position of the contact point with the ground of the front wheel.

\[x_f = x + l \cos \theta\]
\[y_f = y + l \sin \theta\]

The constraint introduced by the front wheel is given by:

\[
\begin{align*}
\dot{x}_f \sin(\theta + \varphi) - \dot{y}_f \cos(\theta + \varphi) &= 0 \\
\dot{x} \sin(\theta + \varphi) - \dot{y} \cos(\theta + \varphi) - l \dot{\varphi} \cos \varphi &= 0
\end{align*}
\]
Kinematic model of the bicycle

The kinematic constraints of the bicycle are:

\[
\begin{align*}
\dot{x}\sin(\theta + \phi) - \dot{y}\cos(\theta + \phi) - l\dot{\phi}\cos \phi &= 0 \\
\dot{x}\sin(\theta) - \dot{y}\cos(\theta) &= 0
\end{align*}
\]

\[
A(q) = \begin{bmatrix}
\sin \theta & -\cos \theta & 0 & 0 \\
\sin(\theta + \phi) & -\cos(\theta + \phi) & -l\cos \phi & 0
\end{bmatrix}
\]

\[
\text{Ker}(A(q)) = \text{span}(\begin{bmatrix}
\cos \theta \cos \phi \\
\sin \theta \cos \phi \\
\frac{1}{l}\sin \phi \\
0
\end{bmatrix}, \begin{bmatrix}0 \\
0 \\
0 \\
1\end{bmatrix}) = \text{Im}(G(q))
\]

\[
\dot{q} = \begin{bmatrix}
\cos \theta \cos \phi \\
\sin \theta \cos \phi \\
\frac{1}{l}\sin \phi \\
0
\end{bmatrix} v + \begin{bmatrix}0 \\
0 \\
0 \\
1\end{bmatrix} \omega = \begin{bmatrix}
\cos \theta \cos \phi & 0 \\
\sin \theta \cos \phi & 0 \\
\frac{1}{l}\sin \phi & 0 \\
0 & 1
\end{bmatrix} \begin{bmatrix}v \\
\omega\end{bmatrix}
\]

\(v = \text{linear velocity}\)

\(\omega = \text{angular velocity}\)
**Kinematic model of the bicycle**

The bicycle has serious static equilibrium problems.

In practice, mechanical structures statically stable but kinematically equivalent to the bicycle are used. The most used structures are the *tricycle* and the *car*.

---

**The tricycle and the car**

**Tricycle**
- It has two fixed wheels on the rear and a steerable wheel on the front.
- The fixed wheels are controlled by one motor that determines the traction. Another motor controls the orientation of the front wheel and determines the steering angle.

**Automobile (Car-like)**
- Two fixed wheels on the rear and two aligned steering wheels on the front.
- It is controlled by two motors. One determines the traction (front or back) and the other determines the steering angle.
## The tricycle and the car

- Both the car and the tricycle have the same kinematic model as the bicycle.
- The point of coordinate \((x,y)\) is the midpoint of the rear axis.
- \(\theta\) is the orientation of the vehicle.
- \(\phi\) is the steering angle.
- \(v\) and \(\omega\) are the traction velocity and the steering velocity respectively.

## Unicycle and Bicycle

- The kinematic structure of the unicycle and of the bicycle are the most used in industrial applications.
- Other kinematic structures are used for specialized applications.
- The kinematic models of more complex structures are obtained considering the constraints introduced by each wheel.
**Elementary motion tasks**

- **Parking**
  ![Parking diagram]

- **Path Following**
  ![Path Following diagram]

- **Trajectory following**
  ![Trajectory Following diagram]

**A conceptual scheme of a wheeled mobile robot**

- **Actuators (Att)**: DC motors with gearbox
- **End-Effectors**: tool, gripper, hand
- **Sensors**:  
  - Proprioceptive (encoders, gyroscopes, ...)
  - Exteroceptive (bumpers, rangefinders (infrared, ultrasound), laser scanner, vision (mono, stereo, ...)
- **Control**:
  - high level
  - low level
The low level control

- The velocity of the motors of the robot is controlled through high gain velocity controlled. If the gains are high enough, the difference between the velocity setpoint and the real velocity is negligible.

- The velocity setpoint for the robot computed by the high-level control is transformed into velocity setpoints for the motors.

- If the gains of the PI controllers are high enough, the low level control makes the robot a purely kinematic system.

The high-level control

- Give a desired position, the high-level control has to determine the velocity the robot should move with.

- From the point of view of the high-level control, the robot is a purely kinematic system with velocity inputs.

- The high level control decides the motion of the robot while the low-level control controls the motors in such a way that the desired motion is implemented.
**Control of a WMR**

- The design of the low-level control
  - It is simple. It is sufficient to design a PI controller for an electric motor (linear system)
  - It is not influenced by the constraints introduced by the wheels
  - Servomotors can be used!

- The planning of the trajectory to track
  - It provides the setpoint to the high-level control
  - It has to take into account the constraints on the mobility of the robot

- The design of the high-level control
  - It is based on the kinematic model of the robot
  - It has to take into account the constraints introduced by the wheels
  - It is complex because it is a **nonlinear** control problem

---

**Planning**

- **Planning problem:** Determine a trajectory in the state space that takes the robot from an initial configuration to a desired final configuration
- Each point of the planned trajectory **must be compatible** with the kinematic constraints of the robot.

Example: Unicycle

*Non admissible trajectory!*

The planned trajectory is not compatible with the kinematic constraints and, therefore, the robot will never be able to track it.
Planning

The planning problem: Find an admissible trajectory $q(t)$ for $t$ in $[t_i, t_f]$ that takes the robot from an initial configuration $q(t_i)=q_i$ to a final configuration $q(t_f)=q_f$.

A trajectory can be split into a path and into a path $q(s)$ and a (time) parameterization $s=s(t)$. Parameterization: The way I go from the point of distance zero from the beginning of the curve to the point of distance $L$ from the beginning of the curve.

Path: The configuration corresponding to a value of parameter $s$ in the curve.

The path represents the geometric shape of the curve while the parameterization represents the way the curve is tracked when considering a certain parameter (time, distance, ...).

