AMR – Useful Facts & Formulary

Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

Index

1. Useful Facts

1.1 Global Mobility vs Local Mobility

The general formulation is that:

- $Not\ controllable \Rightarrow Holonomic \Rightarrow Integrable \Rightarrow Global\ Mobility\ Limitation$ ✅
- $Controllable \Rightarrow Non-Holonomic \Rightarrow Not\ Integrable \Rightarrow NO\ Global\ Mobility\ Limitation$ ✖

Also, we know that:

- If we have a ***GEOMETRIC CONSTRAINT***:
  - $Local\ Mobility\ Limitation$ ☑
  - $Global\ Mobility\ Limitation$ ☑
  - $Any\ GC\ \Rightarrow KC$
- If we have a ***KINEMATIC CONSTRAINT***:
  - *Local Mobility Limitation* ☑
  - *Global Mobility Limitation ->* ⚠ *STUDY CONTROLLABILITY!* – Point (1.4)

1.2 Equivalent Kinematic Models

- $Differential-Drive,\ Synchro-Drive \quad \Rightarrow \underline{UNICYCLE}$
- $Tricycle,\ Car-like \quad \Rightarrow \underline{BICYCLE}$

1.3 Creating a Kinematic Model

1. First of all, we have to rewrite our constraints in Pfaffian form: $a^T(q)\cdot\dot{q} = 0$
2. Write the basis for nullspace of: $\dot{q}\epsilon\mathcal{N}(a^T(q))$
   For writing a base, we need to write m linear independent vectors g_i .
   We can retrieve m by checking the number of constraints (k) and the number of parameters (n), such that $m = n - k$.
3. For writing a linear independent vector g_i, I have to choose a vector which multiplied by $a^T(q)$ gives 0.

   EXAMPLE:

   $$a^T(q) = (sin\theta, -cos\theta, 0) \text{ with } n = 3;\ k = 1$$

   Then a basis for this (considering $m = 2$):

   - $g_1(q) = \begin{pmatrix} cos\theta \\ sin\theta \\ 0 \end{pmatrix}$
   - $g_2(q) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

4. $\dot{q} = g_1(q)u_1 + g_2(q)u_2 = \begin{pmatrix} cos\theta \\ sin\theta \\ 0 \end{pmatrix}\cdot u_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\cdot u_2 \Longrightarrow$ **KINEMATIC MODEL!**

   (Also written in a form like)

$$\begin{cases} \dot{x} = cos\theta\cdot u_1 + 0\cdot u_2 = cos\theta\cdot u_1 \\ \dot{y} = sin\theta\cdot u_1 + 0\cdot u_2 = sin\theta\cdot u_1 \\ \dot{\theta} = 0\cdot u_1 + 1\cdot u_2 = u_2 \end{cases}$$

1.4 Checking controllability

In order to check Controllability, we need to get the **Kinematic model** from point (1.3)

After that we must compute the **Lie-Brackets**:

EXAMPLE:
Given g_1, g_2, q from our KM, we must compute g_3 in the following way:

$$g_3 = \frac{\partial g_2}{\partial q} \cdot g_1 - \frac{\partial g_1}{\partial q} \cdot g_2$$

In this case, both for $\frac{\partial g_2}{\partial q}, \frac{\partial g_1}{\partial q}$, the matrices will have dimension $(n \ x \ n)$, where $n =$ number of parameters in q.

Also:

- Rows = Represent the g_{x_i} element of g_x.
- Columns = Represent the q_i element of q.

EXAMPLE (for a single element of the Jacobian matrix):

If I have to derive $\frac{g(x)}{h(x)} \Leftrightarrow \frac{\mathrm{R} \cdot \cos(\theta)}{\theta}$ with respect to θ, then I have to simply apply derivative of a fraction

$$\frac{\frac{dg}{dx} \cdot h(x) - \frac{dh}{dx} \cdot g(x)}{(h(x))^2} \Rightarrow \frac{(-R \cdot \sin(\theta) - R \cdot \cos(\theta) \cdot 1)}{\theta^2}$$

After computing g_3, we need to add it into the linear space span:

$$\triangle = span \ \{g_1(x), g_2(x), g_3(x)\}$$

Finally, for achieving controllability we have to meet the following condition:

- Rank of $\triangle$ (meaning the rank of the matrix $[g_1, g_2, g_3]$) = n

Now, how do we check the rank of the matrix?

- If the matrix of the span is **NOT** $(n \ x \ n)$, then we know that $rank \neq n \implies$ 2nd order LB
- If the matrix of the span **IS** $(n \ x \ n)$ but we have at least one row or one column$= 0$, then $rank \neq n \implies$ 2nd order LB
- Check then if rank is n through Laplace method or block triangular / diagonal matrix, if we are dealing with $n \ x \ n$ matrix with n > 3. If the matrix is 2 x 2 then it's easy to compute rank.

In all other cases, we must compute 2nd/3rd order lie bracket.

TO BE WRITTEN

When we add Lie Bracket into the Span Matrix $\triangle$, we add the Lie Bracket which is not a linear combination of vectors that are already present in $\triangle$.

1.5 Chained Forms

$$\begin{cases} \dot{z}_1 = v_1 \\ \dot{z}_2 = v_2 \\ \dot{z}_3 = z_2 \cdot v_1 \\ \vdots \\ \dot{z}_n = z_{n-1} \cdot v_1 \end{cases}$$

Flat outputs = z_1, z_n.

1.6 Path and Trajectory

Before explaining path and trajectory, let's recall the definitions of Kinematic Model and Geometric Model:

- Kinematic Model: $\dot{q} = \sum_{i=1}^{m} g_i(q) \cdot u_i$
- Geometric Model: $q' = \sum_{i=1}^{m} g_i(q) \cdot \tilde{u}_i$

Where: $m = n - k$.

We consider $\tilde{u}_i$ the Geometric Inputs of our Geometric model, while u_i the Velocity Inputs of our Kinematic Model.

UNICYCLE EXAMPLE:

KINEMATIC MODEL = $\begin{cases} \dot{x} = cos\theta \cdot v \\ \dot{y} = sin\theta \cdot v \\ \dot{\theta} = \omega \end{cases}$ | *GEOMETRIC MODEL* = $\begin{cases} x' = cos\theta \cdot \tilde{v} \\ y' = sin\theta \cdot \tilde{v} \\ \theta' = \tilde{w} \end{cases}$

Where $v = \tilde{v} \cdot \dot{s}$ and $\omega = \tilde{\omega} \cdot \dot{s}$

- **PATH**:
  - **Definition:**
    - As for path, we define it as $q(s)$ with $s \in [s_i, s_f]$.
    - Here we just have a *geometric description* of the motion. Namely we just have a path.
  - **Planning:**
    - Given $q_i, q_f \in C$, find a feasible (something that a robot can actually follow) path $q(s), s \in [s_i, s_f]$ in such a way that $q(s_i) = q_i$ and $q(s_i) = q_f$.
- **TRAJECTORY:**
  - **Definition:**
    - We define a trajectory $q(t)$ with $t \in [t_i, t_f]$ where, along with geometric description of motion, we also have to take into consideration *time*.
    - Namely, we have *geometric description* of the motion + *speed*.
  - **Planning:**
    - Given $q_i, q_f \in C$, find a feasible trajectory $q(t), t \in [t_i, t_f]$ in such a way that $q(t_i) = q_i$ and $q(t_i) = q_f$.
      - Direct approach: Compute directly $q(t), t \in [t_i, t_f]$.
      - Indirect approach: We first choose $g(s)$ which is a GEOMETRY MOTION, then we choose a timing law $s(t)$, which gives us time information.

As for the models to use, we can sum them up as follows:

- **Path planning** = Geometric Model
- **Trajectory planning**:
  - Direct = Kinematic Model
  - Undirect = Geometric Model + timing law $s(t)$.

For direct trajectory planning and reconstruction formulas, please see formulary at Chapter (2).

1.6.1 Flat Outputs

The Kinematic model is called FLAT if there exist outputs (function of the state q) such that q (whole state) and u (the whole input) can be reconstructed from the history of these outputs.

For UNICYCLE / BICYCLE, the flat outputs are $(X, Y)^T$.

1.6.2 Path & Trajectory Planning

We will first discuss a general approach, then apply the Path planning to the Unicycle model.

1. Compute initial and final w on q, u:

   $$w_i = h(q_i)$$
   $$w_f = h(q_f)$$

   Also, make sure to compute **basic** and **boundary** conditions.

   (*N.B.* In the chained form, we have to reconstruct the missing state variable by using the flat outputs. So, the boundary constraints are the basic constraints applied on the missing state variables explicated in function of the flat outputs.)

   Example: $z_2 = \frac{z_3'}{z_1'}$, so $z_2(S_i) = 0$ AND $z_2(S_f) = 1$

   We'll see them later in the Unicycle example.

2. Interpolate in s or t from w_i to w_f
   - Path planning $\longrightarrow$ Interpolate in s
   - Trajectory planning $\longrightarrow$ Interpolate in t

3. Reconstruct the history of q and u using algebraic formulas:
   - $q = \eta(w, \frac{\partial w}{\partial \gamma}, \frac{\partial^2 w}{\partial^2 \gamma}, \frac{\partial^3 w}{\partial^3 \gamma}, \dots, \frac{\partial^r w}{\partial^r \gamma})$
   - $u = \mu(w, \frac{\partial w}{\partial \gamma}, \frac{\partial^2 w}{\partial^2 \gamma}, \frac{\partial^3 w}{\partial^3 \gamma}, \dots, \frac{\partial^r w}{\partial^r \gamma})$

   Where $\gamma = \begin{cases} S \; if \; PATH \\ t \; if \; TRAJECTORY \end{cases}$

UNICYCLE EXAMPLE:

Our goal is to design a path from q_i $(0,0,0)$ to q_f $(1,1,1)$, as follows:

$$q_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix}; q_f = \begin{pmatrix} x_f \\ y_f \\ \theta_f \end{pmatrix} \rightarrow q(s), s \in [0,1]$$ (Free choice of s)

Recalling now the Geometric Model (since we are working on the path):

GEOMETRIC MODEL = $\begin{cases} x' = cos\theta \cdot \tilde{v} \\ y' = sin\theta \cdot \tilde{v} \\ \theta' = \tilde{w} \end{cases}$ where $W = (x, y) \rightarrow$ Flat outputs

Now, let's compute the first point 1.

1. Compute initial and final w on q, u.

In this case, the flat outputs are really easy to compute since we know they are x, y for the unicycle, so:

$w_i = (x_i, y_i)^T$
$w_f = (x_f, y_f)^T$

Now we have to compute also **basic** conditions and **boundary** conditions for the Unicycle in order to generate two paths $X(S)$ and $Y(S)$ with $s \in [0, 1]$.

- **Basic** conditions:
  - $X(S)$ and $Y(S)$ must satisfy:
    - $\begin{cases} X(0) = Xi \\ X(1) = Xf \\ Y(0) = Yi \\ Y(1) = Yf \end{cases}$
- **Boundary** conditions:
  - From the Geometric Model of the Unicycle, we know that:
    - $\begin{cases} X'(0) = cos\theta_s \cdot \tilde{v}(0) \\ X'(1) = cos\theta_g \cdot \tilde{v}(1) \\ Y'(0) = sin\theta_s \cdot \tilde{v}(0) \\ Y'(1) = sin\theta_g \cdot \tilde{v}(1) \end{cases}$

Now, by simply letting the two velocities $\tilde{v}(0), \tilde{v}(1)$ equal to k, we end up with the following boundary/basic constraints:

- $\begin{cases} X(0) = Xi \\ X(1) = Xf \\ X'(0) = cos\theta_s \cdot k = 1 \cdot k = k \\ X'(1) = cos\theta_g \cdot k = 0 \cdot k = 0 \\ Y(0) = Yi \\ Y(1) = Yf \\ Y'(0) = sin\theta_s \cdot k = 0 \cdot k = 0 \\ Y'(1) = sin\theta_g \cdot k = 1 \cdot k = k \end{cases}$

2. *Interpolate in s or t from w_i to w_f*

So, we have 4 conditions on $X(S)$ and 4 conditions on $Y(S) \rightarrow$ Cubic polynomial

N.B.

