Autonomous Mobile Robotics

Notes

Nijat Mursali

Table of Contents

Lecture 12 - Localization 3: Landmark-Based and SLAM

In this lecture we will complete the discussion about localization and we are going to discuss the use of Kalman filters in context of Landmark-Based localization. If you remember in last class, we discussed the situation where there is a robot with the kinematics of a unicycle moving in an environment. In this environment there are several objects that robots can identify which we call landmarks and the robot has a sensor which can measure the relative position of the robot of the sensor with respect to landmarks or opposite. The idea here was to use a Kalman filter to build a localization system. So how do we do that? First of all, remember that we developed a Kalman filter for a discrete time linear system. Thus, we say that odometric equations can be used as a discrete-time model of the robot, for instance we can use Euler method:

$$
\begin{aligned}
x_{k+1} &= x_k + v_k T_s \cos\theta_k + v_{1,k} \\
y_{k+1} &= y_k + v_k T_s \sin\theta_k + v_{2,k} \\
\theta_{k+1} &= \theta_k + \omega_k T_s + v_{3,k}
\end{aligned}
$$

Where $v_k = (v_{1,k},\ v_{2,k},\ v_{3,k})^T$ is a white gaussian noise with zero mean and covariance matrix V_k.

The second method for that is Runge-Kutta integration which is as in following formula:

$$
\begin{aligned}
x_{k+1} &= x_k + v_k\, T_s \cos\left(\theta_k + \frac{\omega_k\, T_s}{2}\right) \\
y_{k+1} &= y_k + v_k\, T_s \sin\left(\theta_k + \frac{\omega_k\, T_s}{2}\right) \\
\theta_{k+1} &= \theta_k + \omega_k\, T_s
\end{aligned}
$$

Third one is exact integration which is like following:

$$
\begin{aligned}
x_{k+1} &= x_k + \frac{v_k}{\omega_k}(\sin\theta_{k+1} - \sin\theta_k) \\
y_{k+1} &= y_k - \frac{v_k}{\omega_k}(\cos\theta_{k+1} - \cos\theta_k) \\
\theta_{k+1} &= \theta_k + \omega_k\, T_s
\end{aligned}
$$

As we have the process model, we also have the measurement model which will be the output equation that we discussed in the previous class. We have two(range and bearing angle) measurements for each landmark we can see. Let's say the robot can detect the identity of the landmark so the landmarks are tagged (maybe by color), so that the robot can detect to which landmark it's looking at when it measures the landmark. Thus, we build the association map of step k which takes the number of measurements and converts them to landmarks in the environment. At this point we can write the output equation which will be L_k vectors:

$$\boldsymbol{y}_k = \begin{pmatrix} \boldsymbol{h}_1(\boldsymbol{q}_k, a(1)) \\ \vdots \\ \boldsymbol{h}_{L_k}(\boldsymbol{q}_k, a(L_k)) \end{pmatrix} + \begin{pmatrix} w_{1,k} \\ \vdots \\ w_{L_k,k} \end{pmatrix}$$

where

$$\boldsymbol{h}_i(\boldsymbol{q}_k, a(i)) = \begin{pmatrix} \sqrt{(x_k - x_{l,a(i)})^2 + (y_k - y_{l,a(i)})^2} \\ \operatorname{atan2}(y_{l,a(i)} - y_k, x_{l,a(i)} - x_k) - \theta_k \end{pmatrix}$$

i-th landmark range

i-th landmark bearing

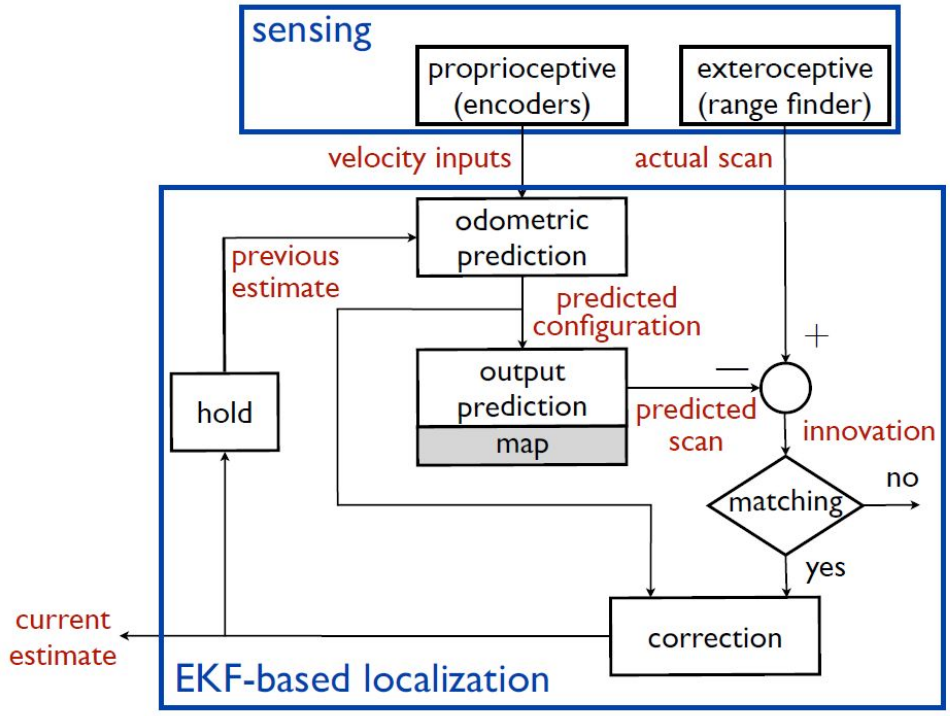
At this point we can use *Kalman filter* because we want to maintain an accurate estimate of the robot configuration in the presence of process and measurement noise which will be the ideal case for KF. this is ideal, but since both process and output equations are nonlinear, we must apply the Extended Kalman Filter which the equations should be linearized.

The fundamental idea of data association is to associate each observation to the landmark that minimizes the magnitude of the innovation. So, for example let's say if we are at the k+1-th step, we need to consider i-th measurement $y_{i,k+1}$ and compute all the candidate innovations. The smaller the innovation $v_{i,j}$, the more likely that *i-th* measurement corresponds to the j-th landmark. However, the innovation magnitude must be weighted with the uncertainty of measurement; in the EKF, this is encoded in the matrix:

EKF localization on a map

We can assume that we know a metric map of the environment where the map can be in several formats like line-based map or an occupancy grid. If we assume that the robot is equipped with a rangefinder (can be laser sensor for example) and it gets the measurements that come in the form of distance reading and uses the whole scan as output vector whose components are range readings in all available directions. Then, we are going to compute the innovation as the difference between these readings and predicted measurements. We compute the predicted measurements geometrically where we take our robot, we put it in our map and we say if the robot is here given the geometry of the environment, what would be my estimate? We compute the innovation as the difference between the actual scan and predicted scan which is computed from pure geometry.

Architecture is as in the following image where proprioceptive sensing is wheel encoders and exteroceptive is range finders. Proprioceptive sensors are used to reconstruct the inputs, and exteroceptive ones are used for actual scan.



EKF SLAM

SLAM means simultaneous localization and map building and it's a situation in which the map for localizing is not given in advance. We tell the robot to build the map and localize itself with respect to the map as it moves. In probabilistic SLAM, the idea is to estimate the map features in addition to the robot configuration. Let's make some assumptions:

1. The robot is an omnidirectional point-robot, whose configuration is then a cartesian position
2. L landmarks are distributed in the environment
3. The robot is equipped with a sensor that can see, identify and measure the relative position of all landmarks with respect to itself

Then we need to define an extended state vector to be estimated:

$$x = (x\, y\, x_{l_1}\, y_{l_1} \ldots x_{1_L}\, y_{1_L})^T$$

We explained the concepts, but let's try to explain how realistic is *KF/EKF* localization. *KF/EKF* assume that the probability distribution of the state is unimodal which means they have a single peak like gaussian. This requires an accurate estimate of the robot initial configuration and also relatively small uncertainties. However, if the robot is released at an unknown/poorly known position, the probability distribution for the state becomes multimodal in the presence of aliasing.

Lecture 13 - Motion Planning 1: Retraction

Motion planning is the problem of planning robot motions among obstacles. As I was saying, robots are expected to execute tasks in the real world and the real world is populated by obstacles. Clearly this is both true in industrial environments where fixed based manipulators need to move to perform certain tasks. We would like to give the robot the ability to autonomously detect the obstacles and move around the environment. Thus, autonomy requires that the robot is able to plan a collision-free motion from an initial to a final posture on the basis of the geometric information. Thus, we can say the information about the geometry can be following:

1. Entirely known in advance which is called off-line planning

2. Gradually discovered by the robot which is called on-line planning where robot accumulates the information about the environment and builds a map and plans the map it built

The canonical problem

It is useful to define a canonical version of the motion planning problem where this canonical problem is relatively simple of the problem, but it captures the essential complexity of the problem itself. So, in this problem we consider a robot which we denote as *B* and this robot might be a kinematic chain with fixed or mobile base (can be really anything) and it is moving in a workspace in 2D or 3D space. In this problem, we will consider B as *free-flying* which means that the representative point of the robot in configuration space can move in all directions which means the robot is not constrained in any way. We also have obstacles in canonical problems where all of them are fixed and they don't deform (fixed rigid objects). So, what is exactly a canonical problem?

- Given a start configuration q_s and a goal configuration q_g of B in C, plan a path that connects q_s to q_g and is safe, i.e., it is completely contained in the free configuration space C_{free}

We have not discussed free configuration space yet, but we will discuss it in the following section.

Let say in workspace we have a manipulator and this robot moves around obstacles (there are several obstacles in the workspace named as $O_1,\ O_2,\ O_3$ which are closed sets in the workspace. We can have a look at this problem in configuration space where the robot is represented by a point. Each point in configuration space represents the posture of the robot. What we would like to do in this problem is to characterize the configurations that put the robot in collision with the obstacles. So, we define the configuration space images of these obstacles. CO_i is the image of the *i-th* object in configuration space which is defined by configuration in configuration space such that the robot placed at *q* has a nonzero intersection with an obstacle.

So as definition says, a path in ***configuration space is free if it lies completely in free configuration space***.

So, what are the extensions to the canonical problem? We can say moving obstacles, on-line planning, kinematic (nonholonomic) constraints and manipulation planning are the extensions to

the canonical problem. Thus, we can say that many methods that can solve the canonical problem can be appropriately modified to address one or more of these extensions. In the next section, we will talk about the motion planning methods.

Motion planning methods

All motion planning methods work in the configuration space C. Why is that? It's because planning in the cartesian space, work space is impossible because in the workspace the robot consists of infinity points. Most of the motion planning methods require preliminary computation of the C-obstacle region *CO*, a highly expensive procedure. Thus, we will perform motion planning in configuration space.

Classification

We essentially consider families of methods which are following:

1. Roadmap methods which represent the connectivity of C_{free} by a sufficiently rich network of safe paths. Thus, roadmap methods are classical methods which are proposed for motion planning. The idea here is to throw a roadmap. Let's say you have a free space and you can represent this space with a roadmap. So the connectivity of the free space is reduced to the connectivity of the roadmap.
2. Probabilistic methods are the most popular and modern algorithms for motion planning and they represent a particular instance of sampling-based methods where samples of C are randomly extracted. Thus, we can say that probabilistic methods are based on the idea of sampling which they represent the configuration space with samples.
3. Artificial potential field methods which are ideal for online case planning

Retraction method

This method is the part of roadmap methods and will be discussed in this section. So, in this problem we assume that configuration space is plane $C = R^2$ and C_{free} is a polygonal limited subset which means it's a subset of a plane whose boundary is made of line segments. We need to explore the concept of clearance of a configuration q in C_{free} which is defined as the

minimum distance between configuration and s which is any other configuration in the boundary of free space (it will be the euclidean distance).

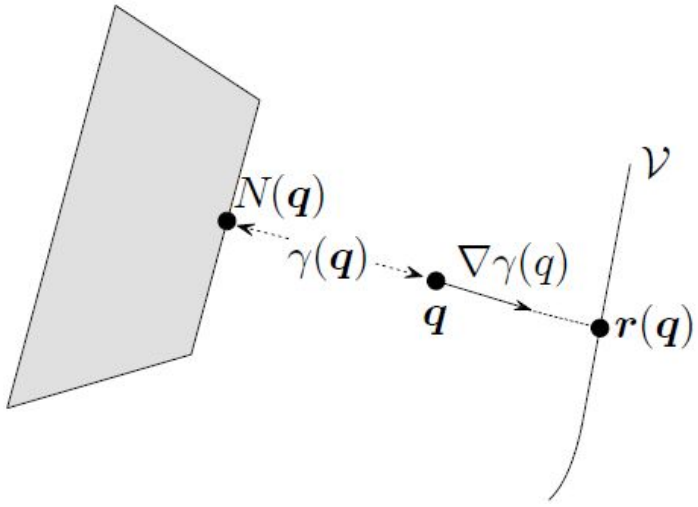
$$\gamma(q) = min \parallel q - s \parallel$$

We can define the neighbors of q as

$$N(q) = \{s \in \partial C_{free} : \parallel q - s \parallel = \gamma(q) \}$$

So, finally, the generalized Voronoi diagram of C_{free} is

$$V(C_{free}) = \{q \in C_{free} : card(N(q) > 1\}$$



In the initial case our q is not in *arc which is V* in the figure. In order to connect any *q to* $V(C_{free})$, we can use a retraction method where we follow the gradient of clearance (by definition gradient of clearance is the direction which clearance increases the most) which is the opposite direction of the neighbor. In our case, retraction r *preserves the connectivity of* C_{free} which means that configuration q and its retraction will always vary in the same connected component of free configuration space. Hence, a safe path exists between q_s and q_g if and only if *a path exists on* $V(C_{free})$ *between* $r(q_s)$ *and* $r(q_g)$. So in the following section we will explain the whole algorithm by showing how it works:

1. build the generalized Voronoi diagram $V(C_{free})$
2. compute the retractions $r(q_s)$ and $r(q_g)$
3. search $V(C_{free})$ for a sequence of arcs such that $r(q_s)$ belongs to the first and $r(q_g)$ to the last

4. if successful, return the ***solution path*** consisting of
    a. line segment from q_s *to* $r(q_s)$
    b. portion of first arc from $r(q_s)$ to its end
    c. second, third, … , penultimate arc
    d. portion of last arc from its start to $r(q_g)$
    e. line segment from $r(q_g)$ *to* q_g

    otherwise, report a ***failure***

Lecture 14 - Motion Planning 1 continuation

To summarize the previous class, we are looking at the problem of planning motion with obstacles in a workspace where we checked the canonical problem where we had a robot with a kinematic chain with any base and works in *2D or 3D*. We also said that robots are not subject to any kinematic constraints of any kind. So in this problem we had to create a path that connects q_s and q_g.

The next method in roadmap methods we would like to discuss is ***cell decomposition methods***. The idea of this method is to build a roadmap (objective is same as before), but here we are going to exploit the different ideas. The idea is to decompose C_{free} in cells where cells are regions. There are two properties to deal with:

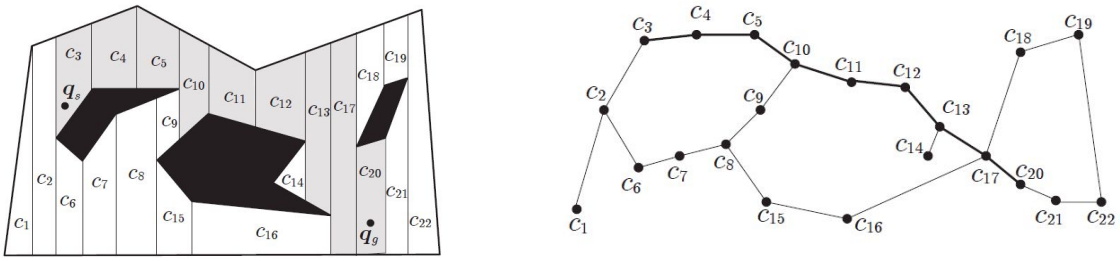
1. it is easy to compute a safe path between two configurations in the same cell. It means that it should be easy to move in the same cell
2. it is easy to compute a safe path between two configurations in adjacent cells

Once we find the decomposition of C_{free}, we can find the sequence of those regions which we will call a *channel* that starts at q_s and ends at q_g. There are several composition types to consider and the first one will deal with exact decomposition.

Exact decomposition

In exact decomposition we want to subdivide the partition in free configuration space into a collection of regions so the union of these regions should give us exactly the free configuration space. We will take the same assumption where configuration space is 2D space and configuration space is a polygonal limited subset. Thus, we will use convex polygons which are

an easy choice for us because if a polygon is convex where if you take two points and join them by segment then the segment is completely contained in the polygon. We will also use variable-shape cells which are needed to decompose exactly C_{free}. To do that we can use ***sweep-line algorithms*** to compute the decomposition of free configuration space into convex polygons. We need to sweep a line over free configuration space so when it goes through a vertex, two segments originate at the vertex. An extension lying in C_{free} is part of the boundary of a cell; the rest are other extensions which we got as the result of trapezoidal decomposition. In the following picture we showed it clearly:



So, in this case we can go from C_3 to C_{20} for instance. But what exactly does the path represent? The path is the sequence of nodes where each node is a cell (channel). So how does the algorithm exactly work?

1. we need to compute a convex polygonal decomposition of free configuration space
2. then we need to build the associated connectivity graph *C*
3. then we need to search *C* for a channel of cells from c_s to c_g
4. if it is successful, we need to extract and return a solution path consisting of
    a. line segment from q_s to the midpoint of the common boundary between the first two cells
    b. line segments between the midpoints of consecutive cells
    c. line segment from the midpoint of the common boundary between the last two cells and q_g
5. Otherwise, we need to report a failure.

