0. Basics

Homogeneous Coordinates

A representation x of a geometric object is homogeneous, if:

$$\mathbf{x} = \lambda \mathbf{x} \rightarrow \text{describe the same object}$$

Allows to represent affine and projective transformations through a single matrix.

Transformation:

$$\mathbf{x} = \begin{bmatrix} U \\ V \\ W \\ T \end{bmatrix} = \begin{bmatrix} U/T \\ V/T \\ W/T \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} U/T \\ V/T \\ W/T \end{bmatrix}$$

homogeneous → Euclidian

→ n-dimensional euclidean vector correspond to (n+1)-dimensional homogeneous coordinates

3D Transformations

Translation:

$$\mathsf{H} = \begin{bmatrix} I & t \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix}, \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \quad \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

3 params: 3 translation

Rotation:

$$\mathsf{H} = \begin{bmatrix} R & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix}$$

3 params: 3 rotation

Rigid Body Transformation:

$$\mathsf{H} = \begin{bmatrix} R & t \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix}$$

6 params: 3 rotation + 3 translation

Similarity Transformation:

$$\mathsf{H} = \begin{bmatrix} mR & t \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix}$$

7 params: 3 rotation + 3 translation + 1 scale

Affine Transformation:

$$\mathsf{H} = \begin{bmatrix} A & t \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix}$$

12 params: 3 translation + 3 rotation + 3 scale + 3 sheer

Rotation Matrices

2D:

$$R^{2D}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

3D:

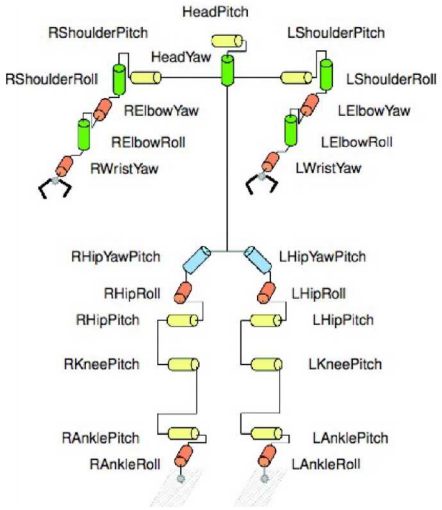
$$R_x^{3D}(\omega) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) \\ 0 & \sin(\omega) & \cos(\omega) \end{bmatrix} \quad R_y^{3D}(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

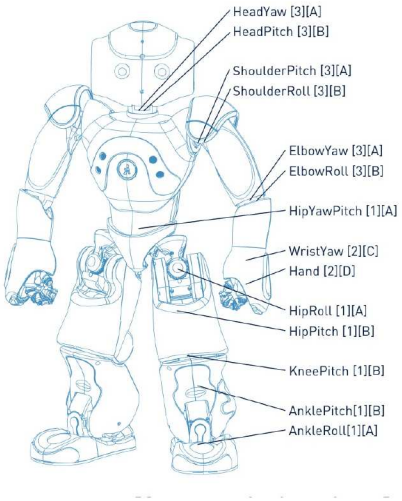
$$R_z^{3D}(\kappa) = \begin{bmatrix} \cos(\kappa) & -\sin(\kappa) & 0 \\ \sin(\kappa) & \cos(\kappa) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R^{3D}(\omega, \phi, \kappa) = R_z^{3D}(\kappa) R_y^{3D}(\phi) R_x^{3D}(\omega)$$

Nao - Robot:

Kinematic Structure



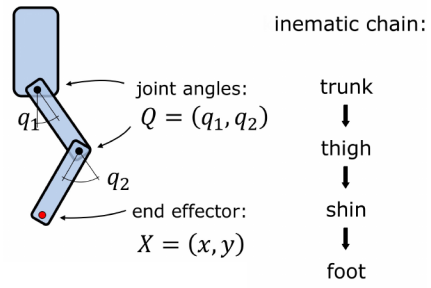


Kinematics

Kinematic Chain:

Chain of joints connected through rigid links

Each joint is transformed relative to the parent joint



Forward Kinematics: $X = f(Q)$

joint angles (measurement) → joint position (end effector)

Follow kinematic chain to retrieve the end-effector position from the angles:

$$P^0 = T_1^0(q_1, l_1) \cdot T_2^1(q_2, l_2) \cdot P^2$$

$$P^0 = T_2^0 \cdot P^2$$

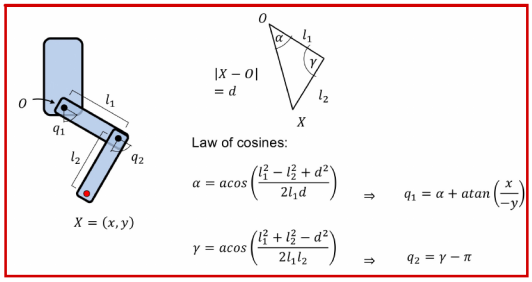
Inverse Kinematics: $Q = f^{-1}(X)$

joint position (end effector) → joint angles (measurement)

Ill-posed problem that has multiple solutions

Analytical Method:

Trigonometry:



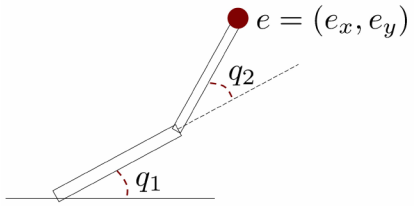
→ computes all solutions
→ equations have to be derived for different kinematic structures
→ fast to compute

Numerical Method:

Given a start configuration, iteratively calculate a sequence that leads to the end-effector pose
→ prone to local minima

Jacobian: $\mathbf{J}(\mathbf{e}, \mathbf{q}) = \begin{pmatrix} \frac{\partial e_x}{\partial q_1} & \frac{\partial e_x}{\partial q_2} \\ \frac{\partial e_y}{\partial q_1} & \frac{\partial e_y}{\partial q_2} \end{pmatrix}$ $[6 \times N]$

→ change of end-effector w.r.t the joint angles



Incremental update:

$$\mathbf{J}\Delta\mathbf{q} = \Delta\mathbf{e}$$

$$\Delta\mathbf{q} = \mathbf{J}^{-1}\Delta\mathbf{e}$$

with: $\Delta\mathbf{e} = \alpha(\mathbf{g} - \mathbf{e}), 0 \leq \alpha \leq 1$

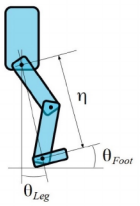
Algorithm:

```
while (e is too far from g) {
    Compute J(e, q) for the current configuration q
    Compute J^-1
    Δe = α(g − e)          // choose a step to take
    Δq = J^-1 Δe           // compute required change in joints
    q = q + Δq             // apply change to joints
    Compute resulting e    // by FK
}
```

Abstract Kinematics: $Q = g(X')$

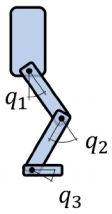
Model the leg angles through abstract parameters that are easier to interpret:

input:
$\eta, \theta_{Leg}, \theta_{Foot}$



$$\alpha = a\cos\eta$$
$$q_1 = \alpha + \theta_{Leg}$$
$$q_2 = -2\alpha$$
$$q_3 = \alpha - \theta_{Leg} + \theta_{Foot}$$

output:
q_1, q_2, q_3



Singularities

Certain changes of the end-effector pose lead to large changes in the joint angles
→ dangerous for IK
→ erratic / uncontrolable movements

1. Least Squares and Sensor Calibration

Definition: Odometry

Estimation of the change of robot position from internal motion sensors
→ systematic errors cause a drift
→ prior calibration is required

Definition: Least Squares

An approach for computing a solution for an overdetermined system by minimizing the sum of the squared errors

Linear System: Closed-form solution

Nonlinear System: Iterative Optimization

Sensor Calibration

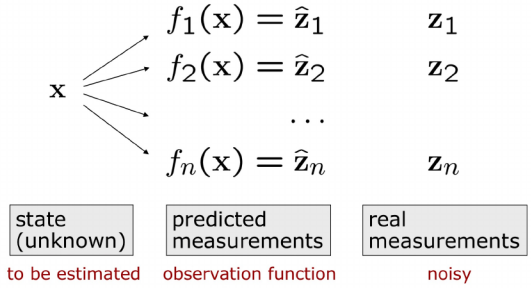
Problem Definition:

Given: $z_{1:n}$: n noisy measurements of state x

Goal: Estimate x that best explains $z_{1:n}$, by minimizing the global error:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \mathbf{e}_i^T(\mathbf{x}) \Omega_i \mathbf{e}_i(\mathbf{x})$$

System:



Error: $\mathbf{e}_i(\mathbf{x}) = \mathbf{z}_i - f_i(\mathbf{x})$

Squared Error: $e_i(\mathbf{x}) = \mathbf{e}_i(\mathbf{x})^T \Omega_i \mathbf{e}_i(\mathbf{x})$

Ω_i: Information matrix for gaussian error with zero mean and normally distributed

Assumptions:

1. "Good" initial guess is available
2. Error functions are smooth around the global/local minimum
3. Error is normal distributed

Gauss-Newton Method

Iterative solving of nonlinear system using local linearization:

1. Linearize error function around X:

$$\mathbf{e}_i(\mathbf{x}+\Delta\mathbf{x}) \simeq \mathbf{e}_i(\mathbf{x}) + \mathbf{J}_i(\mathbf{x})\Delta\mathbf{x}$$

2. Compute terms for the linear system:

$$\mathbf{b}^T = \sum_i \mathbf{e}_i^T \Omega_i \mathbf{J}_i \qquad \mathbf{H} = \sum_i \mathbf{J}_i^T \Omega_i \mathbf{J}_i$$

3. Solve the linear system:

$$\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$$

4. Update State:

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^*$$

Derivation:

1. Approximate error functions via Taylor expansion:

$$\mathbf{e}_i(\mathbf{x}+\Delta\mathbf{x}) \simeq \underbrace{\mathbf{e}_i(\mathbf{x})}_{\mathbf{e}_i} + \mathbf{J}_i(\mathbf{x})\Delta\mathbf{x}$$

2. Insert errors in squared error terms

$$\begin{aligned}
e_i(\mathbf{x}+\Delta\mathbf{x}) &= \mathbf{e}_i^T(\mathbf{x}+\Delta\mathbf{x})\Omega_i\mathbf{e}_i(\mathbf{x}+\Delta\mathbf{x}) \\
&\simeq (\mathbf{e}_i + \mathbf{J}_i\Delta\mathbf{x})^T\Omega_i(\mathbf{e}_i+\mathbf{J}_i\Delta\mathbf{x}) \\
&= \mathbf{e}_i^T\Omega_i\mathbf{e}_i + \\
&\quad \mathbf{e}_i^T\Omega_i\mathbf{J}_i\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{J}_i^T\Omega_i\mathbf{e}_i + \\
&\quad \Delta\mathbf{x}^T\mathbf{J}_i^T\Omega_i\mathbf{J}_i\Delta\mathbf{x} \\
&= \underbrace{\mathbf{e}_i^T\Omega_i\mathbf{e}_i}_{c_i} + 2\underbrace{\mathbf{e}_i^T\Omega_i\mathbf{J}_i}_{\mathbf{b}_i^T}\Delta\mathbf{x} + \Delta\mathbf{x}^T\underbrace{\mathbf{J}_i^T\Omega_i\mathbf{J}_i}_{\mathbf{H}_i}\Delta\mathbf{x} \\
&= c_i + 2\mathbf{b}_i^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}_i\Delta\mathbf{x}
\end{aligned}$$

3. Define Global Error

$$\begin{aligned}
F(\mathbf{x}+\Delta\mathbf{x}) &\simeq \sum_i \left(c_i + 2\mathbf{b}_i^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}_i\Delta\mathbf{x}\right) \\
&= \underbrace{\sum_i c_i}_{c} + 2\underbrace{\left(\sum_i \mathbf{b}_i^T\right)}_{\mathbf{b}^T}\Delta\mathbf{x} + \Delta\mathbf{x}^T\underbrace{\left(\sum_i \mathbf{H}_i\right)}_{\mathbf{H}}\Delta\mathbf{x} \\
&= c + 2\mathbf{b}^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}
\end{aligned}$$

with

$$\mathbf{b}^T = \sum_i \mathbf{e}_i^T\Omega_i\mathbf{J}_i \qquad \mathbf{H} = \sum_i \mathbf{J}_i^T\Omega_i\mathbf{J}_i$$

4. Compute derivative of global error

$$\frac{\partial F(\mathbf{x}+\Delta\mathbf{x})}{\partial\Delta\mathbf{x}} \simeq 2\mathbf{b} + 2\mathbf{H}\Delta\mathbf{x}$$