Planning

Space-time separation of the trajectory: $\dot{q} = \frac{dq}{dt} = \frac{dq}{ds} \dot{s} = q' \dot{s}$

The trajectory is admissible for a WMR if and only if it satisfies the nonholonomic kinematic constraints of the robot. Thus:

$$A(q)\dot{q} = A(q)q' \dot{s} = 0$$

An admissible path is given by: $q' = G(q)\mathbf{u}$

The path can be found by determining the geometric inputs $\mathbf{u}$. Once an admissible path has been selected, it will be necessary to choose a time parameterization $s=s(t)$, namely the velocity at which the path is tracked. These two elements completely determine an admissible trajectory.
Planning

Once the geometric input \( \tilde{u} \) has been selected and the parameterization \( s = s(t) \) has been found, what is the kinematic input that has to be applied to the robot for making it following the desired trajectory?

\[
\dot{q}(s) = G(q)\tilde{u}(s)
\]

\[
\frac{dq}{ds} \dot{s} = G(q)\tilde{u}(s) \dot{s}
\]

\[
\dot{q} = G(q)\tilde{u}(s) \dot{s}
\]

\[
u(t) = \tilde{u}(s) \dot{s}
\]

Example

For a unicycle, the nonholonomic constraints induce the following admissibility condition for a path.

\[
\begin{bmatrix} \sin \theta, -\cos \theta, 0 \end{bmatrix} q' = x' \sin \theta - y' \cos \theta = 0
\]

The condition tells that the cartesian velocity of the robot has to be oriented along the motion direction. The admissible paths of the unicycle are given by:

\[
x' = \cos \theta \ \tilde{v}
\]

\[
y' = \sin \theta \ \tilde{v}
\]

\[
\theta' = \tilde{\omega}
\]

Once the geometric inputs have been determined, the kinematic inputs are given by:

\[
\nu(t) = \tilde{v} \dot{s}
\]

\[
\omega(t) = \tilde{\omega} \dot{s}
\]
Planning using differential flatness

A nonlinear dynamic system
\[ \dot{x} = f(x) + g(x)u \]
has the property of **differential flatness** if there is a set of measurable variables \( y \), called **flat outputs**, such that the state \( x \) and the input \( u \) can be expressed as an algebraic function of \( y \) and of its derivatives.

\[
\begin{align*}
    x &= x(y, \dot{y}, \ddot{y}, \ldots, y^{(r)}) \\
    u &= u(y, \dot{y}, \ddot{y}, \ldots, y^{(r)})
\end{align*}
\]

Once a trajectory \( y(t) \) is determined, the state \( x(t) \) and the input \( u(t) \) are automatically determined.

---

Planning using differential flatness

For the unicycle and the bicycle the cartesian coordinates \( x \) and \( y \) are flat outputs.

**Geometric model of the unicycle**

\[
\begin{align*}
x' &= \cos \theta \, \dot{v} \\
y' &= \sin \theta \, \dot{v} \\
\theta' &= \dot{\theta}
\end{align*}
\]

The steering angle depends algebraically by the first derivative of the flat outputs.

\[
\theta = \theta(x', y') = \arctan(y'/x') + k\pi \quad k = 0, 1
\]

The two possible choices for \( k \) are due to the fact that the same path can be tracked forward \( (k=0) \) or backward \( (k=\pi) \). If the initial steering angle of the robot is assigned, only one of these choices is admissible.
Planning using differential flatness

The flat output $y$ can be used for solving the planning problems. In fact, any interpolation algorithm can be used for planning the path of the flat output $y(s)$. The evolution of the other configuration variables and the kinematic inputs are automatically determined by $y(s)$. The generated path will automatically satisfy the nonholonomic constraints of the robot.

Consider the problem of planning a path for the unicycle that takes the robot from an initial configuration $q_i=(x_i,y_i,\theta_i)$ to a final configuration $q_f=(x_f,y_f,\theta_f)$. Let the curve be parameterized using $s$ in $[0,1]$.

$x$ and $y$ are flat outputs Plan the path on the flat outputs!

Cubic polynomials can be used to generate a path from $x_i (y_i)$ to $x_f (y_f)$

\[
\begin{align*}
x(s) &= s^3 x_f - (s-1)^3 x_i + \alpha_s s^2 (s-1) + \beta_s s(s-1)^2 \\
y(s) &= s^3 y_f - (s-1)^3 y_i + \alpha_s s^2 (s-1) + \beta_s s(s-1)^2
\end{align*}
\]

$x(0)=x_i \quad x(1)=x_f \\
y(0)=y_i \quad y(1)=y_f$

The boundary conditions on $x$ and $y$ are satisfied
Planning using differential flatness

Also the orientation of the robot needs to satisfy the boundary conditions. Recalling that $\theta$ depends on $x'$ and $y'$ algebraically, then the following conditions need to be satisfied:

$$
\begin{align*}
x'(0) &= k_i \cos \theta_i \\
x'(1) &= k_f \cos \theta_f \\
y'(0) &= k_i \sin \theta_i \\
y'(1) &= k_f \cos \theta_f
\end{align*}
$$

$k_i$ and $k_f$ are free non null parameters. They represent the geometric velocities at the initial and at the final instants respectively and they influence the shape of the final path.

The boundary conditions on $\theta$ allow to determine the parameters $\alpha_x, \alpha_y, \beta_x, \beta_y$. For example, if $k_i = k_f = k$ we have that:

$$
\begin{bmatrix}
\alpha_x \\
\alpha_y
\end{bmatrix} = \begin{bmatrix}
k \cos \theta_f - 3x_f \\
k \sin \theta_f - 3y_f
\end{bmatrix},
\begin{bmatrix}
\beta_x \\
\beta_y
\end{bmatrix} = \begin{bmatrix}
k \cos \theta_i + 3x_i \\
k \sin \theta_i + 3y_i
\end{bmatrix}
$$

Planning using differential flatness

Once the path in the configuration space has been found, what are the corresponding geometric and kinematic inputs?