- *4 constraints = Cubic polynomial =* $as^3 + bs^2 + cs + d$
- *3 constraints = Quadratic polynomial =* $as^2 + bs + c$
- *2 constraints = First-order polynomial = as + b*

Now, we interpolate in S (CHECK BOUNDARY CONSTRAINTS WHEN WE PUT SOMETHING EQUAL TO ANOTHER THING. Example: $X(0) = d \Rightarrow d = 0$, due to boundary constraint):

As for $X'(0), X'(1), Y'(0), Y'(1)$ we have to consider the derivative w.r.t s of the cubic polynomial.

$$as^3 + bs^2 + cs + d \Rightarrow \frac{\partial}{\partial s} \Rightarrow s \cdot (3as + 2b) + c$$

- $X(0) = d \Rightarrow d = 0$
- $X(1) = a + b + c = 1$
    - $\Rightarrow a = 1 - b - k$
    - $\Rightarrow a = 1 - (3 - 2k) - k$
    - $\Rightarrow a = 1 - 3 + 2k - k$
    - $\Rightarrow a = -2 + k = k - 2$
- $X'(0) = c = k$
- $X'(1) = 3a + 2b + c = 0$
    - $\Rightarrow b = 3 - 2k$

And in the same way for Y:

- $Y(0) = d \Rightarrow d = 0$
- $Y(1) = a + b + c + d \Rightarrow a = k - 2$
- $Y'(0) = c = 0$
- $Y'(1) = 3a + 2b + c = k \Rightarrow b = 3 - k$

3. *Reconstruct the history of q and u using algebraic formulas:*

Once we reconstructed the history, we can plug in the reconstruction formulas so that we can reconstruct the inputs and the state. For example, we can express θs in the following way:

$$\theta s = ATAN2(y'(s), x'(s)) + K \cdot \pi$$

More formulas in the formulary (Chapter (2))

1.7 Trajectory Tracking

When dealing with a trajectory tracking problem, we have to first set p as the variable we want to track.

$p = \theta \Rightarrow \dot{p} = \dot{\theta}$ | Namely, we want to set our $\dot{\theta} = u'$ using Input-Output linearization.

This is equal now to setting $v \cdot \frac{\sin(\phi - \gamma)}{l \cdot \cos\phi} = u'$.

At this point, we are dealing with a scalar problem, so we need to find a control law for this.

This means setting:

$$\dot{\theta}_d(t) = u' \quad \Rightarrow \quad v = \frac{l \cdot \cos\phi}{\sin(\phi - \gamma)} \cdot u' \Big\} \ \boldsymbol{CONTROL\ LAW}$$

Now, really simply, we conclude the exercise such that:

$u'_{\theta_d} = \dot{\theta}_d + K_\theta(\theta_d - \theta)$ with $K_\theta > 0$ which will guarantee global asymptotically stability.

1.8 Extended Kalman Filter (EKF)

For building the Extended Kalman Filter, we have to follow these steps:

1. **GET KINEMATIC MODEL**
   a. $\begin{cases} \dot{x} = v \cdot cos\theta \\ \dot{y} = v \cdot sin\theta \\ \dot{\theta} = \omega \end{cases}$
   b. **WATCH OUT!** If you are dealing with SLAM, add the *landmarks* in $\dot{q}$ and update the Kinematic model accordingly. In KM & Euler Integration, landmarks are 0.
2. **EULER INTEGRATION**
   a. **Process**
   Here basically, we simply rewrite our parameters in function of $k + 1$ as follows:
$$x_{k+1} = x_k + v_k \cdot \cos\theta_k \cdot T_s + \nu_{1,k}$$
$$y_{k+1} = y_k + v_k \cdot \sin\theta_k \cdot T_s + \nu_{2,k}$$
   Where $\nu_k \sim \mathcal{N}(0, \nu_k)$ with $\nu_k \in \mathbb{R}^{2x2}$
   b. **Measurement**
   In measurement, we define the Y_k vector as follows:
$$\boldsymbol{Y_k} = h_x(q_k) + w_k$$
   Where $W_k \sim \mathcal{N}(0, W_k) \in \mathbb{R}^{2x2}$
   We choose how many $h_x(q_k)$ and w_k to instantiate, based on the exam track.
   Remember that we put only the measurements that don't regard the control inputs!
   Think of getting the distance from 2 landmarks, then we would write:
$$\boldsymbol{Y_k} = \begin{pmatrix} h_1(q_k) \\ h_2(q_k) \end{pmatrix} + \begin{pmatrix} w_{1,k} \\ w_{2,k} \end{pmatrix}$$
   Making sure to explicit $h_1(q_k)$ and $h_2(q_k)$:
$$\boldsymbol{h_1(q_k)} = \sqrt{\left(x_s - x_{l_1}\right)^2 + \left(y_s - y_{l_1}\right)^2}$$
$$\boldsymbol{h_2(q_k)} = \sqrt{\left(x_s - x_{l_2}\right)^2 + \left(y_s - y_{l_2}\right)^2}$$
3. **LINEARIZATION**
   In linearization, we rewrite our q vector in function of q_k, so it becomes $q_k = (x_k, y_k, \theta_k)^T$ and write our F_k function as:
$$\boldsymbol{F_k} = \frac{\partial f(q_k, u_k)}{\partial q_k} |_{q_k = \hat{q}_k} \in \mathbb{R}^{3x3\ (nxn)}$$

Once the F_k matrix has been computed, make sure to replace all the k terms by adding a hat (^) on top, since we are evaluating them on $\hat{q}_k$. Finally, we need to compute the H_{k+1} Jacobian which is:
$$\boldsymbol{H_{k+1}} = \frac{\partial Y_k}{\partial q_k} \in \mathbb{R}^{2x3}$$

4. **EKF FILTER**
   After computing the two Jacobians, we simply write the EKF filter formulas:

PREDICTION = $\begin{cases} \boldsymbol{\hat{q}_{k+1|k}} = f(q_k, u_k) \\ \boldsymbol{P_{k+1|k}} = F_k \cdot P_k + F_k^T \cdot W_k \end{cases}$

CORRECTION = $\begin{cases} \boldsymbol{\hat{q}_{k+1}} = \hat{q}_{k+1|k} + R_{k+1} \cdot \nu_{k+1} \\ \boldsymbol{P_{k+1}} = P_{k+1|k} - R_{k+1} \cdot H_{k+1} \cdot P_{k+1|k} \end{cases}$

$$\boxed{\begin{gathered} \boldsymbol{R_{k+1}} = P_{k+1|k} \cdot H^T{}_{k+1} \cdot (H_{k+1} \cdot P_{k+1|k} \cdot H^T{}_{k+1} + W_{k+1})^{-1} \\ \boldsymbol{\nu_{k+1}} = Y_{k+1} - h(\hat{q}_{k+1}) \end{gathered}}$$

1.9 Motion Planning

In Motion Planning, the problem that one wants to solve is the canonical problem where we make few assumptions.

We have a robot $\boldsymbol{\mathcal{B}}$ moving in a workspace $\boldsymbol{\mathcal{W}} = \mathbb{R}^{\boldsymbol{N}}$, with $\boldsymbol{N} = \mathbf{2}$ or $\mathbf{3}$.

We also assume that the robot $\mathcal{B}$ is free-flying in its configuration space $\boldsymbol{\mathcal{C}}$, namely it has **NO** kinematic constraints of any kind

Finally, we assume the presence of p obstacles $\boldsymbol{\mathcal{O}}$ $(\boldsymbol{\mathcal{O}_1}, \boldsymbol{\mathcal{O}_2}, \boldsymbol{\mathcal{O}_3} .. \boldsymbol{\mathcal{O}_p})$ in our workspace $\mathcal{W}$.

With this being said, given a configuration $\boldsymbol{q_s}$ and a goal configuration $\boldsymbol{q_g}$ of $\mathcal{B}$ in $\mathcal{C}$, we want to plan a path that **connects** both configurations and that is **safe** (does not cause a collision between robot and obstacles).

1.9.1 Obstacles

To formulate the problem, we must define the image of obstacles in our configuration space C.

If we define $\mathcal{B}(q)$ as the volume of $\mathcal{W}$ occupied by the robot at configuration q, then the image of obstacle $\mathcal{O}_i$ in C is:

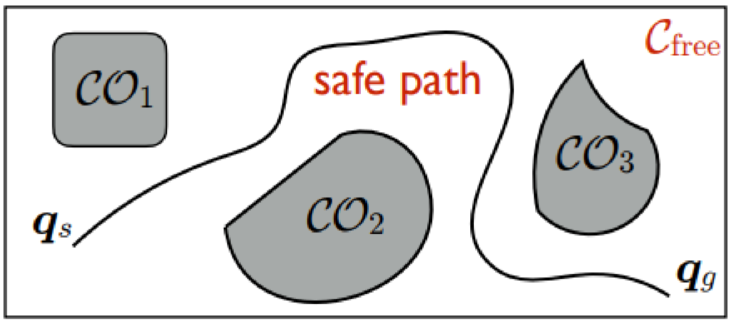
$$\boldsymbol{\mathcal{C}\mathcal{O}_i} = \{q \in \mathcal{C} : \mathcal{B}(q) \cap \mathcal{O}_i \neq \emptyset\}$$

Namely, the intersection of the volume occupied by the robot at configuration q in the workspace $\mathcal{W}$ with that i-th obstacle must be empty.

The union of the C-obstacles creates the **$\boldsymbol{C}$-obstacle region**: $\boldsymbol{\mathcal{C}\mathcal{O}} = \bigcup_{i=1}^{p} \mathcal{C}\mathcal{O}_i$

The complement is the **free configuration space**: $\mathcal{C}_{free} = \mathcal{C} - \mathcal{C}\mathcal{O}$

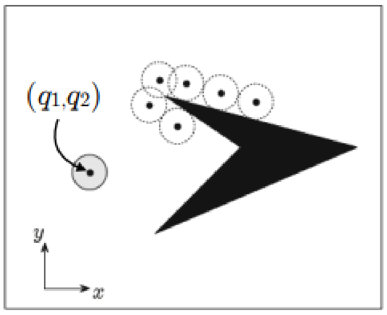
A configuration space path is, then, safe if and only if it belongs to $\mathcal{C}_{free}$, as we can see from the right image.