First derivative of quadratic form

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$$

$$\frac{\partial f}{\partial \mathbf{x}} = \mathbf{H}^T\mathbf{x} + \mathbf{H}\mathbf{x} + \mathbf{b} = (\mathbf{H}+\mathbf{H}^T)\mathbf{x} + \mathbf{b}$$

5. Solve for minimum

$$\begin{aligned}
0 &= 2\mathbf{b} + 2\mathbf{H}\Delta\mathbf{x} \\
\Leftrightarrow \mathbf{H}\Delta\mathbf{x} &= -\mathbf{b} \\
\Leftrightarrow \Delta\mathbf{x}^* &= -\mathbf{H}^{-1}\mathbf{b}
\end{aligned}$$

Application: Odometry Calibration

Given: Noisy odometry measurements $\mathbf{u}_i = (u_{i,x}, u_{i,y}, u_{i,\theta})^T$

Goal: Find parameters x for function $f_i(x)$ to return corrected odometry:

$$\mathbf{u}_i' = f_i(\mathbf{x}) = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \mathbf{u}_i$$

Solution: Solve least squares problem using the Gauss-Newton method with:

The state vector is

$$\mathbf{x} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} & x_{31} & x_{32} & x_{33} \end{pmatrix}^T$$

The error function is

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{u}_i^* - \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \mathbf{u}_i$$

Accordingly, its Jacobian is

$$\mathbf{J}_i = \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{x}} = -\begin{pmatrix} u_{i,x} & u_{i,y} & u_{i,\theta} & & & & & & \\ & & & u_{i,x} & u_{i,y} & u_{i,\theta} & & & \\ & & & & & & u_{i,x} & u_{i,y} & u_{i,\theta} \end{pmatrix}$$

2. Cameras and Calibration

Definition: Projective Mapping

A mapping, such that:

1. Straight lines stay straight
2. Parallel lines may intersect

Image courtesy: Förstner

Pinhole Camera Model

Simple camera model using basic physics to project a point through an infinitesimal small hole onto the image plane.

Assumptions:

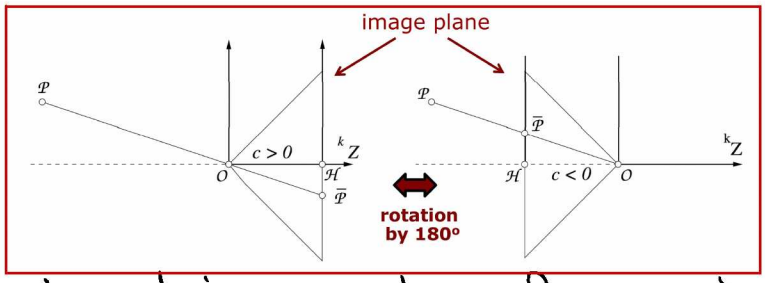
1. All rays intersect in a single point
2. All image points lie on a plane
3. The ray from the object to image point is a straight line

Properties:

1. Line-preserving: straight lines stay straight
2. Not angle-preserving: angles between lines change
3. Not length-preserving: size of objects is inverse proportional to the distance

Method:

Physical-motivated: Most popular:



→ invert image plane for easier computation

Camera constant c: focal length

Camera Calibration

Calibration describes the determination of the intrinsic parameters to define the calibration matrix for the projection from the camera to the sensor frame

Calibration Matrix:

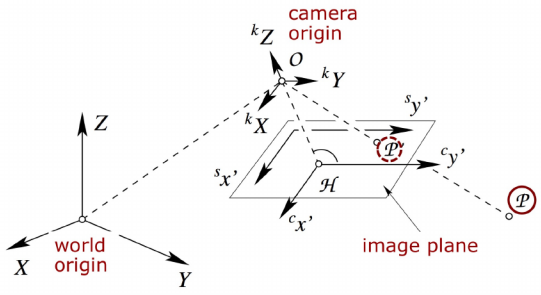
$$ {}^{a}\mathsf{K}(\boldsymbol{x},\boldsymbol{q}) = \begin{bmatrix} c & 0 & x_H + \Delta x(\boldsymbol{x},\boldsymbol{q}) \\ 0 & c(1+m) & y_H + \Delta y(\boldsymbol{x},\boldsymbol{q}) \\ 0 & 0 & 1 \end{bmatrix} $$

General Projection:

$$ {}^{a}\mathbf{x} = {}^{a}\mathsf{P}(\boldsymbol{x},\boldsymbol{q})\ \mathbf{X} $$

$$ {}^{a}\mathsf{P}(\boldsymbol{x},\boldsymbol{q}) = {}^{a}\mathsf{K}(\boldsymbol{x},\boldsymbol{q})\ R[I|-\boldsymbol{X}_O] $$

World-to-Sensor Transformation



Coordinate Frames:

World S_o: $[X, Y, Z]^\mathsf{T}$

Camera S_k: $[{}^kX, {}^kY, {}^kZ]^\mathsf{T}$

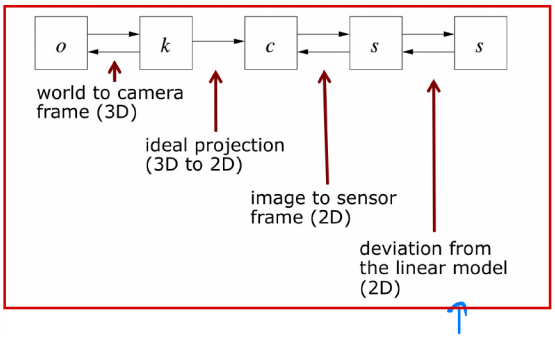
Image S_c: $[{}^cx, {}^cy]^\mathsf{T}$

Sensor S_s: $[{}^sx, {}^sy]^\mathsf{T}$

Transformation:

$$\begin{bmatrix} {}^sx \\ {}^sy \\ 1 \end{bmatrix} = {}^s\mathsf{H}_c \, {}^c\mathsf{H}_k \, {}^k\mathsf{H}_o \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

in the sensor frame | image to sensor | camera to image | world to camera | in the world frame



intrinsics | extrinsics

additional compensation for distortion

Extrinsics: Camera pose

Intrinsics: Projection onto image plane

Transformation Chain:

1. **World → Camera:**

$${}^k\mathbf{X}_P = {}^k\mathsf{H}\,\mathbf{X}_P \quad \text{with} \quad {}^k\mathsf{H} = \begin{bmatrix} R & -R\boldsymbol{X}_O \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix}$$

2. **Camera → Image:**

$${}^c\mathbf{x}_{\overline{P}} = \begin{bmatrix} {}^cu_{\overline{P}} \\ {}^cv_{\overline{P}} \\ {}^cw_{\overline{P}} \end{bmatrix} = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} {}^kX_P \\ {}^kY_P \\ {}^kZ_P \\ 1 \end{bmatrix}$$

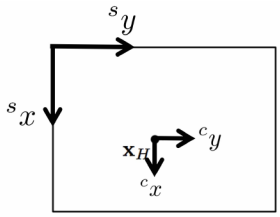
→ projection from camera to image frame
→ typically non-invertable

3. **Image → Sensor:**

$${}^s\mathbf{x} = {}^s\mathsf{H}_c \, {}^c\mathsf{K} R [I_3 | -\boldsymbol{X}_O] \mathbf{X}$$

$${}^s\mathsf{H}_c = \begin{bmatrix} 1 & 0 & x_H \\ 0 & 1+m & y_H \\ 0 & 0 & 1 \end{bmatrix}$$

m: scale difference based on chip design
x_H, y_H: offset between image and sensor frame



4. **Compensate for non-linear errors:**
→ add location-dependent shift to each pixel

$${}^a\mathbf{x} = {}^a\mathsf{H}_s(\boldsymbol{x}, \boldsymbol{q}) \; {}^s\mathbf{x}$$

$${}^a\mathsf{H}_s(\boldsymbol{x}, \boldsymbol{q}) = \begin{bmatrix} 1 & 0 & \Delta x(\boldsymbol{x}, \boldsymbol{q}) \\ 0 & 1 & \Delta y(\boldsymbol{x}, \boldsymbol{q}) \\ 0 & 0 & 1 \end{bmatrix}$$

General Mapping:

$$\begin{aligned} {}^ax &= {}^sx + \Delta x(\boldsymbol{x}, \boldsymbol{q}) \\ {}^ay &= {}^sy + \Delta y(\boldsymbol{x}, \boldsymbol{q}) \end{aligned}$$

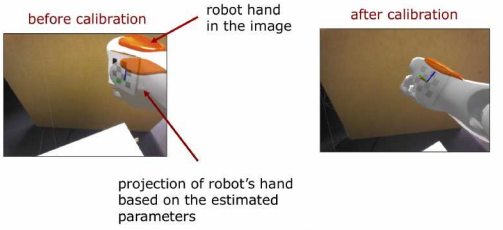
in the image frame

Distortion:

$$\begin{aligned} {}^ax &= x(1 + q\,r^2) \\ {}^ay &= y(1 + q\,r^2) \end{aligned}$$

r: distance to image center
q: additional parameter

Whole-Body Self-Calibration



Parameters to calibrate:

Joint offsets: $\mathbf{q}^{off} = \begin{pmatrix} q_1^{off} \\ q_2^{off} \\ \vdots \\ q_n^{off} \end{pmatrix}$ True position: $q = \hat{q} + q^{off}$

Camera extrinsics: $R,\ t$

Camera intrinsics: c, m, x_H, y_H, q

Calibration Setup (Nao)

Calibration Markers: placed at each of the four end-effectors:

$$\mathbf{m}_{EEF_i} = [x_i, y_i, z_i]^T$$

Parameters: $\theta \in \mathbb{R}^{41}$

Joint offsets: 18 (23 joints - 5 redundant)
Extrinsics: 6
Intrinsics: 5
End-effector poses: 4 × 3 = 12

Least-Squares Optimization

Base: n different robot configurations with encoder readings $\hat{q}$
→ sample configuration space uniformly
→ check, whether marker is in FOV and for self-collisions

Error function: $\mathbf{e}_i(\theta, \mathbf{z}_i, \hat{\mathbf{q}}_i) = \mathbf{z}_i - predictmarker_{M_{EEF}}(\theta, \hat{\mathbf{q}}_i)$

$\mathbf{z}_i$: marker position in image
$predictmarker_{M_{EEF}}(\theta, \hat{\mathbf{q}}_i)$: predicted marker position given by the kinematic structure

Predicted Marker:

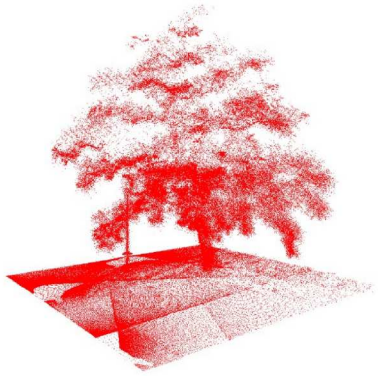
$$predictmarker_{M_{EEF}}(\theta, \hat{\mathbf{q}}_i) = \mathrm{K}_\theta [R_\theta | -R_\theta \mathbf{t}_\theta] \mathcal{F}_{EEF}^N(\theta, \hat{\mathbf{q}}_i) \tilde{\mathbf{m}}_{EEF}$$

K_θ: intrinsics
$[R_\theta | -R_\theta \mathbf{t}_\theta]$: extrinsics
$\mathcal{F}_{EEF}^N(\theta, \hat{\mathbf{q}}_i)$: forward kinematics

Solution: Iterative optimization using Gauss-Newton

3. 3D World Representations

Point Clouds



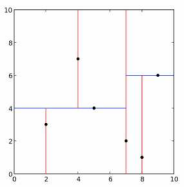
Set of 3D points.

Pros:

- No discretization
- Mapped area is not limited

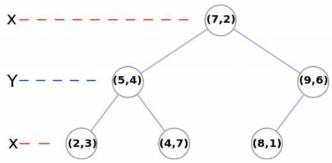
Cons:

- Unbounded memory usage
- No constant time access for location queries
- No distinction between free or unknown space



Kd-trees

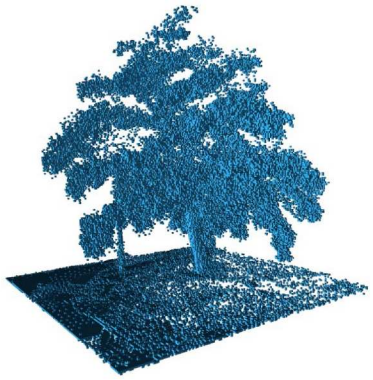
Order point clouds in kd-tree data structure



Pros:

- Search/insert/delete in logarithmic time

3D Voxel Grids



Represent a map through regularly spaced voxels in a grid.

