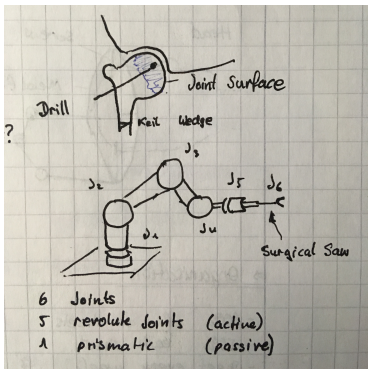
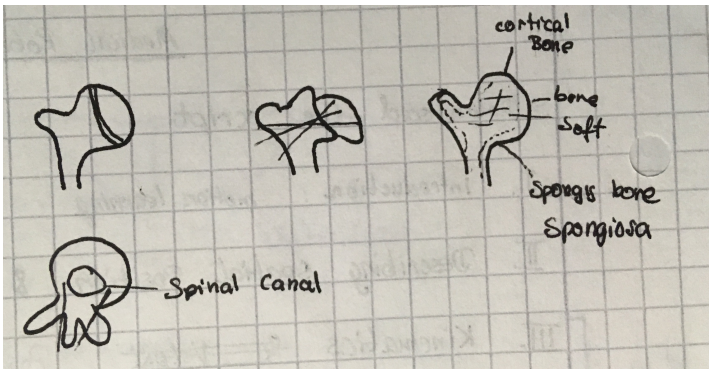
Applications of Medical Robotics

There are 4 types of Robots

- Robots for **Navigation**
  Surgical drill for Precise positioning, motion compensation..
- Robots for **Imaging**
  Minimally-invasive surgery by Motion downscaling, reduce tremor...
- Robots for **Motion Replication**
  Robotic ultrasound for Automation, speed...
- **Rehabilitation** and **Prosthetics**
  Exoskeletons that Replace damaged structures, autonomous rehabilitation...

Application: Epiphyseolysis





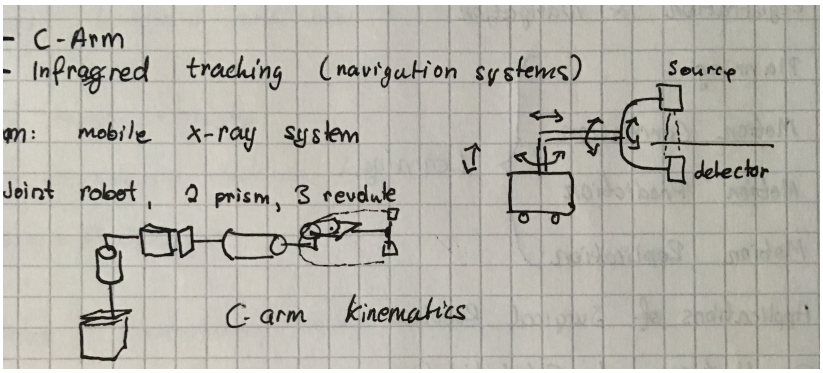
Navigation

Problem: We have an image.

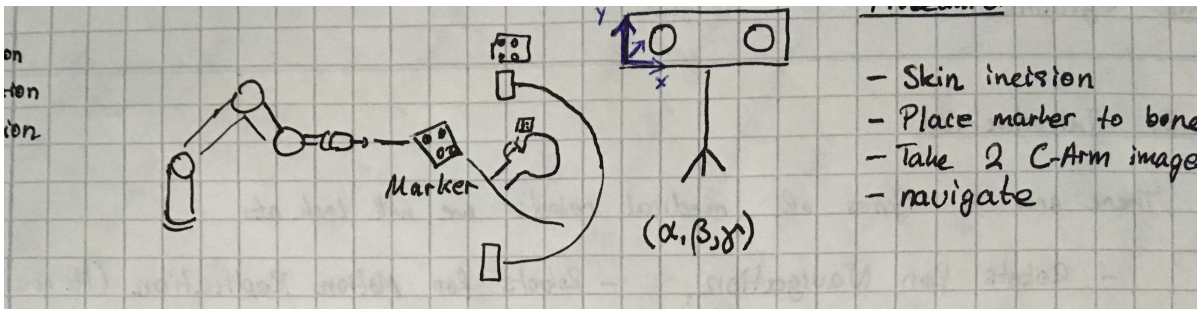
- Where is the **Robot** in the image coordinate system?
- Where is the **Target** in the image?

Ingredients for Radiologic Navigation:

- **C-Arm Robot:**
  A mobile x-ray system, robot containing 2 prismatic and 3 revolute Joints



- **Infrared Tracking:**
  Place Marker on Endeffector, on Target and on C-Arm. A Camera then calculates the positions:
  - Skin incision
  - Place marker on bone
  - Take 2 C-Arm images
  - Tavigate



$\Longrightarrow$ Navigation, Registration, Calibration!

Spatial position and orientation

Every Transformation is a multiplication, simply write down the vectors.
Rotatory Matrices:

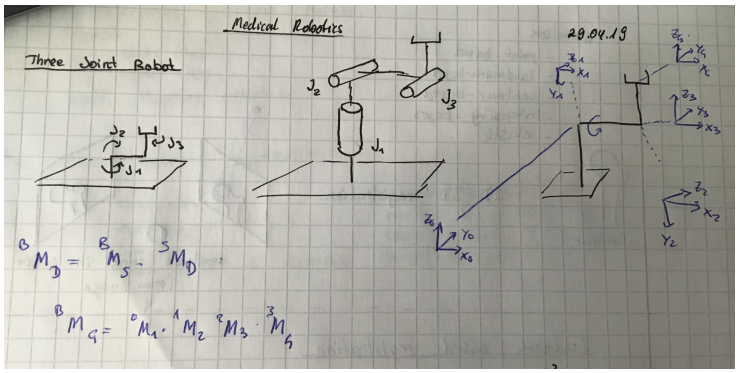
$$R(x,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C_\theta & -S_\theta & 0 \\ 0 & S_\theta & C_\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; R(y,\theta) = \begin{pmatrix} C_\theta & 0 & S_\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S_\theta & 0 & C_\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; R(z,\theta) = \begin{pmatrix} C_\theta & -S_\theta & 0 & 0 \\ S_\theta & C_\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translatory Matrices:

$$T(x,a) = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; T(y,a) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & a \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; T(z,a) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Denavit Hartenberg Parameter

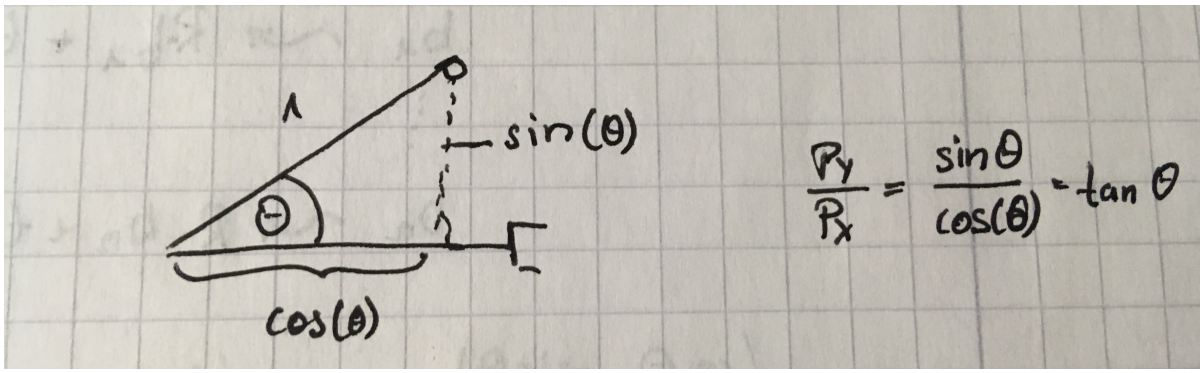
TODO



In **Forward Kinematics**, we derive the Endeffector Position by multiplication of given Joint Matrices, e.g.