From the geometric model of the robot we obtain that:

$$
\begin{align*}
\dot{v}(s) &= \pm \sqrt{(x'(s))^2 + (y'(s))^2} \\
\ddot{\omega}(s) &= \frac{y''(s)x'(s) - x''(s)y'(s)}{(x'(s))^2 + (y'(s))^2}
\end{align*}
$$

The sign of the linear velocity depends on the direction of motion (backward or forward).
Planning of the unicycle - Algorithm

- **Require:** $q_i = [x_i, y_i, \theta_i]$, $q_f = [x_f, y_f, \theta_f]$

1. Plan the path on the flat outputs $(x, y)$
   
   \[
   \begin{align*}
   x(s) &= x'_i x_f - (x_i - x_f) + \alpha_i (x_f - x_i) + \beta_i (x_f - x_i)^2 \\
   y(s) &= y'_i y_f - (y_i - y_f) + \alpha_i (y_f - y_i) + \beta_i (y_f - y_i)^2
   \end{align*}
   \]

2. Consider the boundary conditions for determining the coefficient of the polynomial

   \[
   \begin{bmatrix}
   \alpha_i \\
   \alpha_f
   \end{bmatrix} = \begin{bmatrix}
   k \cos \theta_f - 3x_f \\
   k \sin \theta_f - 3y_f
   \end{bmatrix}, \quad
   \begin{bmatrix}
   \beta_i \\
   \beta_f
   \end{bmatrix} = \begin{bmatrix}
   k \cos \theta_f + 3x_i \\
   k \sin \theta_f + 3y_i
   \end{bmatrix}
   \]

3. Determine the geometric inputs

   \[
   \begin{align*}
   \tilde{v}(s) &= \pm \sqrt{(x'(s))^2 + (y'(s))^2} \\
   \tilde{\omega}(s) &= \frac{y''(s) x'(s) - x''(s) y'(s)}{(x'(s))^2 + (y'(s))^2}
   \end{align*}
   \]

4. Determine the time law and obtain the kinematic inputs

   \[
   v(t) = \tilde{v}(s) \dot{s} \\
   \omega(t) = \tilde{\omega}(s) \dot{s}
   \]

Planning using differential flatness

$q_i = [0, 10, 0]$  Parallel parking  $q_f = [0, 0, 0]$
**Planning using differential flatness**

\[ q_i = \begin{bmatrix} 10 & 7 & -\pi/3 \end{bmatrix} \quad k=10 \quad q_f = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \]

\[ k=50 \]

\[ k=100 \]

Cristian Secchi

**Motion Control**

- Given a desired trajectory (or configuration), build a controller that makes the robot tracking the desired trajectory (or configuration).

- The kinematic model of the robot is used for designing the controller.
- It is assumed that the kinematic inputs directly influence the configuration variables. For the unicycle and the bicycle, the control inputs are \( v \) and \( \omega \).
- This can be done because in most of the mobile robots it is not possible to directly control the torque on the wheels since it is controlled by inner control loops integrated in the hardware or software architecture.
- We will consider the motion control of the unicycle. All the results can be easily extended to the bicycle.
**Motion Control**

**Regulation problem:** The robot has to reach a desired configuration $q_d=(x_d, y_d, \theta_d)^T$ starting from an initial configuration $q_0=(x_0, y_0, \theta_0)^T$.

It is the harder problem.

**Trajectory tracking:** The robot has to asymptotically reproduce a cartesian trajectory $(x_d(t), y_d(t))$ starting from an initial configuration $q_0=(x_0, y_0, \theta_0)^T$.

Easier and more interesting problem.

---

**Trajectory tracking – I-O SFL**

**I-O SFL:** Input-Output state feedback linearization

- It depends on the reference point of the robot (output of the system), namely the point we want to track the desired trajectory
  - All the points on the axis connecting the wheels can never assume a lateral velocity.

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \omega
\end{align*}
\]

The system needs to stop and rotate. It is not possible to track a discontinuous trajectory with a constant velocity.

Forbidden instantaneous motion

Considering a point on the wheels axis as an output to control, the mobility constraints of the robot create problems.
Trajectory tracking – I-O SFL

Define as an output to control a point out of the wheels axis:

\[ x_b = x + b \cos \theta \]
\[ y_b = y + b \sin \theta \]

\( b \neq 0 \)

The point \((x_b, y_b)\) is no more subject to the kinematic constraints and it can move, instantaneously, in all directions.

It is possible to define two kinematic inputs that determine the velocity of the point B.

\[ \dot{x}_b = \dot{x} - b \omega \sin \theta = v \cos \theta - b \omega \sin \theta \]
\[ \dot{y}_b = \dot{y} + b \omega \cos \theta = v \sin \theta + b \omega \cos \theta \]
\[ \dot{\theta} = \omega \]

The matrix \(T(\theta)\) is always invertible.

\[
\begin{bmatrix}
\dot{x}_b \\
\dot{y}_b \\
v \\
\omega
\end{bmatrix} = T(\theta) \begin{bmatrix}
\dot{v}_x \\
\dot{v}_y \\
v_x \\
v_y
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -b \sin \theta & v_x \\
\sin \theta & b \cos \theta & v_y \\
1 & 0 & v_x \\
0 & 1 & v_y
\end{bmatrix}
\]

\[ \det T(\theta) = b \neq 0 \]

\[
\begin{bmatrix}
v \\
\omega
\end{bmatrix} = T^{-1}(\theta) \begin{bmatrix}
\dot{v}_x \\
\dot{v}_y
\end{bmatrix} = \begin{bmatrix}
\cos \theta & \sin \theta & v_x \\
\frac{1}{b} \sin \theta & \frac{1}{b} \cos \theta & v_y
\end{bmatrix} \begin{bmatrix}
\dot{v}_x \\
\dot{v}_y
\end{bmatrix}
\]
Trajectory tracking – I-O SFL

From the previous equations, the following system, decoupled on the cartesian coordinates, is obtained.

\[
\begin{align*}
\dot{x}_B &= v_{dx} \\
\dot{y}_B &= v_{dy} \\
\dot{\theta} &= \frac{1}{b}(v_{dy} \cos \theta - v_{dx} \sin \theta)
\end{align*}
\]

The x and the y directions of point B can be controlled independently by the inputs \(v_{dx}\) and \(v_{dy}\).