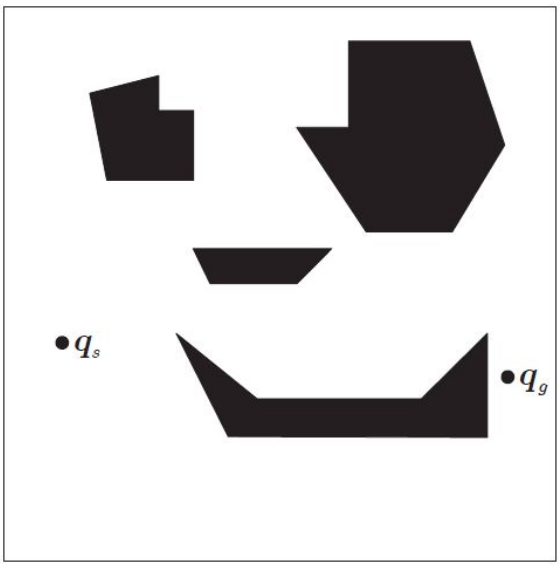
So, to be more precise, we need to build the associated connectivity graph *C* and identify the nodes(cells) c_s and c_g and then we need to use the graph search to find a path on *C* from c_g that represents the channel of cells. Finally, we need to extract from the channel a safe solution

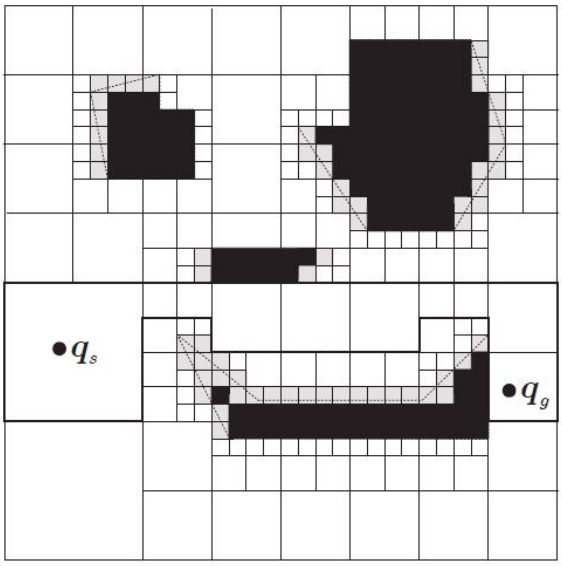
path such as joining q_s to q_g via midpoints of common boundaries. The time complexity of this solution is ***O(v logv) where v is the number of vertices of the free configuration space***.

A channel is more flexible than a roadmap because it contains an infinite number of paths which can be exploited to take into account non-holonomic constraints or to avoid unexpected obstacles during the motion. The solution path is a broken line, but smoothing may be performed in a post-processing phase. Everything we said was about *2D* dimensional space and if we move it to *3D* dimensional space the polygonal limited subset becomes a polyhedral limited subset, the sweep-plane algorithm to compute a decomposition of free configuration space into convex polyhedra. It is possible to extend the idea of cell composition to configuration space or arbitrary dimension, however, it is very inefficient because the complexity is exponential in the dimension of *C*.

Approximate decomposition

In this method we also take the same assumptions as before such as configuration space is 2D and free configuration space is a polygonal limited subset. In this case, we use fixed-shape cells to obtain an approximate decomposition of free configuration space such as squares. As in exact decomposition, if we use convex it guarantees that it is easy to plan in a cell and between adjacent cells which means it is easy to move in the cell or between the cells that are adjacent. The nice thing is that we can use a recursive algorithm to read a trade-off between simplicity and accuracy.





Let's say you have the image on the left which is the polygonal region which is limited by its external boundaries and all parts of boundaries are segments. So, in this case we have a start and a goal where we need to build a convex cell decomposition of this area using squares. We can use recursive algorithm in this case by starting to divide C in 4 cells and classifying each cell as

1. Free, if its interior is completely in free configuration space
2. Occupied, if it is completely contained in CO
3. Mixed, if it is neither nor occupied

---------- EXAMPLE AT 50MIN ----------

Lecture 14:15 - Motion Planning 2: Probabilistic Planning

Probabilistic planning methods are instances of sampling-based methods and we will be talking about sampling-based methods in the following chapter. This method is used extensively in mobile robotics and not only for motion planning, but also other purposes.

Sampling-based methods

These are the most modern methods in motion planning and they are used extensively in mobile robotics and the idea is still to build a roadmap, but ***this roadmap should not be built by knowing free configuration space in advance***. In the previous method, we already knew the obstacles and the area we worked on. In this method, we don't know free configuration space and we don't want the obstacles. We want to build an approximate representation of this free configuration area as a collection of collision free configurations which are connected among them, so it would still be a roadmap. So we need to to the basic iterations as below:

1. we need to extract a sample q of C
2. then use forward kinematics to compute the volume occupied by the robot B(q) at this particular sample
3. then, we need to check the collision between the robot and obstacles
4. finally, the idea is that if there's no collision in the sample we add it to roadmap, else we discard it

So, if you extract the samples randomly, you will get much faster planning by extracting samples in a deterministic way. It's a really fast algorithm which takes only milliseconds. In the following

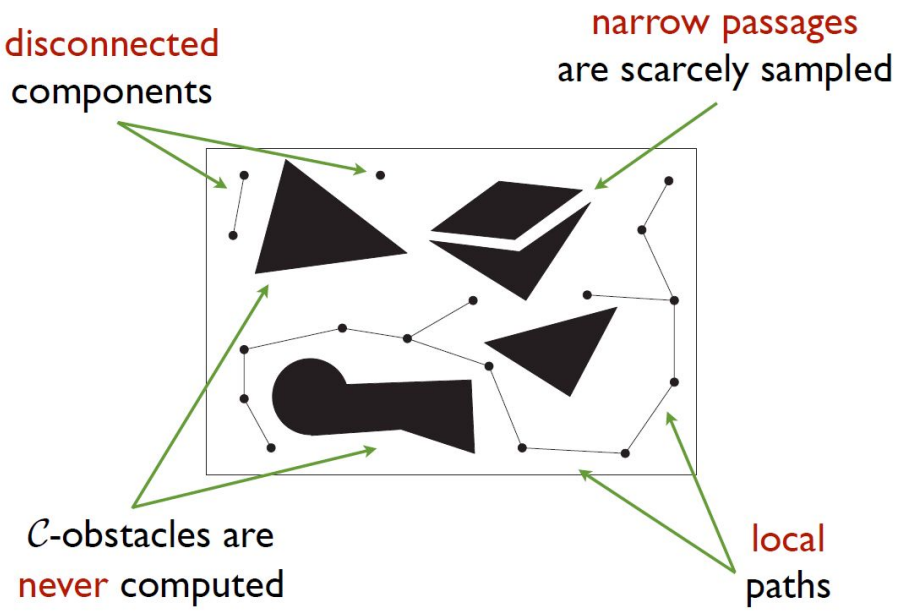
chapters, we will be explaining two important methods which are PRM (Probabilistic Roadmap) and RRT (Random Tree) methods.

PRM (Probabilistic Roadmap)

The following shows the basic iteration to build the PRM:

1. we first need to extract sample q of free configuration space using uniform probability distribution. It means that any sample has the same probability to be extracted
2. then we need to compute the robot $B(q)$ and check for collision
3. if it is free we keep it and add q to the PRM, if it is not we simply discard it
4. once we add q to the PRM, we search the PRM for sufficiently near configurations
5. if possible, we connect q *to* q_{near} with free local path

The PRM method is probabilistically complete which means the probability of finding a solution whenever one exists tends to 1 as the execution time tends to infinity. The fundamental advantage of these methods is speed where the time PRM needs to find a solution in high-dimensional spaces can be orders of magnitude smaller than previous planners.



As in the above example, we have disconnected components, configuration space obstacles that are never computed, narrow passages that are scarcely sampled and local paths. Of course, construction of the PRM is not always perfect, so we say it is "arrested" when

1. disconnected components become less than a threshold

2. a maximum number of iterations is reached

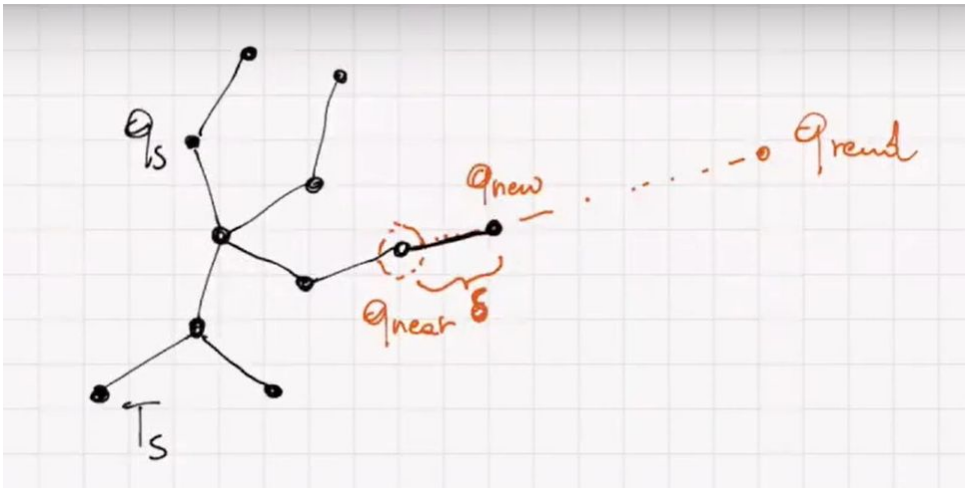
So, if $q_s and\, q_g$ can be connected to the same component, a solution can be found by a graph search, else, we can enhance the PRM by performing more iterations.

The PRM method is probabilistically complete where the probability of finding a solution whenever one exists tends to 1 as the execution time tends to infinity and it is called multiple-query where new queries enhance the PRM. The fundamental advantage of this algorithm is speed where the time PRM needs to find a solution in high-dimensional spaces can be orders of magnitude smaller than previous planners. However, it is not always easy to find a solution because narrow passages are critical and heuristics may be used to design biased probability distributions aimed at increasing sampling in such areas.