$$^{B}M_{G} = ^{B}M_{1} \cdot^{1} M_{2} \cdot^{2} M_{3} \cdot^{3} M_{G}$$

In **Reverse Kinematics**, we calculate the Joint angles from a given Endeffector Position, e.g. One Joint Robot:

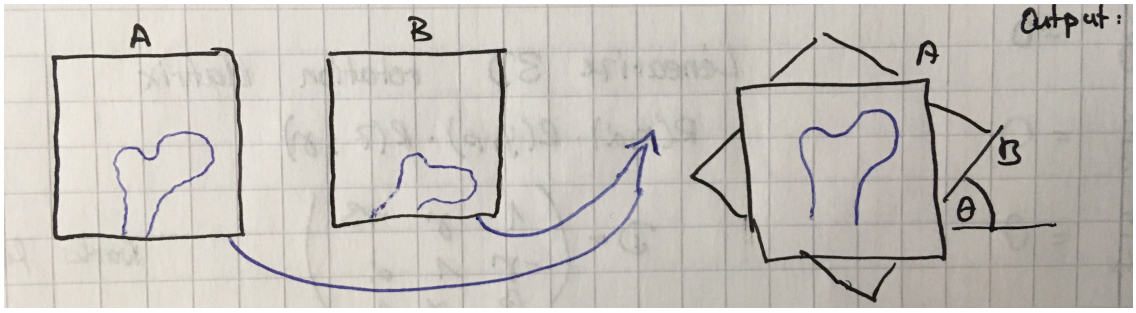


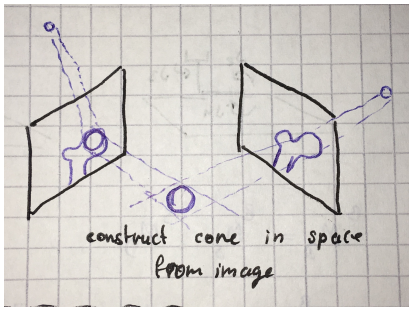
$P = (P_X, P_Y)$, Get Angle θ by: $\frac{P_Y}{P_X} = \frac{\sin\theta}{\cos\theta} = \tan\theta \implies \theta = atan2(P_Y, P_X)$
Use atan2-Function, as it gives back the angles of full circle depending on signs

Navigation & Registration

Problem: We have >1 images.

- Rotate images in such way, that images align
- Output is $\theta, \triangle x, \triangle y$





There are various types of Registration:

- point-based
- landmark-based
- contour-based
- intensity-based
- elastic

Landmark based registration

Situation: We have an MR image of a head. We place 4 Landmarks (Nose and Mouth). The Target Point is given with respect to the four landmark points.
From this, two point clouds A, B are generated.
We assume: $\#A = \#B,\ 2D,\ a_1 \to b_1$ *etc.*
If we move B, this leads to:

$$b_1 \rightsquigarrow R \cdot b_1 + t,\ R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, t = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$
$$b_2 \rightsquigarrow R \cdot b_2 + t$$
$$\vdots$$
$$b_n \rightsquigarrow R \cdot b_n + t$$

We create a function f, which is the distance between the two point clouds A and B. We want to minimize this function:

$$f = ||R \cdot b_1 + t - a_1||^2 + \cdots + ||R \cdot b_n + t - a_n||^2$$

Gaussian Least Square

If we assume that the angle θ is small, we can linearize the rotational matrix R such that:

$$\cos\theta \sim 1,\ \sin\theta \sim \theta \implies R = \begin{pmatrix} 1 & -\theta \\ \theta & 1 \end{pmatrix}$$

This matrix R is linear in θ.
We can also linearize the 3D rotation matrix as follows:

$$R(x,\alpha)\cdot R(y,\beta)\cdot R(z,\gamma) = \begin{pmatrix} 1 & \gamma & -\beta \\ -\gamma & 1 & \alpha \\ \beta & \alpha & 1 \end{pmatrix}$$

As the multiplication of two small values leads to an even smaller value. This method works fine until angles of 10 degrees.

Iterative Closest Points (ICP)

Assume we have a point cloud A and a point cloud B. We want know the transformation needed to overlap B over A. We do **not** deform B, as all motion is rigid (rotation and translation).

Cloud A:

1	1	3	3
1	1	3	3
2	2	3	3
2	2	3	3

Cloud B

2	2	1	1
2	2	1	1
3	3	3	3
3	3	3	3

$\Longrightarrow$ Just rotate 90^o to align
But sometimes, coloring differs in images depending on the imaging process. For Example, in CT the bone has a bright white color, while in MR the bone is more gray:

Cloud A:

1	1	3	3
1	1	3	3
2	2	3	3
2	2	3	3

Cloud B

3	3	1	1
3	3	1	1
2	2	2	2
2	2	2	2

$\Longrightarrow$ rotate 90^o enough?

Mutual Information (MI)

For **Mutual Information** Registration, the color mapping does not have to be exactly 1x1. It can be different, as shown in the example above.
Basic Definition:
We generate Images A, B in a random process. For each pixel of the image, we throw a dice (random generation), so we map 6 colors $\{1,\dots,6\}$ to each pixel. A, B are random variables with distributions P_A, P_B, namely:

- $P_A(a)$ = Probability of grey level value a in image A
- $P_B(b)$ = Probability of grey level value b in image B

$\Longrightarrow$ $P_{A,B}(a,b)$ = Probability of grey level value a in image A occuring at the same position in image B

Examples:

- $A \neq B$: A tells nothing about B, 2 different images
- $A = B$: A tells everything about B
- A tells something about B

Keep in mind the law of Independence of two random variables:

$$P_{A,B}(a,b) = P_A(a) \cdot P_B(b)$$

We want to know the maximum Mutual Information, calculated as follows:

$$I(A,B) = \sum_{a,b} P_{A,B}(a,b) \cdot \log_2 \left(\frac{P_{A,B}(a,b)}{P_A(a) \cdot P_B(b)} \right)$$

The Sum ranges over all greyscale pairs of diefferent colors. $I(A,B)$ measures the information dependencies. We maximize the Mutual Information to find exact position.
Example 1:

$$A = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array}$$

$$P_A(0) = 0.5, \; P_B(0) = 0.5$$
$$P_A(1) = 0.5, \; P_B(1) = 0.5$$
$$P_{A,B}(0,0) = 0.5, \; P_{A,B}(0,1) = 0$$
$$P_{A,B}(1,1) = 0.5, \; P_{A,B}(1,0) = 0$$
$$\sum = 1$$

$$I(A,B) = \sum_{a,b} P_{A,B}(a,b) \cdot \log_2 \left(\frac{P_{A,B}(a,b)}{P_A(a) \cdot P_B(b)} \right)$$

$$= 0.5 \cdot \log_2 \left(\frac{P_{A,B}(0,0)}{P_A(0) \cdot P_B(0)} \right) + 0.5 \cdot \log_2 \left(\frac{P_{A,B}(1,1)}{P_A(1) \cdot P_B(1)} \right) + 0 \cdot \log_2 \left(\frac{P_{A,B}(0,1)}{P_A(0) \cdot P_B(1)} \right) + 0 \cdot \log_2 \left(\frac{P_{A,B}(1,0)}{P_A(1) \cdot P_B(0)} \right)$$

$$= 0.5 \cdot \log_2(2) + 0.5 \cdot \log_2(2)$$

$$= 1$$

Example 2:

$$A = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

$MI(A, B) = 1$, as we care for the structure of the image. The color values do not really matter. Example 3:

$$A = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array}$$

$MI(A, B) = 0$, as the structures are very different. Example 4:

$$A = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 1 \\ \hline \end{array}$$

$MI(A, B) = 0.32$
Keep in mind that $MI \geq 0$, but can go up to 100 for large images!

Calculating P_A, P_B, $P_{A,B}$

Just create a table, then divide the right hand side by the amount of pixels/voxels:

0	number of Pixels in image A with greylevel 0
$\vdots$	$\vdots$
255	number of Pixels in image A with greylevel 255

Calculating $P_{A,B}$ is a little difficult:

	$0 \ldots$ i $\ldots$ 255
0	
$\vdots$	$\vdots$
i	$\ldots$ # pixel pins with colors i, j
$\vdots$	
255	

There are 3 algorithms/methods for calculation:

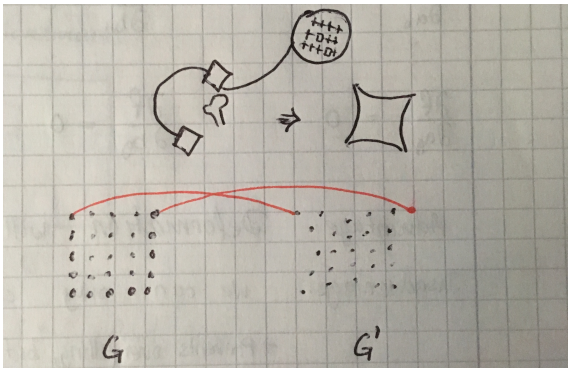
1. **Nearest Neighbour (NN)**
2. **Trilinear Interpolation (TI)**
3. **Trilinear Partial Volume (TPV)**

Imaging

Image Deformation

There are multiple applications:

- Elastic Registration
- Distortion Correction



Bilinear Interpolation

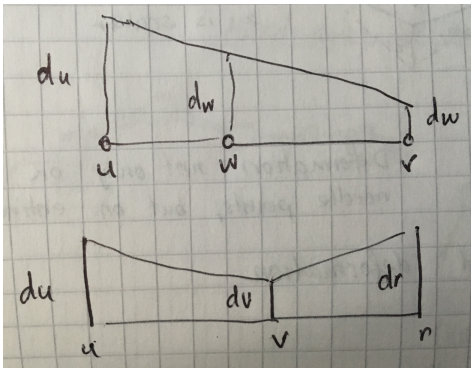
Assume we have 2 adjacent grid points u, v. To get from G' to G we have to deform u in x-direction by du. We obtain information for w by **linear interpolation**. Same for the y-direction.

Advantage:

- Simplicity
- We will move all points to its goal $\implies$ Accuracy!

Disadvantage:

- Not smooth



Cubic Spline Interpolation

We define a correction polynomial, two variables x, y

$$a_0 + a_1x + a_2y + a_3xy + a_4x^2 + a_5y^2 + a_6xy^2 + a_7x^2y + a_8x^3 + a_9y^3$$

$$b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2 + b_6xy^2 + b_7x^2y + b_8x^3 + b_9y^3$$

We have our model: (x_m, y_m) and our distorted points (x_d, y_d).
We further assume, that a's and b's are variables, while x, y are constant.
We set up f as:

$$f = [(x_m, y_m) - (x_d, y_d)]^2$$

For N grid points, we adjust f to:

$$f = [(x_m^{(1)}, y_m^{(1)}) - (x_d^{(1)}, y_d^{(1)})]^2 + \cdots + [(x_m^{(N)}, y_m^{(N)}) - (x_d^{(N)}, y_d^{(N)})]^2$$

With x_d being the a-Polynomial and x_m being the b-Polynomial, function f is quadratic in the a's and b's.
We deriviate f after a_i, b_i, with:

$$\frac{\delta f}{\delta a_i} = \frac{\delta f}{\delta b_i} = 0$$

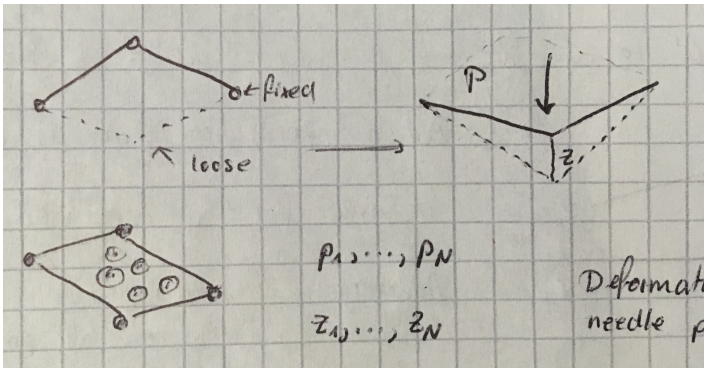
This is a 20x20 linear system. We perform Gaussian Elimination
$\Longrightarrow$ **Gaussian Least Squares**
Advantage:

- Deformation will be smooth

Disadvantage:

- We can only create $\sim$-formed Formations. This prevents overfitting, but, regarding grid points, they will not always match.

Thin Plate Splines



We describe deformation not only as a force on a specific point, but as da force on some entire sheet. So, we have N points $p_1, \ldots, p_N$ and we have a vector $z_1, \ldots, z_N$ for each point. We set up function f to represent deformation:

$$f(p) = \sum_{i=1}^{N} w_i F(||p - p_i||)$$

where w_i are the weights and F is our **Kernel function**

$$F(r) = r^2 \quad \text{or} \quad F(r) = r^2 \log r$$

How to find the weight values?
We have N equations

$$f(p_1) = \sum_{i=1}^{N} w_i F(||p_1 - p_i||) = z_1$$

$$\vdots$$

$$f(p_N) = \sum_{i=1}^{N} w_i F(||p_N - p_i||) = z_N$$

Our method of choice is to add 3 more equations to obtain a full linear system:

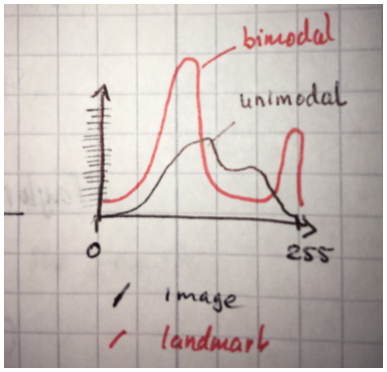
$$f(p) = \sum_{i=1}^{N} w_i F(||p - p_i||) + a_0 + a_1 \cdot x + a_2 \cdot y$$

where

$$\sum_{i=1}^{N} w_i = \sum_{i=1}^{N} x_i \cdot w_i = \sum_{i=1}^{N} y_i \cdot w_i = 0$$

Elastic Registration

Problem: We have to find correspondencies between landmarks.
We select/take a small subwindow on (e.g. 9×9 Pixel Array), and find the best match. We can find good landmarks by the use of **color contrasting** and **bimodal histograms.** We then use the histograms to replace elastic registration by correlation and prediction.

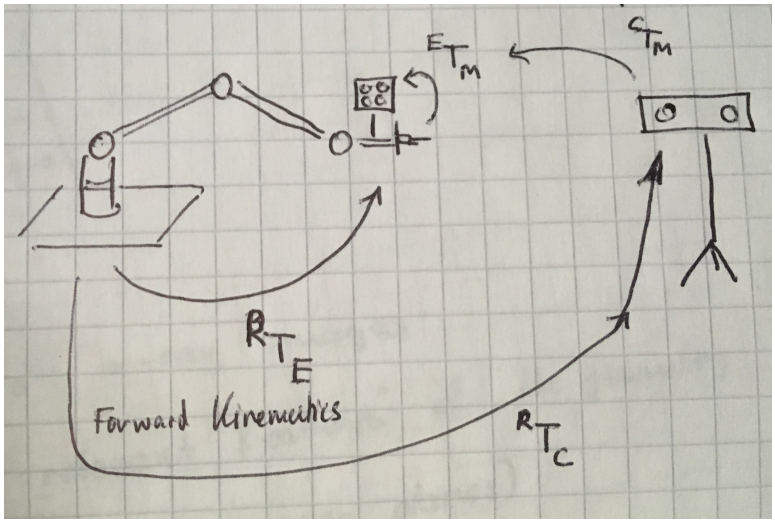


Hand-Eye-Calibration

We have a Robot. We know the Robots Forward Kinematics from base to endeffector (see. DH-Parameters or Handbook).
We place a marker on the bone/target and a marker on the endeffector. We observe via a camera.

$$\underbrace{{}^{R}T_{E}}_{known} \cdot \underbrace{{}^{E}T_{M}}_{unknown} = \underbrace{{}^{R}T_{C}}_{unknown} \cdot \underbrace{{}^{C}T_{M}}_{known}$$

$$\rightsquigarrow A \cdot x = y \cdot B$$



We have 24 unknown variables, but only 12 equations. What shall we do?
The trick is to move the robot, sinse x, y are constant and won't change at all!

$$A_1 \cdot x = y \cdot B_1$$

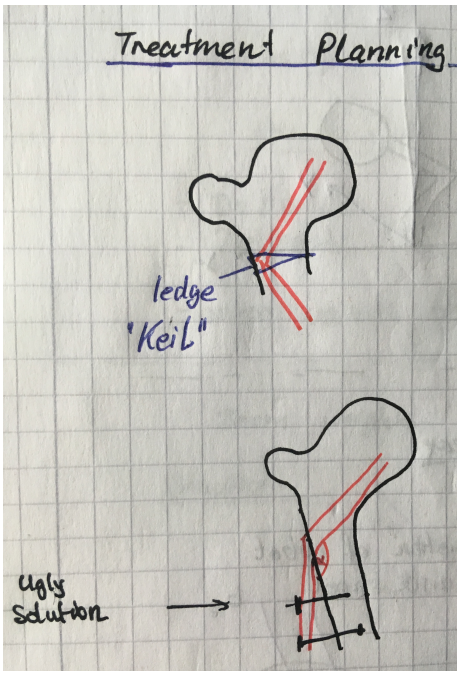
$$A_2 \cdot x = y \cdot B_2$$

$$\vdots$$

$$A_{500} \cdot x = y \cdot B_{500}$$

Treatment Planning

Situation: We perform a surgery without CT (very slow), only X-Ray. We want to fix a bone, see picture. This is a heavily invasive operation, as we have to place implants to hold together the pieces.
To place an implant, we need to dril a hole into a bone to create a cavity for the implant. We drill before we cut the bone $\implies$ Cunjuction with Navigation. We have to drill at a correct angle, otherwise a (ugly) correction with screws is needed. As we are in 3D space, we don't have only one but **three** angles.



Steps:

1. Take two x-ray images
2. Extract relevant "features" of the geometry
3. Plan (compute wedge planes)
4. Navigate

1. ✓(AP and lateral x-ray)
2.
   - femur shaft axis
   - femur neck isthmus
   - femur head center point
   - cutting plane

   Steps for feature extraction (point)

   - mark point in x-ray in both images. The C-Arm has amarker, the positions are known
   - Get the line
   - Intersect two lines
   - Define the base coordinate system
   - Origin is defined as the intersection of shaft axis and cutting plane
   - z-axis points along the shaft axis
   - y-axis: AP in images
   - x-axis: Put in such way that the right hand coordinate system is complete
3. Plan (compute wedge planes)
4. Navigate

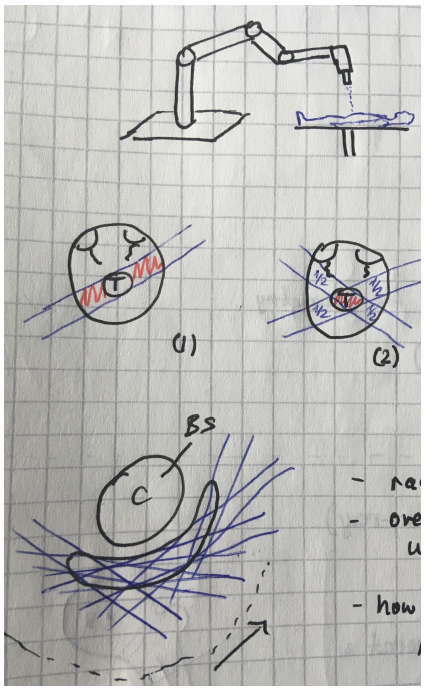
For the Computer Simulation, we define 3 angles:

- flexion angle (x-axis)
- valgus angle (y-axis)
- rotation (z-axis)

Planning for radiosurgery

Scenario: We perform radiosurgery, which means we shoot beams to destroy a specific target in a human body. In order not to damage the surrounding tissue unnecessarily, we shoot various low power beams from various directions, instead of a high power beam from one. By this, we have a high dose in the target and a low intensity in the tissue. The robot performs "Step-And-Shoot".
Problems:

- Radiation is physical, it spreads
- If we go around some important object, we will have an overdose on the inner track and an underdose on the outer track
- How long do we have to activate a beam at which position?
  $\Longrightarrow$ We have to place **beams** and calculate the **weights**.