Given a trajectory \((x_{des}, y_{des})\) to track, it is possible to find \(v_{dx}\) and \(v_{dy}\) that guarantee the asymptotic tracking:

\[
\begin{align*}
\dot{e}_x &= x_{des} - x \\
\dot{e}_y &= y_{des} - y \\
\dot{e}_x + k_1 e_x &= 0 & e_x \rightarrow 0 \\
\dot{e}_y + k_2 e_y &= 0 & e_y \rightarrow 0
\end{align*}
\]

Trajectory tracking – I-O SFL

Once \(v_{dx}\) and \(v_{dy}\) have been found, the kinematic inputs of the unicycle for obtaining the desired motion of point B can be found by:

\[
\begin{bmatrix}
v \\
\omega
\end{bmatrix} = T^{-1}(\theta) \begin{bmatrix} v_{dx} \\ v_{dy} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ \frac{1}{b} \sin \theta & \frac{1}{b} \cos \theta \end{bmatrix} \begin{bmatrix} v_{dx} \\ v_{dy} \end{bmatrix}
\]
Control of Industrial Robot

Trajectory tracking – I-O SFL

Il punto \((x,y)\) arrotonda l’angolo vivo. L’arrotondamento può essere reso piccolo a piacere scegliendo \(b\) abbastanza piccolo.

\[
P_1 \rightarrow P_2 \quad v_{dx} = 0 \quad v_{dy} = v \\
P_2 \rightarrow P_3 \quad v_{dx} = v \quad v_{dy} = 0 \\
P_3 \rightarrow P_4 \quad v_{dx} = 0 \quad v_{dy} = -v \\
P_4 \rightarrow P_1 \quad v_{dx} = -v \quad v_{dy} = 0 
\]

IO-SFL for the unicycle- Algorithm

**Require:** \(q_{des}(t) = (x_{des}(t), y_{des}(t), \dot{q}_{des}(t))\)

1. Choose a point out of the axis \((x_B, y_B) = (x + b \cos \theta, y + b \sin \theta)\) defining \(b\)
2. **while** True
3. Determine \(v_{dx}, v_{dy}\) by
   \[
   v_{dx}(t) = \dot{x}_{des}(t) + \dot{k}(x_{des}(t) - x_y(t)) \\
   v_{dy}(t) = \dot{y}_{des}(t) + \dot{k}(y_{des}(t) - y_y(t))
   \]
4. Determine \(v(t)\) and \(\omega(t)\) by
   \[
   \begin{bmatrix}
   v(t) \\
   \omega(t)
   \end{bmatrix} = T^{-1}(\theta(t))
   \begin{bmatrix}
   v_x(t) \\
   v_y(t)
   \end{bmatrix} =
   \begin{bmatrix}
   \cos \theta(t) & \sin \theta(t) \\
   -\frac{1}{b} \sin \theta(t) & \frac{1}{b} \cos \theta(t)
   \end{bmatrix}
   \begin{bmatrix}
   v_{dx}(t) \\
   v_{dy}(t)
   \end{bmatrix}
   \]
5. **end while**

Cristian Secchi
A robot often needs to move in an environment populated by obstacles (fixed or mobile). The robot, by its knowledge of the environment and/or by the use of onboard sensors, has to detect the obstacles and avoid them in order to navigate in a safe way (i.e. without collisions) in the workspace.

**Navigation:** Given a start and a goal configurations, find a collision-free path that takes the robot from the start to the goal.
**Bug Algorithms**

Bug algorithms are based on the possibility of the robot to use exteroceptive sensors (bumpers, range finders,...) for detecting an obstacle when they are very close to it or even when they are in contact with it.

**Basic Idea:** Move the robot along a straight line towards the objective. When an obstacle is met, circumnavigate it until it is possible to move along a straight line towards the goal.

The robot is assumed to be a point (e.g. no control on the orientation for the unicycle) endowed with a contact sensor. It is also assumed that the robot knows its position, the goal positions and the start position.

**Bug1**

An m-line is the segment connecting a point to $q_{\text{goal}}$.

The robot switches between two possible behaviors:

**motion to goal:** Starting from a leave point, the robot moves along the m-line that connects the leave point to the goal until the goal is reached or an obstacle is met. If the robot meets an obstacle, the contact point is marked as hit point and the robot switches to the boundary following behavior.

**boundary following:** Starting from a hit point, the robot circumnavigates the obstacle until it reaches the hit point again. Among the points laying on the border of the obstacle, the robot determines the one closest to the goal and it follows the border of the obstacle to reach such a point, marked as leave point. The robot commutes to the motion to goal behavior.
**Bug1**

$q_i^\text{L}$: i-th leave point

$q_i^\text{H}$: i-th hit point

\[ q_i^0 = q_{\text{start}} \quad i = 0 \] Initialization

The robot starts from a leave point and implements the motion to goal behavior.

If the line connecting the i-th leave point with the goal intersects the i-th obstacle, then there is no path to reach the goal. The algorithm can detect such a situation.
**Bug1 Algorithm**

**Input:** A robot with a contact sensor  
**Output:** a path until \( q_{goal} \) or a conclusion that such a path does not exist

0: \( i=1 \); \( q_{L0} = q_{start} \);  
1: while Forever do  
2: repeat  
3: \( Da q_{L,i-1} \), move towards \( q_{goal} \).  
4: until \( q_{goal} \) is reached or an obstacle is touched at \( q_{H} \).  
5: if the goal is reached then  
6: Exit.  
7: end if  
8: repeat  
9: Follow the border of the obstacle  
10: until \( q_{goal} \) is reached or \( q_{H} \) is reached again  
11: Determine the point of the border \( q_{H} \) which is at the minimum distance from the goal  
12: go to \( q_{H} \).  
13: if the m-line connecting \( q_{H} \) to the goal intersects the obstacle on which \( q_{H} \) lies then  
14: Conclude that \( q_{goal} \) cannot be reached and exit.  
15: end if  
16: \( i=i+1 \)  
16: end while

**Bug2**

- As the Bug1, the Bug2 commutes between two behaviors: the motion to goal and the il boundary following  
- In Bug2, the m-line is fixed and it is the segment that connects \( q_{start} \) to \( q_{goal} \)

**motion to goal:** Starting from a leave point, the robot moves along the m-line that connects \( q_{start} \) to \( q_{goal} \) until the goal is reached or an obstacle is met. If the robot meets an obstacle, the contact point is marked as hit point and the behavior switches to boundary following.

**boundary following:** Starting from an hit point, the robot circumnavigates the obstacle until it reaches a point on the m-line, marked as leave point and the behavior of the robot switches to motion to goal.
Bug2

$q_i^L$: i-th leave point

$q_i^H$: i-th hit point

$q_i^L = q_{start} \quad i = 0 \quad$ **Initialization**

The robot starts from a leave point and implements the motion to goal behavior.