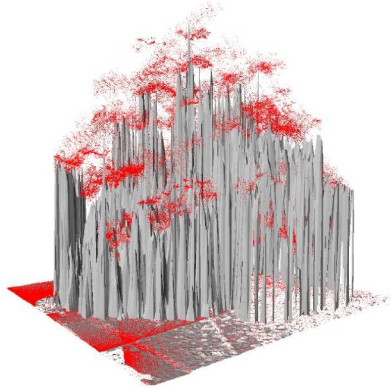
Pros:

- Volumetric representation
- Constant access time
- Probabilistic update possible

Cons:

- Memory requirement: complete grid is allocated in memory
- Extent of map is pre-defined
- Discretization errors

2.5D Maps: Height Maps



Represent map as 2D cells and consider the average height over all points as the height value.

Pros:

- Memory efficient (2D)
- Constant time access

Cons:

- No vertical objects
- Only one level is represented

Multi-Level Surface Maps

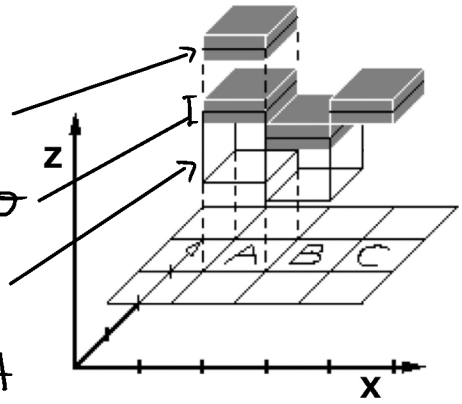
Represent the map through 2D cells that each consist of multiple patches

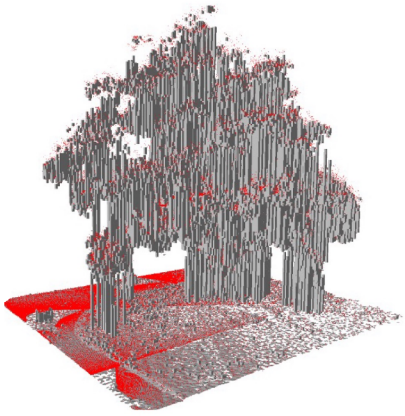
Cell: set of patches

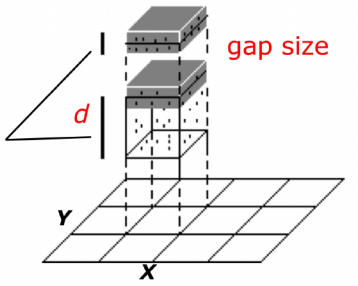
Patch: height mean μ

height variance σ

depth value d
→ diff between lowest & highest







Convert from point cloud:

1. Determine 2D cell for each 3D point
2. Compute vertical intervals based on a threshold
3. Determine heigh mean, variance and depth

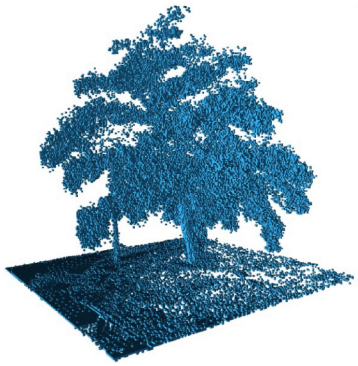
Pros:

- Represent multiple surfaces per cell

Cons:

- No volumetric representation but a discretization in the vertical dimension
- Several tasks are not straightforward to realize

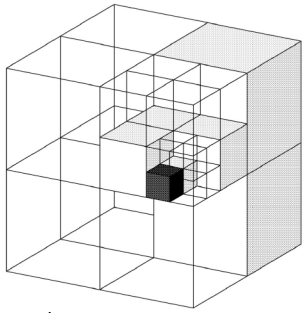
Octree Representation

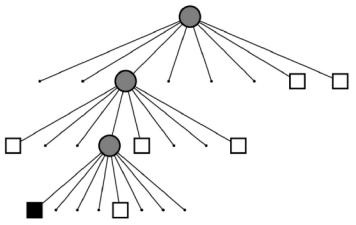


Tree-based structure that recursively divides the space in octants
→ Split until max depth to insert point
→ Merge upwards if all octants are free or occupied
→ volume and resolution allocated as needed

Pros:

- 3D model
- Multi-resolutional
- Memory efficient
- Probabilistic update possible



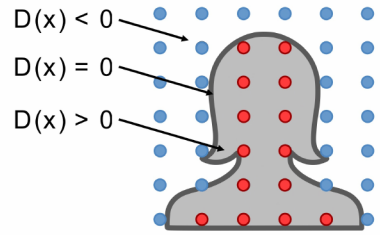


Cons:

- Efficient implementation can be tricky

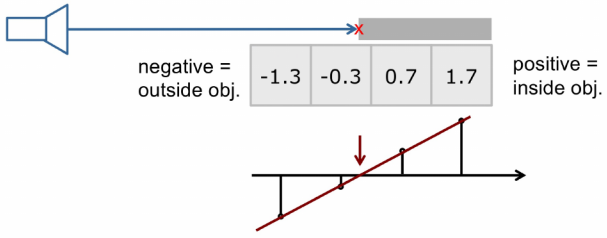
Signed Distance Function

Represent the map through a voxel grid where each voxel represents its distance to the nearest measured surface
→ Surfaces are located at zero-crossings
→ surface extraction at sub-voxel accuracy through interpolation



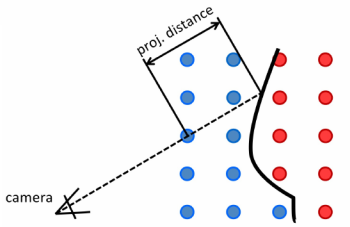
SDF:

Negative: outside object
Positive: inside object
Zero-crossing: surface



Truncated SDF:

Distance of each voxel to the observed surface is computed and only close voxels are updated.



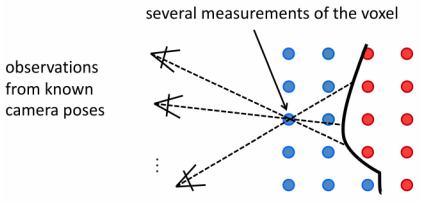
Weighted Update

For each voxel compute weighted average over all measurements

Computation (each cell):

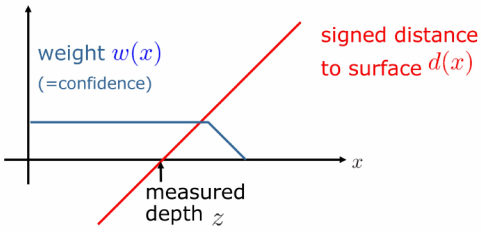
$$D \leftarrow \frac{WD + w_t d_t}{W + w_t}$$
$$W \leftarrow W + w_t$$

D: Weighted sum of signed distances
W: Sum of all weights



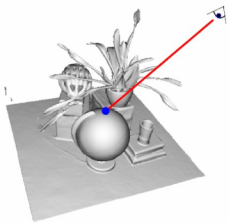
Weight function:

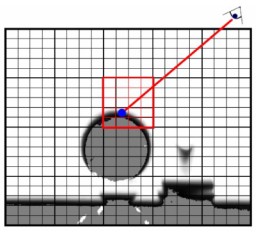
Weight function should represent confidence and relevance
→ small values ensure that values can be updated by new observations



Surface Rendering

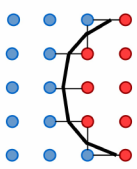
Ray Casting: Cast ray and find zero-crossing
→ fast computation on GPU

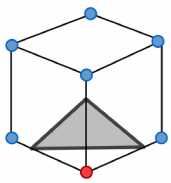




Polygonization: Marching cubes algorithm to generate a triangle mesh

→ evaluate each cell seperately and generate triangles according to a look-up table





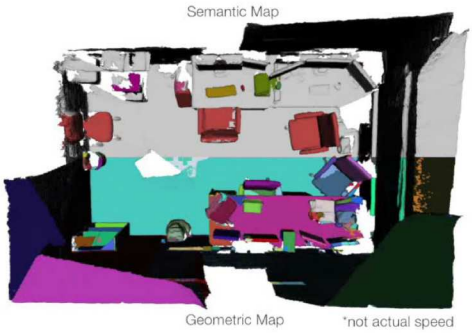
Pros:

- Full 3D model
- Sub-voxel accuracy
- supports fast GPU implementations

Cons:

- Space-consuming voxel grid
- Polygonization: slow

Semantic Aware Volumetric Mapping



Combine semantic scene understanding and combine it with a geometric representation

→ Use deep learning for semantic understanding

Semantic Segmentation:

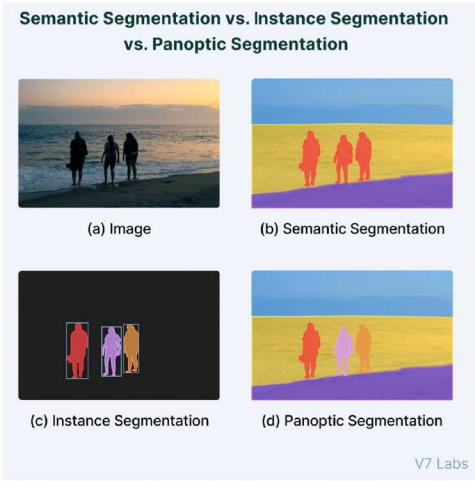
Assign each pixel a class label

Instance Segmentation:

Find instances in the foreground and label pixels of each instance with a unique Id.

Panoptic Segmentation:

Combination of semantic and instance segmentation.



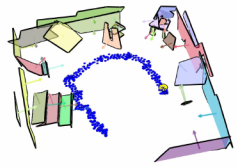
Pros:

- Semantic rich maps
- Enables semantic based planning and interaction
- Closer to how humans abstract their environment

Cons:

- Requires large neural networks
- Complex and expensive

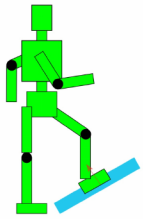
3D Polygonal Maps



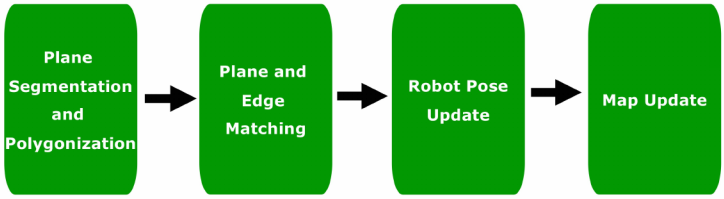
Represent the map through 3D polygonal planes. Each plane can be represented through 4 parameters and a boundary.

→ Empty space represented implicitely

→ Footstep location can be extracted from polygons directly



Iterative Map Update:

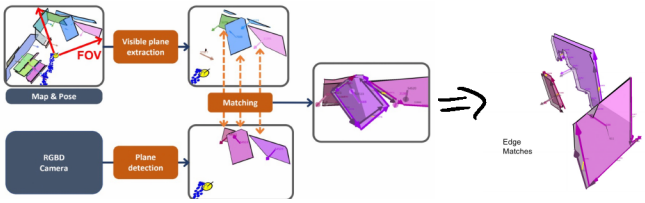


Plane Segmentation and Polygonization:

1. Segment RGB-D image
2. Polygonize using contour detection
3. Identify and remove ground plane

Plane Matching:

Match planes via normals



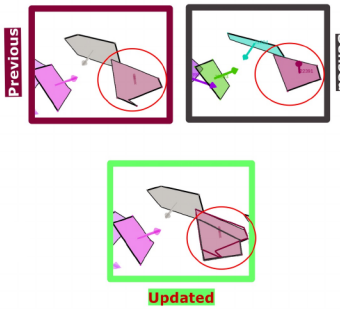
Edge Matching:

Match edges between two previously matched planes

Pose Update:

Sample covensus across all matched planes

Map Update:



1. Compute expected planes visible in the updated pose
2. Re-compute plane matches
3. Update polygon parameters and add new polygons

Pros:

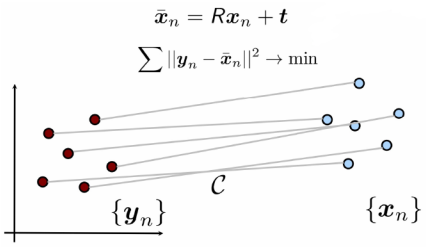
- Footstep placement directly known from map
- Memory efficient
- Fast to compute

Cons:

- Coarse representation
- General objects are not well represented

Iterative Closest Point

ICP registers a point cloud obtained from a Lidar scan or RGB-D camera against a second scan. Iteratively, correspondences in the point clouds are established and alligned.