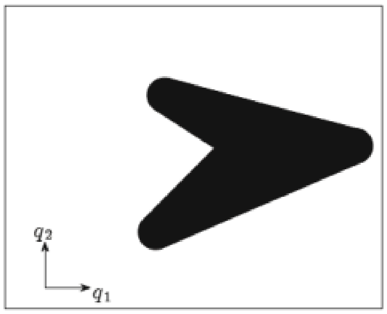


1.9.2 $\mathcal{C}$-Obstacles

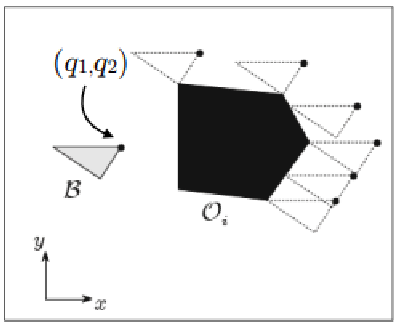
But, how exactly we define the $\mathcal{C}$-obstacles? We actually have different types, based on the kind of robot that we are dealing with:

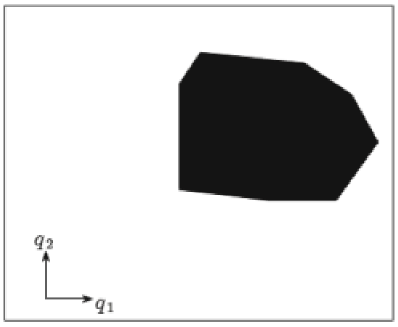
- Point robot:
    - $\mathcal{C}$-obstacles are simply copies of the original obstacles (due to the fact that q are the robot Cartesian coordinates and that $\mathcal{C}$ is a copy of $\mathcal{W}$).
- Disk robot:
    - These robots are free to translate/rotate in $\mathbb{R}^2$, so the $\mathcal{C}$-obstacles grows isotropically w.r.t the obstacle. We just basically slide the disk all over the obstacle. Watch out for vertices.



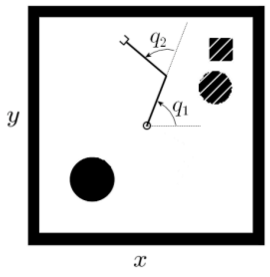


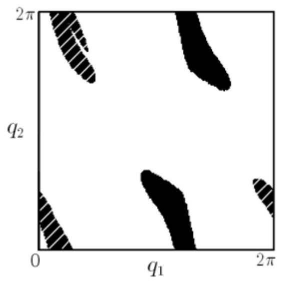
- Polygonal robot:
    - These robots have a fixed orientation and are free to translate in $\mathbb{R}^2$. The important thing to notice here is that the obstacle grows anisotropically w.r.t the obstacle, since we must choose a representative point for our robot, in order to slide it all over the obstacle:

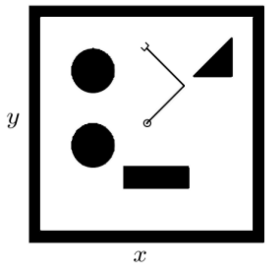


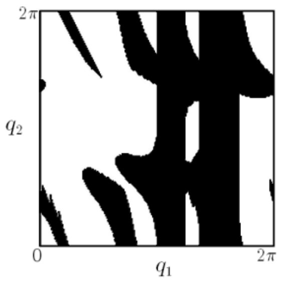


We also have examples for manipulators, where the configuration is given by joint angles, so we build the $\mathcal{C}$-obstacles by looking at the manipulator configurations which would collide with the obstacles:









In order to build an appropriate path, we have different kinds of motion planning methods, which either require an exact computation of $\mathcal{CO}$ or an approximation of it (due to the fact that building the $\mathcal{CO}$ is really expensive). They are:

- **Roadmap methods**
  - Retraction
  - Cell decomposition
    - Exact decomposition
    - Approximate decomposition
- **Probabilistic methods**
  - PRM
  - RRT
- **Artificial Potential Field**

Let's analyze them in detail.

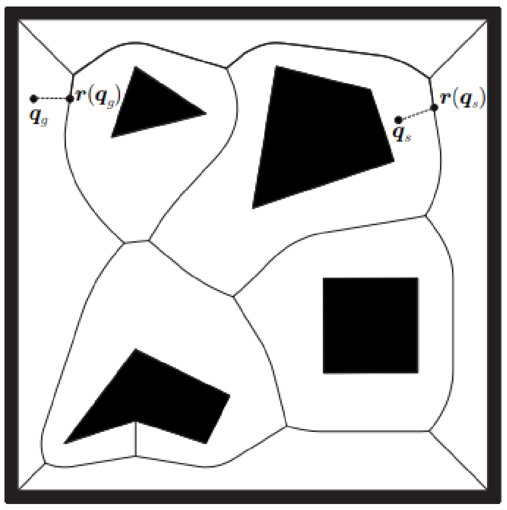
1.9.3 Roadmap methods

1.9.3.1 Retraction

In Retraction, along with the assumption that we have done for the canonical problem, we assume that $\mathcal{C} = \mathbb{R}^2$ and that the boundary of $\boldsymbol{C_{free}}$ (∂C_{free}) is a polygonal limited subset. Given these, we define the following terms:

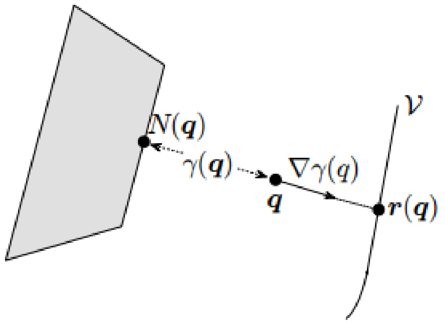
- **Clearance** of q in C_{free}:
  - It's the minimum distance between the current configuration and whatever is not in C_{free} (obstacle).
  - $\boldsymbol{\gamma(q)} = \min_{s \in \partial C_{free}} \|q - s\|$
- **Neighbors** of q:
  - It's exactly that point that comes out of the clearance.
  - $\boldsymbol{N(q)} = \{s \in \partial C_{free} : \|q - s\| = \gamma(q)\}$
- **Generalized Voronoi diagram:**
  - If we have multiple configurations q which have AT LEAST 2 neighbors, we can pass a line segment through these configurations, starting composing the Voronoi diagram.
  - $\boldsymbol{v}(\boldsymbol{C_{free}}) = \{q \in C_{free} : card(N(q)) > 1\}$

- its elementary arcs are
  - rectilinear (edge-edge, vertex-vertex)
  - parabolic (edge-vertex)
- can be seen as a graph
  - elementary arcs as arcs
  - arc endpoints as nodes
- a natural roadmap as it maximizes safety



As for the algorithm itself:

i. **Build the generalized Voronoi diagram $\nu(C_{free})$**
ii. **Compute the retractions $r(q_s)$ and $r(q_g)$.**
    a. The retractions are points in the Voronoi Diagram which connect any configuration (in this case we are interested in q_s, q_g) to the Diagram itself. They are easy to compute, as we simply need to follow the gradient of the clearance up to the intersection with the Voronoi Diagram itself. A safe path exists between $q_s - q_g$ if and only if a path exists on the Diagram between $r(q_s)$ and $r(q_g)$.

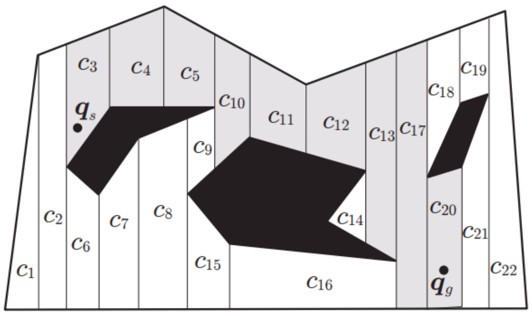


iii. **Search $\nu(C_{free})$ for a sequence of arcs such that $r(q_s)$ belongs to the first and $r(q_g)$ to the last**
iv. **If successful, return the solution path (q_s to $r(q_s)$ -> $r(q_g)$ to q_g) otherwise report a failure.**

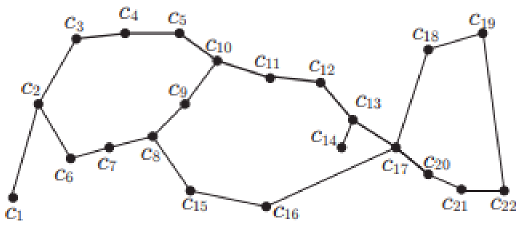
1.9.3.2 Exact decomposition

Here we also assume that $\mathcal{C} = \mathbb{R}^2$ and that the boundary of $\mathcal{C}_{free}$ ($\partial\mathcal{C}_{free}$) is a polygonal limited subset. We decompose $\mathcal{C}_{free}$ in cells (typical choice: convex polygons). Straight to the algorithm:

i. **Compute convex polygonal decomposition**
    a. This is really simple, as we have to simply use the sweep-line algorithm which states that we must pass a vertical line whenever we encounter a vertex. In the example here, we can see that C_{16} has no lines going straight to the obstacle. That's because the bottom part of the obstacle has no vertex. If it had one at the center, then the bottom part would be cut in half into C_{16} and C_{17}. It happens at C_{11} and C_{12} because the vertex is given by the world boundaries.



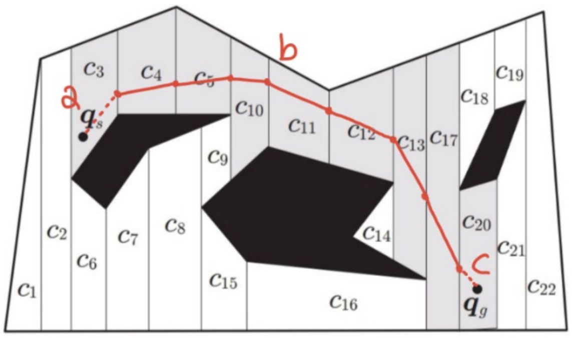
ii. **Build the associated connectivity graph C**
    a. Here, we simply transfer the cells to a graph connected by links and we identify c_s/c_g (nodes in which q_s/q_g belong).



iii. **Search $\mathcal{C}$ for a channel of cells (path) from c_s to c_g through graph search.**

iv. **If successful, extract and return a solution path**

   a. If we look at the polygonal decomposition, we would return a path starting from q_s to the segment midpoint between C_3-C_4- … -C_{17}-C_{20} and then to q_g.



1.9.3.3 Approximate decomposition

In approximate decomposition, we approximate our C_{free} using squares, where we use a recursive algorithm which reaches a trade-off between simplicity and accuracy. Straight to the algorithm:

i. **Start by diving $\mathcal{C}$ in 4 cells and classify each cell as free/occupied/mixed.**

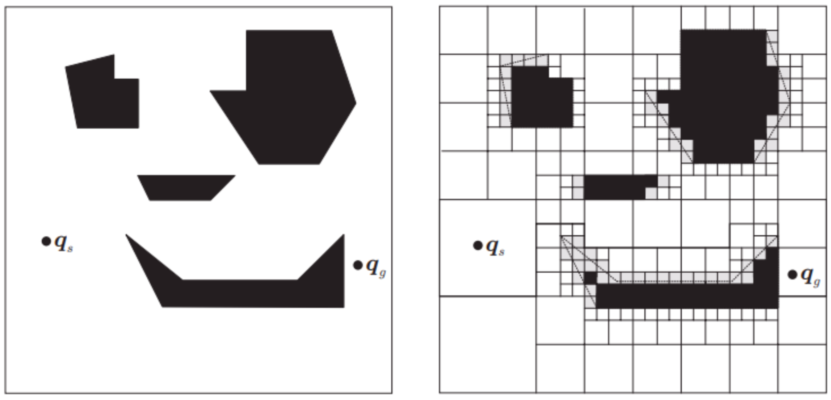
ii. **Build the associated connectivity graph $\mathcal{C}$**

   a. We must pay attention here since we have to take into consideration free and mixed cell as nodes. So mixed are included! We'll explain why later.

iii. **Search $\mathcal{C}$ for a channel of cells (path) from c_s to c_g through graph search.**

iv. **If a path exists, the mixed cells are further partitioned and the above cycle is repeated until a free channel is found.**

   a. Basically, if we find a path where there are mixed cells, we partition them and run the cycle again until we find a path composed only by free cells.