If during the boundary following the robot meets again the hit point from which it started to circumnavigate the obstacle, then the algorithm detects that there is no solution to the navigation problem.
**Bug2 Algorithm**

**Input:** A robot with a contact sensor  
**Output:** A path until $q_{goal}$ or a conclusion that such a path does not exist

0: $i=1$; $q_{L0}=q_{start}$;
1: while True do
2: repeat
3: Da $q_{L_{i-1}}$, move towards $q_{goal}$ along the $m$-line.
4: until $q_{goal}$ has been reached or an obstacle is touched at the hit point $q_{hi}$.
5: Turn left (or right).
6: repeat
7: Follow the border of the obstacle
8: until
9: $q_{goal}$ is reached or
10: $q_{hi}$ is reached again or
11: the $m$-line is reached again at a point $m$ such that
12: $m \neq q_{hi}$ (the robot didn’t reach the hit point),
13: $d(m, q_{goal}) < d(m, q_{hi})$ (the robot is closer)
14: If the robot moves towards the goal, it does not touch the obstacle
15: Set $q_{L_{i-1}} = m$
16: $i=i+1$
17: end while

---

**Bug1 or Bug2?**

Bug2 may seem better than Bug1 since the robot does not need to fully circumnavigate the obstacles. Nevertheless, this is not true. The choice of the bug algorithm depends on the shape of the obstacles.
**Bug1 or Bug2?**

- Bug1 makes an exhaustive search and finds the best leave point. This requires the robot fully circumnavigates the obstacle.
- Bug2 exploits a greedy approach. When it finds a leave point that is better than the ones previously seen, it chooses it.
- When the obstacles are “simple”, typically Bug2 is better than Bug1.
- In case of “complex” obstacles, Bug1 is better than Bug2.

---

**Tangent Bug**

- It is an improvement of Bug2.
- Proposed in:
  

  available at
  
  http://ieeexplore.ieee.org/stamp/stamp.jsp?
  
  tp=&arnumber=503814&isnumber=10815

  from an UNIMORE IP.
Navigation through potential functions

- The bug algorithms are very simple but they work only for robots moving on a plane and they do not consider the full configuration variable.

- Using potential function navigation, it is possible to plan the motion of complex systems with several configuration variables.

- As for the bug algorithms, they allow to generate a path incrementally, changing it when an obstacle is detected.

A potential function can be interpreted as an energy function on the configuration space. The gradient of $U$ represents the force induced on a robot because of the presence of the potential. The gradient of $U$ is a vector directed towards the increasing direction of $U$. 

$$U : \mathbb{R}^m \rightarrow \mathbb{R}$$

$$\nabla U(q) = \left[ \frac{\partial U}{\partial q_1}(q) \ldots \frac{\partial U}{\partial q_n}(q) \right]^T$$

Gradient of $U$
Navigation through potential functions

The potential functions approach guides the robots as if it was a particle immersed in a potential field given by the sum of different potential functions. The potential field generates a force on the particle. The goal configuration is associated to a potential function that produces a force that attracts the particle. Each obstacle is associated to a potential function that generates a repulsive force on the particle. The combination of all the gradients produces a force that drives the robot towards the goal while avoiding obstacles.

\[ U(q) = \frac{1}{2} q^T \cdot q = \frac{1}{2} q_1^2 + \frac{1}{2} q_2^2 \]

\[ \nabla U(q) = \begin{bmatrix} q_1 & q_2 \end{bmatrix} \]
Navigation through potential functions

The potential functions can also be seen as landscapes where the robot is immersed. The robot moves "downhill" in this landscape. This can be done by forcing the robot to move in the opposite direction of the gradient of the potential function (gradient descent technique).

\[ \dot{q}(t) = -\nabla U(q) \]

The robot stops when it reaches a configuration \( q^* \) where the gradient is zero. This point can be either a minimum, a maximum or a saddle point. We will design the potential in such a way that \( q^* \) will be a (local) minimum.

Attractive potential. It has a global minimum in \( q_{\text{goal}} \). Its role is to attract the robot towards the goal.

\[ U_{\text{att}}(q) \]

Repulsive potential. It is given by the sum of all the repulsive potentials associated to the obstacles. It has the role to repel the robot from the obstacles in the environment.

\[ U_{\text{rep}}(q) = \sum_{i=1}^{n_{\text{obs}}} U_{\text{rep}_i}(q) \]

\[ U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q) \]

\[ \dot{q}(t) = -\nabla U(q) \]
The attractive potential

The potential $U_{\text{att}}$ has to grow as the distance from $q_{\text{goal}}$ grows. In this way the robot, using the gradient descent, is attracted towards the configuration $q_{\text{goal}}$. This is equivalent to say that $q_{\text{goal}}$ is a global minimum. Indicating with $d(q,q_{\text{goal}})$ the distance between $q$ and $q_{\text{goal}}$, a possible choice is:

$$U_{\text{att}} = \frac{1}{2}k_a d^2(q,q_{\text{goal}}) \quad \nabla U_{\text{att}} = \frac{1}{2}k_a \nabla d^2(q,q_{\text{goal}}) = k_a (q - q_{\text{goal}})$$

If the distance is big, the robot approaches to the goal quickly while if the distance is small the robot approaches to the goal slowly.
The repulsive potential

It is a sum of repulsive potentials terms, each of which is associated to an obstacle. Then $i$-th term, has to be such to generate a repulsive effect that is bigger when the robot is closer to the $i$-th obstacle. Furthermore, no effect should be produced by the $i$-th potential is the robot is sufficiently distant form the $i$-th obstacle. A possible choice is:

$$U_{rep_i}(q) = \begin{cases} 
\frac{1}{2} k_i \left( \frac{1}{d_i(q)} - \frac{1}{Q^*} \right)^2 & d_i(q) \leq Q^* \\
0 & d_i(q) > Q^*
\end{cases}$$

$$U_{rep}(q) = \sum_{i=1}^{n_{obs}} U_{rep_i}(q)$$

$Q^*$= distance beyond which the robot disregards the $i$-th obstacle.
$d_i(q)$= distance between the robot and the $i$-th obstacle
$n_{obs}$= number of obstacles.