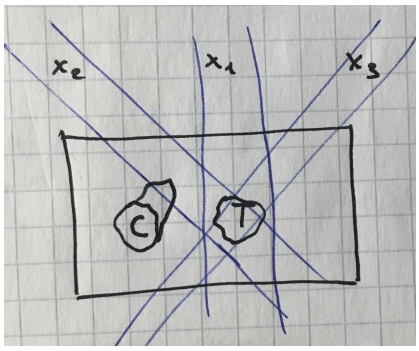


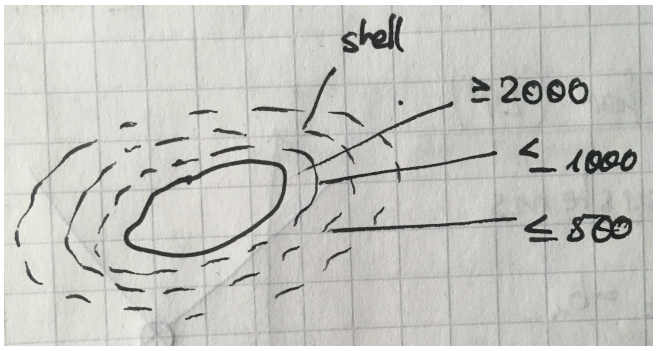
Specification of Goals

1. We want to achieve dose homogenity. It has to be homogenious in the tumor
2. We want to protect the critical region / minimal dose threshold. We specify a minimum value in tumor and a maximum value in tissue.
3. We want to have a high dose gradient around the target
4. Conformity

To achieve this, we have to **deliniate the target** on each positive CT slice, **deliniate the critical region** and **enter the values** as (upper/lower) threshold.

Beam Weights Computation (LP)





We have multiple equations, the a's and b's are either 0 or 1 (Beam Active or Not):

$$a_{1,1}x_1 + \cdots + a_{1,n}x_n \leq critical_1$$

$$\vdots$$

$$a_{m,1}x_1 + \cdots + a_{m,n}x_n \leq c_m$$

$$b_{1,1}x_1 + \cdots + b_{1,n}x_n \leq tumor_1$$

$$\vdots$$

$$b_{m,1}x_1 + \cdots + b_{m,n}x_n \leq t_m$$

We call this the **Linear Feasability Problem (F)**

General Linear Programming

Minimize

$$\min c_1x_1 + \ldots c_nx_n$$

Such that all m constraints are met

$$a_{1,1}x_1 + \cdots + a_{1,n}x_n \leq b_1$$

$$\vdots$$

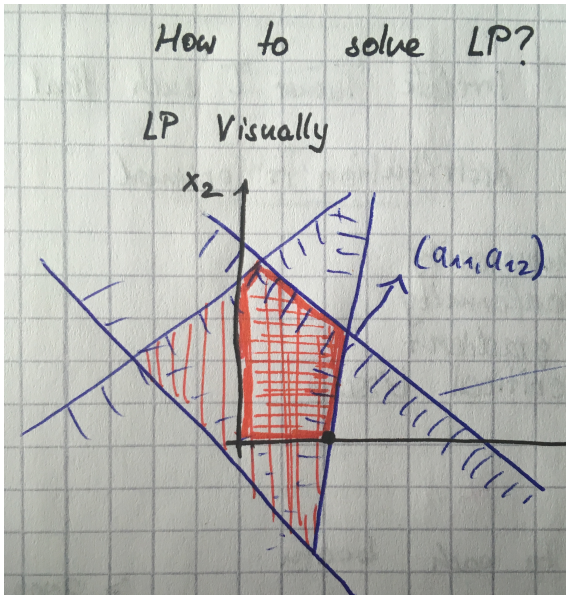
$$a_{m,1}x_1 + \cdots + a_{m,n}x_n \leq b_m$$

Implicit constraint: $x_i \geq 0$, beams can not be negative.

(F) and (LP) are very similar. In fact, (LP) is just (F) with additional minimization or maximization function. Each constraint specifiec one line, alltogether they specify the solvable region containing feasable point, as they satisfy the equation. Linear Programming is the origin for **Support Vector Machines**.

How do we solve (LP)?
Each constraint gives us 1 line. (LP) always returns a vertex of the feasable region.

The **Naive Method for LP** is to look at linear equation systems, and to look at all n-pels (pairs of n constraints) from all $m+n$ constraints. The solutions are, as always, the intersection points.



Infeasibility

How do we deal with infeasibility? Well, we take the original constraint

$$a_{1,1}x_1 + \cdots + a_{1,n}x_n \leq b_1$$

and modify it by inserting 2 new variables s_1, s_1'

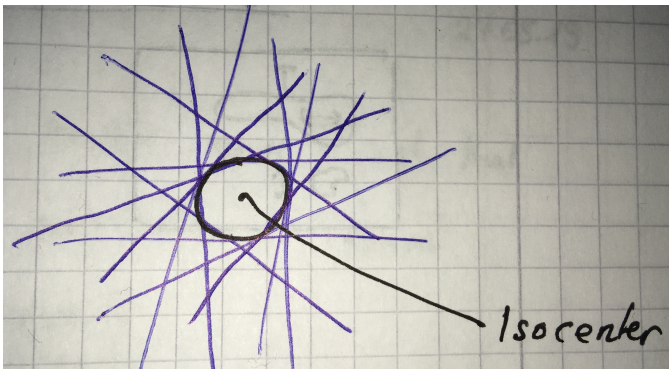
$$a_{1,1}x_1 + \cdots + a_{1,n}x_n + s_1 - s_1' \leq b_1$$

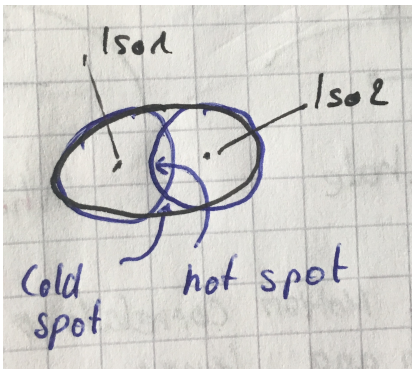
Now, we want to minimize the deviation from the original constraint:

$$\min s_1 + \cdots + s_m + s_1' + \cdots + s_m'$$

Hierarchy of shapes

The most simple shape is a sphere. The **Isocenter** is in the center of the sphere. Next would be an ellipse. The problem is, that an ellipse fits 2 spheres (and two isocenters), but not the whole area is covered, creating hot spots (area covered multiple times) and cold spots (area not covered at all). The solution is to sweep through the area, and the cold spots will vanish.





The area of a tumor will get smaller, if we put isocenters around the tumor (on it's surface) and erase the covered area.
Problems:

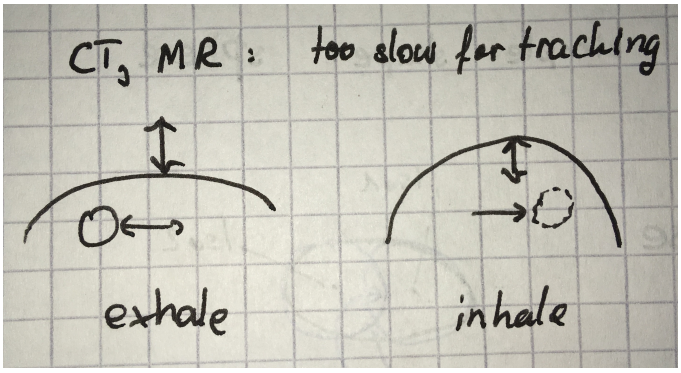
- The dose decreases with depth in tissue
- There is an off center ratio