Algorithm: Find transform $T=(R,t)$ to transform $P \longrightarrow X$

```
0: WHILE (not finished)
1:   Select points in the point sets
2:   IF (termination criterion)
3:     Return R, t
4:   Establish correspondences between the points in X and P
5:   Find R,t that minimizes: E(R,t) = 1/N_p Σ_{i=1}^{N_p} ||x_i − R p_i − t||^2
                                  x_i and p_i are corresponding points
6:   Transform P
```

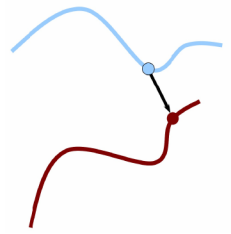
Sub-sample Point Clouds:

1. All points
2. Uniform sub-sampling
3. Random sampling
4. Feature-based sampling
   → sample out floor points
5. Normal-space sampling
   → Normals are distributed uniformly

Data Association:

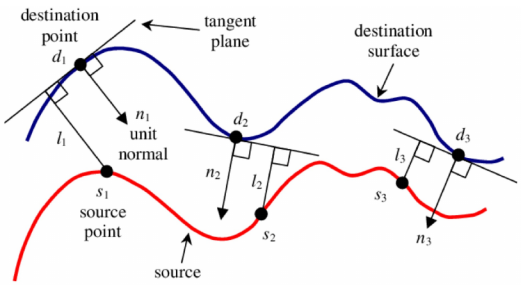
Closest-Point Matching:

Take the closest point in the other point cloud as correspondence.



Point-to-Plane Matching

Compute the distance of each source point to the tangent space of each target and make correspondences accordingly.



Compute Transformation

1. Mean Shift

$$\mu_Q = \frac{1}{|C|}\sum_{(i,j)\in C} q_i \qquad \mu_P = \frac{1}{|C|}\sum_{(i,j)\in C} p_j \longrightarrow \begin{aligned} Q' &= \{q_i - \mu_Q\} = \{q'_i\} \\ P' &= \{p_j - \mu_P\} = \{p'_j\} \end{aligned}$$

2. Compute cross-covariance matrix

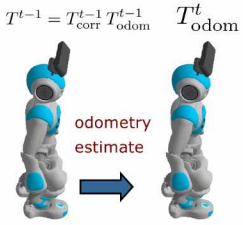
$$W = \sum_{(i,j)\in C} q'_i p'^{\mathsf{T}}_j$$

3. Apply SVD

$$W = UDV^{\mathsf{T}}$$

4. Compute Transformation

$$R = UV^{\mathsf{T}} \quad t = \mu_Q - R\mu_P$$

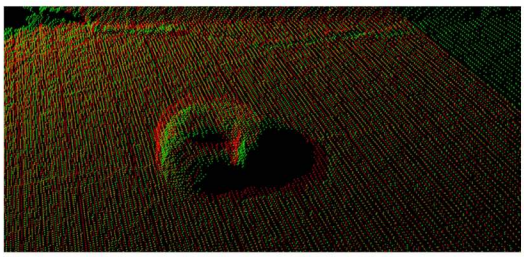


Application: Refine odometry measurements

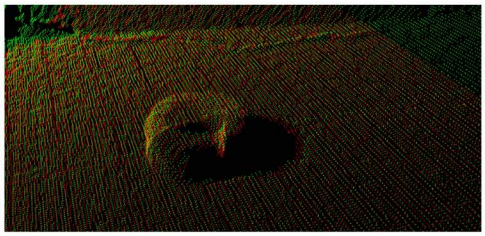
Initialize ICP with odometry measurement and iteratively correct the transformation to prevent accumulative errors:

Corrected pose $T^t = T^t_{\text{corr}} T^t_{\text{odom}}$ given by

$$T^t_{\text{corr}} = \text{argmin}_{T'} \sum_i \|T^{t-1}\mathbf{q}_i - T' T^t_{\text{odom}} \mathbf{p}_i\|^2$$



$T^t = T^t_{\text{corr}} T^t_{\text{odom}}$



4. Localization

Bayes Filtering

Bayes Filtering Formula:

$$Bel(x) = \eta\, P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

Correction | Prediction

Prediction:

$$\overline{Bel}(x_t) = \int P(x_t \mid u_t, x_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

Action Model

Correction:

$$Bel(x_t) = \eta\, P(z_t \mid x_t)\, \overline{Bel}(x_t)$$

Sensor Model

Derivation:

$$\begin{aligned}
Bel(x_t) &= P(x_t \mid u_1, z_1, \ldots, u_t, z_t) \\
&= \eta\, P(z_t \mid x_t, u_1, z_1, \ldots, u_t)\, P(x_t \mid u_1, z_1, \ldots, u_t) && \text{| Bayes} \\
&= \eta\, P(z_t \mid x_t)\, P(x_t \mid u_1, z_1, \ldots, u_t) && \text{| Markov} \\
&= \eta\, P(z_t \mid x_t) \int P(x_t \mid u_1, z_1, \ldots, u_t, z_t, x_{t-1})\, P(x_{t-1} \mid u_1, z_1, \ldots, u_t)\, dx_{t-1} && \text{| Total Prob} \\
&= \eta\, P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\, Bel(x_{t-1})\, dx_{t-1} && \text{| Markov}
\end{aligned}$$

Algorithm:

1. Algorithm **Bayes_filter**(*Bel(x), d*):
2. $\eta = 0$
3. If *d* is a **perceptual** data item *z* then
4. For all *x* do
5. $Bel'(x) = P(z \mid x)\, Bel(x)$
6. $\eta = \eta + Bel'(x)$
7. For all *x* do
8. $Bel'(x) = \eta^{-1}\, Bel'(x)$
9. Else if *d* is an **action** data item *u* then
10. For all *x* do
11. $Bel'(x) = \int P(x \mid u, x')\, Bel(x')\, dx'$
12. Return *Bel'(x)*

Assumptions:

1. Markov assumption
2. Static world (only actions change state)
3. Independent noise
4. Perfect model, no approximation error

Particle Filters (Monte Carlo Localization)

Particle Filters model a target distribution through a set of samples drawn from a proposal distribution and weighted according to its difference to the target distribution.

General Particle Filter:

1. Sample the particles using the proposal distribution
$$x_t^{[j]} \sim proposal(x_t \mid \ldots)$$
2. Compute the importance weights
$$w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})}$$
3. Resampling: Draw sample i with probability $w_t^{[i]}$ and repeat J times

Particle Set: $\mathcal{X} = \{\langle x^{[j]}, w^{[j]} \rangle\}_{j=1,\ldots,J}$

state hypothesis — importance weight

Posterior Distribution:
$$p(x) = \sum_{j=1}^{J} w^{[j]} \, \delta_{x^{[j]}}(x)$$

Monte Carlo Localization:

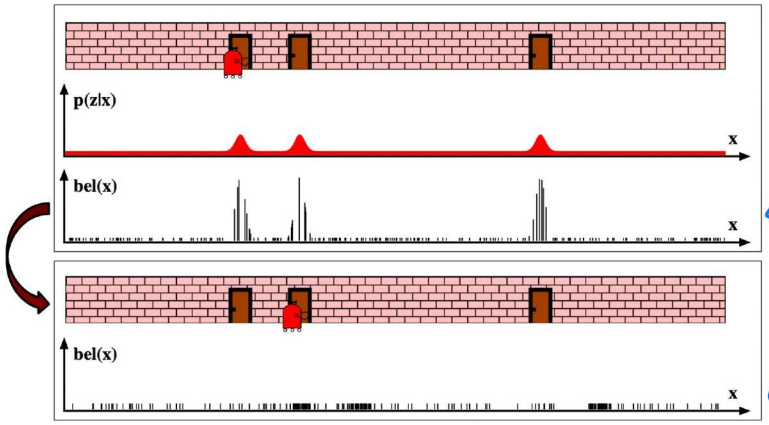
Particle Set: Each particle represents a pose hypothesis

Prediction: For each particle, sample a new pose from the motion model.
$$x_t^{[j]} \sim p(x_t \mid x_{t-1}^{[j]}, u_t)$$

Correction: Weigh samples according to sensor models
$$w_t^{[j]} \propto p(z_t \mid x_t^{[j]})$$

Derivation:
$$w_t^{[j]} = \frac{target}{proposal} = \frac{\eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})}{p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1})}$$
$$= \eta\, p(z_t \mid x_t)$$

Resampling: Resample from the set of particles with their respective probability



← Prediction

← Correction

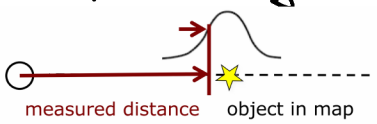
Motion Model: $p(x_t \mid u_t, x_{t-1})$

Probabilistic representation of the robots inaccurate motion. Motion model is specific to each robot.

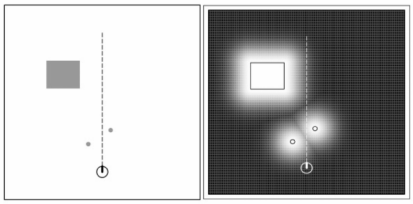
Observation Model: $p(\mathbf{o}_t \mid \mathbf{x}_t)$

Likelihood of a measurement given a pose

Ray-Casting: For each measurement cast ray and compare against the obtained measurement



End-Point: Compute the likelihood of each point on the beam by evaluating its distance to the closest obstacle.



Humanoid Localization

Estimate the robots pose using MCL

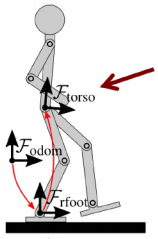
Motion Model:

$$\mathbf{x}_t^{[i]} = \mathbf{x}_{t-1}^{[i]} \oplus \mathcal{N}(\mathbf{M}\mathbf{u}_t, \boldsymbol{\Sigma}_{\mathbf{u}_t})$$

calibration matrix; covariance matrix; motion composition; odometry estimate

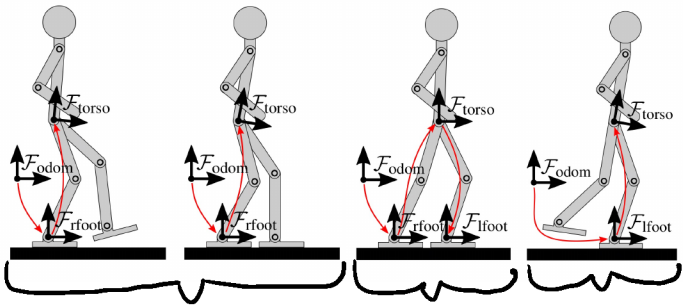
Odometry Estimation:

Estimate the robots torso pose from the stance foot pose using forward kinematics



frame of the torso, transform can be computed with FK over the right leg

frame of the current stance foot



Compute torso relative to the right foot

support exchange

compute relative to new stance foot

Observation Model:

$$p(\mathbf{o}_t \mid \mathbf{x}_t) = p(\mathbf{r}_t, \tilde{z}_t, \tilde{\varphi}_t, \tilde{\psi}_t \mid \mathbf{x}_t) = p(\mathbf{r}_t \mid \mathbf{x}_t)\, p(\underline{\tilde{z}_t} \mid \mathbf{x}_t)\, p(\underline{\tilde{\varphi}_t} \mid \mathbf{x}_t)\, p(\underline{\tilde{\psi}_t} \mid \mathbf{x}_t)$$

$$p(\tilde{z}_t \mid \mathbf{x}_t) = \phi(z_t - \tilde{z}_t, \underline{\sigma_z})$$
$$p(\tilde{\varphi}_t \mid \mathbf{x}_t) = \phi(\varphi_t - \tilde{\varphi}_t, \underline{\sigma_\varphi})$$
$$p(\tilde{\theta}_t \mid \mathbf{x}_t) = \phi(\theta_t - \tilde{\theta}_t, \underline{\sigma_\theta})$$

r_t: range measurement $\tilde{z}_t$: height computed by FK

$\tilde{\varphi}_t$: roll from IMU $\tilde{\psi}_t$: pitch from IMU

5. Footstep Planning

A*-Search

Expand iteratively nodes with the minimal evaluation function until a goal state is reached and a path is planned

Evaluation Function: $f(n)=g(n)+h(n)$

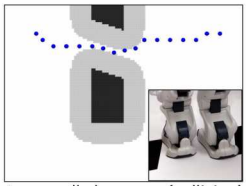
- $g(n)$: actual costs from start to state n
- $h(n)$: heuristic, admissible, if: $h(n) \leq h^*(n)$

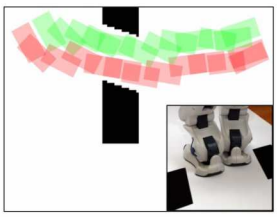
Node expansion: generate successor states, compute their f-values and insert them into the priority queue