\[
\nabla U_{rep_i}(q) = \begin{cases} 
k_i \left( \frac{1}{Q^*} - \frac{1}{d_i(q)} \right) \nabla d_i(q) & d_i(q) \leq Q^* \\
0 & d_i(q) > Q^*
\end{cases}
\]

If $d_i(q)$ goes to zero then the gradient goes to infinity!

$$\nabla U_{rep}(q) = \sum_{i=1}^{n_{obs}} \nabla U_{rep_i}(q)$$
The gradient descent algorithm

\[ \dot{q}(t) = -\nabla U = -\nabla U_{\text{att}} - \nabla U_{\text{rep}} \]

Input: A way of computing \( \nabla U[q] \)
Output: A sequence of points \((q[0], q[1], \ldots, q[i])\)

1: \(q[0] = q_{\text{start}}\)
2: \(i = 0\)
3: while \(\nabla U(q[i]) \neq 0\) do
4: \(q[i + 1] = q[i] - \alpha(i) \nabla U(q[i])\)
5: \(i = i + 1\)
6: end while

The value \(\alpha(i)\) determines the length of the step the robot
descent the gradient of \(U\). If \(\alpha(i)\) is small, the algorithm is
accurate but computationally expensive while if \(\alpha(i)\) is big the
algorithm is fast but less accurate. The choice of \(\alpha(i)\) depends
on the application.

Local minima

The gradient descent algorithm can get stuck into a local
minimum.

The attractive potential is counterbalanced by the repulsive
potential.
Local minima

There are several approaches for solving the local minima problem:


2) Navigation Functions (see the book Principles of Robot motion)

Other navigation algorithms

- There are many other navigation strategies:
  - Roadmaps
  - Cell Decomposition
  - Sampling based algorithms
  - ...

- Please have a look at:
  - Choset, Lynch, Hutchinson, Kantor, Burgard, Kavraki, Thrun, Principle of Robot Motion- Theory, algorithms and Implementations, MIT Press 2005
Motion Control

- Other practically interesting problems
  - Planning considering kinematic limits (e.g., bounded velocity)
  - Shortest path
  - Multi-robot systems
  - ...  

SLAM - Simultaneous Localization and Mapping

- In order to track a desired trajectory, a mobile robot needs to know its current configuration.
- In order to execute a non-trivial navigation strategy, a robot needs to know a map of the surrounding environment.
- Each feedback controller requires the knowledge of the state of the robot (i.e., the configuration).
- In general, for a mobile robot really autonomous, it is necessary that it can localize itself and build a map of the surrounding environment.
**SLAM – Simultaneous Localization and Mapping**

SLAM searches an answer to the question:

"Is it possible to use a robot starting from:

1. **an unknown initial position**, in an
2. **unknown environment**, for
3. **incrementally building** a map of the
   environment and
4. **at the same time**

use the map for determining the position
of the robot?"

---

**Origin: the odometric localization**

**Odometry**

**Odometry** is a technique for estimating the configuration of a
WMR by integrating the kinematic model.
Odometric localization: the unicycle

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \omega
\end{align*}
\]

Assume that the inputs are constant during each sample time (this is true if the system if a digital controller is used).

- If at time \(k(=kT)\) are known
  - the configuration of the robot \(q(k)\)
  - the inputs \(v(k)\) and \(w(k)\) that will be applied in \([k,k+1)\)
  - The sampling period \(T\)

- Then it is possible to use the kinematic model of the robot for determining \(q(k+1)\)


Odometric localization: the unicycle

- Discretizing the model with the Euler method:
  \[
  \begin{align*}
  x(k+1) &= x(k) + v(k)T \cos \theta(k) \\
  y(k+1) &= y(k) + v(k)T \sin \theta(k) \\
  \theta(k+1) &= \theta(k) + T\omega(k)
  \end{align*}
  \]

- It is very simple but the computation can be affected by some errors due to:
  - \(\theta(k)\) is considered constant in \([k,k+1)\)
  - The commanded inputs \(v(k)\) and \(\omega(k)\) may be different from those really applied (non ideality of the actuation system)
  - Errors due to wheels slipping
  - Calibration errors (\(d\) and \(r\))
  - Errors introduced by the traction system
  - ...

- The odometric localization produces an unreliable value and characterized by a significant random error that is accumulated over time.
Using only odometry

![GPS (for reference)](image)

Exteroceptive sensors

- For compensating the errors introduced by the odometry and for building a map of the surrounding environment, exteroceptive sensors are used. These sensors are capable of measuring the position of the robot with respect to the environment.

- The most used exteroceptive sensors are laser scanner, cameras (mono and stereo), GPS and, for simple application, sonar.
**The laser scanner**

- It emits a pencil of rays with a certain angular step. Each ray returns the distance of the obstacle it hits.
- For each ray, the information about range (distance) and bearing (emission angle of the ray) is returned.
- It provides the polar coordinates of the elements in the environment.

**The localization problem**

Given a map $m$ of reference points (landmarks) whose position is known a priori:

- Measure the relative position of the landmarks $y(k)$ (e.g.: range and bearing)
- Determine the configuration of the robot $x(k)$ using the measure $y(k)$
- A filter is necessary because the measurements are noisy
- The position of the landmarks is used for reducing the uncertainty about the configuration of the robot.

- $x_k$: configuration of the robot at time $k$
- $u_k$: control input applied at time $k-1$ and held constant in the interval $[k-1,k)$
- $y_k$: measure of the landmarks taken at time $k$
- $X$: configuration history $\{x_1,..,x_k\}$
- $U$: input history $\{u_1,..,u_k\}$
- $m$: set of landmarks
The mapping problem

Given the configurations of the robot $X^k$

- Measure the relative position of the landmarks $y(k)$ (e.g.: range and bearing)
- Determine the map using the measure $y(k)$
  - A filter is necessary because the measurements are noisy
  - The configuration of the robot is used for decreasing the uncertainty of the landmark detection.