Overall Planning

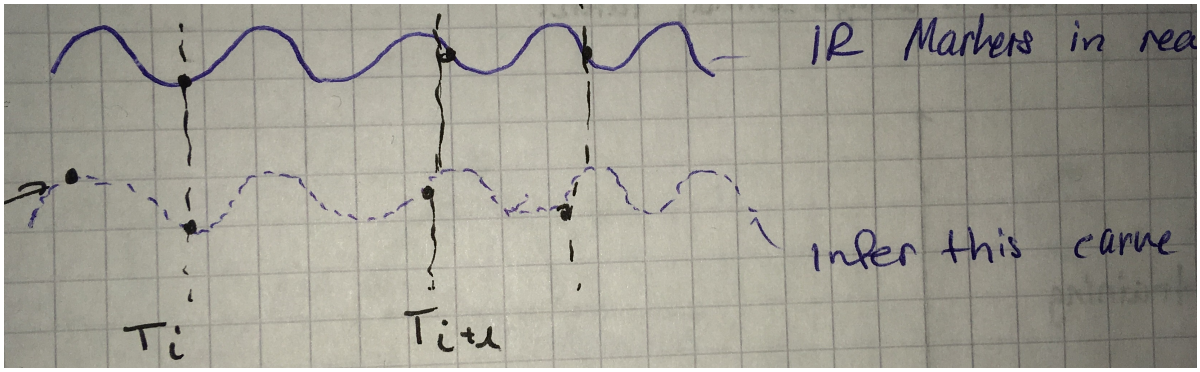
1. Deliniate target (polygon), set dose thresholds (initial and target)
2. Inverse Planning
    - Beam Placement
    - Beam Weights
3. Forward dose and visualize

Motion Correlation and Tracking

Most of the time, we have to use some sort of 4D Planning, because the target moves (e.g. patient breathes). To track a moving target with a robot, we need special sort of imaging that is fast enough to calculate.

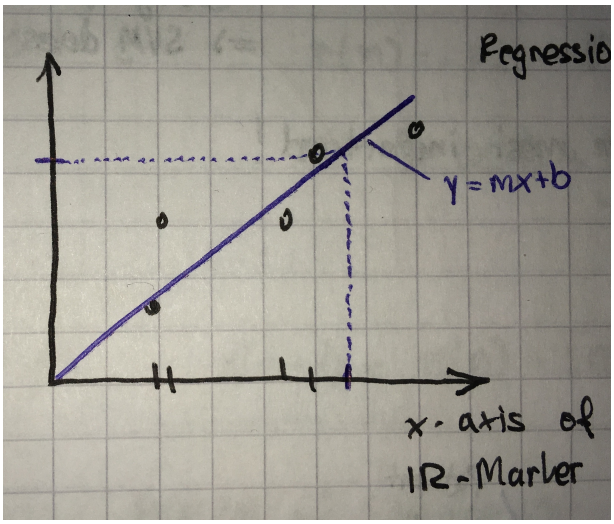


CT and MR are too slow for tracking. So, we create a model that predicts certain motion patterns. We claim that the motions correlate, as they tend to be similar over time.



Thus, the general idea is to **infer the target position** (which we don't see) from skin motion (which we see). We place IR Markers on the skin and gold markers near the tumor to perceive the movement. We then get the real position every timestep T_i, create the correlation, and predict the positions between T_i and T_{i+1}. Summing up, we capture the **Motion Correlation** between skin motion and tumor and, by correlation, we then imply the target's position.

Mathematics behind it: Computing Correlation



In the set of data points, we want to find a regression line that fits to the data. To find the correlation, we use the **Gaussian Least Squares Method**.
For every data point, we have the x and y coordinates. Now, we have to find the m (slope) and the b (offset). we do so by minimizing the function f:

$$f = (y_1 - mx_1 - b)^2 + (y_2 - mx_2 - b)^2 + (y_3 - mx_3 - b)^2 + \dots$$

Problem: Hysteresis (variant delayed behaviour of the caused output variable) Reality is very noisy, and perceived motions are curves, not lines.
Solution: Machine Learning We use support vector machines, as they are flexible and suitable for heterogeneous data.

Motion Prediction

We do get the real time position of the target, but the information is always delayed. We have to compensate for the time lag of robot movement, as the robot is always behind/latent.
We prepare training data set with equidistant spacing between the data points. By this, the information is evenly spaced, but the SVM does not know that. Also, we have to tell the SVM that the most recent calculated point is the most important one.

MULIN: Multi-step Linear Interpolation

$p(n)$: patient in time step n
$r(n)$: robot in time step n
$\Longrightarrow$ To predict p means to compute $p(n+1)$!

We define

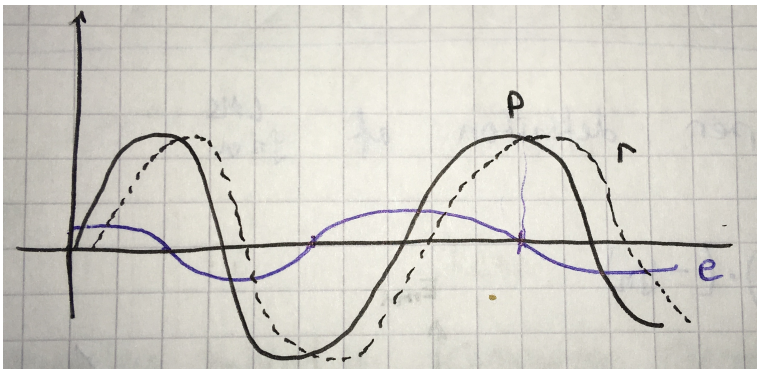
$$p(n+1) = p(n) + [\underbrace{p(n) - r(n)}_{\text{Error in last step}}]$$

By this, error e should be easier to predict than p, as e has a way smaller amplitude than p:

$r(n) = p(n-1)$
$p(n+1) = p(n) + [\underbrace{p(n) - p(n-1)}_{e(n)}]$
$e(n+1) = e(n) + [e(n) - e(n-1)] = e^*$, this is MULIN of order 1
$p(n+1) = p(n) + e(n+1)$
Now, our goal is to predict e^*:

$$e^*(n+1) = e^*(n) + [e^*(n) - e^*(n-1)]$$

How far do we look into the past?



Fast Lane Method

We select a parameter k which is the best/fastest. We keep k until some other k' becomes best. (Switch lanes like on ze gud old Autobahn).

LMS Prediction

We see that Robot Control has a lot to do with learning.
$e_{n+1} = y_{n+1} - y_{n+1}^{\text{LMS}}$
$y_{n+1}^{\text{LMS}} = w_n \cdot y_n$, where w_n is the weight factor $w_{n+1} = w_n + \mu \cdot e_n \cdot y_n$

$w_0 = 1$
$y_0^{\text{LMS}} = y_0$
$\mu = 2$ $E_{n+1} = (y_{n+1} - y_{n+1}^{\text{LMS}})^2$
$\frac{\delta E_{n+1}}{\delta w} = ?$

$E_{n+1} = (y_{n+1} - w_n \cdot y_n)^2$ per definition of y_{n+1}^{LMS}
$$\frac{\delta E_{n+1}}{\delta w} = 2(y_{n+1} - w_n \cdot y_n) \cdot (-y_n)$$
$$= 2(\underbrace{y_{n+1} - y_{n+1}^{\text{LMS}}}_{e_{n+1}}) \cdot (-y_n)$$
$$= 2 \cdot e_{n+1} \cdot (-y_n) = -2 \cdot e_{n+1} \cdot y_n$$

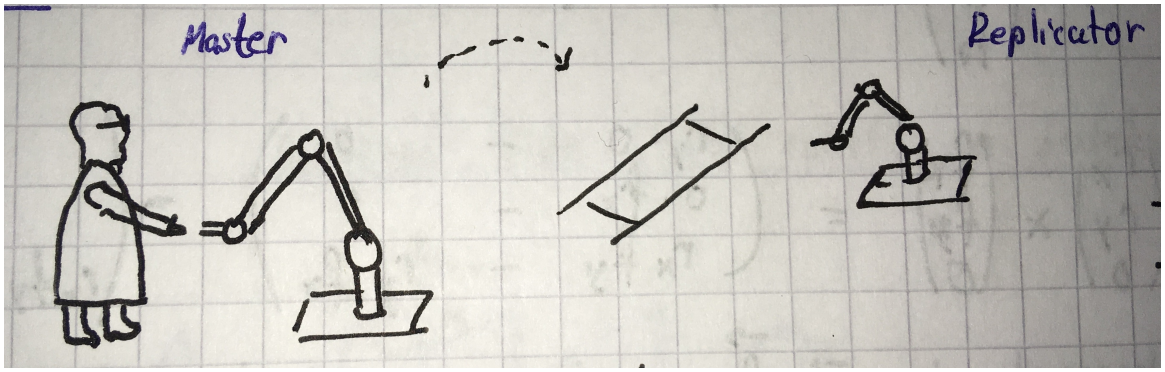
Assume error in last step is equal to this step. We then replace e by e_{n+1}. To reduce the error, we must reduce w. Vice Versa, if the Gradient is negative, we must increase w:

$\Longrightarrow$ If Gradient **pos**, reduce w to reduce E

$\Longrightarrow$ If Gradient **neg**, increase w to reduce E

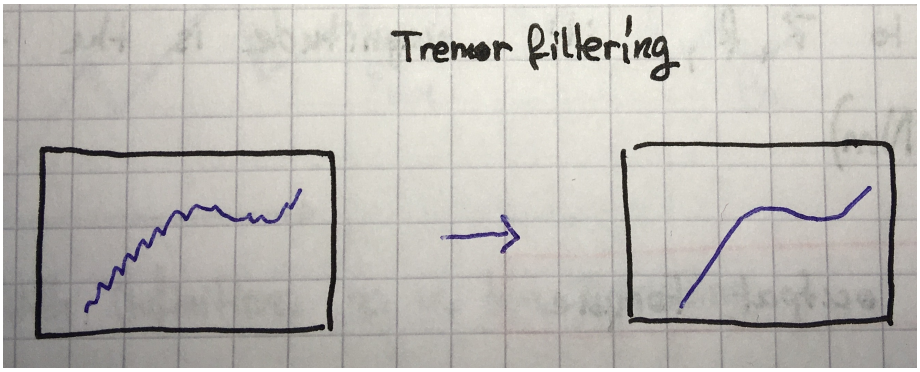
In comparision with MULIN, LMS is not as robust.

Motion Replication



Motion Replication is basically a scenarion of two robots, where the **replicator** imitates the motion of the **master**. This is a popular choice for

- Microsurgery
- Motion scaling for deliquate operations
- Tremor filtering



We want to conclude from the past, what is what and which is which. Most of the time, Robots are not position oriented, but rather force oriented (You usually calculate Current→Force→Angle→Position)
Problems:

- We have to compensate gravity (which is also different in different parts on the planet!)
- We have to compensate lag (unwanted Overreaction)

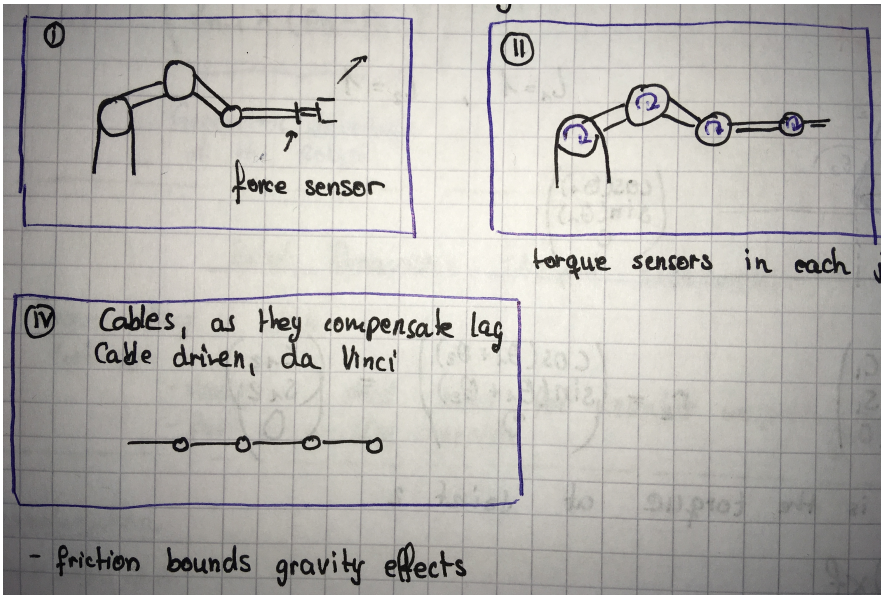
For data acquisition, we can use:

(i) Force Sensors in every joint of the robot

(ii) Torque Sensors in every joint

(iii) Redundant Force measurements by combination of (i) and (ii)

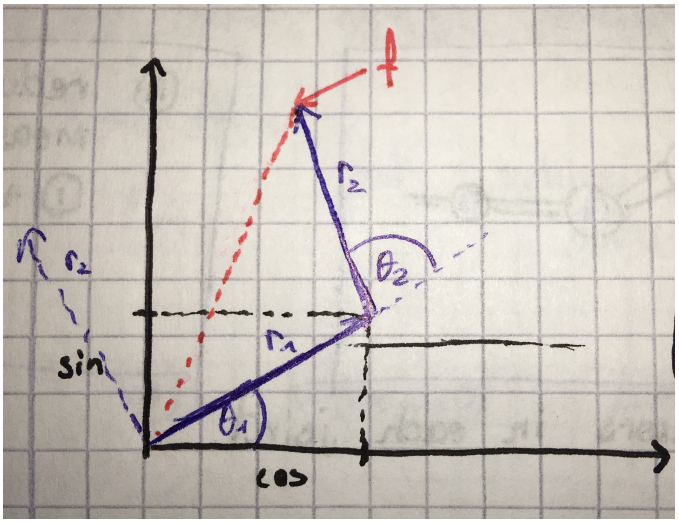
(iv) Cables. This is a popular choice as they compensate lag and the friction bounds gravity effects, see daVinci Robot