1.9.4 Probabilistic methods

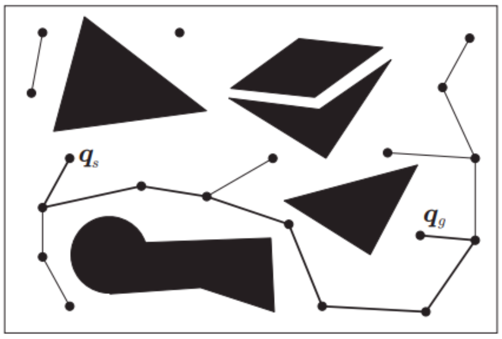
In probabilistic methods we build a roadmap of the configuration space $\mathcal{C}$ where the preliminary computation of $\mathcal{CO}$ is completely avoided, in fact an approximate representation of $\mathcal{C}_{free}$ is directly built as a collection of connected configurations (also called *roadmaps*).

1.9.4.1 PRM (Probabilistic Roadmap)

In PRM, we follow a simple algorithm:

i. Extract a sample $\boldsymbol{q}$ of $\mathcal{C}$ with Uniform probability distribution
ii. Compute $\boldsymbol{\mathcal{B}(q)}$ (which is the volume that $\mathcal{B}$ occupies in $\mathcal{W}$ at configuration q) and check for eventual collisions.
iii. If $\boldsymbol{q} \in \boldsymbol{\mathcal{C}_{free}}$, add that configuration $\boldsymbol{q}$ to PRM, otherwise discard it.
iv. Search the PRM for “sufficiently near” configurations called $\boldsymbol{q_{near}}$.
v. If possible, connect $\boldsymbol{q}$ and $\boldsymbol{q_{near}}$ with a free local path.

This connection is performed by the local planner (throw a linear path and check for collision).



Construction of the PRM is arrested when a max. number of iterations is reached.

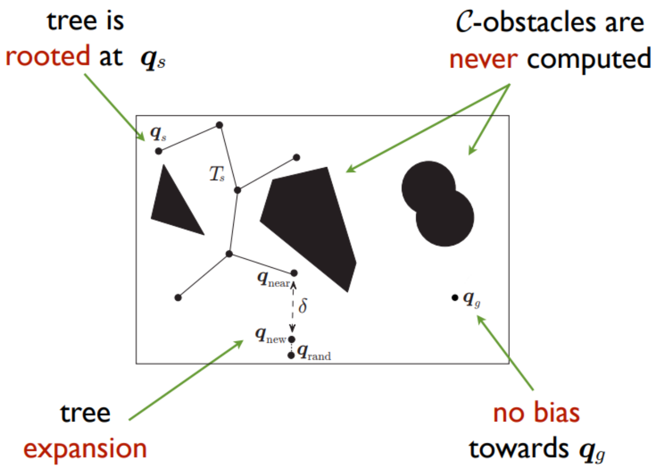
Also, if q_s and q_g can be connected to the same component, a solution can be found by graph search.

1.9.4.2 RRT (Rapidly-exploring Random Tree)

In order to build the tree T_s rooted at q_s:

i. Generate q_{rand} in $\mathcal{C}$ with Uniform probability distribution
ii. Search the tree for the nearest configuration q_{near}
iii. Choose q_{new} at a distance δ from q_{near} in the direction of q_{rand}
iv. Check q_{new} and the segment from q_{near} to q_{new} for any collision
v. If no collisions are found, add q_{new} to the tree T_s (expansion)

One important thing to notice is that T_s rapidly covers $\mathcal{C}_{free}$ because the expansion is biased towards unexplored areas (actually towards larger Voronoi regions).



When, instead, we deal with an RRT in an empty 2D space, we might introduce a bias towards q_g by growing two trees T_s and T_g respectively rooted at q_s and q_g (**Bidirectional RRT**). At the end they will meet. This is probabilistically complete and single-query. We can also speed up motion in wide open areas with greedy exploration.

- Classical expansion → q_{rand} = $random(q)$
- Greedy behaviour → q_{rand} = q_{goal}

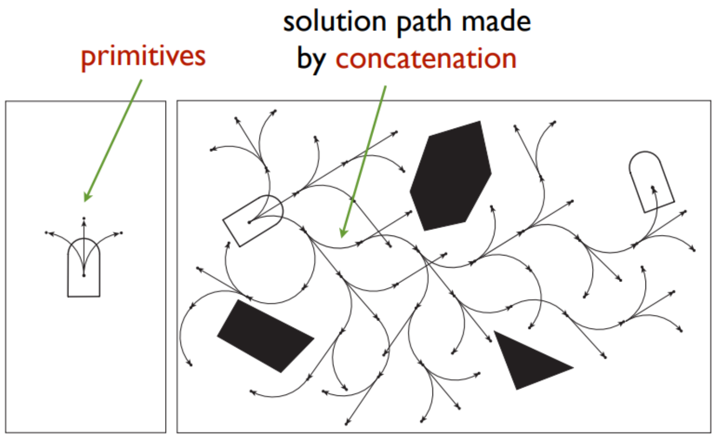
If we apply RRT to nonholonomic robots, one might think of the unicycle in $\mathcal{C} = \mathbb{R}^2 \times SO(2)$, where we know that:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos\theta \\ sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega$$

In this case, we might use the Dubins car, which results in 3 possible paths in forward motion:

$$v = \bar{v} \qquad\qquad \omega = \{-\bar{\omega}, 0, \bar{\omega}\}$$

The algorithm here is the same with the only difference that q_{new} is generated from q_{near} selecting **one of the possible paths that the unicycle can take**. If q_g can be reached from q_s with a collision-free concatenation of primitives, the probability that a solution is found tends to 1.



1.9.5 Complexity

METHOD	COMPLEXITY	COMPLETENESS	QUERY	REASON
Retraction	O(vlogv)	Yes	Multiple	We just need to build the Generalized Voronoi Diagram once
Exact Decomposition	O(vlogv)	Yes	Multiple	Build the connectivity graph once for all
Approx. Decomposition	Exponential in C	Resolution complete*	Single	Decomposition guided by $\boldsymbol{q_s, q_g}$
PRM	Fast	Probabilistically complete*	Multiple	New queries enhance the PRM
RRT	Quick (Also could use Greedy exploration)	Probabilistically complete*	Single	Trees are rooted at q_s and q_g.

*Resolution complete = a solution is found if and only if one exists at the maximum allowed resolution.

*Probabilistically complete = the probability of finding a solution whenever one exists tends to 1 as the execution time tends to ∞.

1.9.6 Artificial Potential Fields

We know that autonomous robots must be able to plan on-line (meaning that they must travel using partial workspace information). In APF, we are directly working on this, following a stimulus-response paradigm.

In fact, the idea is to build potential fields in C so that the point that represents the robot is attracted by the goal q_g and repelled by the C-obstacle region CO.

The total potential U is the sum of attractive/repulsive potential, whose negative gradient $-\nabla U(q)$ indicates the most promising direction of motion.

1.9.6.1 Attractive Potential

In attractive potential, the objective is to guide the robot to the goal q_g.

We have two configurations to keep in mind here:

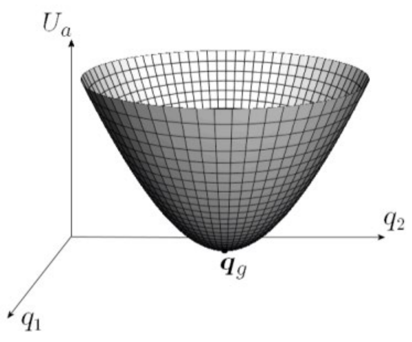
i. **Paraboloidal**:

$$U_{a_1}(q) = \frac{1}{2} \cdot k_a \cdot \|e(q)\|^2 = \frac{1}{2} \cdot k_a \cdot (e_1^2 + \cdots + e_n^2)$$

where $e = q_g - q$.

In this case, the force related to this potential is clearly the negative gradient:

$$f_{a_1} = -\nabla U_{a_1} = -\begin{pmatrix} \frac{\partial U_{a_1}}{\partial q_1} \\ \vdots \\ \frac{\partial U_{a_1}}{\partial q_n} \end{pmatrix}$$



If we want to calculate one element i from this gradient, it would be:

$$\frac{\partial U_{a_1}}{\partial q_i} = -\frac{1}{2} \cdot k_a \cdot 2e_i = -k_a \cdot e_i$$

An important thing to notice is that the only element which depends on q is exactly the i-th element from the e parenthesis, since $e_i = q_{g_i} - q_i$, so we are only interested in that element.

Finally, $-\nabla U_{a_1} = -\begin{pmatrix} -k_a \cdot e_1 \\ .. \\ -k_a \cdot e_n \end{pmatrix} = k_a \cdot \begin{pmatrix} e_1 \\ .. \\ e_n \end{pmatrix}$

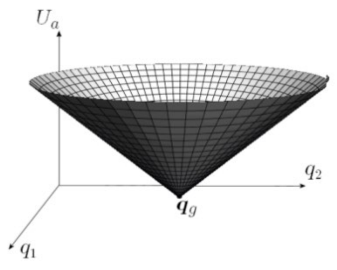
ii. **Conical**:

In a similar manner, we can say that the total potential is given by:

$U_{a_2}(q) = k_a \|e(q)\|$

And that the resulting attractive force is constant, equal to:

$$f_{a_2}(q) = -U_{a_2}(q) = k_a \cdot \frac{e(q)}{\|e(q)\|}$$
$$= \frac{k_a}{\|e(q)\|} \cdot \begin{pmatrix} e_1 \\ .. \\ e_n \end{pmatrix}$$



In general, the only difference is the magnitude of the force.

- *Conical* = The gradient is constant in conical potential. Good if we are very far from goal, but bad if we are near (chattering).
- *Paraboloid* = The gradient is proportional to the distance from goal. Nice convergence behavior since force gets gentler towards the goal.

We can also think of combining the two: When we are away, we use conical. When we are close to q_g, use paraboloidal. We must take into consideration continuity of f in order to have a smooth transition between the two settings.

1.9.6.2 Repulsive Potential

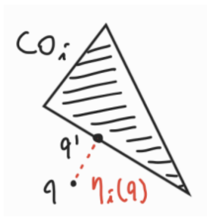
In the repulsive potential, what we do is to keep the robot away from the obstacles (CO).

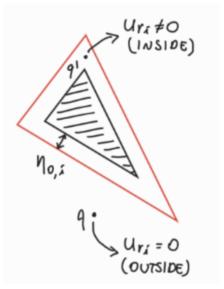
Dividing the CO in partitions, we can calculate the repulsive field for each CO_i as follows:

$$U_{r,i}(q) = \begin{cases} \frac{k_{r,i}}{\gamma} \cdot \left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}}\right)^{\gamma}, & if\ \eta_i(q) \le \eta_{0,i} \\ 0, & if\ \eta_i(q) > \eta_{0,i} \end{cases}$$

Where:

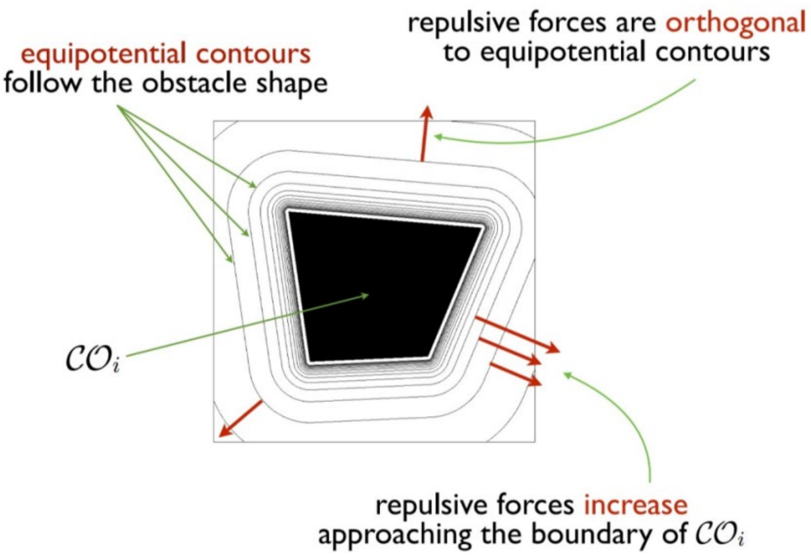
- $k_{r,i} > 0$
- $\gamma = 2, 3, \ldots$
- $\eta_{0,i}$ = Range of influence of CO_i
- $\eta_i(q)$ = Clearance w.r.t. CO_i





It is important to note that this repulsive field has only a local action, indeed outside $\eta_{0,i}$, the robot does not perceive it.