- $X^k$: configuration history $\{x_1,...,x_k\}$
- $m_i$: position of landmark $i$
- $m$: set of landmarks

The SLAM problem

- From the knowledge of the measurements $Y^k$
- Determine the configuration of the robot $X^k$
- Build a map $m$ of the landmarks

- $x_k$: configuration of the robot at time $k$
- $u_k$: control input applied at time $k-1$ and held constant in the interval $[k-1,k)$
- $y_k$: measure of the landmarks taken at time $k$
- $X^t$: configuration history $\{x_1,...,x_t\}$
- $U^t$: input history $\{u_1,...,u_t\}$
- $Y^t$: history of the measurements $\{y_1,...,y_t\}$
- $m$: set of landmarks
SLAM
- Localization and mapping are two correlated problems
  - two uncertain quantities have to be inferred by a single measurement
  - A solution can be obtained only if the localization and mapping problems are considered together.

The uncertainty in the position is correlated with the uncertainty of the map.

SLAM - Setting
- A robot with a known kinematic model moves in an environment populated by fixed landmarks (process model)
- The robot is endowed with a sensor that can measure the relative position between each landmark and the robot itself (observation model)
**Process model**

- Assume, by now, that the kinematic model of the robot is a linear system described by:

\[ x_v(k) = Ax_v(k-1) + u_v(k) + w_v(k) \]

where

- \( A \) is the state matrix
- \( u_v \) is the input vector
- \( w_v \) is a gaussian random vector with zero mean and covariance matrix \( Q_v(k) \) that models the uncertainty of the model.

- The model of the i-th landmark is given by:

\[ p_i(k) = p_i(k-1) \]

---

**Process model**

- It is possible to group the robot model and the landmarks models into a unique model of the process.

- The state of the process is given by:

\[ x(k) = \begin{pmatrix} x_v^T(k) & p_1^T & \cdots & p_N^T \end{pmatrix}^T \]

- The model of the process is given by:

\[
\begin{pmatrix}
  x_v(k) \\
  p_1(k) \\
  \vdots \\
  p_N(k)
\end{pmatrix} = \begin{pmatrix}
  A & 0 & \cdots & 0 \\
  0 & I_{p_1} & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & I_{p_N}
\end{pmatrix} \begin{pmatrix}
  x_v(k-1) \\
  p_1(k-1) \\
  \vdots \\
  p_N(k-1)
\end{pmatrix} + \begin{pmatrix}
  u_v(k) \\
  0_{p_1} \\
  \vdots \\
  0_{p_N}
\end{pmatrix} + \begin{pmatrix}
  w_v(k) \\
  0_{p_1} \\
  \vdots \\
  0_{p_N}
\end{pmatrix} = Ax(k-1) + u(k) + w(k)
\]

where \( I_{p_i} \) and \( 0_{p_i} \) and the identity matrix and the null matrix of dimension \( p_i \), respectively.
Observation model

- Assume, by now, that the sensors measuring the landmarks are linear. The observation model of the $i$-th landmark is given by:

$$y_i(k) = C_i x(k) + v_i(k)$$

where:

- $v_i(k)$ is a gaussian random vector with zero mean and covariance matrix $R_i(k)$ that models the measure uncertainty.
- $C_i$ is the observation matrix (i.e. the output matrix) that relates the state of the process with the relative measure of the landmark and it can be written as:

$$C_i = \begin{pmatrix} -A & 0 & \cdots & 0 & C_{i_p} & 0 & \cdots & 0 \end{pmatrix}$$

- The observation model can be rewritten as:

$$y_i(k) = C_{i_p} p_i - Ax_i(k)$$

- If $n$ landmarks are measured then

$$y(k) = \begin{pmatrix} y_1(k) \\ \vdots \\ y_n(k) \end{pmatrix} = C x(k) + v(k)$$

where

$$C = \begin{pmatrix} C_1 \\ \vdots \\ C_n \end{pmatrix} \quad v(k) = \begin{pmatrix} v_1(k) \\ \cdots \\ v_n(k) \end{pmatrix} \quad R(k) = diag(R_1(k), \ldots, R_n(k))$$

- Since the number of landmarks observed by the robot changes as the robot moves, $C$, $v(k)$ and $R(k)$ have a variable dimension.
**SLAM**

- Determining the position of the robot and that of the landmarks is equivalent to estimate the state of the process using the measurements.

- Both the process model and the observation model are affected by a gaussian uncertainty.

- The Kalman filter solves the problem! And it solves it online!

---

**The Kalman Filter**

**Prediction**
- Predicted state $\hat{x}(k | k-1) = A\hat{x}(k-1 | k-1) + u(k-1)$
- Predicted covariance $\Sigma(k | k-1) = A\Sigma(k-1 | k-1)A^T + Q(k)$

**Update**
- Innovation $e(k) = y(k) - C\hat{x}(k | k-1)$
- Covariance of the innovation $S(k) = C\Sigma(k | k-1)C^T + R(k)$
- Kalman gain $K(k) = \Sigma(k | k-1)C^TS(k)^{-1}$
- State updated $\hat{x}(k | k) = \hat{x}(k | k-1) + K(k)e(k)$
- Covariance update $\Sigma(k | k) = (I - K(k)C)\Sigma(k | k-1)$
The Kalman Filter

**Prediction**

1. State prediction
\[ \hat{x}(k) = A\hat{x}(k-1) + Bu(k-1) \]
2. Covariance prediction
\[ \Sigma(k) = A\Sigma(k-1)A^T + Q(k) \]

**Update**

1. Compute Kalman gain
\[ K(k) = \Sigma(k)C^T(C\Sigma(k)C^T + R)^{-1} \]
2. Update the estimate
\[ \hat{x}(k) = \hat{x}(k)^{-} + K(k)e(k) \]
3. Update covariance
\[ \Sigma(k) = (I - K(k)C)\Sigma(k)^{-} \]

The Extended Kalman Filter – EKF

- The Kalman filter can be used only if the process model and the observation model are linear.
- The kinematic model of a mobile robot is often nonlinear (e.g. the unicycle).
- The observation model is often non linear (e.g. when using a laser scanner)
- The Extended Kalman Filter (EKF) allows to get rid of the linearity assumption.
The process and the measurements are described by nonlinear functions

\[
\begin{align*}
x(k) &= f(x(k-1),u(k-1)) + w(k) \\
y(k) &= h(x(k)) + v(k)
\end{align*}
\]

where \( w(k) \) and \( v(k) \) are gaussian random vectors with zero mean and covariance matrices \( Q(k) \) and \( R(k) \) respectively.

---

The idea of the EKF is to approximate the system whose state has to be estimated with a linear system. The approximation is done by linearizing the system around the estimated state. Given the estimate at time \( k-1 \), the system is linearized around that estimate and then the state at time \( k \) is estimated.

At each instant, the system is treated as a linear system and therefore it is possible to use the Kalman filter for estimating the state.

At each instant, the considered linear system is different because the linearization is executed around different points.

\[
f(x(k-1),u(k-1)) = f(\hat{x}(k-1),u(k-1)) + \left. \frac{\partial f}{\partial x} \right|_{x(\hat{x}(k-1)),u(\hat{x}(k-1))} (x(k-1) - \hat{x}(k-1))
\]

\[
h(x(k)) = h(\hat{x}(k)) + \left. \frac{\partial h}{\partial x} \right|_{x(\hat{x}(k))} (x(k) - \hat{x}(k)) = h(\hat{x}(k)) + H(k)(x(k) - \hat{x}(k))
\]
The EKF algorithm

- **Prediction**
  - Predicted state
    \[ \hat{x}(k) = f(\hat{x}(k-1), u(k-1)) \]
  - Predicted Covariance
    \[ \Sigma(k) = F(k)\Sigma(k-1)F(k)^T + Q(k) \]

- **Update**
  - Innovation
    \[ e(k) = y(k) - h(\hat{x}(k)) \]
  - Innovation Covariance
    \[ S(k) = H(k)\Sigma(k)H(k)^T + R(k) \]
  - Kalman gain
    \[ K(k) = \Sigma(k)H(k)^T S(k)^{-1} \]
  - State update
    \[ \hat{x}(k) = \hat{x}(k) - K(k)e(k) \]
  - Covariance update
    \[ \Sigma(k) = (I - K(k)H(k))\Sigma(k) \]

EKF – Localization

A differential drive robot needs to follow a predefined circular trajectory. The trajectory tracking is obtained by IO-SFL and, therefore, it is necessary to know the configuration of the robot. The kinematic model of the robot is affected by a process uncertainty. The position of the robot is detected by a camera and the measure is affected by an uncertainty.
Process model

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
y &= v \cos \theta \\
\dot{\theta} &= \omega
\end{align*}
\]

\[
\begin{align*}
x(k) &= x(k-1) + T v(k-1) \cos \theta(k-1) + w_x(k) \\
y(k) &= y(k-1) + T v(k-1) \sin \theta(k-1) + w_y(k) \\
\theta(k) &= \theta(k) + T \omega(k-1) + w_\theta(k)
\end{align*}
\]

\[
T = 0.001 \quad x_B = x + b \cos \theta \quad v_{xv} = \dot{x}_{des} + k_1 (x_{des} - x_B) \\
b = 0.01 \quad y_B = y + b \sin \theta \quad v_{yv} = \dot{y}_{des} + k_2 (y_{des} - y_B)
\]

\[
\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\frac{1}{b} \sin \theta & \frac{1}{b} \cos \theta \end{bmatrix} \begin{bmatrix} v_{xv} \\ v_{yv} \end{bmatrix}
\]

\[
\text{cov} \left( \begin{bmatrix} w_x \\ w_y \\ w_\theta \end{bmatrix} \right) = Q = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}
\]

Observation model

\[
\begin{align*}
y(k) &= \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \begin{bmatrix} v_x(k) \\ v_y(k) \\ v_\theta(k) \end{bmatrix} \\
\text{cov} \left( \begin{bmatrix} v_x(k) \\ v_y(k) \\ v_\theta(k) \end{bmatrix} \right) &= R(k) = \begin{bmatrix} 0.8 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.8 \end{bmatrix}
\end{align*}
\]

- The output gives an information about the full state!
- The output is characterized by an uncertainty
Results – Feedback the output

The robot gets lost and the real trajectory (blue) is different from the desired one (black). The red circles indicate the readings of the sensor.

EKF

- Using the EKF it is possible to build an estimate of the state that is robust with respect to uncertainties.
- The model of the robot is nonlinear and, therefore, it is necessary to build the Jacobians of the process model at each step.
**EKF Prediction**

- Consider the process model without noise

\[
x(k) = f(x(k-1), u(k-1)) = \begin{pmatrix} x(k-1) + T_v(k-1) \cos \theta(k-1) \\ y(k-1) + T_v(k-1) \sin \theta(k-1) \\ \theta(k) + T_o(k-1) \end{pmatrix}
\]

- Build the Jacobian of \( f \) evaluated in \( \hat{x}(k-1), u(k-1) \)

\[
F(k) = \begin{pmatrix}
\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \theta} \\
\frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial \theta} \\
\frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial \theta}
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & -T_v \sin \theta \\
0 & 1 & T_v \cos \theta \\
0 & 0 & 1
\end{pmatrix}
\]

**EKF Prediction**

- Predict the state using the mode and update the covariance

\[
\hat{x}(k)^+ = f(\hat{x}(k-1), u(k-1))
\]

\[
\Sigma(k)^+ = F(k) \Sigma(k-1) F(k)^T + Q(k)
\]
EKF Update

- Build the Jacobian of the observation model

\[
y(k) = h(x(k)) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \begin{bmatrix} v_x(k) \\ v_y(k) \\ \nu(k) \end{bmatrix}
\]

\[
H(k) = \begin{bmatrix} \frac{\partial h_x}{\partial x} & \frac{\partial h_x}{\partial y} & \frac{\partial h_x}{\partial \theta} \\ \frac{\partial h_y}{\partial x} & \frac{\partial h_y}{\partial y} & \frac{\partial h_y}{\partial \theta} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

- Build the innovation and its covariance

\[
e(k) = y(k) - h(\hat{x}(k^-))
\]

\[
S(k) = H(k)\Sigma(k^-)H(k)^T + R(k)
\]

EKF Update

- Build the Kalman gain and update the estimate

\[
K(k) = \Sigma(k^-)H(k)^T S(k)^{-1}
\]

\[
\hat{x}(k) = \hat{x}(k^-) + K(k)e(k)
\]

\[
\Sigma(k) = (I - K(k)H(k))\Sigma(k^-)
\]
Results

The robot (blue) follows the desired trajectory (black) and the estimate of the position of the robot (green) is very accurate.

The initial estimate is very uncertain but the Kalman filter makes it more and more accurate.