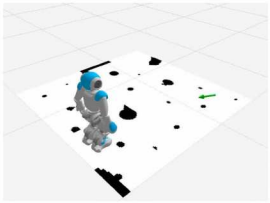
Optimal Path: A* yields optimum if h(n) is admissible

Motivation Footstep Planning

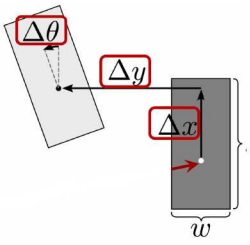
- Navigate through narrow passages
- Step over obstacles
- Navigate through cluttered/complicated environments

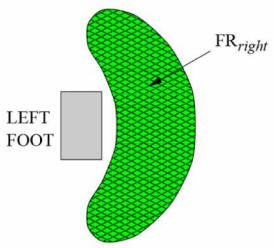

too small clearance (collision)

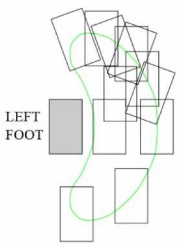




Footstep Planning with A*







State: $s = (x, y, \theta)$

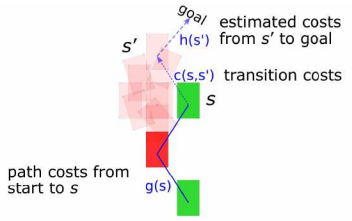
Footstep: $a = (\Delta x, \Delta y, \Delta\theta)$

Action set: $F = \{a_1, \ldots, a_n\}$

→ sampled from physical possible steps and/or commanded GCV values

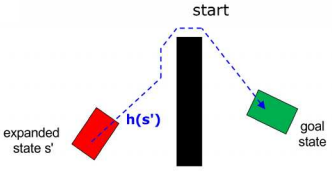
Evaluation Function:

Transition costs: $c(s, s') = \|(\Delta x, \Delta y)^\top\|$

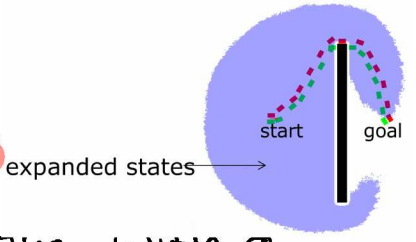


Heuristics:

1. Euclidean distance
   → likely to get stuck in local optima
2. Shortest 2D path (Dijkstra)
   → generally not admissable



Footstep Planning with Anytime Repairing A*

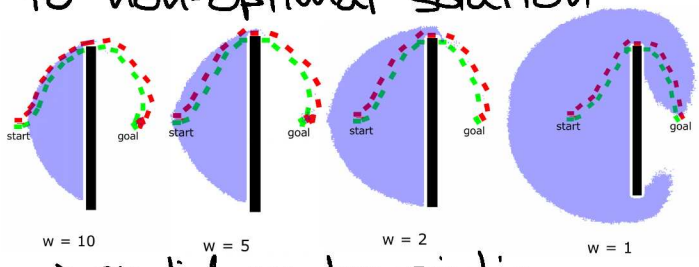


Run multiple Weighted A* searches, starting with a high weight until a given time limit is reached
→ finds sub-optimal solution fast
→ iteratively find better solution until time runs out

Weighted A*: Inflate heuristic by factor w

Large w: Fast convergence to non-optimal solution

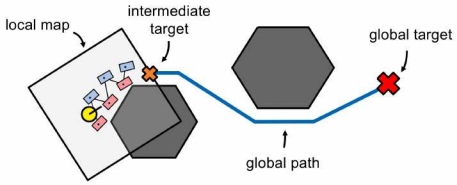
w=1: Optimal solution



→ euclidean heuristic

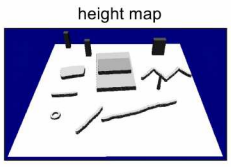
Local Footstep Planning

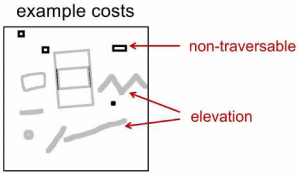
Only compute an optimal path to an intermediate goal on a global path

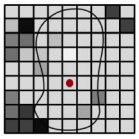


Planning on a Heightmap

Simply add a modified heuristic on the height of a point.
→ Additionally check, if stepping location has a small height difference

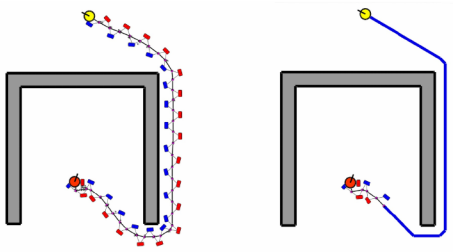






Footstep Planning with Aborting A*

Abort A* search at 50Hz to compute local footstep plans reactive to measurements and controls.

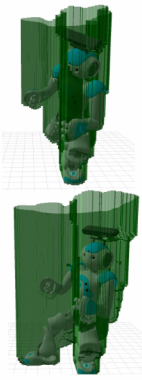


Heuristic: Path RTR

Whole-Body Collision Checking

Before footstep actions are taken check collision with obstacles

1. Pre-compute volume covered by the robot
2. Store minimum height value for footstep actions
3. Use the inverse height map (IHM) for fast collision checking



volume covered by the robot

Action a can be executed safely, if:

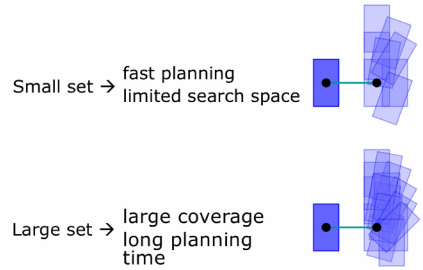
$$\forall (u,v) \in \mathrm{IHM}^a : \mathrm{IHM}^a_{(u,v)} + \underbrace{z_s}_{\text{height of stance foot at s}} > \underbrace{h_{(u,v)}}_{\text{heightmap value}}$$

Adaptive Node Expansion

Systematically search for a small number of viable successors

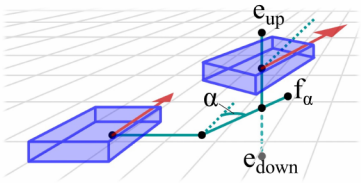
→ limit search space

→ speed up computation by only taking "good" successors



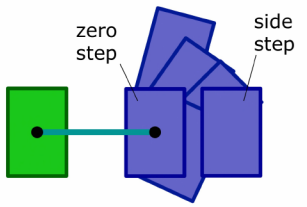
Reachability Map:

Provides a pre-computed discretization of feasible footsteps. This includes the footstep with maximal displacement from the stance foot.



Adaptive Action Set:

Local search around prototype steps (forward-, zero-, sidestep)



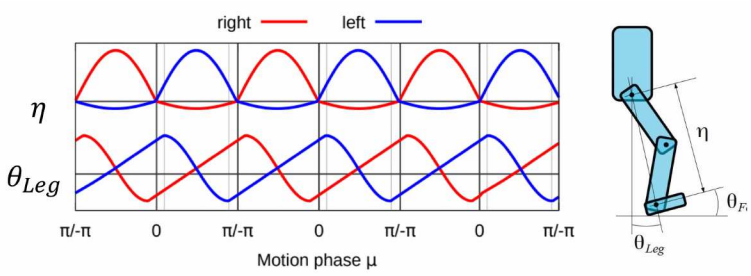
Validation of each successor:

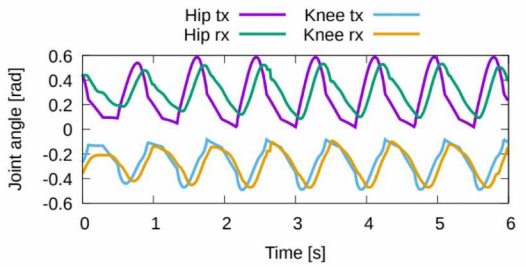
1. Check feasibility according to reachability map
2. Check for planar surface and height difference
3. Check for robot collisions

6. Bipedal Walking

Central Pattern Generator

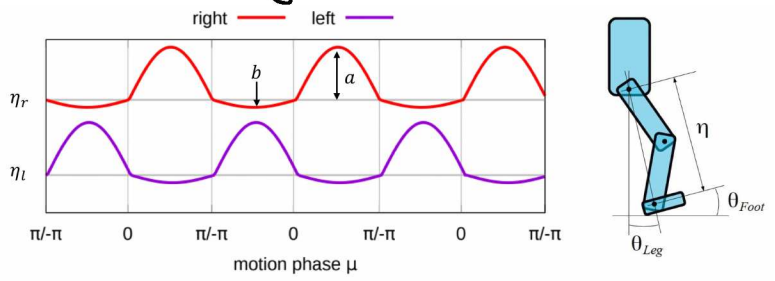
Module to generate open-loop stepping motions by providing a periodic signal to the abstract leg interface. This creates an "optimal" walk pattern.





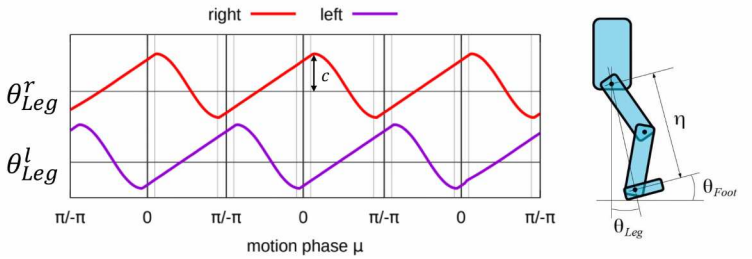
Leg lifting pattern: Modulates the leg extension

$$\eta(\mu) = \begin{cases} asin(\mu), if \mu \geq 0 \\ bsin(\mu), if \mu < 0 \end{cases}$$



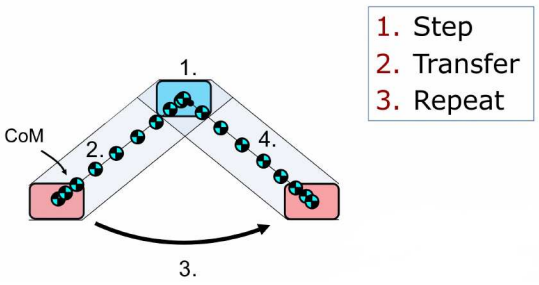
Leg swing pattern: Modulates the leg angle

$$\theta_{Leg}(\mu) = \begin{cases} ccos(\mu), if \mu \geq 0 \\ c\left(2\frac{\mu+\pi}{\pi} - 1\right), if \mu < 0 \end{cases}$$



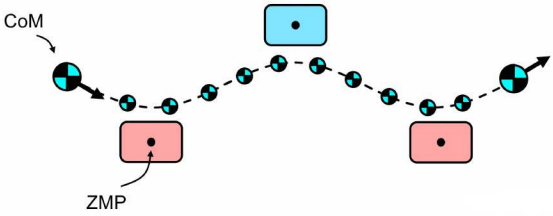
Static Walking

The COM never leaves the support polygon
→ stable but slow and unnatural



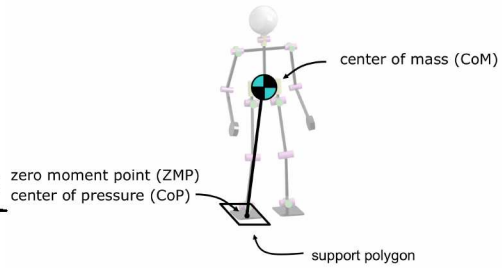
Dynamic Walking

COM is disregarded but the ZMP is always kept within the support polygon

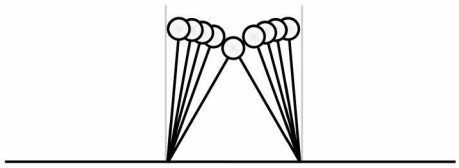


Inverted Pendulum Model

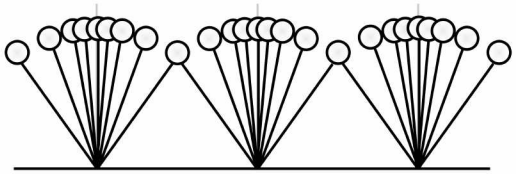
Model the movement of the COM in lateral and sagittal direction using an inverted pendulum for balance control



Lateral motion:

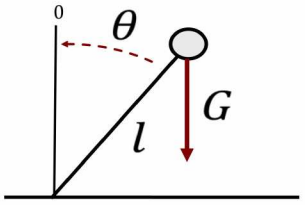


Sagittal motion:



Differential Equation:

$$\ddot{\theta}(t) = \frac{G}{l}\sin\theta(t)$$



Problem: Given an initial values compute future states

Solution via Euler-Method:

→ simulate for small timesteps
→ accurate but slow

Algorithm:

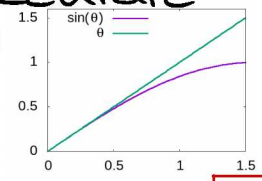
$function\ (\theta,\dot{\theta})(T)\ given\ (\theta_0,\dot{\theta}_0)$
{
$\theta = \theta_0, \dot{\theta} = \dot{\theta}_0, t = 0$
$dt = 0.000001s$
$while(t < T)$
{
$t \mathrel{+}= dt$
$\ddot{\theta} = \frac{G}{l}\sin\theta$
$\dot{\theta} \mathrel{+}= dt\,\ddot{\theta}$
$\theta \mathrel{+}= dt\,\dot{\theta}$
}
$return\ (\theta,\dot{\theta})$
}

Solution via Linearization:

→ fast but less accurate

Linearize:

$$\ddot{\theta}(t) = \frac{G}{l}\sin\theta(t)$$
$$\ddot{\theta}(t) \approx \frac{G}{l}\theta(t)$$
$$\ddot{\theta}(t) \approx C\theta(t)$$



Solution linear system:

$$\theta(t) = c_1 e^{\sqrt{C}t} + c_2 e^{-\sqrt{C}t} \qquad c_1 = \frac{1}{2}\left(\theta_0 + \frac{\dot{\theta}_0}{\sqrt{C}}\right)$$
$$\dot{\theta}(t) = c_1\sqrt{C}e^{\sqrt{C}t} - c_2\sqrt{C}e^{-\sqrt{C}t} \qquad c_2 = \frac{1}{2}\left(\theta_0 - \frac{\dot{\theta}_0}{\sqrt{C}}\right)$$

Initial condition:

$$\theta_0 = c_1 + c_2$$
$$\dot{\theta}_0 = c_1\sqrt{C} - c_2\sqrt{C}$$

Solution for t:

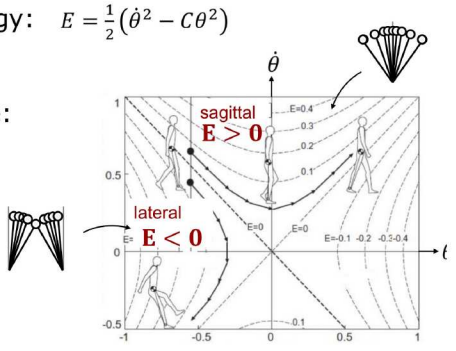
$$t(\theta) = \frac{1}{\sqrt{C}}\ln\left(\frac{\theta}{2c_1} + \sqrt{\frac{\theta^2}{4c_1^2} - \frac{c_2}{c_1}}\right)$$
$$t(\dot{\theta}) = \frac{1}{\sqrt{C}}\ln\left(\frac{\dot{\theta}}{2c_1\sqrt{C}} + \sqrt{\frac{\dot{\theta}^2}{4c_1^2 C} + \frac{c_2}{c_1}}\right)$$

Algorithm:

$function\ (\theta,\dot{\theta})(T)\ given\ (\theta_0,\dot{\theta}_0)$
{
$c_1 = \frac{1}{2}\left(\theta_0 + \frac{\dot{\theta}_0}{\sqrt{C}}\right)$
$c_2 = \frac{1}{2}\left(\theta_0 - \frac{\dot{\theta}_0}{\sqrt{C}}\right)$
$\theta = c_1 e^{\sqrt{C}t} + c_2 e^{-\sqrt{C}t}$
$\dot{\theta} = c_1\sqrt{C}e^{\sqrt{C}t} - c_2\sqrt{C}e^{-\sqrt{C}t}$
$return\ (\theta,\dot{\theta})$
}

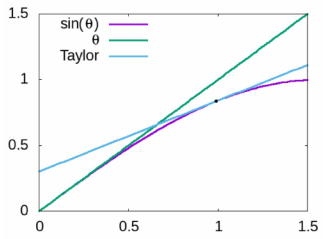
Orbital Energy: $E = \frac{1}{2}(\dot{\theta}^2 - C\theta^2)$

Phase Space:

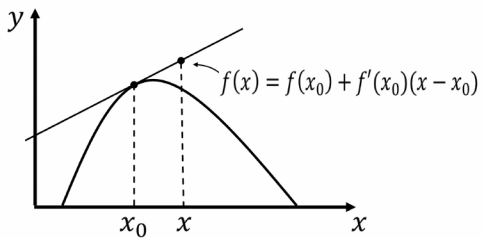


Solution via First-Order Taylor Expansion

→ very fast and locally accurate



Taylor Expansion: $f(x) = f(x_0) + f'(x_0)(x - x_0)$



Linearize:

$$\ddot{\theta}(t) = \frac{G}{l}\sin\theta(t)$$

$$\ddot{\theta}(t) \approx C(\sin\theta_0 + \cos\theta_0(\theta(t) - \theta_0))$$

Reorder: $\ddot{\theta}(t) \approx a\,\theta(t) + b$

$$\ddot{\theta}(t) \approx C(\sin\theta_0 + \cos\theta_0(\theta(t) - \theta_0))$$

$$\ddot{\theta}(t) \approx \underbrace{C\cos\theta_0}_{a}\theta(t) + \underbrace{C(\sin\theta_0 - \theta_0\cos\theta_0)}_{b}$$

Solution:

$$\theta(t) = -\frac{b}{a} + c_1 e^{\sqrt{a}t} + c_2 e^{-\sqrt{a}t} \qquad c_1 = \frac{1}{2}\left(\frac{b}{a} + \theta_0 + \frac{\dot{\theta}_0}{\sqrt{a}}\right)$$

$$\dot{\theta}(t) = c_1\sqrt{a}e^{\sqrt{a}t} - c_2\sqrt{a}e^{-\sqrt{a}t} \qquad c_2 = \frac{1}{2}\left(\frac{b}{a} + \theta_0 - \frac{\dot{\theta}_0}{\sqrt{a}}\right)$$

Initial Conditions:

$$\theta_0 = -\frac{b}{a} + c_1 + c_2$$

$$\dot{\theta}_0 = c_1\sqrt{a} - c_2\sqrt{a}$$

Algorithm:

function $(\theta, \dot{\theta})(T)$ *given* $(\theta_0, \dot{\theta}_0)$

{

$a = C\cos\theta_0$

$b = C(\sin\theta_0 - \theta_0\cos\theta_0)$

$c_1 = \frac{1}{2}\left(\frac{b}{a} + \theta_0 + \frac{\dot{\theta}_0}{\sqrt{a}}\right)$

$c_2 = \frac{1}{2}\left(\frac{b}{a} + \theta_0 - \frac{\dot{\theta}_0}{\sqrt{a}}\right)$

$\theta = -\frac{b}{a} + c_1 e^{\sqrt{a}t} + c_2 e^{-\sqrt{a}t}$

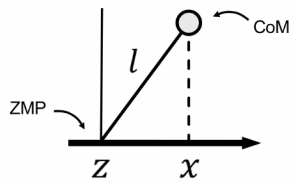
$\dot{\theta} = c_1\sqrt{a}e^{\sqrt{a}t} - c_2\sqrt{a}e^{-\sqrt{a}t}$

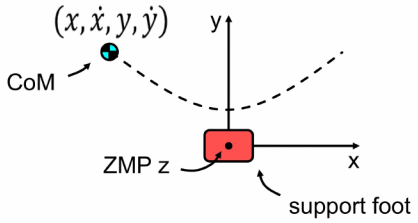
return $(\theta, \dot{\theta})$

}

2D Linear Inverted Pendulum

Use 2D coordinates instead of angles





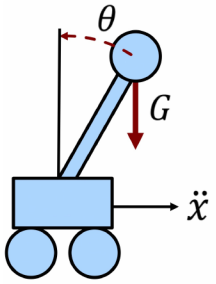
CoM Equation:

$$\ddot{x} = C(x - z_x)$$

$$\ddot{y} = C(y - z_y)$$

ZMP Equation: $z = x - \frac{\ddot{x}}{C}$

Cart-Pole Model



Differential equation: $\ddot{\theta}(t) = \frac{G}{l}\sin\theta(t) + \frac{\ddot{x}}{l}\cos\theta(t)$

Local linearization: $\ddot{\theta}(t) \approx a\,\theta(t) + b$

$$\ddot{\theta}(t) \approx \frac{G}{l}(\sin\theta_0 + \cos\theta_0(\theta(t) - \theta_0)) + \frac{\ddot{x}}{l}(\cos\theta_0 - \sin\theta_0(\theta(t) - \theta_0)) \longrightarrow$$

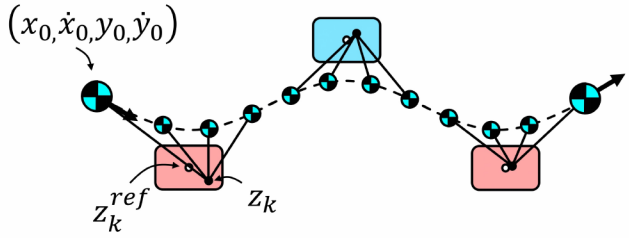
$$\ddot{\theta}(t) \approx \overbrace{\left(\frac{G}{l}\cos\theta_0 - \frac{\ddot{x}}{l}\sin\theta_0\right)}^{a}\theta(t) + \underbrace{\frac{G}{l}\sin\theta_0 + \frac{\ddot{x}}{l}\cos\theta_0 - \frac{G}{l}\theta_0\cos\theta_0 + \frac{\ddot{x}}{l}\theta_0\sin\theta_0}_{b}$$

State Estimation (COM)

1. Apply forward kinematics to retrieve the robots pose
2. Tilt the robot using the trunk angle reported by the IMU
3. Extract COM
4. Determine COM velocity using numerical differentation
5. Output state $(x_0, \dot{x}_0, y_0, \dot{y}_0)$

Zero Moment Point Preview Control

Using the ZMP for balance creates dynamic walking behavior:



State estimation through quadratic programming:

$$\min_{\dddot{X}_k} \frac{1}{2} Q \left(Z_{k+1} - Z_{k+1}^{ref} \right)^2 + \frac{1}{2} R \, \dddot{X}_k^2$$

Preview Scheme:

$$Z_{k+1} = P_x \, \hat{x}_k + P_u \, \dddot{X}_k$$

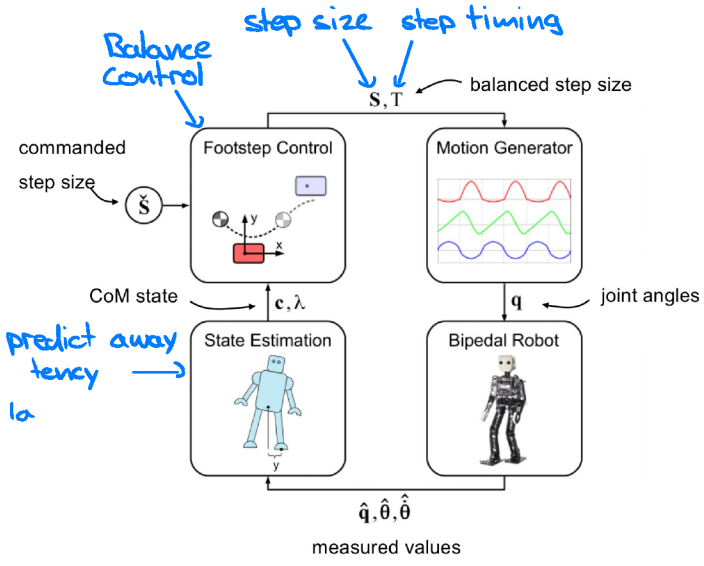
output / input

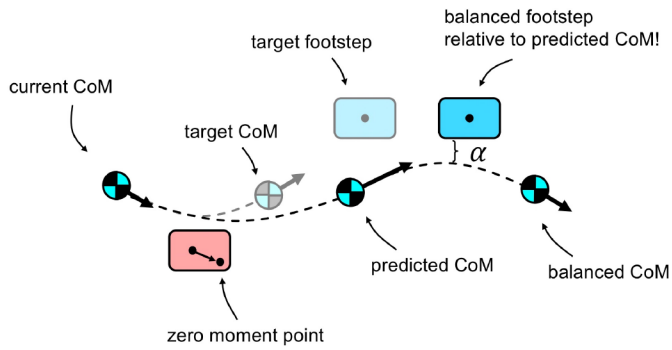
$$\begin{bmatrix} z_{k+1} \\ \vdots \\ z_{k+N} \end{bmatrix} = \begin{bmatrix} 1 & T & T^2/2 - h_{CoM}/g \\ \vdots & \vdots & \vdots \\ 1 & NT & N^2T^2/2 - h_{CoM}/g \end{bmatrix} \hat{x}_k + \begin{bmatrix} T^3/6 - Th_{CoM}/g & 0 & 0 \\ \vdots & \ddots & 0 \\ (1+3N+3N^2)\,T^3/6 - Th_{CoM}/g & \dots & T^3/6 - Th_{CoM}/g \end{bmatrix} \times \begin{bmatrix} \dddot{x}_k \\ \vdots \\ \dddot{x}_{k+N-1} \end{bmatrix}$$

Solution:

$$\dddot{X}_k = - \left(P_u^T P_u + \frac{R}{Q} I_{N \times N} \right)^{-1} P_u^T \left(P_x \, \hat{x}_k - Z_k^{ref} \right)$$

Capture Step Framework





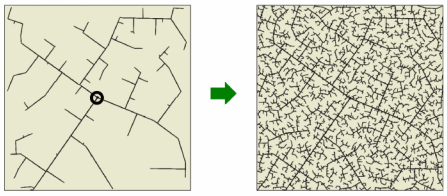
7. Whole-Body Motion Planning

Motivation:

- Inverse kinematics produce unpredictable joint configurations
- Instead plan a path in a high-dimensional configuration space
  - → complete search intractable
  - → use sampling-based or randomized algorithms
- Consider constraints such as joint limits, self- and obstacle collisions, and balance

Rapidly Exploring Random Trees (RRTs)

Iteratively explore the configuration space by randomly expanding a tree from the initial configuration.