RRT (Rapidly-exploring Random Tree)

This is the second algorithm we will discuss for probabilistic motion planning. It is fairly simple algorithm where the basic iteration to build the tree T_s rooted at q_s :

1. generate q_{rand} in C with uniform probability distribution which is same as in PRM
2. search the tree for the nearest configuration q_{near}
3. then we choose q_{new} at a distance δ from q_{near} in the direction of q_{rand}
4. then we check for collision q_{new} and the segment from q_{near} to q_{new}
5. if check is negative, we add $q_{new} to\, T_s$ (expansion)



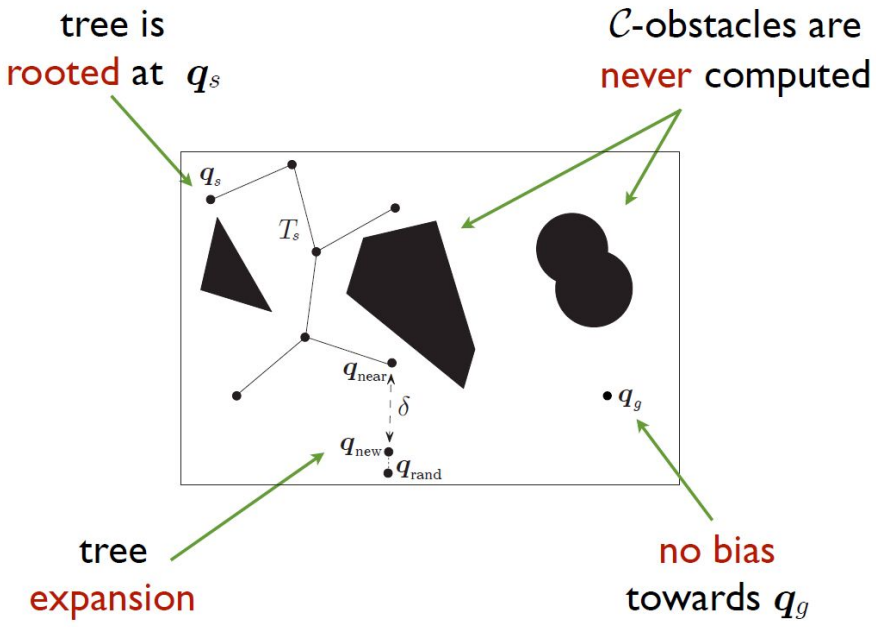


Fig. Rapidly exploring Random Tree

T_s rapidly covers C_free because the expansion is biased towards unexplored areas (actually, towards larger Voronoi regions). Another advantage of this algorithm is that it quickly explores all areas much more efficiently than other simple strategies.

To make this algorithm more effective in approaching a goal we need to introduce some sort of bias which we can do by ***bidirectional RRT.*** In this case, we grow two trees: T_s and T_g. Each tree expands by choosing the random configuration (same as before, but two trees), but in this case we move to alternate expansion and connection phases which use the last generated q_new of T_s as a q_rand for T_g and then switches the rule of T_s and T_g.

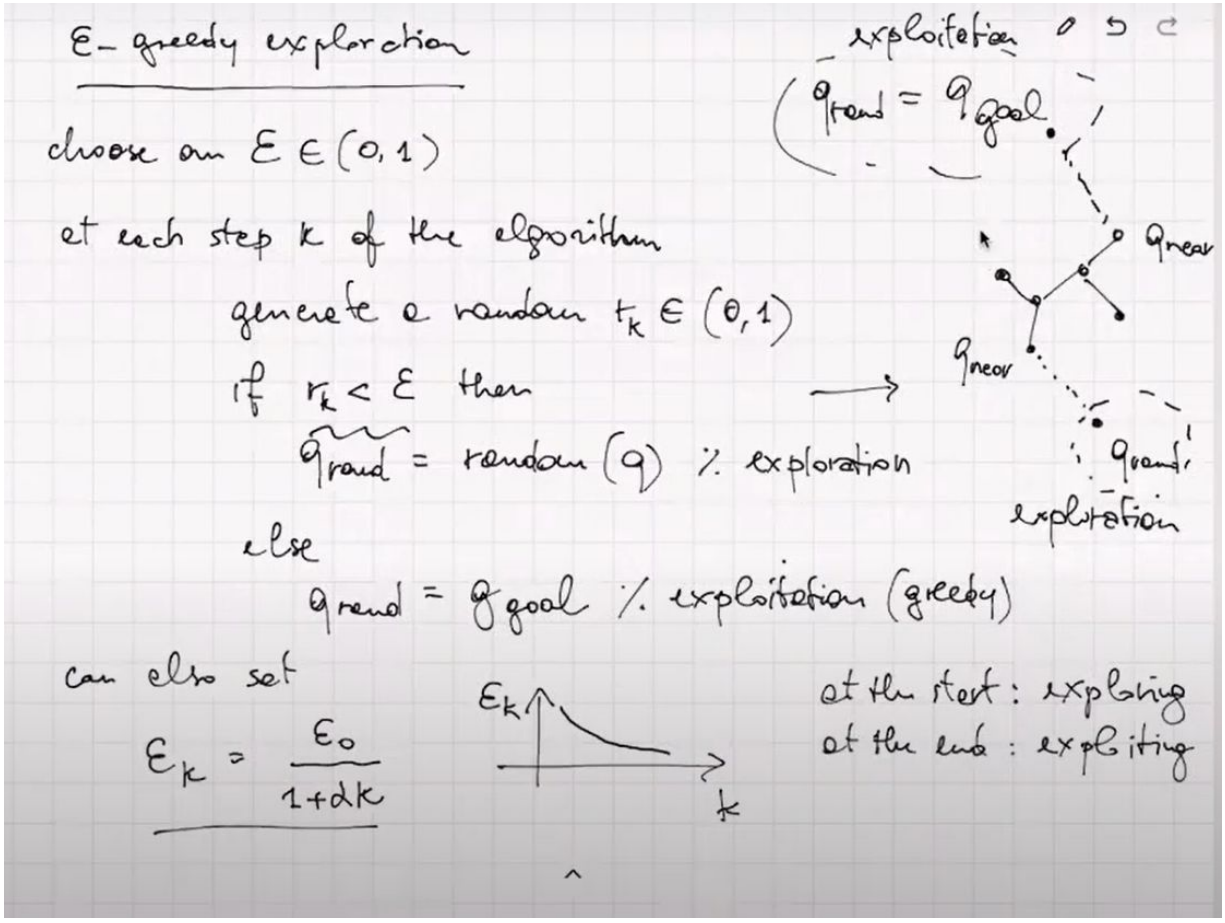


Fig. Greedy Algorithm

Bidirectional RRT is probabilistically complete and single-query which means trees are rooted at q_s and q_g, and in any case new queries may require significant work. There are also many variations such as one may use and adaptive stepsize δ to speed up the motion in wide open areas. This can be modified to address many extensions to the canonical planning problem such as moving obstacles, non-holonomic constraints and manipulation planning.

RRT: extension to nonholonomic robots

The idea here for extending to nonholonomic robots is fairly simple. Let's say, we want to plan for unicycle which is three dimensional configuration space (we have two cartesian coordinates and one rotational coordinate) $C = R^2 \times SO(2)$. The problem in standard RRT to this system is that linear paths which we use in configuration space to connect q_{near} to q_{rand} ***are not admissible*** in general. One possibility is to use motion primitives such as a finite set of admissible local

paths, produced by a specific choice of the velocity inputs. For instance, we can use *Dubin's Car* which is

$$v = \overline{v} \quad w = \{-\overline{w},\ 0,\ \overline{w}\} \quad t \in [0,\ \Delta]$$

The algorithm is the same with the only difference is that q_new is generated from q_near selecting one of the possible paths (can be chosen randomly or as the one that leads the unicycle closer to q_rand). 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 as the time tends to infinity.

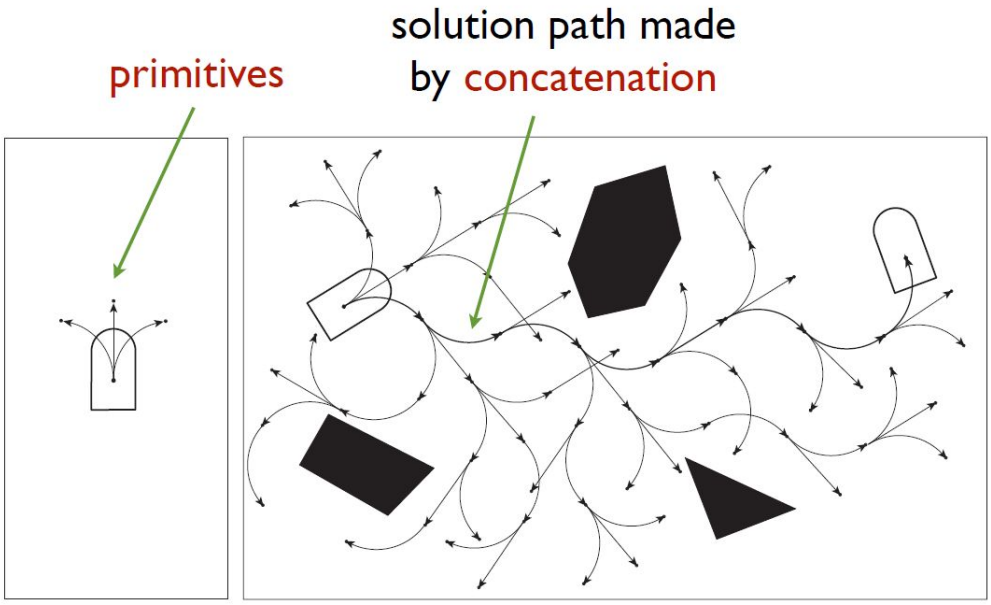


Fig. Using Primitives

Lecture 16 - Motion Planning 3: Artificial Potential Fields

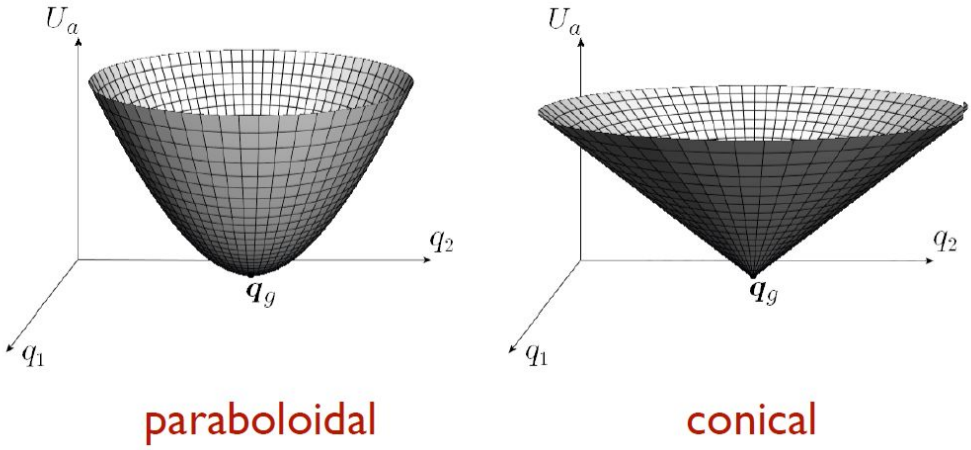
The idea here is that these methods are aimed at online planning which means that a ***robot does not have any prior knowledge about the environment and it has to plan online while collecting information about the environment***. Incremental workspace information may be integrated in a map and used in a ***sense-plan-move*** paradigm.Alternatively, incremental workspace information may be used to plan motions following a memoryless stimulus-response paradigm (reactive navigation).

Artificial potential fields

The fundamental idea here is that we need to build potential fields in configuration space so that the point that represents the robot is attracted by the goal q_g and repelled by the configuration space obstacle region CO. So, the main idea was to obtain a total potential by superposition of the attracted potential and the repulsive potential. We want to follow the most promising local direction of motion which is the direction of the entire negative gradient $-\nabla U(q)$. Why is it negative? It's negative because it is the direction in which the function decreases the most.

Attractive potential

We have discussed the structure of attractive potential and we have seen particular possibilities like paraboloidal and conical as in the following picture. Both have the vertex and the goal and as seen they are in *2D* space, but they can be in any space, it does not matter. The fundamental idea of this algorithm is to guide the robot to the goal q_g which we will build in configuration space. We essentially have two possibilities for building the attractive potential: paraboloidal and conical.



The following formula is the analytical expression of paraboloidal potential and the expression for conical potential. For the paraboloidal we have the square norm and in the conical it is just a norm of error. As a consequence, the result of attractive forces are different. In a paraboloidal case as the norm is square the force will be linear function, but in conical it will be constant. It is

actually easy to combine those two notions where conical will be away from q_g and paraboloidal close to q_g.

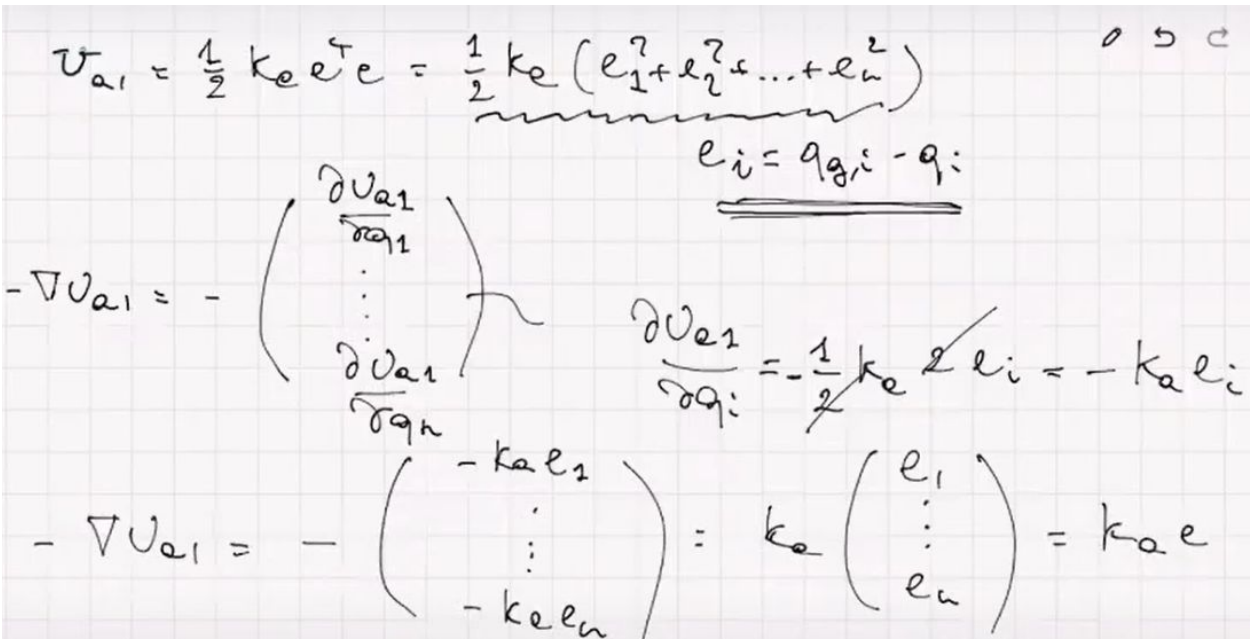
However, it is not important to have those two, we can also have anything other than these two possibilities. For the first case, we can say $e = q_g - q$ which is the error between goal and current configuration and we can compute U by following formula:

$$U_{a1}(\boldsymbol{q}) = \frac{1}{2} k_a \boldsymbol{e}^T(\boldsymbol{q})\boldsymbol{e}(\boldsymbol{q}) = \frac{1}{2} k_a \|\boldsymbol{e}(\boldsymbol{q})\|^2$$

The resulting attractive force is linear in e

$$\boldsymbol{f}_{a1}(\boldsymbol{q}) = -\nabla U_{a1}(\boldsymbol{q}) = k_a \boldsymbol{e}(\boldsymbol{q})$$

Professor wrote the formula like this:



For conical case, we simply compute *U and resulting attractive force* by following formula which attractive force will be constant:

$$U_{a2}(\boldsymbol{q}) = k_a \|\boldsymbol{e}(\boldsymbol{q})\|$$

$$\boldsymbol{f}_{a2}(\boldsymbol{q}) = -\nabla U_{a2}(\boldsymbol{q}) = k_a \frac{\boldsymbol{e}(\boldsymbol{q})}{\|\boldsymbol{e}(\boldsymbol{q})\|}$$

Attractive force for the paraboloidal is better than conical in the vicinity of q_g, however, it increases indefinitely with e. Thus, a convenient solution is to combine the two profiles: conical away from q_g and paraboloidal close to q_g.

$$U_a(\boldsymbol{q}) = \begin{cases} \frac{1}{2} k_a \|\boldsymbol{e}(\boldsymbol{q})\|^2 & \text{if} \quad \|\boldsymbol{e}(\boldsymbol{q})\| \leq \rho \\ k_b \|\boldsymbol{e}(\boldsymbol{q})\| & \text{if} \quad \|\boldsymbol{e}(\boldsymbol{q})\| > \rho \end{cases}$$

Continuity of paraboloidal attractive force at the transition requires

$$k_a \boldsymbol{e}(\boldsymbol{q}) = k_b \frac{\boldsymbol{e}(\boldsymbol{q})}{\|\boldsymbol{e}(\boldsymbol{q})\|} \qquad \text{for} \quad \|\boldsymbol{e}(\boldsymbol{q})\| = \rho$$

such as $k_b = \rho k_a$

Repulsive potential

The fundamental objective in this case is ***to keep the robot away from the configuration space obstacles***. We know that CO are regions which are made by the union if configuration space obstacles. Let's assume that this region is partitioned in advance into convex components. So, from now on, every CO_i will be assumed to be convex. If it is not convex, then we have decomposed it into several components each of which will be convex. So, for each CO_i we define a repulsive field with the following formula:

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

where $k_{r,i} > 0; \gamma = 2, 3, \ldots;$ $\eta_{0,i}$ is the range of influence of CO_i *and* $\eta_i(q)$ *is the clearance*

$$\eta_i(\boldsymbol{q}) = \min_{\boldsymbol{q}' \in \mathcal{CO}_i} \|\boldsymbol{q} - \boldsymbol{q}'\|$$

The clearance is the minimum between the configuration space and any other configuration space that belongs to the configuration space obstacle.

Planning techniques

There are 3 ways to generate motion on the basis of the total force:

1. The first would be to consider this artificial force as a true force. So, we would use this as a generalized force in dynamics in which the forces are f_t which are total forces. It means that the robot will move as a consequence of these generalized forces. The effect on the robot is filtered by its dynamics
2. In the second case we consider total force not as generalized force, but like generalized accelerations. In this case the effect on the robot is independent on its dynamics and will be slow
3. In the last case we consider total force as generalized velocity where the effect on the robot is independent on its dynamics and fast

In offline planning the most popular choice is the third one where the next sample will be the sum of previous sample and sampling interval multiplied by $f_t(q_k)$ (which is a gradient method). In online planning, the first and second technique directly provides control inputs, and the third one provides reference velocities for low-level control loops. The most popular choice is the third one.

Local minima

If planned path enters the basin of attraction of a local minimum, it will reach the local minimum q_m and stop there because if the gradient is zero there's no exit direction as we remember our robot follows the anti gradient, so if anti gradient is zero robot will be staying in zero as well. Repulsive fields generally create local minima, hence motion planning based on artificial potential fields is not complete. There are of course workarounds existing, but we need to remember that artificial potential fields are mainly used for online motion planning so completeness may not be required. There are several workarounds that we will explain:

Workaround no. I: best-first algorithm

The first one is to use an algorithm which enters the basin of attraction, but then realizes so it gets out from the local minimum. This workaround is not efficient because paths generated by this algorithm are not efficient because local minima are not avoided.

Workaround no. 2: navigation functions

The second option is to use a different approach where we need to build navigation functions such as potentials without local minima. In this case, if the configuration space obstacles are star-shaped, one can map configuration space obstacles to a collection of spheres via a diffeomorphism, which builds a potential in transformed space and maps it back to configuration space *C*. Another possibility is to define the potential as an harmonic function which is the solution of Laplace's equations. All these techniques required complete knowledge of the environment which is only suitable for off-line planning.

Workaround no.3: vortex fields

This approach is the alternative to navigation functions in which one directly assigns a force field rather than a potential. The fundamental idea is to replace the repulsive action (which is responsible for appearance of local minima) with an actio forcing the robot to go around the configuration space obstacles.

Lecture 17 - Motion Planning 3 and V-REP

V-REP is a virtual robot experimentation platform which is simply a robotic simulator that is a software environment aimed at generic robotic applications (not only motion planning). It's new, relatively new (2014) and open source which is available for Windows, Linux and Mac. As an example of applications we can say fast prototyping and verification, fast algorithm development, hardware control etc. V-REP simulator provides physical engines for dynamic simulations which means in this software you can simulate real physics of the real world. It allows the simulation of sensors where you can deploy sensor based algorithms. It also provides a large and continuously growing library of robot models. In your scene you can have basic components like following:

1. Shapes
    a. Shapes are rigid mesh objects that are composed of triangular faces
    b. Can be grouped and ungrouped
    c. There are different types of shapes such as random, convex, pure shapes
2. Joints

a. Joints can be either revolute or prismatic where revolute is rotational and prismatic is translational movement
b. They can also be screw which is translational while rotational movement and spherical which have three rotational movements

There are also sensors that can be proximity,vision, torque/force sensors and more. Proximity sensors are more than simple ray-type detection and they are configurable detection volume. Vision sensors render the objects that are in the field of view and they are embedded image processing.

V-REP has both advantages and disadvantages which are following:

1. It has very good online documentation, it has four different types of physics engines, most complete open source software for dynamic simulations, very good set of APIs and control mechanisms.
2. The vision sensors have high computational payload, since it is a huge software, it is not so friendly at the beginning, multi-robot simulations have high computational payload and collision checking library is slow.

Task-constrained motion planning with moving obstacles(TCMP-MO)

It's the problem of finding feasible, collision-free motions for a robot that is assigned a task constraint in an environment containing moving obstacles. We assume that the robot configuration q takes values in a n_q dimensional configuration space. We will take into consideration that the task is defined by a desired task path y_d(s) which is between 0 and 1 and takes values in an n_y dimensional task space. We have two assumptions in this case:

1. The robot is kinematically redundant with respect to task (n_q > n_y)
2. The obstacle trajectories are known (we are working off-line)

This can be addressed into two cases: kinematic and dynamic cases. Kinematic case the robot is described by the kinematic-level model with generalized velocities and state. However, in the dynamic case the robot is described in the Euler-Lagrange form with generalized accelerations and state. The fundamental solution to TCMP-MO problem is a configuration-space trajectory q(t), i.e., a path q(s) + a time history s(t) such that:

1. The assigned task path is continuously satisfied
2. Collisions are always avoided

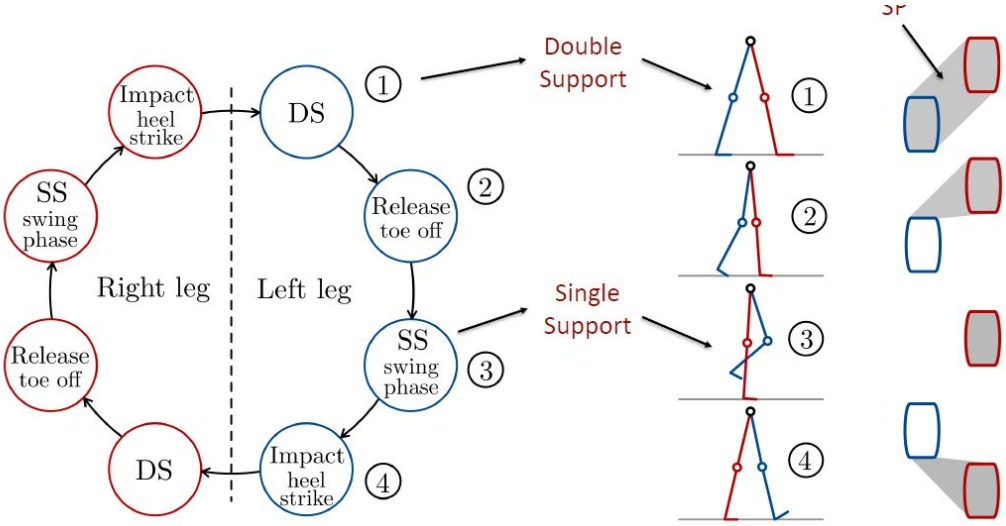
3. Joint velocity limits (kinematic case) or joint velocity/torque limits (dynamic case) are respected

Lecture 18 - Humanoid Robots 1 and 2

Let's begin with the reason why humanoid robots should be used. First of all, there are practical reasons. We can think of the environments that are built for humans like stairs, obstacles. Another reason is psychological and commercial because humanoids have a major appeal. There are several reasons why humanoids are good.

1. multipurpose: sensing, manipulation, locomotion etc..
2. adaptability: humanoids can work in environments suitable for humans and expand their capabilities by using machines designed for humans
3. collaboration: humanoid motion is easy for humans to understand and predict
4. human-like appearance: empathy

Humanoid robots are made like human structures where it has a torso, legs, knees, arm etc. It also has the center of mass which is the moving point. The following image shows the basic terminology of the moving robot:

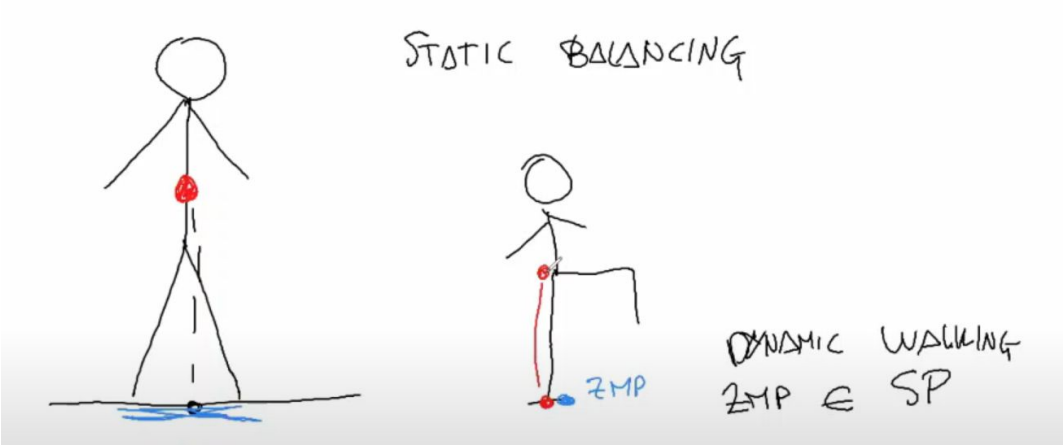


The above image shows the phases of walking during a motion where we have several phases. The one phase is double support phase in which both feet are in contact with ground which is shown in #1. The next step is releasing the toe off where only the tip of the robot is contact with the ground. The next step is a single support phase (swing phase) where one foot is in contact

with ground and the other one is in swing motion. The final phase happens when the heel is in contact with the ground.

Gaits

Gait is basically a regular motion while walking and there are different types we can generate. First one is static gait which is the projection of the COM (Center of Mass) on the ground is always inside the SP (Support Polygon). The second one is a dynamic one which happens when the robot is moving to balance. Zero Moment Point (ZMP) is the point on the ground where the resultant of the reaction forces acts. In dynamic gait the ZMP is always inside the SP.



Passive(dynamic) walkers

Passive walkers are basically a mechanism that performs local motion without using any actuation. They are energy-efficient and natural gait (limit cycle). However, they do not work on horizontal ground. They also have the limited agility and responsiveness of motion.

Active (dynamic) walkers have actuated joints which can cause more energy consumption and in this type the feedback control is needed. In this case, robots with flat feet or non-trivial feet are powerful.

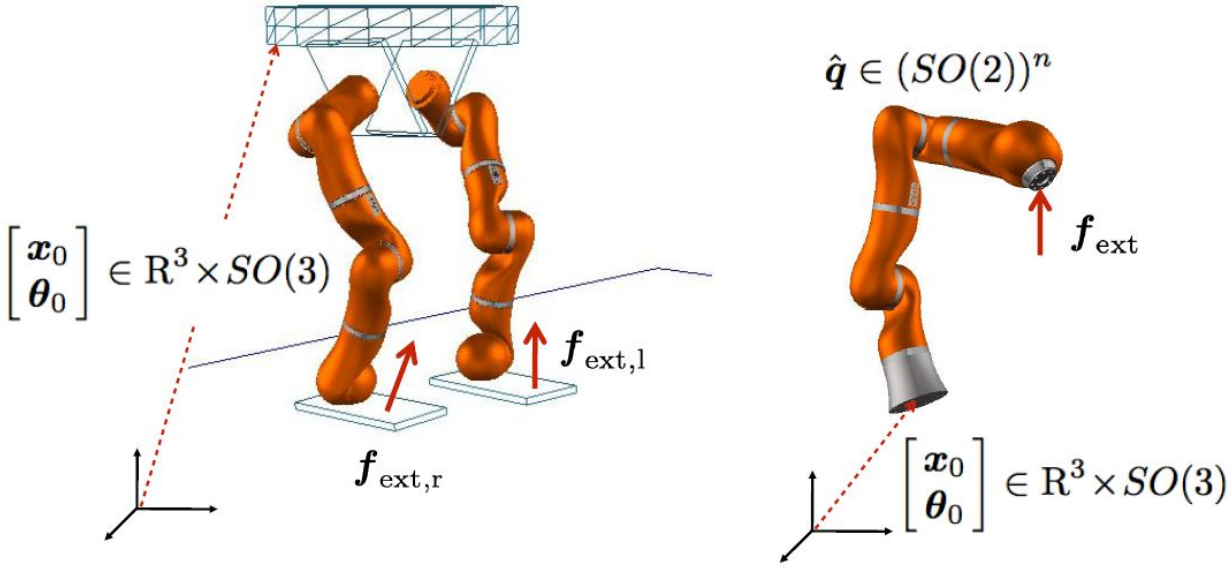
Humanoid Robots 2: Dynamic Modeling

Till now we just talked about the starting concepts, but did not define the points for ZMP and COM. In this lecture we will talk about modeling which is a very complex topic and there are many ways to model a humanoid robot. We will talk about the most important models while

walking. For the humanoid robots modeling is important because we need to get the mathematical notions from the robot while it moves. Can we also say the manipulators that are attached to the floor like a part of a legged robot? No, because this type of manipulator cannot fall because its base is clamped to the ground.

Floating-base model

The difference lies in the contact forces which means one may look at these contact configurations as different fixed-base robots, each with a specific kinematic and dynamic model or we can consider a single floating-base system with limbs that may establish contacts. The general model is that of a floating-base multi-body which is shown in following image:



State of the manipulator is given by q_i, but the state of robot is combination of configuration of joints plus the position and orientation of the any point of the robot which we can write like $\{q_i\} \cup \{x_b, v_b\}$. The dynamic model for this floating-base model would be very similar to the dynamic model of a fixed-base manipulator, but in this case the state is given by vector q which is a combination of joint coordinates and position/orientation of the base link. The following formula is the formula for dynamic model for this type of robot:

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{e}(\boldsymbol{q}) = \boldsymbol{\tau} + \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{f}_{ext}$$

but we can express the dynamic model like following image:

$$\boldsymbol{B}(\boldsymbol{q})\left(\begin{bmatrix}\ddot{\hat{\boldsymbol{q}}}\\ \ddot{\boldsymbol{x}}_0\\ \ddot{\boldsymbol{\theta}}_0\end{bmatrix}+\begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{g}\\ \boldsymbol{0}\end{bmatrix}\right)+\boldsymbol{n}(\boldsymbol{q},\dot{\boldsymbol{q}})=\begin{bmatrix}\boldsymbol{\tau}\\ \boldsymbol{0}\\ \boldsymbol{0}\end{bmatrix}+\sum_i \boldsymbol{J}_i^T(\boldsymbol{q})\boldsymbol{f}_i$$

Newton-Euler equations

For this case, we need to consider the linear momentum which is basically mass times acceleration of COM and we need to consider all the forces acting on the robot. The forces in this case are gravity and contact forces.

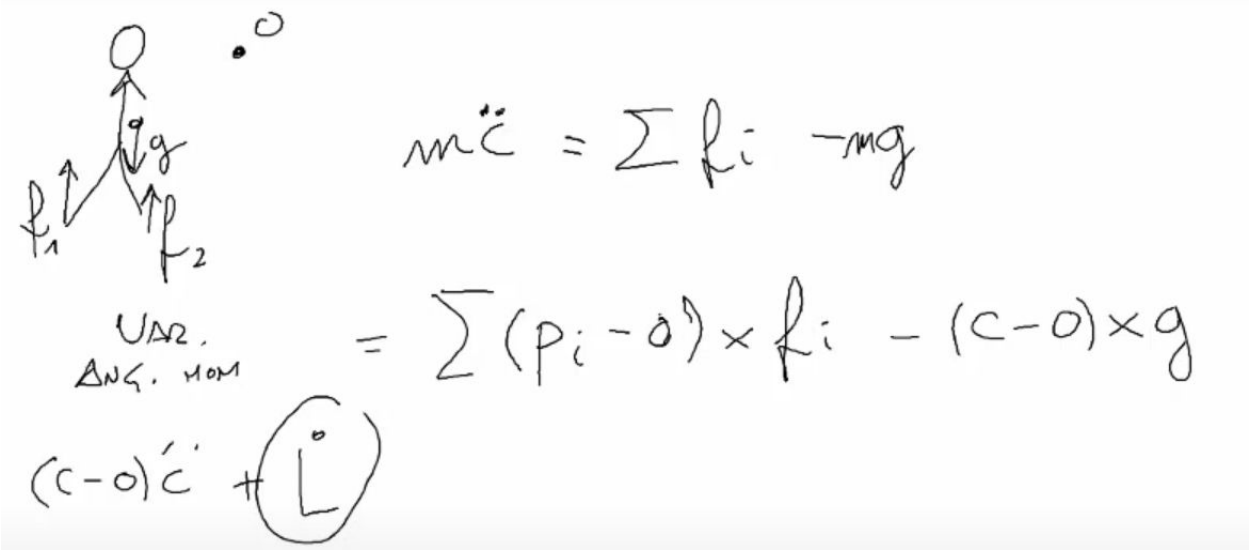
$$M\ddot{\boldsymbol{c}}=\sum_i \boldsymbol{f}_i - M\boldsymbol{g}$$

Where c is CoM position and M is total mass of the system. Hence, we need contact forces to move the CoM in a direction different from that of gravity.

Euler equation

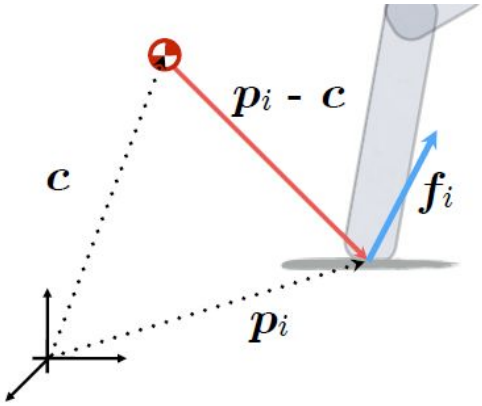
Previous case was for force balance, and this case is for moment balance. We have the same thing: two forces acting on the robot which are the contact forces f_i and gravity. P_i-o is the moment of the contact forces

$$(\boldsymbol{c}-\boldsymbol{o})\times M\ddot{\boldsymbol{c}}+\dot{\boldsymbol{L}}=\sum_i(\boldsymbol{p}_i-\boldsymbol{o})\times\boldsymbol{f}_i-(\boldsymbol{c}-\boldsymbol{o})\times M\boldsymbol{g}$$



Newton-Euler equations

As we have said before the moment of a force (or torque) is a measure of its tendency to cause a body to rotate about a specific point or axis.



$$(\boldsymbol{p}_i - \boldsymbol{c}) \times \boldsymbol{f}_i$$

moment generated by the contact force $\boldsymbol{f}_i$ around the CoM

To compute the angular momentum around the CoM we need to sum all the robot link momentums like in following formula

$$\boldsymbol{L} = \sum_k (\boldsymbol{x}_k - \boldsymbol{c}) \times m_k \dot{\boldsymbol{x}}_k + I_k \boldsymbol{\omega}_k$$

$\boldsymbol{\omega}_k$: angular velocity of the k-th link

Zero Moment Point

This case happens when *o is zero* where the point with respect to which the moment of the contact forces is zero. We denote this point by *z*.

Newton-Euler on flat ground

So, starting from this we can make some computations by knowing the Newton and Euler equations which is $M(c^z + g^z)\sum_i f_i^z$ which leads to

$$\frac{M(\boldsymbol{c}-\boldsymbol{z})\times(\ddot{\boldsymbol{c}}+\boldsymbol{g})+\dot{\boldsymbol{L}}}{m(\ddot{c}^z+g)} = \frac{\sum_i(\boldsymbol{p}_i-\boldsymbol{z})\times\boldsymbol{f}^i}{\sum_i f_i}$$

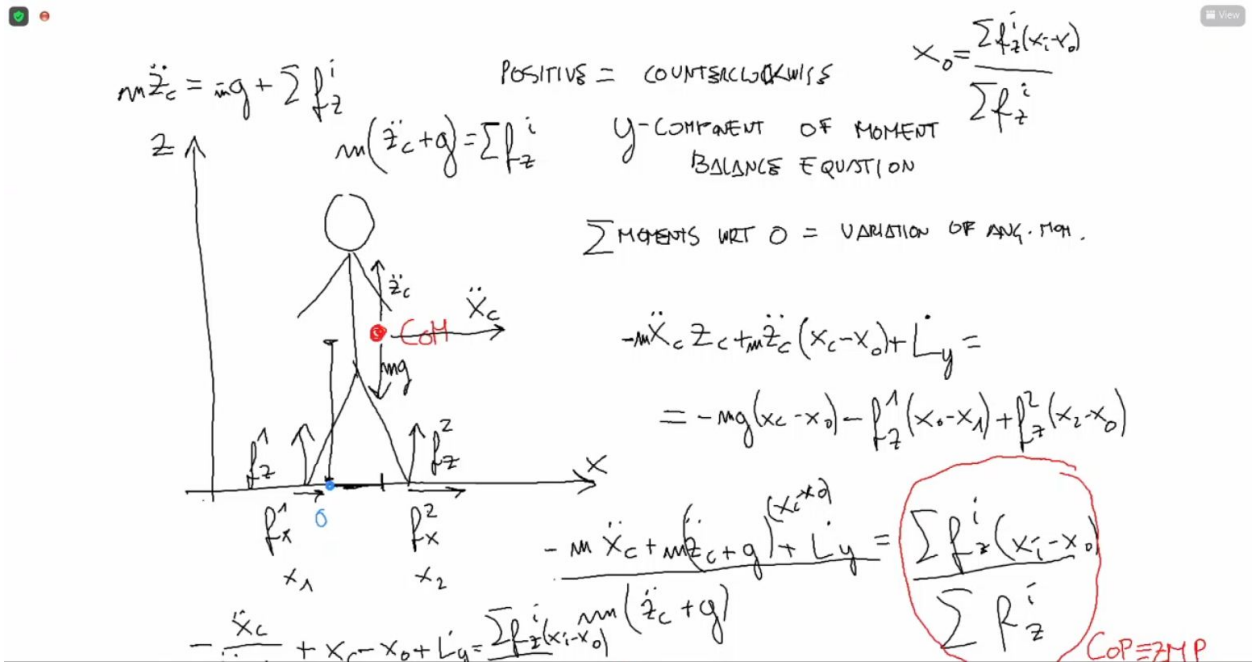
Lecture 19 - Humanoid Robots 2 continuation and 3

In the previous lecture we learned about modeling which can be Lagrange (floating base = COM) and Newton-Euler equations. y-component of Moment Balance Equations. We look at y because we are in the x,y plane. In the following picture we describe the notion of momentum. Moment balance says that the sum of moments with respect to point O is equal to the variation of angular momentum. The formula is as follows:

$$\boldsymbol{c}^{x,y} - \frac{c^z}{\ddot{c}^z+g^z}(\ddot{\boldsymbol{c}}^{x,y}+\boldsymbol{g}^{x,y}) + \frac{\boldsymbol{S}\dot{\boldsymbol{L}}^{x,y}}{M(\ddot{c}^z+g^z)} = \frac{\sum_i f_i^z \boldsymbol{p}_{\boldsymbol{i}}^{x,y}}{\sum_i f_i^z}$$

In this case: c^x,y is the center of mass position, g^x,y is horizontal line gravity.

We can find acceleration of the center of mass by getting the minus of COM with ZMP (c-z).



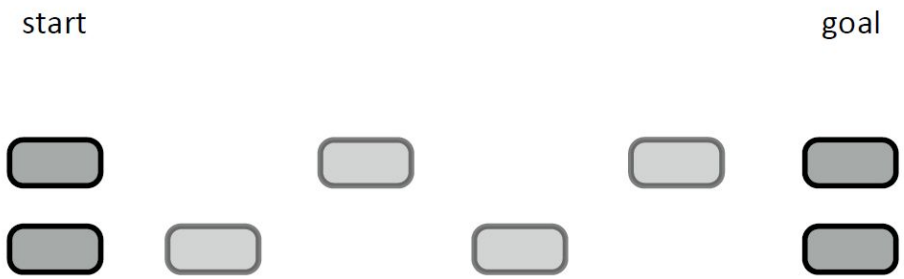
Point foot is the simplest interpretation of the LIP is that of a telescoping massless leg in contact with the ground at p^x.

----------3RD PART STARTS AT 1:01----------

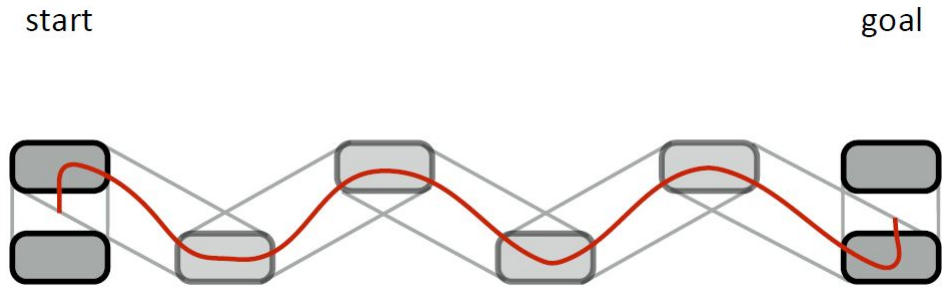
Lecture 20 - Humanoid Robots: Gait Generation

So, by gait generation we mean that we want to generate a gait which is a way for robots to walk ***while maintaining the balance*** which is also called walking pattern generation. We will be talking about the way to generate gait generation which the best one is ZMP-based. ZMP-based gait generation is an algorithm based on the strategy where we need to keep the ZMP inside the Support Polygon (SP). There are several steps for that:

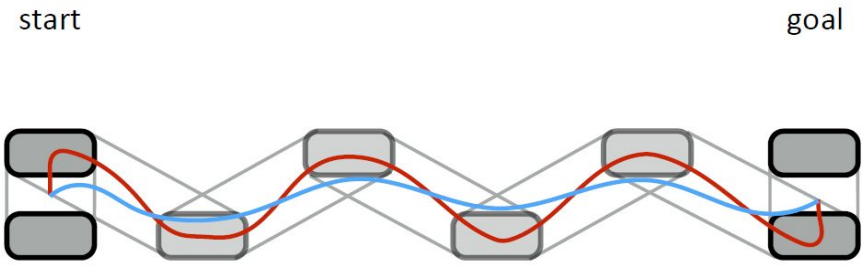
1. plan the footsteps
    a. we want to plan the footsteps because it gives the shape of polygon overtime
    b. let say we want to go from start to goal position with some average speed, we can say $v=L/T$ where L is the distance between the movement of feet



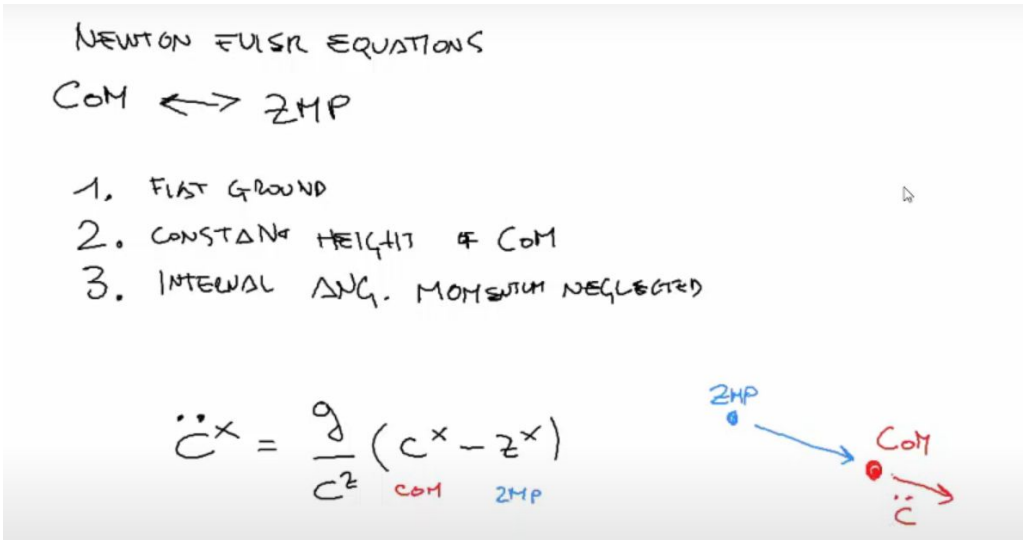
2. Once we have the footsteps and shape of polygon, we can now plan a ZMP trajectory that is always the SP
    a. it's about interpolation



3. then we need to compute a COM trajectory such that the ZMP moves as planned
    a. we need to use the LIP model in this case



4. then, we track the COM trajectory
    a. define a swinging foot trajectory
    b. use kinematic control to obtain reference joint trajectories that realize the COM and foot trajectories
    c. send the reference joint profiles to the joint servos (for a position-controlled humanoid)



Instability problem: a control perspective

Pendulum as we know is a stable model and it has two eigenvalues which are minus and plus omega. Minus omega is stable and another one is unstable.