The potential which forces away the robot, follows the geometry of the obstacle as seen in the image below.


equipotential contours follow the obstacle shape
repulsive forces are orthogonal to equipotential contours
$\mathcal{CO}_i$
repulsive forces increase approaching the boundary of $\mathcal{CO}_i$

1.9.6.3 Repulsive Force

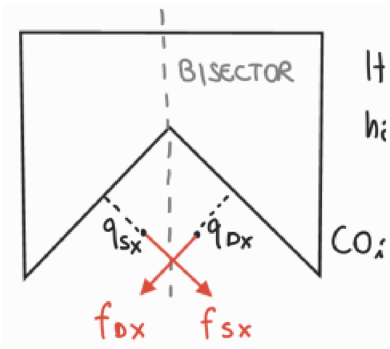
We can simply compute the repulsive force associated with $U_{r,i}(q)$ as follows:

$$f_{r,i}(q) = -\nabla U_{r,i}(q) = \begin{cases} \frac{k_{r,i}}{\eta_i^2(q)} \cdot \left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}}\right)^{\gamma-1} \cdot \nabla_q \eta_i(q), \ if\ \eta_i(q) \le \eta_{0,i} \\ 0, \ if\ \eta_i(q) > \eta_{0,i} \end{cases}$$

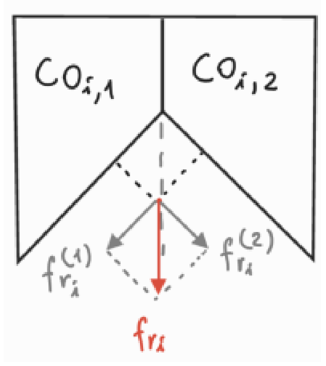
The total repulsive potential then, is given by the sum of the potential $U_{r,i}$ associated to each CO_i:

$$U_r(q) = \sum_{i=1}^{p} U_{r,i}(q) \ \mid p = N. of\ obstacles$$

It is crucial for us to assume that objects are convex and that we have then a convex decomposition of the C-obstacle region. In fact, let's suppose we have a no-convex obstacle and that we want to compute the force at its bisectrix:



We can already see that here there is a problem, since the robot will get a push from right to left. If we now separate this obstacle in two obstacles $CO_{i,1}$ and $CO_{i,2}$, then the repulsive force will be given by the sum of the two forces $f_{r,i}^{(1)}, f_{r,i}^{(2)}$ as shown below:



1.9.6.4 Total potential and local minimum / saddle point

By summing up the attractive and repulsive potential we said at the very beginning, we'll obtain the total potential and then the total force field:

SUPERPOSITION: $U_t(q) = U_a(q) + U_r(q)$

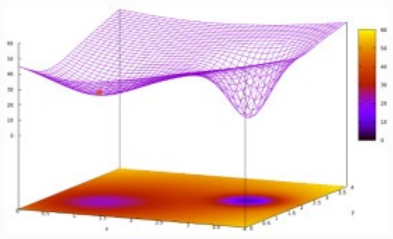
FORCE FIELD: $f_t(q) = -\nabla_q U_t(q) = f_a(q) + \sum_{i=1}^{p} f_{r,i}(q)$

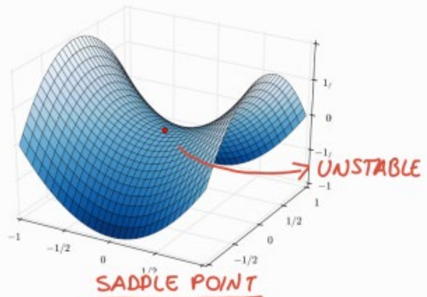
TOTAL POTENTIAL FIELD

_ Local minimum

LOCAL MINIMUM are a problem, since the robot is driven by $-\nabla_q U_t$, at a local minimum it will stop although it does NOT reach the goal.

Note that a local minimum is different wrt a saddle point, indeed this last is a minimum in one direction and a maximum in the other one.


LOCAL MINIMUM



SADDLE POINT

1.9.6.5 Planning techniques

We might use 3 different techniques for planning, based on f_t.

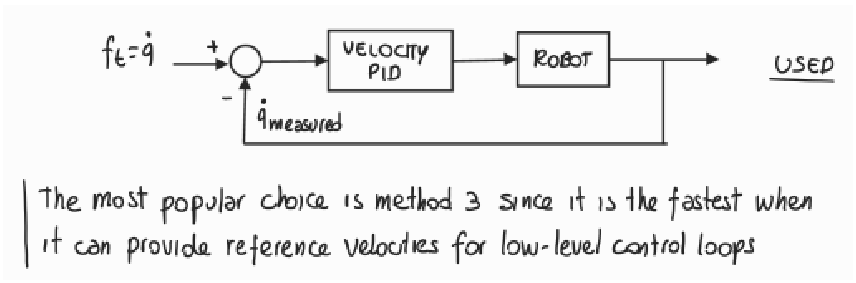
First, though, let's recall the Robot Dynamics formula (general one):

$$M(q) \cdot \ddot{q} + \eta(q, \dot{q}) = \tau$$

1. Consider f_t as generalized forces: $\tau = f_t(q)$.
   In this case, the effect on the robot is **filtered by its dynamics** (we isolate $\ddot{q}$) and it's **slow** due to the *double integrator process*. Gives really natural/smooth motion to the robot, though.
2. Consider f_t as generalized accelerations: $\ddot{q} = f_t(q)$.
   In this case, the effect on the robot is **independent on its dynamics** and it's **slow** due to the *double integrator process*, again. This gives intermediate results.
3. Consider f_t as generalized velocities: $\dot{q} = f_t(q)$.
   In this case, the effect on the robot is **independent on its dynamics** and it's **fast** due to a *single integrator*. This is the most reactive motion, due to fast reaction and due to guarantee of asymptotic stability of q_g.

In offline planning, one might want to use the **3rd case**, integrating the generalized coordinates through Euler integration: $q_{k+1} = q_k + f_t \cdot T$, where $f_t = -\nabla U_t$ (Maximum descent). If using 1, we must integrate the dynamic model, if using 2 we must integrate the generalized accelerations.

In online planning, instead, if we use the 1st technique, we are acting directly on control inputs, technique 2 acts through inverse dynamics and **technique 3** provides reference velocities for low-level control loops.



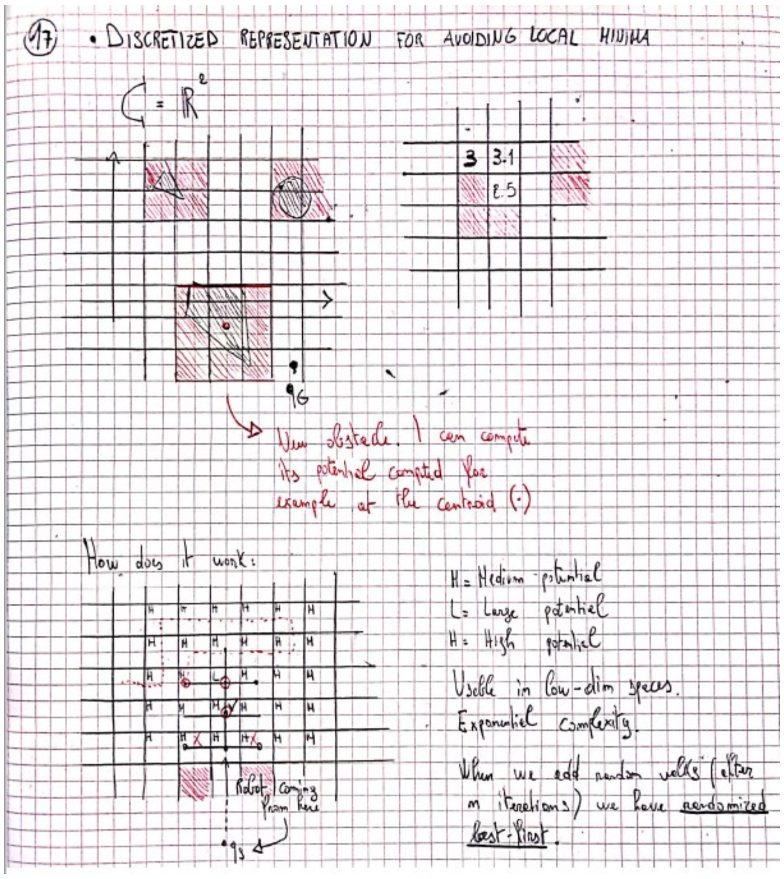
The most popular choice is method 3 since it is the fastest when it can provide reference velocities for low-level control loops

1.9.6.6 Addressing local minima issue

As we have seen before, if a robot enters in a local minimum, we might have a problem, while saddle points are not a problem.

For this reason, we say that motion planning based on artificial potential fields is not complete, but some workaround exists (mainly used in online motion planning):

1. **Best-first algorithm:**
   Here, we build a discretized representation of C_{free} using a regular grid, and associate to each free cell of the grid the value of U_t at its centroid. After this, we build a tree T rooted at q_s where in each iteration, we select the leaf with the minimum value of U_t and select as children its adjacent free cells that are not in T. This results in success if q_g is reached, or in failure if no further cells can be added to T.
   This solution is resolution complete and at a local minimum, the algorithm will find a way out. It's complexity, though, it's exponential in C, so it's only applicable in low-dimensional spaces.



• DISCRETIZED REPRESENTATION FOR AVOIDING LOCAL MINIMA

New obstacle. I can compute its potential computed for example at the centroid (·)

How does it work:

H = Medium potential
L = Large potential
H = High potential

Usable in low-dim spaces.
Exponential complexity.

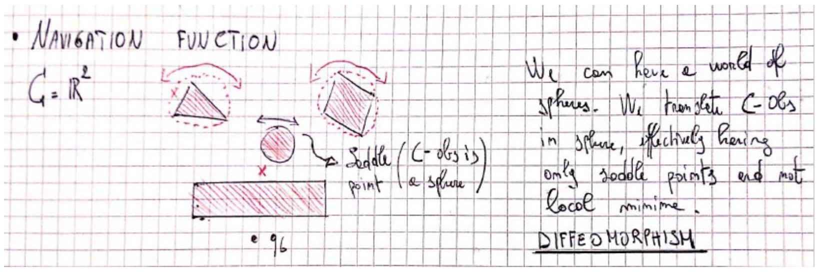
When we add random walks (after m iterations) we have randomized best-first.

2. Navigation functions:

In the best-first approach, we can escape from a local minimum, but we do not avoid it in principle. This is not the case for navigation functions, where we basically use diffeomorphism in order to translate C-obstacles in spheres, effectively having only saddle points and not local minima. This technique, though, is only suitable in off-line planning as it requires complete knowledge of the environment.

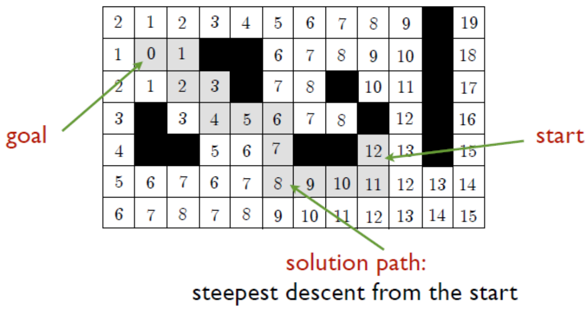
Once we have done this, we simply represent C_{free} as a grid map, assigning:

- 0: Goal cell
- 1: Cells adjacent to the 0-cell
- 2: Unvisited cells adjacent to 1-cell
- etc.



We can have a world of spheres. We translate C-obs in sphere, effectively having only saddle points and not local minima.

DIFFEOMORPHISM


goal
start
solution path:
steepest descent from the start

2	1	2	3	4	5	6	7	8	9	■	19
1	0	1	■	■	6	7	8	9	10	■	18
2	1	2	3	■	7	8	■	10	11	■	17
3	■	3	4	5	6	7	8	■	12	■	16
4	■	■	5	6	7	■	■	12	13	■	15
5	6	7	6	7	8	9	10	11	12	13	14
6	7	8	7	8	9	10	11	12	13	14	15

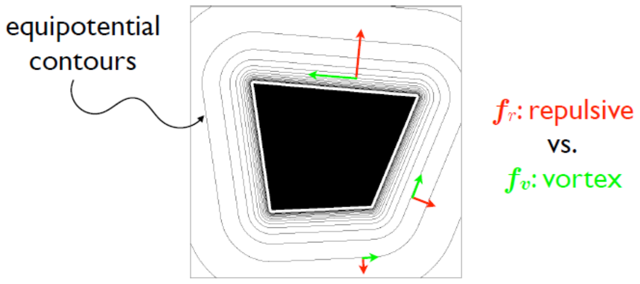
3. Vortex fields:

In Vortex field, we directly assign a force field (rather than a potential). In this way we replace the repulsive action with an action forcing the robot to go around the C-obstacle.

The Vortex field for $\mathcal{CO}_i$ is:

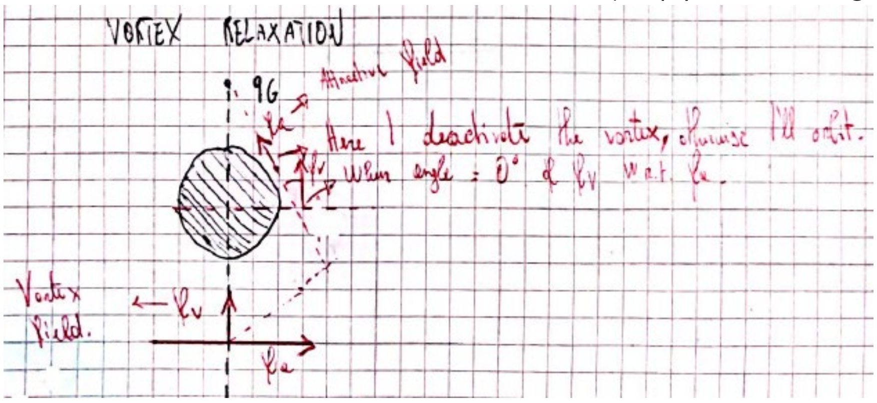
$$f_v = \pm \begin{pmatrix} \dfrac{\partial U_{r,i}}{\partial y} \\ -\dfrac{\partial U_{r,i}}{\partial x} \end{pmatrix}$$

Where we basically take the tangent w.r.t the equipotential countours.



• the intensity of the two fields is the same, only the direction changes

The vortex sense (hence the $\pm$) should be chosen depending on the entrance point of the robot w.r.t the range of influence of C-obstacle. One important aspect to consider is vortex relaxation, where we deactivate the vortex if certain conditions are met (simply when the angle of f_v is 0° w.r.t f_a), otherwise we'll be orbiting around the obstacle following the equipotential countour.


VORTEX RELAXATION
• 96
Attractive field
f_a
f_v
Here I deactivate the vortex, otherwise I'll orbit.
When angle = 0° of f_v w.r.t. f_a.
Vortex field.
f_v
f_a

1.9.6.7 Artificial Potentials for WMR

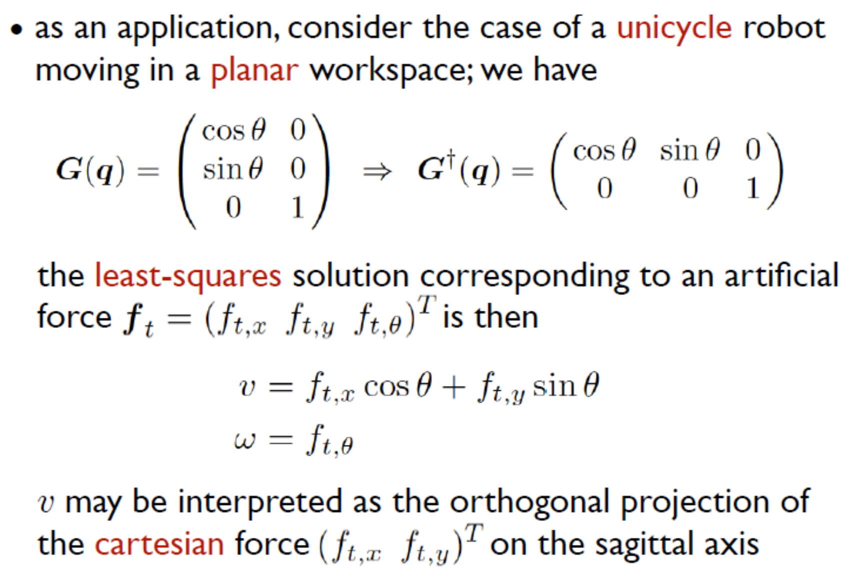
We know there are robots subject to nonholonomic constraints which violate the free-flying assumption we've done at the beginning, thus the artificial force f_t **can't** be directly imposed as a generalized velocity $\dot{q}$.

We know that the kinematic model of a WMR is in this form:

$\dot{q} = G(q) \cdot u$, but in this case, we would end up in an overdetermined system where we have more equations than unknown (considering that equation as $b = A \cdot x$)

In order to solve this, we can use a least-squares solution where we take into consideration the Pseudo-Inverse of G, as follows:

$$u = G^{\dagger}(q) \cdot \dot{q}_{des} = G^{\dagger}(q) \cdot f_t$$



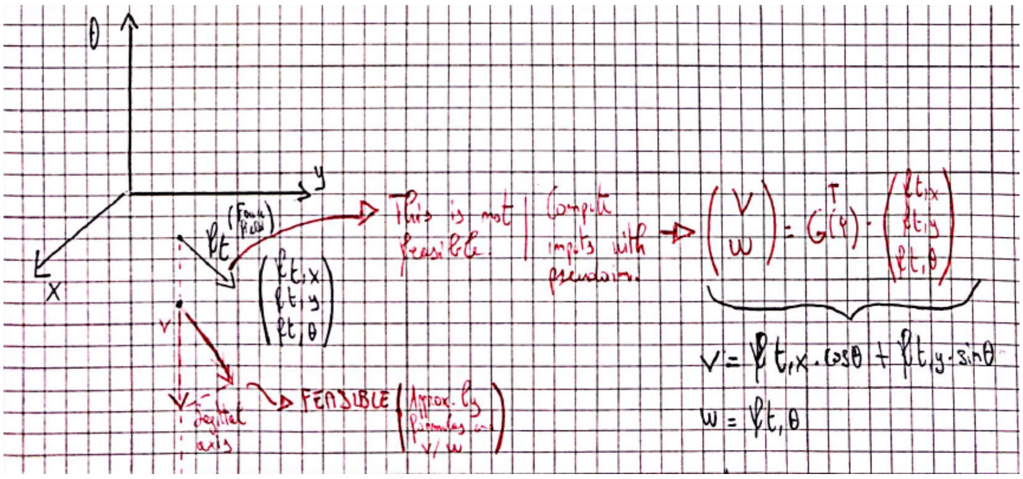
- as an application, consider the case of a unicycle robot moving in a planar workspace; we have

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \boldsymbol{G}^{\dagger}(\boldsymbol{q}) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

the least-squares solution corresponding to an artificial force $\boldsymbol{f}_t = (f_{t,x} \;\; f_{t,y} \;\; f_{t,\theta})^T$ is then

$$v = f_{t,x}\cos\theta + f_{t,y}\sin\theta$$
$$\omega = f_{t,\theta}$$

v may be interpreted as the orthogonal projection of the cartesian force $(f_{t,x} \;\; f_{t,y})^T$ on the sagittal axis



θ
y
X
ft
(Total field)
$\begin{pmatrix} f_{t,x} \\ f_{t,y} \\ f_{t,\theta} \end{pmatrix}$
This is not feasible!
Compute inputs with pseudoinv.
$\underbrace{\begin{pmatrix} v \\ \omega \end{pmatrix} = G^T(q) \cdot \begin{pmatrix} f_{t,x} \\ f_{t,y} \\ f_{t,\theta} \end{pmatrix}}$
$v = f_{t,x} \cdot \cos\theta + f_{t,y} \cdot \sin\theta$
$\omega = f_{t,\theta}$
v
V
sagittal axis
FEASIBLE
(Approx. by formulas on v/w)

Also, if we assume that the unicycle robot has a circular shape, so that its orientation is irrelevant for collision and that the obstacles are polygonal, then we can build artificial potentials in a reduced $C = C' = \mathbb{R}^2$ with C'-obstacles simply obtained by growing the workspace obstacles by the robot radius.

Since in C', we do not take into consideration the orientation, the total field will not include a component $f_{t,\theta}$, hence:

$$\omega = f_{t,\theta} = k_\theta \cdot (ATAN2(f_{t,y}, f_{t,x}) - \theta)$$

If we are close to the goal, then $f_t = f_a$ and the controls become:

$$v = -k_a(x\cos\theta + y\sin\theta)$$
$$\omega = k_\theta\,(\text{Atan2}(-y, -x) - \theta)$$

1.9.6.8 Artificial Potentials for manipulators

The complexity of motion planning is high, due to the configuration space having $n \geq 4$.

The idea is to reduce the dimensionality (for example in 6-DoF robots, we can replace the wrist with a "ball" denoting the total volume it can sweep).

For offline planning, one might want to use probabilistic methods

For online planning, one wants to adapt artificial potential fields

In the latter, one chooses a set of control points $p_1, \dots, p_p$ distributed on the robot body Generally:

- **One point per link**: $p_1, \dots, p_{p-1}$
- **End-effector**: p_p

In this case, the attractive potential U_a acts only on the EE (P_p), while the repulsive potential U_r acts on the whole set on control points (including the EE = P_p).

For this reason, the p_p is subject to the total potential: $U_t = U_a + U_r$

In the case of planning with control points, we can either **impose** to the robot joints the **generalized forces** resulting from the combined force fields:

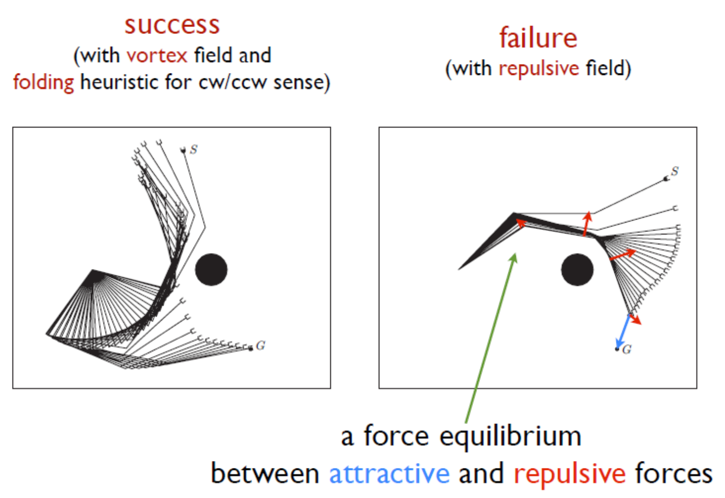
$$\tau = -\sum_{i=1}^{P-1} \mathcal{J}_i^T(q) \cdot \nabla U_r(p_i) - \mathcal{J}_p^T(q) \cdot \nabla U_t(p_P)$$

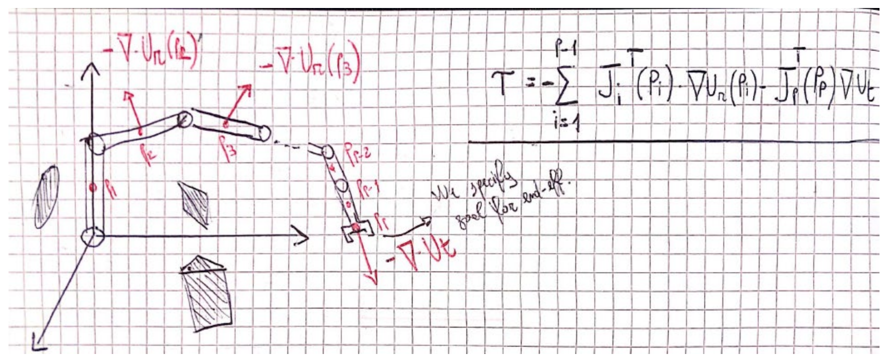
Where $\mathcal{J}_i(q), i = 1, \dots P$ is the Jacobian matrix of direct Kinematics associated to $p_i(q)$.

Or, we can use the above formula directly as **reference velocities** $\dot{q}$ to be **fed** to the low-level control loops.

In the first case, we generate smoother movements but in a slower manner, while in the second case, the computation is way faster (actually being a gradient-based minimization step in $\mathcal{C}$).

We could encounter, though, force equilibria, where the various forces balance each other even if the total potential U_t is not at a local minimum. For this reason, we must use both methods with some workarounds in order to avoid this problem.

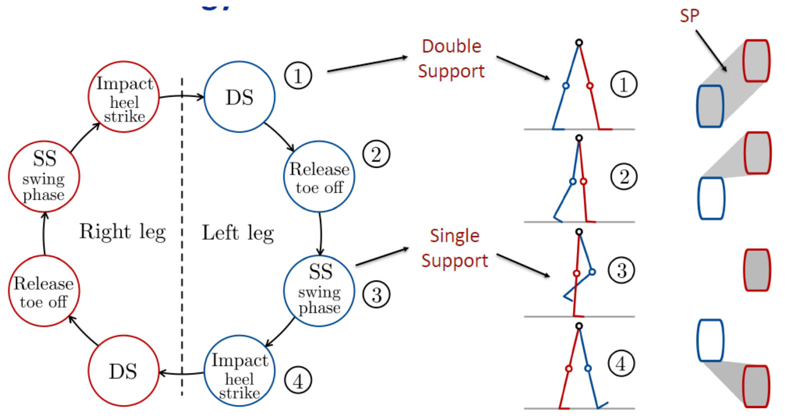


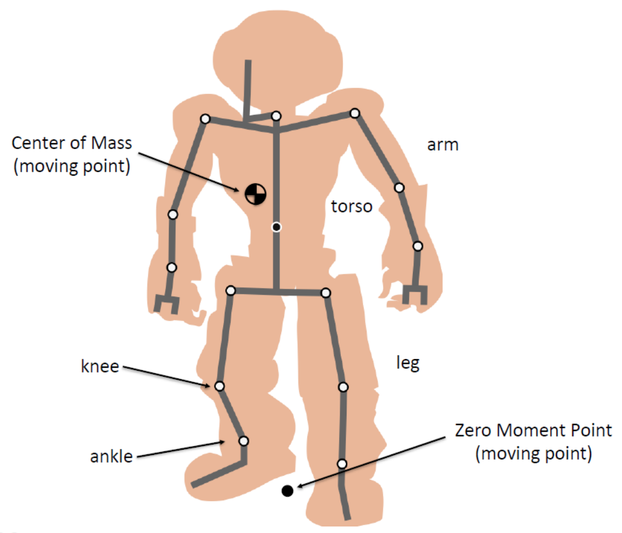


$$T = -\sum_{i=1}^{p-1} J_i^T(P_i)\,\nabla U_r(P_i) - J_p^T(P_p)\,\nabla U_t$$

1.10 Humanoids

Humanoids are robots that can adapt and collaborate in environments suitable for humans, having also a wide range of tasks: Sensing, manipulation of objects, locomotion etc.




Center of Mass (moving point)
arm
torso
knee
leg
ankle
Zero Moment Point (moving point)

- walking: cyclic alternation of 4 phases
- Support Polygon (SP): convex hull of the contact points
- robots with flat feet have only Single and Double Support phases

1.10.1 Static/Dynamic Gaits

The projection of the Center of Mass on the ground is **always** inside the Support Polygon, however **static gaits** are very **slow** and conservative (static walk).
As for the **ZMP** (Zero Moment Point), it is a point on the ground where the resultant of the reaction forces acts.
When we say that the ZMP is always inside the Support Polygon, we are talking about a **dynamically stable gait** (smoother walk).

1.10.2 Dynamic Modeling

In order to describe the configuration of a Robot, we say that it is equal to:

$$C = SE(3) \times \mathbb{R}^n$$

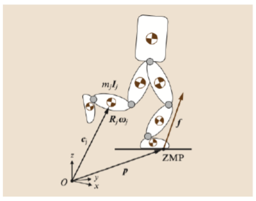
In total, we have $N + 6$ generalized coordinates for a robot with N joints., since we are representing the position and orientation in 3D space and the internal configuration as joint angles.

Actually, we know from theory, that joint angles are referred as $SO(2)^N$, so we might rewrite the whole configuration of the Humanoid to be:

$$C = SE(3) \times SO(2)^N$$

$$\boldsymbol{q} = (\boldsymbol{x}_0, \boldsymbol{\theta}_0, \hat{\boldsymbol{q}})$$

- here, $\boldsymbol{x}_0$ and $\boldsymbol{\theta}_0$ represent the position and orientation of a robot link called the **floating base**
- any link can be chosen, although it is more common to chose the torso or one of the feet



- $\hat{\boldsymbol{q}}$ is the vector of joint angles: if the floating base is fixed, a humanoid robot is exactly described like a manipulator

With this being said, we can write the **Lagrange equations** to derive a Dynamic model for the robot:

$$B(q) \cdot \ddot{q} + N(q, \dot{q}) = Q \cdot \tau + \sum_i \mathcal{J}_i^T \cdot f_i$$

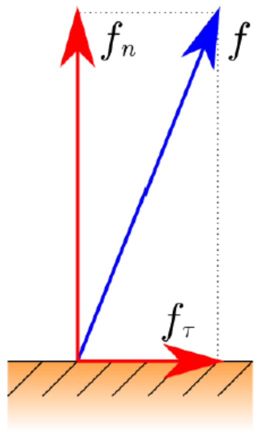
i. $\boldsymbol{B}$ = Inertia matrix
ii. $\boldsymbol{N}$ = Coriolis + Centrifugal + Gravity terms
iii. $\boldsymbol{Q}$ = Maps joint torques to generalized coordinates
iv. $\boldsymbol{\mathcal{J}_i}$ = Jacobian of the i-th contact point
v. $\boldsymbol{f_i}$ = Is the i-th external force (applied forces, like pushing, or reaction force, like reacting to ground surface)

1.10.3 Contacts

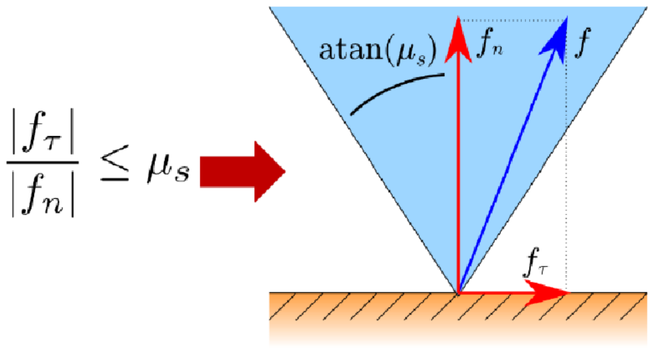
Reaction contact forces are really important in dynamic modeling because we need them in order to move the robot in the environment.

These forces are composed by two components:

- Normal forces (f_n) = Express the action of a non-holonomic constraint
- Tangential forces (f_τ) = Express the action of a friction which can be either static or dynamic depending on whether there is sliding.



One might need the **Coulomb friction model** which expresses the total contact force being inside a "friction cone" with its apex at the contact point.



1.10.4 Newton-Euler equations

In order to simplify our problem, one might consider a reduced-model which takes into account Newton-Euler equations on the entire robot.

The first Newton-Euler equation acts on the **force balance** and is defined as follows:

$$M\ddot{c} = \sum_i f_i - Mg$$

Where c is the Center of Mass position and M is the total mass of the system.

This equation says that we need the contact forces to move the CoM in a direction different from that of gravity (remember, we are in a dynamic system, not static).

In the second case, we defined the **moment balance** as moments computed with respect to a generic point o:

$$(c - o) \times M\ddot{c} + \dot{L} = M(c - o) \times g + \sum_i (p_i - o) \times f_i$$

p_i = Position of the contact point of force f_i, while L is the angular momentum (*actually, sum of the angular momentum of each robot link*) of the robot with respect to its Center of Mass.

Also, Newton-Euler equations are used when we want to derive a relation between CoM and ZMP.

1.10.5 Zero Moment Point (ZMP)

If we take into consideration the **moment balance** equation above and choose a generic point o, such that:

$$\sum_i (p_i - o) \times f_i = 0$$

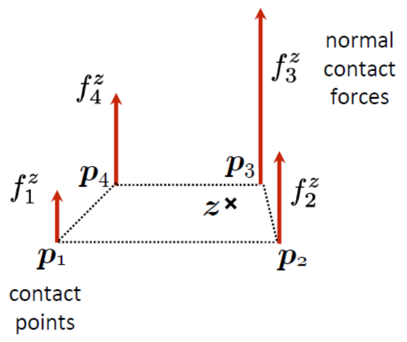
Then, the equation would become:

$$M(c - \boldsymbol{z}) \times (\ddot{c} + g) + \dot{L} = 0$$

Basically, we consider z as the **ZMP** which represents that point where the moment of the contact force is exactly zero.

We can consider the ZMP through a graphical notion as follows:

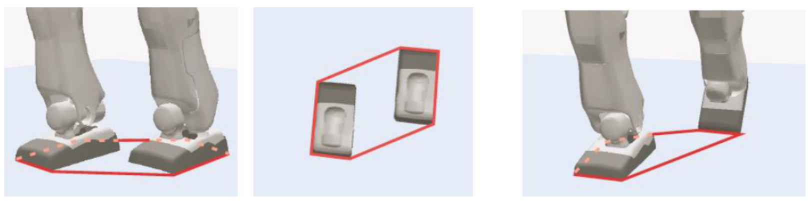
$$\boldsymbol{z} = \frac{\sum_i \boldsymbol{p}_i f_i^z}{\sum_i f_i^z}$$



In this case, $p_1 \dots p_4$ are points all on the same plane.

If we consider the left equation, we see that the ZMP is the sum of the ratio between the position vector of each contact point weighted by the normal contact forces. Namely, the sum of all these ratios sum up to 1, due to the fact that each denominator is a value < 1.

- the region where the ZMP can be is the region of all points that can be expressed as a **convex combination** of the contact points
- this region is called the **support polygon**: it is the **convex hull** of the contact surfaces between robot and ground



If the ZMP is inside the Support polygon, then we are sure about the fact that the **foot** is **well planted** on the ground.

Also, <u>we have the condition of unilaterality of contact forces</u>.

1.10.6 Approximate LIP model & Balance

Let's consider the Linear Inverted Pendulum (LIP) model:

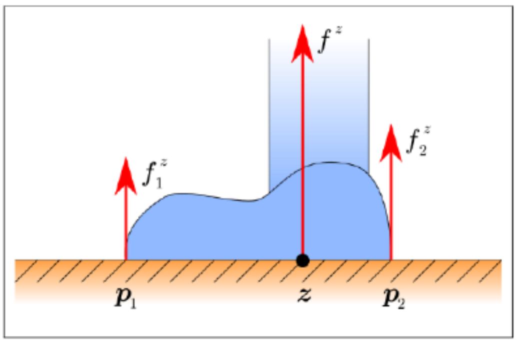
$$\ddot{\boldsymbol{c}}^{x,y} = \frac{\boldsymbol{g}^z}{\boldsymbol{c}^z} \cdot (\boldsymbol{c}^{x,y} - \boldsymbol{z}^{x,y})$$

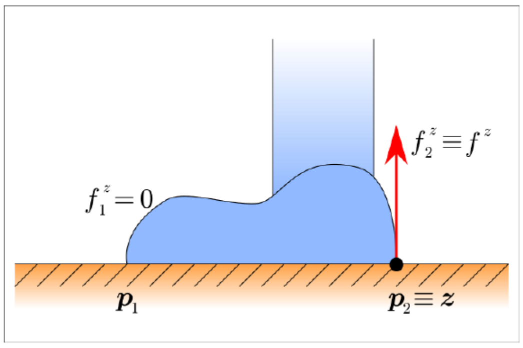
Although this is simplified, the LIP equation describes an approximation of the time evolution of the CoM trajectory, which we are interested in. Its scope is to represent a differential relationship between the CoM trajectory and ZMP time evolution.

One might want a suitable ZMP trajectory such that it satisfies the dynamic balance we are about to see next. Also, the associated CoM trajectory can be tracked with a complete robot model.

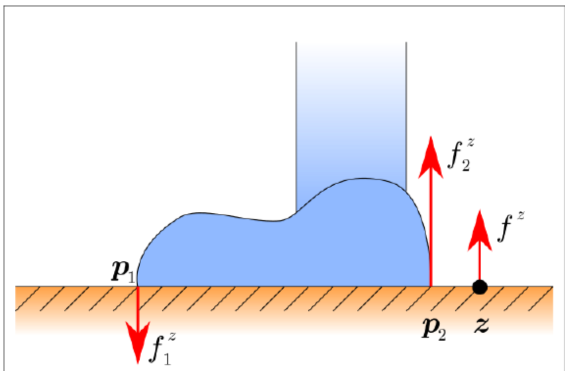
A note on the ZMP:

If we have two forces applied at point p_1, p_2, then the ZMP is somewhere in between them shifted towards the larger of the two forces (Left image). If we reduce the magnitude of one force (f_1) until it's equal to zero, then the ZMP exactly coincides with the second point p_2 (Right image).

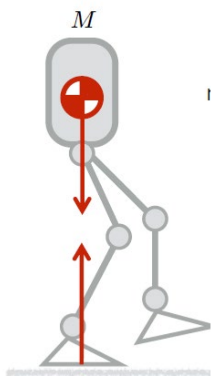




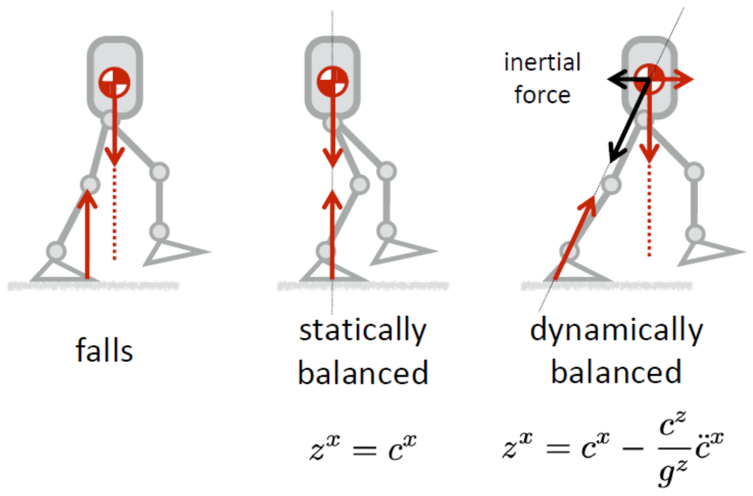
If we keep on moving the ZMP to the right, it would end up outside the support polygon, requiring a negative force f_1 which is impossible due to the unilateral condition explained in the previous sub-chapter. The force can only push on the robot and not pull! (See iamge on the right).



As we have already seen, if we consider the static balance, the robot will keep the center of mass within the support polygon (actually ZMP) in order to maintain postural stability (it's very slow, though – Right picture).



Things get different when we consider a dynamic balance. Actually, it's the same as the static one, but we add an inertial force (if we consider an accelerating frame and a pendulum, this one starts to swing as it the frame moves. If the frame is non-inertial, the pendulum stands still). We can transfer this notion in a humanoid as well, considering that, as it moves, we have motion that requires the exchange of horizontal friction force with the ground.



Where c is the Center of Mass at constant height and z is the ZMP.

2. Formulary

2.1 Pure Rolling Constraint

$$sin\theta \cdot \dot{x} - cos\theta \cdot \dot{y} = 0$$

2.2 Rigid body Constraint

If we consider the front wheel drive of the bicycle, we have to rewrite the front wheel pure rolling constraint:

$$sin\theta \cdot \dot{x}_f - cos\theta \cdot \dot{y}_f = 0$$

w.r.t to the parameters in q.

For this, we deploy the rigid body constraint:

$$x_f = x + l \cdot cos\theta \mid \dot{x}_f = \dot{x} - l \cdot sin\theta \cdot \dot{\theta}$$
$$y_f = y + l \cdot sin\theta \mid \dot{y}_f = \dot{y} + l \cdot cos\theta \cdot \dot{\theta}$$

2.3 Unicycle

2.3.1 Kinematic model

- $\begin{cases} \dot{x} = v \cdot cos\theta \\ \dot{y} = v \cdot sin\theta \\ \dot{\theta} = \omega \end{cases}$, where $v = \pm\sqrt{\dot{x}^2 + \dot{y}^2}$

2.3.2 Reconstruction formulas

- $\theta = ATAN2(\dot{y}, \dot{x}) + K \cdot \pi$
- $v = \pm\sqrt{\dot{x}^2 + \dot{y}^2}$

$$\omega = \frac{\ddot{y}\dot{x} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}$$

2.3.3 Reconstruction formulas (Motion Planning)

- $v = f_{t.x} \cdot cos\theta + f_{t.y} \cdot \sin\theta$
- $\omega = f_{t,\theta}$

  If we do not care about the orientation, working only in $\mathbb{R}^2$, then ω becomes:
- $\omega = f_{t,\theta} = k_\theta \cdot (ATAN2(f_{t,y}, f_{t,x}) - \theta)$

  The velocity/driving input, <u>when close to the goal</u> (0,0 w.l.o.g.) become:
- $v = -k_a \cdot (x \cdot cos\theta + y \cdot sin\theta)$
- $\omega = k_\theta \cdot (ATAN2(-y, -x) - \theta)$

2.4 Bicycle

2.4.1 Kinematic model for “rear” wheel drive

- $$\begin{cases} \dot{x} = v \cdot cos\theta \\ \dot{y} = v \cdot sin\theta \\ \dot{\theta} = v \cdot \frac{tan\phi}{l} \\ \dot{\phi} = \omega \end{cases}$$

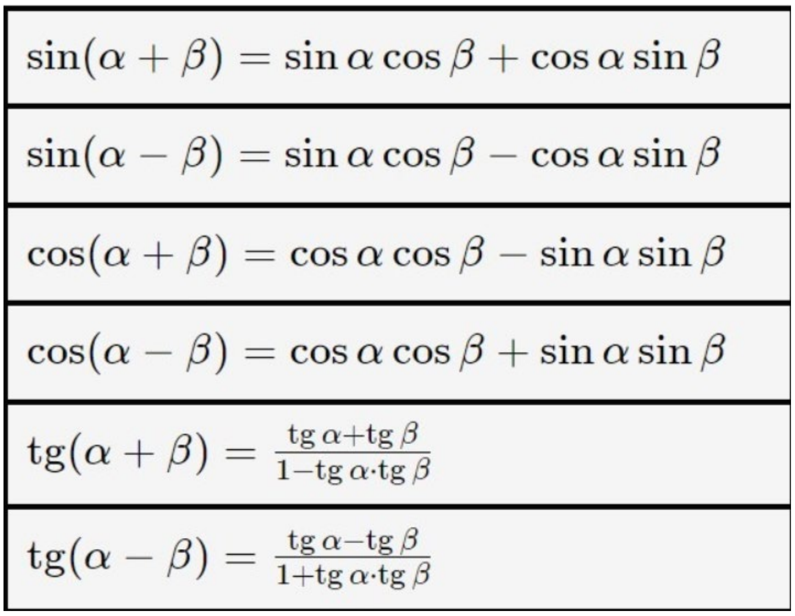
2.4.2 Kinematic model for “front” wheel drive

- $$\begin{cases} \dot{x} = v \cdot \cos{(\theta + \phi)} \\ \dot{y} = v \cdot sin(\theta + \phi) \\ \dot{\theta} = v \cdot \frac{sin\phi}{l} \\ \dot{\phi} = \omega \end{cases}$$

2.4.3 Reconstruction formulas

- $\theta = ATAN2(\dot{y}, \dot{x}) + K \cdot \pi$
- $\phi = \arctan\left[l \cdot \frac{\ddot{y}\dot{x} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}}\right]$
- $v = \pm\sqrt{\dot{x}^2 + \dot{y}^2}$
- $\omega = \dot{\phi}$

2.5 Sin/Cos sum-difference



$\sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$
$\sin(\alpha - \beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$
$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$
$\cos(\alpha - \beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$
$\mathrm{tg}(\alpha + \beta) = \frac{\mathrm{tg}\,\alpha + \mathrm{tg}\,\beta}{1 - \mathrm{tg}\,\alpha \cdot \mathrm{tg}\,\beta}$
$\mathrm{tg}(\alpha - \beta) = \frac{\mathrm{tg}\,\alpha - \mathrm{tg}\,\beta}{1 + \mathrm{tg}\,\alpha \cdot \mathrm{tg}\,\beta}$