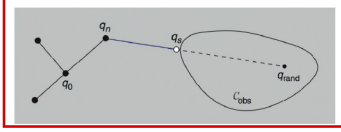
Algorithm:

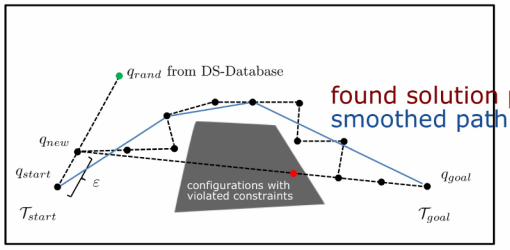
Algorithm 1: RRT

```
1 G.init(q0)
2 repeat
3   qrand ← RANDOM_CONFIG(C)
4   qnear ← NEAREST(G, qrand)
5   G.add_edge(qnear, qrand)
6 until condition
```

Connect q_{rand} with q_{near} using a **local planner**:

- No collision: add edge
- Collision: new vertex is q_s, as close as possible to C_{obs}



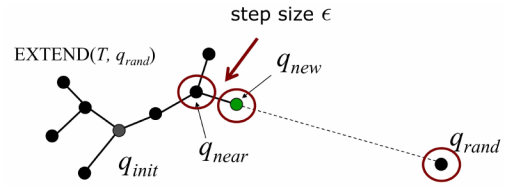


Goal Bias:

Exchange q_{rand} with the goal configuration with probability 5% - 10% to search towards goal

Variations:

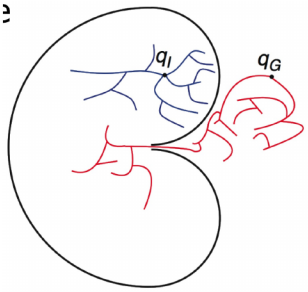
Fixed step size:



→ terminate when close enough

Bidirectional:

Grow to trees from start and goal. In every other steps try to reach q_{new} from the other tree.



Algorithm:

```
RRT_CONNECT (qinit, qgoal) {
  Ta.init(qinit); Tb.init(qgoal);
  for k = 1 to K do
    qrand = RANDOM_CONFIG();
    if not (EXTEND(Ta, qrand) = Trapped) then
      if (EXTEND(Tb, qnew) = Reached) then
        Return PATH(Ta, Tb);
    SWAP(Ta, Tb);
  Return Failure;
}
```

Constraints for humanoids:

Sample only double support configurations

Pros:

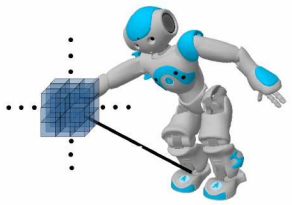
- fast
- easy to implement
- good balance between greedy search and exploration
- Finds a path to goal, if one exists

Cons:

- non-optimal paths
- post-processing is neccessary
- paths are not repeatable and unpredictable
- unknown rate of convergence

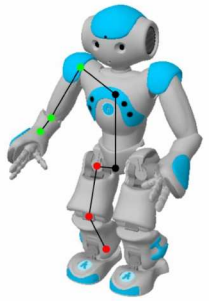
Reachability Maps:

A map over the end-effector space represented by systematic sampled joint configuration. Each voxel contains the corresponding configuration and quality measure.
→ can be pre-computed

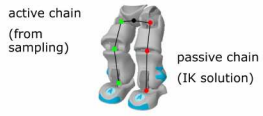


Configuration Sampling

1. Sample discrete steps through the range of motions of the kinematic chain
   Kinematic chain: right/left foot → gripper link

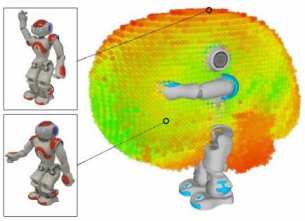


2. Generation of a double support configuration
   Given hip and desired foot pose, solve IK for configuration.



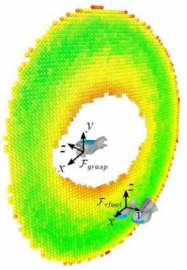
Measurement of Manipulability

Penalize configurations with limited maneuverability
→ distance to joint limits and singularities
→ distance to itself or objects



Inverse Reachability Maps

Represents the potential stance poses w.r.t. an end-effector pose
→ select optimal stance for given EE



red=low
green=high

Cross section through the IRM showing potential feet locations

Inversion RM:

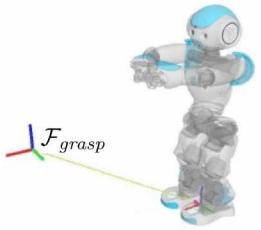
```
while v ← GET_VOXEL(RM) do
    n_c ← GET_NUM_CONFIGS(v)
    for i = 1 to n_c do
        (q_c, q_SWL, w) ← GET_CONFIG_DATA(v, i)
        p_tcp ← COMPUTE_TCP_POSE(q_c)   // end-effector pose via FK
        p_base ← (p_tcp)^-1   // inverse transform to get pose of foot wrt EE frame
        idx ← FIND_EE_VOXEL(p_base)   // determine voxel of foot
        IRM ← ADD_CONF_TO_VOXEL(idx, q_c, q_SWL, w)
    end
end
```

configuration of sampled chain
leg configuration of passive chain
manipulability measure

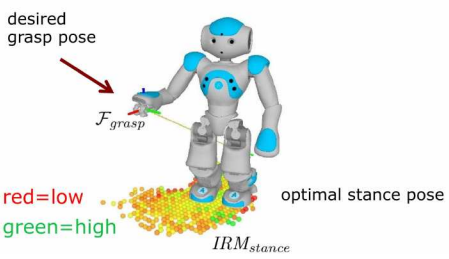
1. Invert Fk in RM and retrieve foot pose
2. Find corresponding voxel in IRM and copy configuration from RM

Determining the Optimal Stance Pose

Given a desired end-effector pose find the optimal stance.



1. Align the origin of the IRM with the grasp frame
2. Intersect the transformed IRM with the floor plane: $IRM_{floor} = tIRM \cap F$
3. Remove unfeasible configurations



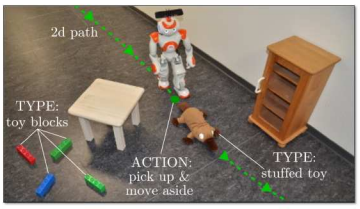
Whole-Body Planning

1. Use IRM to find optimal stance pose
2. Let robot walk to stance pose
3. Determine whole-body configuration to reach the desired grasp pose
4. Plan a whole-body motion using RRT-connect

8. Foresighted Robot Navigation

Motivation:

- Navigation through cluttered scenes
  → reason about possible actions
- Navigation through human environments
  → predict human actions



Foresighted Robot Positioning

Position robot close to user in case service is needed while not interfering and considering possible navigation goals

Evaluate cost function and choose position with smallest cost

Cost function:

$$C_{dist}(\mathcal{X}) = \sum_{o_j \in O} P(o_j|S) \cdot \mathcal{L}(\mathcal{P}_{\mathcal{X} \to \mathcal{X}_{o_j}})$$

costs of 2D map cells

length of A* path from the cell to possible navigation goal

probability of possible next navigation goal

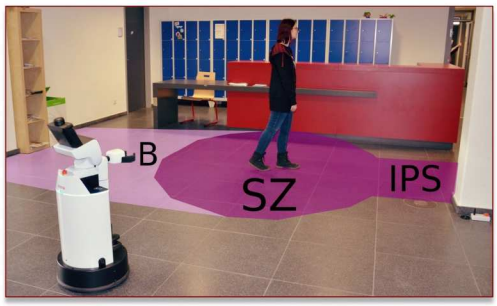
Human Comfort Constraints

Robot should not enter:

- Social zone (SZ)
- Information process space (IPS)

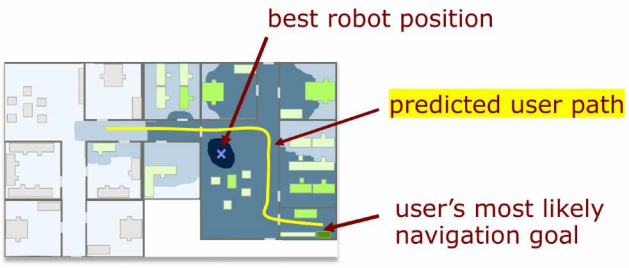
Increased costs:

- Area behind human (B)



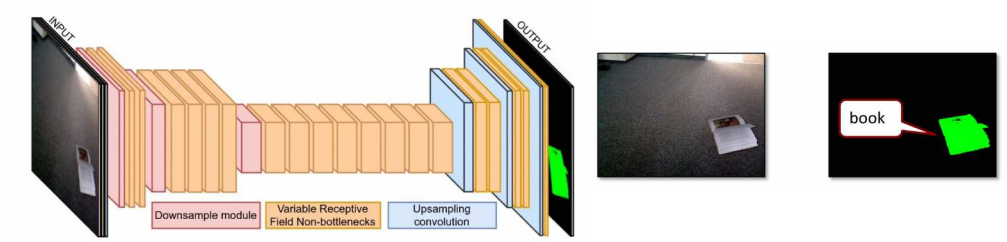
Time-dependent Planning

Determine the best position and user path from the navigation goal considering the human comfort constraints



Plan actions based on objects

1. Segment objects in images



2. Estimate the costs of possible actions

Possible actions:

object class	action type
balls	push, step over, pick up
cars, toy blocks	step over, pick up
stuffed toys, dolls	pick up
boxes, books	step onto, pick up

→ use depth information to exclude actions depending on the object size

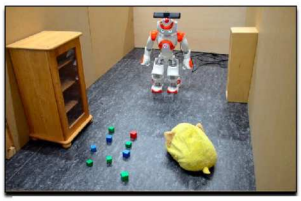
Action costs: estimated execution time

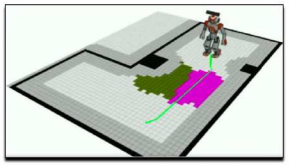
→ can be learned from experimental runs

→ assign cheapest possible action to each object

3. Encode the action costs in a 2D cost grid of the environment

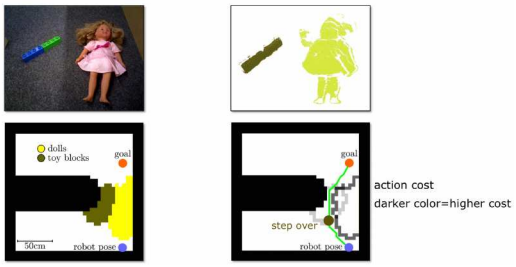
Project obstacles with their cost onto a 2D grid





4. Search for 2D path on the cost grid

Obtain the cheapest path and action for each grid cell



Plan execution

Path segments without objects:

Walking control along the 2D path

Path segments with action:

Push/pickup: Walk to the last free grid cell on the 2D path and perform the action on the segmented point cloud

Step over/onto: 3D footstep planning in the corresponding region on the height map computed from the point cloud