Assume a force f acts on position r. We can calculate vector $\boldsymbol{t}$ by building the cross product of $\boldsymbol{r}$ and $\boldsymbol{f}$. Vector $\boldsymbol{t}$ is then orthogonal to the $\boldsymbol{r}$ and $\boldsymbol{f}$, and it's magnitude is the torque τ:

$$\boldsymbol{t} = \boldsymbol{r} \times \boldsymbol{f}$$

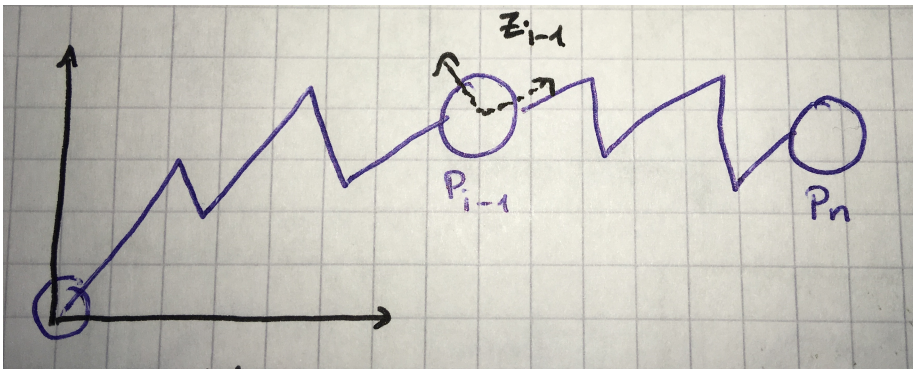
$$\tau = ||\boldsymbol{t}||, \quad \text{unit Nm}$$



Calculation: $r_1 = \begin{pmatrix} l_1 \cdot \cos_{\theta_1} \\ l_1 \cdot \sin_{\theta_1} \\ 0 \end{pmatrix} \implies$ For further $r_i$just add the lengths and angles

$\boldsymbol{t_n} = \boldsymbol{r_n} \times \boldsymbol{f}, \quad \boldsymbol{t_{n-1}} = (\boldsymbol{r_n} + \boldsymbol{r_{n-1}}) \times \boldsymbol{f}$ and so on

General Case (3D Robot, n joints)



We want to transfor $\boldsymbol{f}$ to effector coordinate system.
For an n-Joint robot, the torques are calculated through

$$\begin{pmatrix} \tau_1 \\ \vdots \\ \tau_n \end{pmatrix} = \begin{pmatrix} z_0 \times (p_n - p_0) \\ \vdots \\ z_{n-1} \times (p_n - p_{n-1}) \end{pmatrix} .^{\mathbf{B}} f$$

where the matrix is the **Geometric Jacobian of the Robot**

Robot Dynamics: A thought experiment

Assume we have a person sitting in a chair. We have to possibilities to get up:

- get up slowly: We need a counter weight (lean more forward)
- get up fast: We use our inertia

Interestingly, the paths from both approaches are different.

In **Robot Dynamics**, we use the equation of motion, velocity and acceleration as funtion of position:

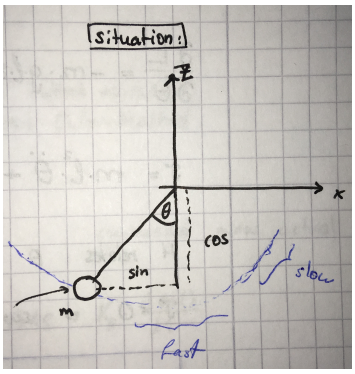
$\theta(t)$ is the joint **angle** as function of time
$\dot{\theta} = \theta'(t)$ is the **velocity** of the joint
$\ddot{\theta} = \theta''(t)$ is the **acceleration** in the joint

We further recall the **Kinetic Energy** $\mathcal{K}$ as

$$\mathcal{K} = \frac{1}{2} \cdot m \cdot v^2$$

and the **Potential Energy** $\mathcal{P}$ as

$$\mathcal{P} = m \cdot g \cdot h$$



The position $r(t)$ is calculated by (assume only x and y axis as in figure)

$$r(t) = \begin{pmatrix} l \cdot \sin(\theta) \\ 0 \\ -l \cdot \cos(\theta) \end{pmatrix}, \quad \dot{r} = \begin{pmatrix} l \cdot \cos(\theta) \cdot \dot{\theta} \\ 0 \\ l \cdot \sin(\theta) \cdot \dot{\theta} \end{pmatrix}$$

$$\mathcal{K} = \frac{1}{2} \cdot m \cdot (\dot{r})^2$$

$$= \frac{1}{2} \cdot m \cdot l^2 \cdot \dot{\theta}^2 \cdot \underbrace{[\cos^2(\theta) + \sin^2(\theta)]}_{=1} = \frac{1}{2} \cdot m \cdot l^2 \cdot \dot{\theta}^2$$

$$\mathcal{P} = m \cdot g \cdot h$$

$$= -m \cdot g \cdot l \cdot \cos(\theta)$$

Lagrange Theorem

We define the Lagragne Function with independent parameters $\theta, \dot{\theta}$

$$L(\theta, \dot{\theta}) = \mathcal{K}(\theta, \dot{\theta}) - \mathcal{P}(\theta)$$

We then calculate the torque magnitude τ through Lagrange Theorem

$$\frac{d}{dt}\frac{\delta L}{\delta \dot{\theta}} - \frac{\delta L}{\delta \theta} = \tau$$

Note that dt is a 1D differentiation, and $\delta\dot{\theta}$ is a partial differentiation.
Continuing:

$$L = \frac{1}{2} \cdot m \cdot l^2 \cdot \dot{\theta}^2 + m \cdot g \cdot l \cdot \cos(\theta)$$

$$\frac{\delta L}{\delta \dot{\theta}} = m \cdot l^2 \cdot \dot{\theta}$$

$$\frac{d}{dt}\frac{\delta L}{\delta \dot{\theta}} = m \cdot l^2 \cdot \dot{\theta}$$

$$\frac{\delta L}{\delta \theta} = -m \cdot g \cdot l \cdot \sin(\theta)$$

In conclusion, our torque is

$$\tau = m \cdot l^2 \cdot \dot{\theta} + m \cdot g \cdot l \cdot \sin(\theta)$$

We see that the Lagrange Theorem relates θ and $\dot{\theta}$ to τ. Assuming a free swing ($\tau = 0$), we get

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta)$$

If we want to hold the robot in a static position, the acceleration $\ddot{\theta}$ is 0. This leads to τ being

$$\tau = m \cdot g \cdot l \cdot \sin(\theta)$$

as the only force actiong on the robot is the Gravity vecotor $(0, 0, -mg)^T$

The Lagrange Theorem can be applied to every joint i by

$$\tau_i = \frac{d}{dt}\frac{\delta L}{\delta \dot{\theta}_i} - \frac{\delta L}{\delta \theta_i}$$

But what if the joints are not rotational? We generalize further to

$$y_i = \frac{d}{dt}\frac{\delta L}{\delta \dot{q}_i} - \frac{\delta L}{\delta q_i}$$

However, the standard method nowadays is the **Newton-Euler-Method**, not deeply discussed in the lecture.

PID Controller

We define the error function $e(t)$ as

$$e(t) = \theta_{Command} - \theta_{Actual}(t)$$

Then, the **PID Controller** is defined as

$$\tau(t) = K_p \cdot e(t) + K_i \int_0^t e(\xi)d\xi + K_D \cdot \frac{de(t)}{dt}$$

where the parameters K_p, K_i, K_D are chosen in advance in training phase.