Human Motion Prediction

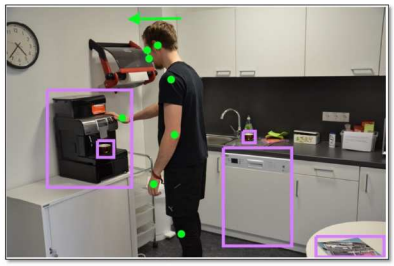
Predict the users navigation goal by modelling the transition probabilities for object interactions

Realize foresighted navigation by choosing poses close to the use while avoiding interference

Learning Transitions:

R-CNN trained on Open Images dataset to detect objects in RGB-D videos

OpenPose for user detection and human pose estimation



Detection of Interactions

Estimate users orientation and hand positions

Infer interaction from overlap between hand and bounding boxes and check depth

Belief about Navigation Goal:

object class + position

observed user position and pose

$$p(o_i|S) = \eta \cdot p(S|o_i)p(o_i) = \eta \cdot p(S|o_i)I(o_i|\tau)$$

observation likelihood

learned object transitions

observed human-object interaction

user's observed position and pose

user's observed position

user's observed orientation

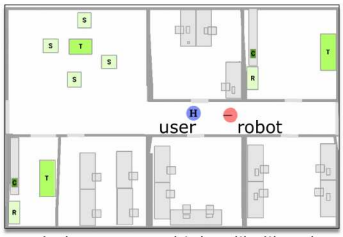
$$p(S|o_i) = dist(x_h, x_i)^{-1} \cdot \Delta(\theta_h, \theta_{opt})^{-1}$$

object position

optimal orientation wrt. to the object

Method:

1. Extract human-object interactions and derive object transitions
2. Integrate the observations of the users position and orientation
   → Use inverse of distance and orientation difference
3. Update navigation goal belief



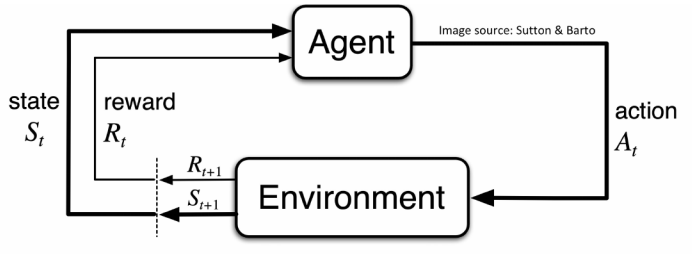
darker green = higher likelihood

9. Reinforcement Learning

Reinforcement Learning Model

An agent learns a policy by interacting with the environment and getting spase feedback.

Markovian Model:



Goal: Maximize cumulative reward

Reward: $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3+...}$

γ: discount factor

State values:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | S_t = s] \\ &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s\right] \end{aligned}$$

Value of the state to reach a specific goal using a fixed policy

Policy: $\Pi: S \longrightarrow A$

Optimal: $\pi^*(s) = \underset{\pi}{\operatorname{argmax}}\ V_\pi(s)$

Bellman equation:

$$\begin{aligned} V^\pi(s) &= E_\pi\left[R_t | s_t = s\right] \\ &= E_\pi\left[r_{t+1} + \gamma V(s_{t+1}) | s_t = s\right] \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^\pi(s')\right] \end{aligned}$$

Value successor

sum over all actions acc. to Π

transition prob.

immediate reward

Expresses the relationship between the value of a state and its successor states and gives a recursive formulation

RL Paradigms

Model free: No prior knowledge about the world

Model based: Model of world exist and used for training

Learning Paradigms:

On-Policy: Updates the values by assuming to follow the current policy's action.

Off-policy: Updates the values using a different policy, i.e. greedy policy, than the one the decisions are based upon.

RL-Agent Categories

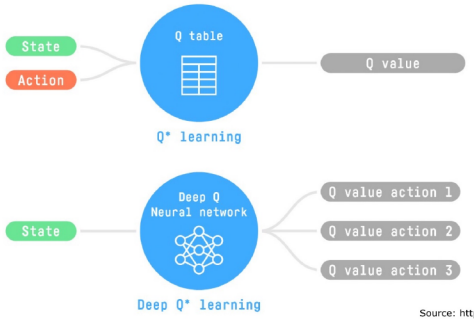
Value based: Train to approximate the value function

Policy based: Learn a probability distribution over the set of possible actions

Actor-critic: A critic estimates the value and the agent updates its policy distribution accordingly

Deep Reinforcement Learning

Use neural networks to estimate value functions

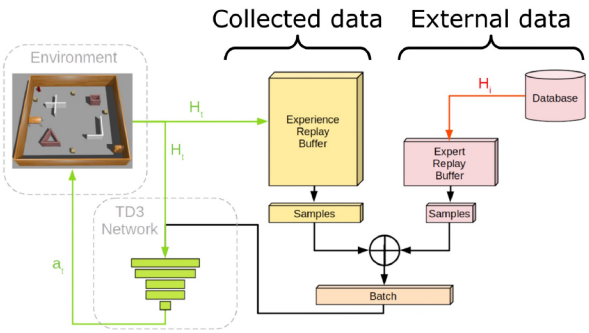


Experience Replay Buffer

Increase efficiency by reusing experiences instead of resampling every time.

Functioning:

After agent took an action and received an award, write experience to the buffer



Hindsight Experience Replay

Goal: Overcome failures in learning

In case of failure: Rewrite state and reward assuming the reached state was a success

Advantages:

- effective utilization of failures
- improved exploration and exploitation
- better convergence and stability

Prioritized Experience Replay

Goal: Prioritize experiences based on their importance

Priority: Difference between the estimated state-action value and the actual value

Advantages:

- focused utilization and repitition of important experiences
- better adaptability to changing environments

Attentive Experience Replay

Goal: Prioritize experiences based on their similarity

Probability of selection is defined by the similarity to the current observation
→ use k·n samples and choose n most similar for further computation

Advantages:

- reduces sampling of irrelevant state-action pairs
- focusses on task of current situation

Object Pushing

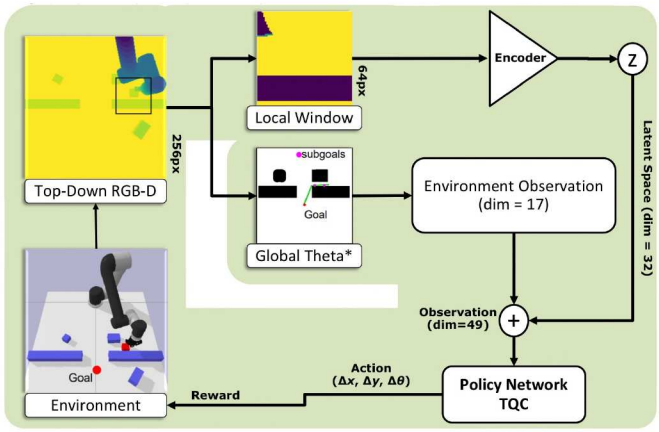
Applications:

- Re-positioning & re-orientating
- Move large, heavy or uneven shaped objects
- Move fragile objects to target positions

Goal:

- efficient goal-oriented pushing
- avoiding obstacles
- encourage gentle motion

Approach:



Actions:

- **increment** of current end effector position
- (Θx, Θy , Θyaw)

Observations:

- Latent space of local window **(32)**
- End effector position **(5)**
- 6D joint angle pose **(6)**
- Sub-goal at t-1 **(2)**
- Sub-goal at t-5 **(2)**
- Contact with obstacle **(1)**
- Object to goal distance **(1)**

Total: 49

Reward:

$$r_{dist} = \begin{cases} 50, & \text{if goal reached} \\ -r_{g\_dist} - r_{o\_dist}, & \text{otherwise} \end{cases}$$

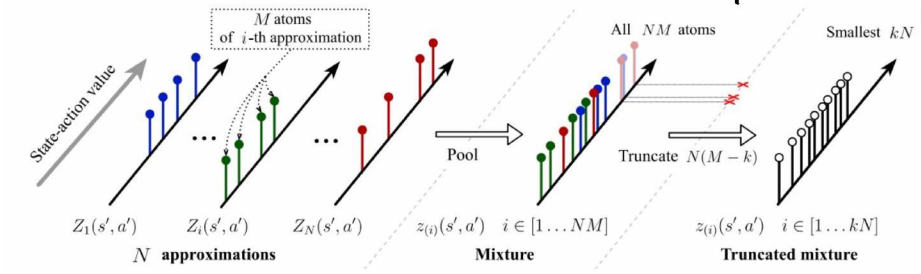
$$r_{collison} = \begin{cases} -10, & \text{if object out of bounds} \\ -5 & \text{if collision occurred} \end{cases}$$

$$r_{touch} = \begin{cases} r_{o\_dist}, & \text{contact to object} \\ 0 & \text{otherwise} \end{cases}$$

$$r_{total} = r_{dist} + r_{collision} + r_{touch}$$

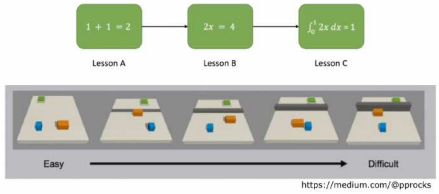
Learning: Truncated Quantile Critics

→ Attentive experience replay



Curriculum learning

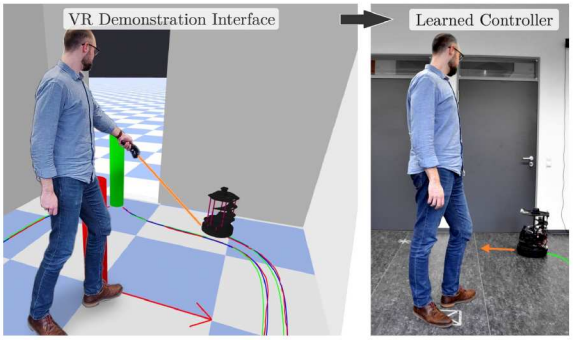
- Increasing start-goal distance (6 cm to 60 cm)
- Increasing difficulty of environment (closer gaps)

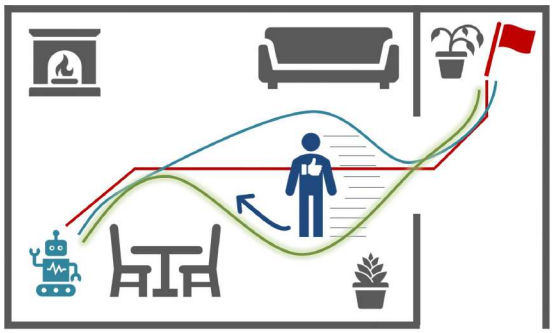


https://medium.com/@pprocks

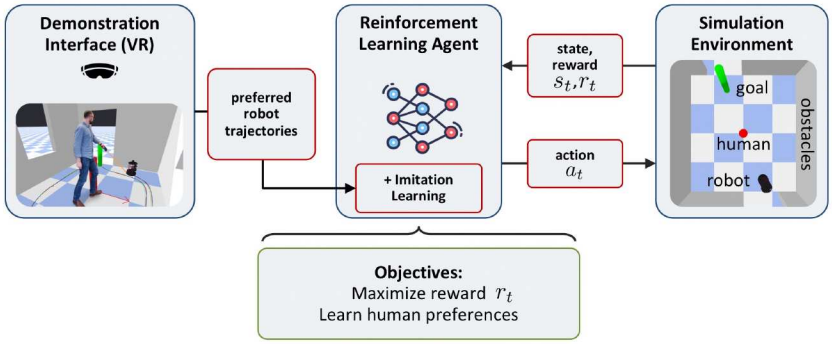
Robot Navigation

Goal: Learn path from demonstration

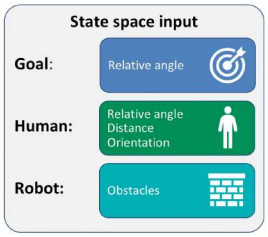


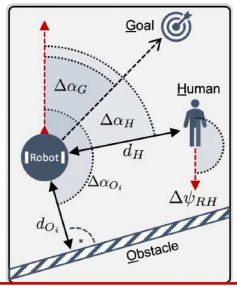


Approach:



State Space:





Reward:

$$r = r_{\text{collision}} + r_{\text{goal}} + r_{\text{timeout}}$$

$$r_{\text{collision}} = \begin{cases} -c_{\text{rew}} & \text{if collision} \\ 0 & \text{else.} \end{cases}$$

Demo boost

$$r_{\text{goal}} = \begin{cases} +c_{\text{rew}} & \text{if goal reached in demonstration data} \\ 0 & \text{if goal reached during training} \\ 0 & \text{else} \end{cases}$$

$$r_{\text{timeout}} = \begin{cases} -\frac{c_{\text{rew}}}{2} & \text{if episode timeout } (n > N_{\text{ep}}) \\ 0 & \text{else} \end{cases}$$

Perception: