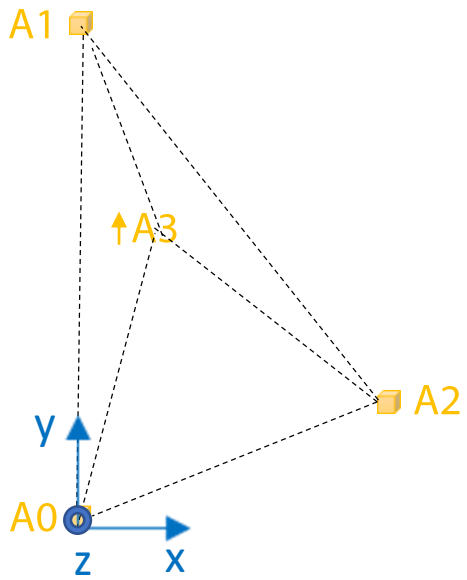
Algoritmi per calibrazione di ancore DW1000

Sistemi di guida e navigazione - Federica Fioretti

Presentazione del problema



Supponendo di avere il setup, costituito da 4 ancore, il sistema di riferimento è stato costruito nella maniera seguente:

- L'origine del sistema di riferimento è colocato rispetto all'*ancora0.* Quindi si assume che essa occupi la posizione A0={0, 0, 0}.
- L'asse y viene costruito sulla base della posizione dell'*ancora1*. La sua direzione è data, infatti, dalla retta passante per A0 e A1, con A1 sui valori positivi di y.
- La direzione delle x crescenti viene determinata affinché sia positiva la coordinata x della posizione dell'*ancora2*, A2.

Questa notazione viene mantenuta nella presentazione del lavoro. Vengono, inoltre, indicati i range tra due generiche ancore i e j come: r_{ij}. Dalla conoscenza dei range, si vogliono ottenere le posizioni delle ancore.

Caso 1: stessa quota da terra e range esatti

Sono stati applicati i seguenti algoritmi per effettuare il calcolo delle posizioni delle ancore a partire dai range esatti forniti come input:

- *Classical Multidimensional Scaling:* attraverso il comando di Matlab `cmdscale`, oppure usando la funzione realizzata `computeCMDS.m`, che richiede in ingresso le distanze euclidee relative tra i punti occupati dalle ancore, ordinati come:

$$E = \{r_{01}, r_{02}, r_{03}, r_{12}, r_{13}, r_{23}\}$$

  Restituisce i punti ricostruiti secondo un certo sistema di riferimento rispettando questi range. È necessario individuare una **matrice di rotazione** che riporti queste coordinate nel sistema riferimento di interesse.

- *Algoritmo algebrico*: sfrutta l'ipotesi di base che le ancore alla stessa quota da terra formino un triangolo di cui tutte le caratteristiche sono note. Ricavando gli angoli interni con il teorema di Carnot, possono essere calcolate le coordinate delle tre ancore sul piano xy = 0. Avendole determinate, è possibile ricondursi alle coordinate dell'ancora 3 (a quota più alta) sfruttando le distanze euclidee note, relative alle ancore di cui sono state ricavate le coordinate.

Algoritmo algebrico[1/2]

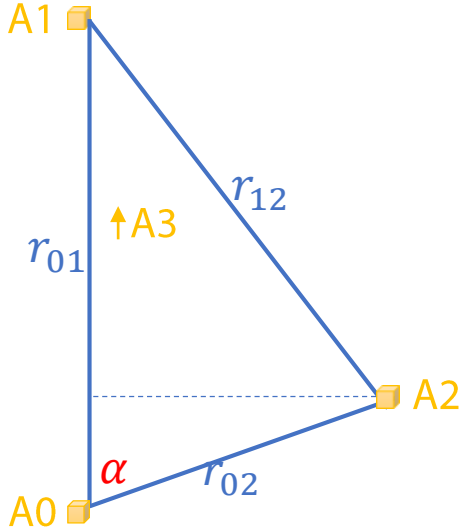
Considerando A0, A1, A2 alla stessa quota da terra, si sceglie un sistema di coordinate t. c.

- A0 sia l'origine [0, 0, 0]
- A1 sia sull'asse y, lungo la direzione positiva, con coordinate [0, r_{01}, 0].

Le coordinate di A2, si trovano come:

$$[r_{02}\sin\alpha\,, r_{02}\cos\alpha\,, 0], \qquad \alpha = \cos^{-1}\frac{{r_{01}}^2 + {r_{02}}^2 - {r_{12}}^2}{2r_{01}r_{02}}$$

Dalle coordinate di A0, A1, A2 ricavate e dai range relativi di queste con l'ancora 3, si ottengono le coordinate di A3.



$$\sqrt{x_3^2 + y_3^2 + z_3^2} = r_{03}$$

$$\sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2 + z_3^2} = r_{13}$$

$$\sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + z_3^2} = r_{23}$$

Algoritmo algebrico[2/2]

Infatti dalla conoscenza di r_{03}, r_{13}, r_{23} e avendo ricavato y_1, x_2, y_2, si esplicitano le incognite x_3, y_3, z_3 dalle equazioni:

$$\sqrt{x_3^2 + y_3^2 + z_3^2} = r_{03}, \qquad \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2 + z_3^2} = r_{13}, \qquad \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + z_3^2} = r_{23}$$

Ottenendo:

$$z_3 = \sqrt{r_{03}^2 - x_3^2 - y_3^2} \qquad (1)$$

$$x_3^2 - 2x_2x_3 + x_2^2 + (y_3 - y_2)^2 + r_{03}^2 - x_3^2 - y_3^2 = r_{23}^2 \;\Rightarrow\; x_3 = \frac{x_2^2 + (y_3 - y_2)^2 + r_{03}^2 - y_3^2 - r_{23}^2}{2x_2}$$

Sostituendo nella seconda equazione z_3 e x_3 si ricava y_3: $y_3 = \frac{y_1^2 + r_{03}^2 - r_{13}^2}{2y_1}$

Da sostituire in z_3 e x_3 per ricavarne le espressioni finali.

Classical Multidimensional Scaling

Partendo da una matrice quadrata, contenente le *disparity* tra elementi di un insieme, l'algoritmo di scaling multidimensionale assegna a ognuno una posizione in uno spazio *n*-dimensionale, con n stabilito a priori.

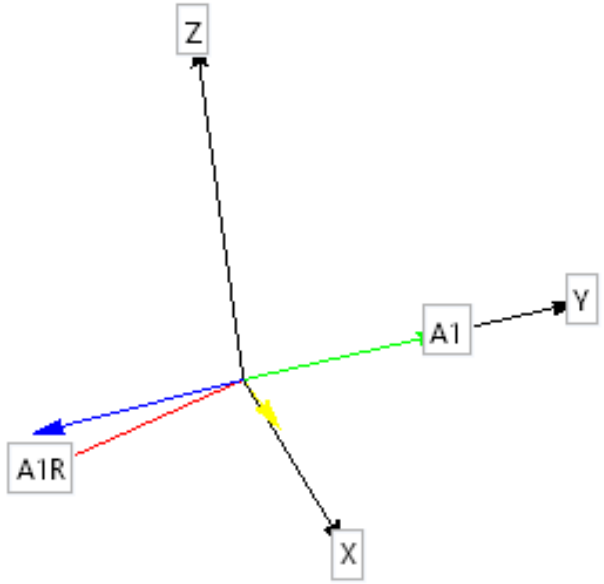
Nello scaling multidimensionale classico, le *disparity* sono le distanze euclidee. Date quindi le distanze euclidee è possibile costruire una matrice *X* di coordinate cartesiane di questi punti nello spazio euclideo tridimensionale. È già presente in Matlab il comando `cmdscale` che lo implementa.

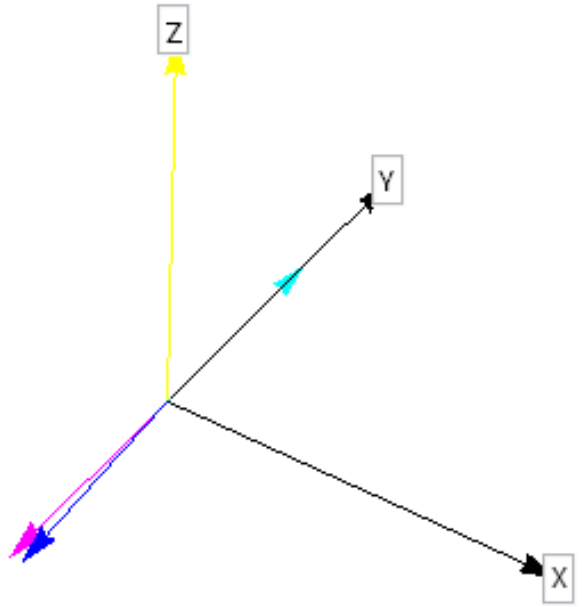
In previsione di utilizzare questa tecnica in altri linguaggi di programmazione vengono esplicitati di seguito i passaggi di cui è composta, che sono stati implementati nella funzione `computeCMDS.m`:

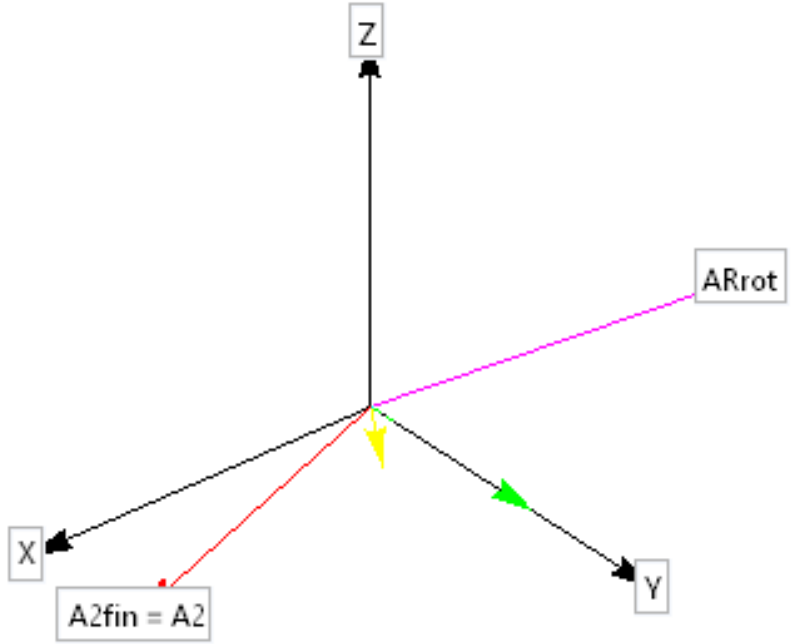
- Costruire la matrice di distanze al quadrato: $D = [d_{ij}^2]$ Effettuare la doppia centratura: $B = -\frac{1}{2}JDJ$, $J = I - \frac{1}{N}11^T, \quad N = numero\ di\ punti$ nel nostro caso 4.
- Determinare gli *m* autovalori a modulo massimo e corrispondenti autovettori di B, dove *m* è la dimensione dello spazio di arrivo, nel nostro caso lo spazio tridimensionale.
- Calcolare $X = E_m\Lambda_m^{1/2}$, E_m *matrice degli autovettori* e Λ_m *matrice diagonale degli autovalori di* B.

Caso 1- Multidimensional Scaling

(rotazione implementata nella funzione `rotateSet.m`)







Prima rotazione:

Individuazione dell'angolo descritto dal punto A1 ricostruito e traslato (A1R) con il piano xy=0.

Ruotare di questo angolo attorno all'asse dato dal prodotto vettoriale del vettore applicato in A (proiezione su xy=0 di A1R) e A1R.

Seconda rotazione:

Individuazione dell'angolo ϑ descritto dal punto A1R ruotato (x1r, y1r, 0) e l'asse y.

Ruotare di un opportuno angolo attorno all'asse z definito secondo due casi a seconda del valore di y1r:

- Se y1r < 0 $\quad \theta = \pi + \frac{x1r\ y1r}{|x1r\ y1r|}\vartheta$
- Se y1r > 0 $\quad \theta = \frac{x1r\ y1r}{|x1r\ y1r|}\vartheta$

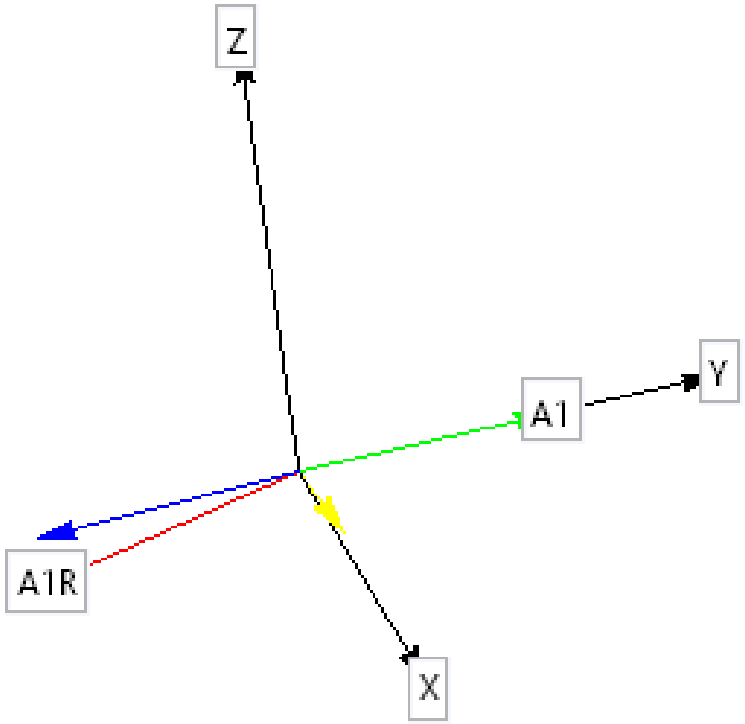
Terza rotazione:

Individuazione dell'angolo di cui ruotare il punto A2 ricostruito a cui sono state applicate le precedenti trasformazioni (ARrot).

Ruotare di questo angolo attorno all'asse dato dal prodotto vettoriale del versore di y e del vettore applicato in ARrot.

Caso 1- Multidimensional Scaling

(Prima rotazione implementata nella funzione `rotateSet.m`)



Considerando il set di punti ricostruiti con coordinate secondo un generico sistema di riferimento, ottenuto dall'algoritmo di *Classical Multidimensional Scaling*. Poiché si può discriminare quali di esse siano relative a una specifica ancora, si trasla tutto il set della coordinata associata all'*ancora0*, che risulta essere, così, l'origine del sistema. Si indicano le coordinate dei punti ricostruiti traslati come: A0R, A1R, A2R, A3R.

Prima rotazione:

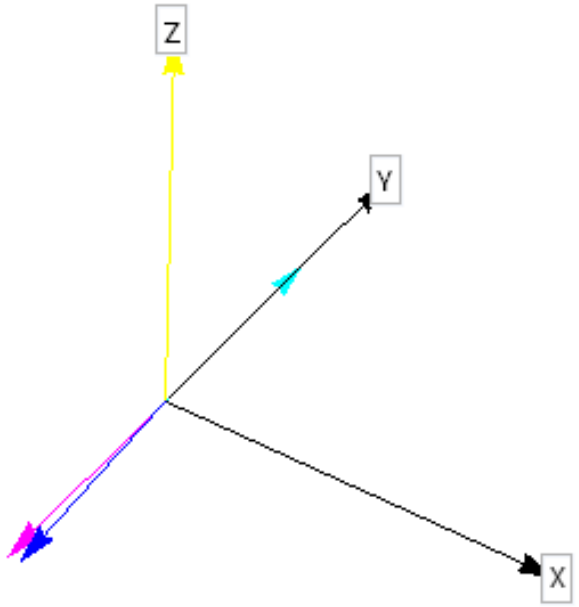
- Definizione del punto A, proiezione di A1R sul piano xy=0.
- Individuazione dell'angolo descritto con il piano xy=0 dal vettore applicato in A0R=O congiungente il punto A1R:

$$\psi = \cos^{-1}\frac{\|A\|}{\|A1R\|}$$

- Ruotare di questo angolo attorno all'asse (in giallo) dato dal prodotto vettoriale del vettore applicato in O con direzione definita dal punto A e A1R.

Caso 1- Multidimensional Scaling

(Seconda rotazione implementata nella funzione `rotateSet.m`)



Seconda rotazione:

- Definizione del punto B sull'asse y, collegato dal vettore (in magenta) applicato al punto A0R.
- Individuazione dell'angolo ϑ descritto dal vettore applicato in A0R che congiunge A1R trasformato (in blu) secondo la precedente rotazione, di coordinate (x1r, y1r, 0) e quello congiungente il punto B:

$$\vartheta = \cos^{-1} \frac{\|B\|}{\|A1Rtras\|}$$

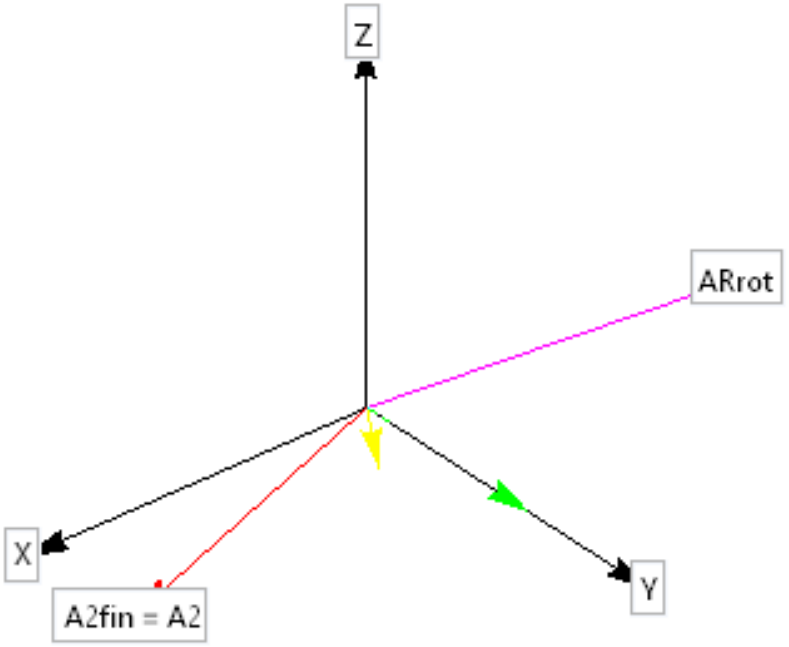
- Ruotare di un opportuno angolo θ attorno all'asse z definito secondo due casi a seconda del valore di y1r:

a) Se y1r < 0 $\quad \theta = \pi + \frac{x1r\ y1r}{|x1r\ y1r|}\vartheta$

b) Se y1r > 0 $\quad \theta = \frac{x1r\ y1r}{|x1r\ y1r|}\vartheta$

Caso 1- Multidimensional Scaling

(Terza rotazione implementata nella funzione `rotateSet.m`)



Terza rotazione:

- Individuazione dell'angolo di cui ruotare il punto A2 ricostruito a cui sono state applicate le precedenti trasformazioni (ARrot). Considerando le componenti del vettore risultato del prodotto vettoriale del versore di y e del vettore applicato in ARrot, normalizzato (in giallo), la componente su z $axis_z$ corrisponde al coseno dell'angolo per cui effettuare la rotazione. Si vuole infatti ruotare di un angolo per cui portare a 0 la quota del punto ARrot, che corrisponde a allineare l'asse con il versore dell'asse z.
$$\phi = \cos^{-1} axis_z.$$

- Ruotare di questo angolo attorno all'asse (in giallo)

Caso 1 - Risultati

Command Window

```
P =

         0         0    4.7220    0.1430
         0    2.9810    1.7210    1.6420
         0         0         0    2.2240


Pr_CMD =

         0   -0.0000    4.7220    0.1430
         0    2.9810    1.7210    1.6420
         0   -0.0000    0.0000   -2.2240


P_algeb =

         0         0    4.7220    0.1430
         0    2.9810    1.7210    1.6420
         0         0         0    2.2240

fx >>
```

Command Window

```
P =

         0         0   -4.7220    0.1430
         0    2.9810   -1.7210   -1.6420
         0         0         0    2.2240


Pr_CMD =

         0   -0.0000    4.7220   -0.1430
         0    2.9810   -1.7210   -1.6420
         0   -0.0000   -0.0000    2.2240


P_algeb =

         0         0    4.7220   -0.1430
         0    2.9810   -1.7210   -1.6420
         0         0         0    2.2240

fx >>
```

Caso 1 - Risultati

Si nota dai risultati come vi siano ambiguità di soluzione intrinseche al problema:

- La direzione delle y crescenti è ben definita per come è stato costruito l'asse. La **direzione positiva** dell'**asse x non è definita** dalla posizione delle ancore nel setup. In entrambi gli algoritmi il punto occupato dall'ancora2 viene ricostruito con coordinata x positiva e questo influenza anche la determinazione del punto occupato dall'ancora3. Per risolvere questo problema è necessario fornire come **input** se la posizione occupata dall'ancora2 abbia coordinata x positiva o meno nel nostro sistema di riferimento. Compensare eventualmente il segno delle x di ancora2 e 3.

- La **direzione positiva dell'asse z non è definita** dalla posizione delle ancore nel setup. Sulla base del sistema di riferimento scelto, fornire come input se l'ancora3 sia posta ad una coordinata z positiva o meno.

- Compensando i segni di queste coordinate nei punti ricostruiti si riesce ad ottenere i punti originali. Si mostra di seguito come è stata gestita la compensazione dei segni in codice Matlab.

Compensazione dei segni

Assumendo quindi come input i segni delle coordinate x del punto A2 e z del punto A3 nel nostro sistema di riferimento, si effettua la compensazione dei segni.

Si riporta il pezzo di codice che gestisce questo problema all'interno dello script `SimulazioneCal.m` (simula la misura dei range relativi soggetta a rumore e delay con ricostruzione di un certo set di punti).

```
%% Compensazione Segni
if a2_real(1)*a2_alg(1)<0
    a2_alg(1)=(-1)*a2_alg(1);
    a3_alg(1)=(-1)*a3_alg(1);
end
if a2_real(1)*Pr_CMD(3,1) < 0
    Pr_CMD(3,1)=(-1)*Pr_CMD(3,1);
    Pr_CMD(4,1)=(-1)*Pr_CMD(4,1);
end

if a3_real(3)>0
   if a3_alg(3)< 0
       a3_alg(3)=(-1)*a3_alg(3);
   end
   if Pr_CMD(4,3) < 0
      Pr_CMD(4,3) = (-1)*Pr_CMD(4,3);
   end
end
```

Caso 2: diversa quota da terra e range esatti

Questo problema è stato affrontato in due modi, applicando l'*Algoritmo Algebrico*, a seconda degli input disponibili.

A. Fornendo in **input**: le distanze euclidee relative e, in aggiunta, tutte le **quote da terra** delle posizioni delle ancore è possibile ricostruire i punti, modificando le equazioni dell'algoritmo. Se la quota dell'ancora 3 non viene fornita non risulta possibile calcolare le posizioni con questa tecnica.

B. Fornendo come input le distanze euclidee relative: l'algoritmo lavora con le equazioni per Ancore 0,1,2 poste sul piano xy=0, dunque i punti ricostruiti andrebbero ruotati secondo una **matrice di rotazione** da individuare, al fine di ottenere i punti nel sistema secondo le vere coordinate delle ancore.

Caso 2,A: Input range e quote

Avendo fornito come input tutte le coordinate z dei punti dello spazio occupati dalle ancore, si riescono a ricavare i punti A0, A1, A2, A3 risolvendo il seguente sistema di equazioni:

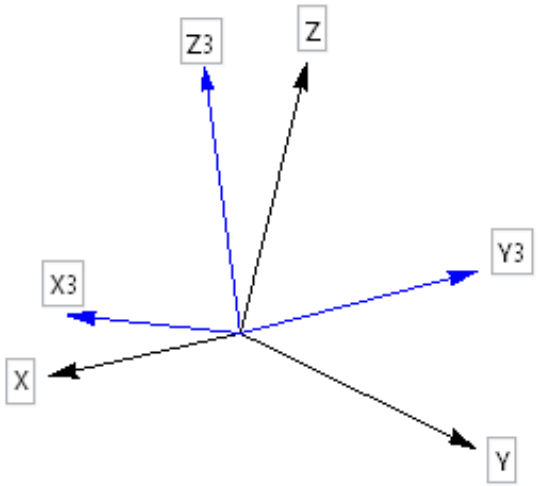
$$r_{01} = \sqrt{y_1^2 + z_1^2} \qquad z_1 = h_1 - h_0 \qquad \Rightarrow\ y_1 = \sqrt{r_{01}^2 - z_1^2} \Rightarrow A1 = (0, y_1, z_1)$$

$$r_{02} = \sqrt{x_2^2 + y_2^2 + z_2^2} \qquad z_2 = h_2 - h_0 \qquad x_2^2 = r_{02}^2 - z_2^2 - y_2^2$$

$\Rightarrow r_{12}^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$ da cui ricavare y_2 e sostituirlo in x_2, ottenendo A2.

Dai range relativi da A3 rispetto i punti A0, A1, A2 e la quota h_3 si ottengono le coordinate di A3.

Caso 2,B: Input range




```
In[80]:= A1R = {0, 3.5898, 0};
In[81]:= R.A1R
Out[81]= {9.60694×10^-6, 2.96234, 2.02761}
In[82]:= A1
Out[82]= {0, 2.981, 2}
In[83]:= A2R = {4.7237 , 1.9863 , 0};
In[84]:= R.A2R
Out[84]= {4.65037, 1.17081, 1.80609}
In[85]:= A2
Out[85]= {4.722, 1.721, 1}
In[86]:= A3R = {0.1176 , 2.6026 , 0.9356};
In[87]:= R.A3R
Out[87]= {-0.0484354, 1.61579, 2.24713}
In[88]:= A3 = [0.143 , 1.642 , 2.224]
```

Considerando i sistemi di riferimento costruiti rispettivamente secondo i set di punti con A0, A1, A2 a stessa quota da terra {S} e con diverse quote {S3}.

Si vuole applicare l'*algoritmo algebrico*, che lavora sull'ipotesi di base che le ancore 0, 1, 2 si trovino alle stesse quote da terra ma si forniscono in input i **range** dei punti a **quote z generiche**.

Individuando la **matrice di rotazione** che allinei il sistema S al sistema S3, si vuole verificare se i punti ottenuti sono quelli originali a quote arbitrarie.

Si riporta il risultato della rotazione totale.

Difficoltà nell'individuare gli angoli. (vedere notebook *Rotation.nb*)

Algoritmo ML - Formulazione del problema [1/2]

tratto dalla tesi ”Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS” di Jeroen Hol

Considerando come variabili misurate i TOA da ogni ancora m-sima, descritte nella maniera seguente:

$$y_{u,m} = \tau + \|r_m - t\|_2 + \Delta\tau_m + \delta_{u,m} + e_{u,m}$$

Dove:

- τ è il tempo di trasmissione dell’impulso
- r_m è la posizione dell’ancora m-sima (Rx)
- t è la posizione del target (Tx)
- $\Delta\tau_m$ è l’offset di clock dell’ancora m-sima
- $\delta_{u,m}$ è il contributo di ritardo dovuto a cause quali NLOS, multipath..
- $e_{u,m}$ è il contributo di rumore di misura gaussiano

Algoritmo ML - Formulazione del problema [2/2]

tratto dalla tesi ”Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS” di Jeroen Hol

- Si considera di avere a disposizione *K* misure di TOA per ogni ancora m-sima.

- Si nota come il TOA sia affetto da componenti di disturbo, che hanno un effetto negativo sulla qualità della stima delle posizioni di ancore e target.

- OBIETTIVO: Stimare i parametri θ che producono i TOA misurati, attraverso una formulazione del problema come una stima *Maximum Likelihood* vincolata.

$$\theta = (\{t, \tau_k\}, \{r_m, \Delta\tau_m\}) \qquad k = 1, \dots, K \qquad m = 0, \dots, 3$$

Stima *ML* Vincolata

tratto dalla tesi ”Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS” di Jeroen Hol

Avendo assunto rumore di misura Gaussiano, questo permette di modellare le variabili TOA $y_{u,mk}$ come:

$$p\left(y_{u,mk}, \theta\right) = \frac{1}{\sqrt{2\pi\sigma_u^2}} \exp\left(-\frac{1}{2}\varepsilon_{u,mk}^2(\theta)\right) \qquad (1)$$

Dove i residui normalizzati sono:

$$\varepsilon_{u,mk}(\theta) = \sigma_u^{-1}\left(\tau_k + \|r_m - t\|_2 + \Delta\tau_m - y_{u,mk}\right) \qquad (2)$$

L’obiettivo della stima *ML* è di identificare i parametri che più verosimilmente hanno prodotto quei dati, ovvero che massimizzino (1), o equivalentemente minimizzino l’argomento dell’esponenziale, nel nostro caso.

Stima *ML* Vincolata

tratto dalla tesi ”Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS” di Jeroen Hol

$$\hat{\theta}_{ML} = \arg\min_{\theta} \frac{1}{2} \sum_{k=1}^{K} \sum_{m=0}^{3} \varepsilon_{u,mk}^{2}(\theta)$$

Vincoli:

- $\boldsymbol{A_m r_m = 0}$, $m = 0, 1, 2$ (derivante dalla scelta del sistema di coordinate)
  $A_0 = [e_1, e_2, e_3]^T$, $A_1 = [e_2, e_3]^T$, $A_2 = [e_3]^T$, e_i, vettori base canonica di $\mathbb{R}^3$
- $\boldsymbol{\Delta\tau_0 = 0}$
- $\boldsymbol{r_m - t = 0}$ in fase di calibrazione si pone il target vicino all’ancora m-esima.

Calibrazione

$$\hat{\theta}_{ML} = \arg\min_{\theta} \frac{1}{2} \sum_{k=1}^{K} \sum_{m=0}^{3} \varepsilon_{u,mk}^2(\theta) \qquad \varepsilon_{u,mk}(\theta) = \sigma_u^{-1}\left(\tau_k + \tfrac{1}{c}\|r_m - t\|_2 + \Delta\tau_m - y_{u,mk}\right)$$

Risolvere questo problema ai minimi quadrati per individuare i $\theta_0 = \{\tau_k, \Delta\tau_m, r_m\}$, con i seguenti vincoli:

1. $r_0 = [0,0,0], \quad r_1 = [0, r_{01}, 0], \quad r_2 = [x_2, y_2, 0], \qquad \delta_{u,m} = 0$
2. $\Delta\tau_0 = 0$
3. $r_0 - t = 0$, considerando il target in prossimità dell'ancora 0, $r_0 = t = [0,0,0]$.

Derivando parzialmente $J = \frac{1}{2}\sum_{k=0}^{K} \varepsilon_{u,mk}^2$ rispetto a ciascuno dei parametri, ponendo a 0:

$$\frac{\partial J}{\partial \tau} = \sum_{k=1}^{K} 4\tau_k - y_{u,0k} + \frac{1}{c}\|r_1\|_2 + \Delta\tau_1 - y_{u,1k} + \frac{1}{c}\|r_2\|_2 + \Delta\tau_2 - y_{u,2k} + \frac{1}{c}\|r_3\|_2 + \Delta\tau_3 - y_{u,3k} = 0 \quad (1)$$

$$\frac{\partial J}{\partial \Delta\tau_m} = \sum_{k=1}^{K} \tau_k + \frac{1}{c}\|r_m\|_2 + \Delta\tau_m - y_{u,mk} = 0 \quad (2)$$

$$\frac{\partial J}{\partial x_m} = \sum_{k=1}^{K} \frac{1}{c}\left(\tau_k + \frac{1}{c}\|r_m\|_2 + \Delta\tau_m - y_{u,mk}\right)\frac{x_m}{\|r_m\|_2} = 0\,, \qquad \text{non fornisce un'altra equazione}$$

Calibrazione – Equazioni

Utilizzando i valori dei range misurati, si ottengono:

τ: dalla derivata parziale di J rispetto $\Delta\tau_0$, sostituendo poi $\Delta\tau_0 = 0$ e $\|r_0\|_2 = 0$;

$$\boldsymbol{\tau_k} = y_{u,0k}$$

$$\boldsymbol{\Delta\tau_m} = \frac{1}{K}\sum_{k=1}^{K} y_{u,mk} - \frac{1}{K}\sum_{k=1}^{K} \tau_k - \frac{1}{c}\|r_m\|_2$$

$$4\frac{1}{K}\sum_{k=1}^{K}\tau_k - \frac{1}{K}\sum_{k=1}^{K} y_{u,0k} + \frac{1}{c}\|r_1\|_2 + \Delta\tau_1 - \frac{1}{K}\sum_{k=1}^{K} y_{u,1k} + \frac{1}{c}\|r_2\|_2 + \Delta\tau_2 - \frac{1}{K}\sum_{k=1}^{K} y_{u,2k} + \frac{1}{c}\|r_3\|_2 + \Delta\tau_3 - \frac{1}{K}\sum_{k=1}^{K} y_{u,3k} = 0$$

$$\sum_{k=1}^{K}\frac{1}{c}\left(\tau_k + \frac{1}{c}\|r_m\|_2 + \Delta\tau_m - y_{u,mk}\right)\frac{x_m}{\|r_m\|_2} = 0, \qquad \sum_{k=1}^{K}\frac{1}{c}\left(\tau_k + \frac{1}{c}\|r_m\|_2 + \Delta\tau_m - y_{u,mk}\right)\frac{y_m}{\|r_m\|_2} = 0,$$

$$\sum_{k=1}^{K}\frac{1}{c}\left(\tau_k + \frac{1}{c}\|r_m\|_2 + \Delta\tau_m - y_{u,mk}\right)\frac{z_m}{\|r_m\|_2} = 0$$

Stima della posizione del Target

tratto dalla tesi ”Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS” di Jeroen Hol

- Si raccoglie un data set di misure di TOA $D_2 = \{y_{u,mk}\}$ con il target in movimento.
- Utilizzando i valori dei parametri individuati in fase di calibrazione, si applica la multilaterazione su D_2 per determinare $\{t_0, \tau_0\}$

$$\min_{\tau,t} \sum_{m=1}^{M} \frac{1}{2} \left\| y_{u,m} - \tau - \|r_m - t\|_2 - \Delta\tau_m \right\|^2 \Sigma_u^{-1}$$

- Partendo quindi dall’insieme di valori per i parametri stimati θ_0:

$$\theta_0 = (\{t_0, \tau_0\}, \{r_{m,0}, \Delta T_{m,0}\})$$

può essere effettuata l’ottimizzazione:

$$\hat{\theta}_{ML} = \arg\min_{\theta} \frac{1}{2} \sum_{k=1}^{K} \sum_{m=0}^{3} \varepsilon_{u,mk}^2(\theta)$$

Algoritmo

1. Costruire setup stazionario di M ricevitori.
2. Posizionare il target in prossimità delle ancore e raccogliere il data set di TOA $D1 = \{y_{u,mk}\}$, con cui effettuare la calibrazione.
3. Individuare i parametri nella fase di calibrazione, inizializzando il problema con delle posizioni rumorose fornite dall'utente.
4. Raccogliere un data set di TOA $D2 = \{y_{u,mk}\}$ con il target in movimento.
5. Applicare la multilaterazione sulla base dei valori dei parametri stimati in fase di calibrazione per determinare $\{t_0, \tau_0\}$.
6. Inizializzare l'ottimizzazione con i valori $\theta_0 = (\{t_0, \tau_0\}, \{r_{m,cal}, \Delta T_{m,cal}\})$ per la determinazione di $\hat{\theta}_{ML} = \arg\min_{\theta} \frac{1}{2} \sum_{k=1}^{K} \sum_{m=0}^{3} \varepsilon_{u,mk}^2 (\theta)$.

Calibrazione – Stima dei parametri

riscrittura del problema per applicarlo al nostro caso

$$\hat{\theta}_{ML} = \arg\min_{\theta} \frac{1}{2} \sum_{k=1}^{K} \sum_{m=0}^{3} \varepsilon_{u,mk}^2(\theta) \qquad \varepsilon_{u,mk}(\theta) = \sigma_u^{-1}\left(\|r_m - t\|_2 + \Delta\tau_m + \Delta\tau_{target} - R_{mis,mt}\right)$$

$$R_{mis,mt} = \|r_m - t\|_2 + \Delta\tau_m + \Delta\tau_{target} + \delta_{u,m} + e_{u,m}$$

Dove:

- r_m è la posizione dell'ancora m-sima (Rx)
- t è la posizione del target (Tx)
- $\Delta\tau_m$ è il *delay* dell'ancora m-sima
- $\Delta\tau_{target}$ è il *delay* del target
- $R_{u,m}$ è il range rispetto al target, misurato dall'ancora m-esima.
- $\delta_{u,m}$ è il contributo di ritardo dovuto a cause quali NLOS, multipath..
- $e_{u,m}$ è il contributo di rumore di misura gaussiano

Calibrazione

procedura di acquisizione dei range relativi grezzi e simulazione in `SimulazioneCal.m`

- Si posiziona il target vicino a ciascuna delle ancore, a rotazione.

- Si avrà che la distanza euclidea tra il target e l'ancora vicina, in quella fase di calibrazione, è nulla. Da qui si può stimare il contributo $\Delta\tau_m + \Delta\tau_{target}$.

- In ogni fase, per una generica ancora, si ha che la minimizzazione del funzionale di costo porta all'equazione: $\|r_m - t\|_2 + \Delta\tau_m + \Delta\tau_{target} - R_{mis,mt} = 0$

- Dopo aver raccolto i dati relativi a tutte e quattro le configurazioni del target, è possibile ricavare tutti i termini $\Delta\tau_m + \Delta\tau_{target}$, con cui effettuare la compensazione dei range misurati.

Algoritmo ML con rumore e clock offset Risultati

[1/4]

Questi risultati fanno riferimento allo script Matlab `SimulazioneCal.m`

Assumendo la presenza di un errore di misura gaussiano di ampiezza 0.02 m e valori [m] degli errori introdotti nel range, dovuti agli offset di clock di ogni ancora:

$$\Delta\tau' = [0.06, 0.08, 0.07, 0.055]$$

Questi si vanno a sommare all'errore sul range a causa del delay del target di 0.02 m. Dunque il vettore degli offset complessivi sui range è:

$$\Delta\tau = [0.06 + 0.02, 0.08 + 0.02, 0.07 + 0.02, 0.055 + 0.02]$$

Ad ogni fase della calibrazione si riesce ad individuare l'offset sull'ancora in prossimità del tag, grazie alle precedenti assunzioni.

Algoritmo ML con rumore e clock offset Risultati

[2/4]

Questi risultati fanno riferimento allo script Matlab `SimulazioneCal.m`

```
%% CALIBRAZIONE
% fase 1, ponendo il target vicino all'ancora 0. Determino DT_0t
% in quanto il valore misurato dall'ancora 0 è dato solo dai disturbi, quelli delle
% altre ancore sono dati dalle distanze relative a quell'ancora, perturbate
% dai disturbi agenti su di esse.
D1 = [DT(1)+n(:,1), R_mis(:,4), R_mis(:,7), R_mis(:,10)];

% Indicando DT_mt = DT_m + DT_t
DT_0t = mean(D1(:,1));

R_mis = R_rd + n_rd21 + ones(K,1)*DT_rd;

% fase 2, target vicino all'ancora 1. Trovo DT_1t
D2 = [R_mis(:,1), DT(2)+n1(:,2), R_mis(:,8), R_mis(:,11)];

DT_1t = mean(D2(:,2));

R_mis = R_rd + n_rd22 + ones(K,1)*DT_rd;
% fase 3, target vicino all'ancora 2.
D3 = [R_mis(:,2), R_mis(:,5), DT(3)+n2(:,3), R_mis(:,12)];
DT_2t = mean(D3(:,3));

R_mis = R_rd + n_rd23 + ones(K,1)*DT_rd;
% fase 4, target vicino all'ancora 3.
D4 = [R_mis(:,3), R_mis(:,6), R_mis(:,9), DT(4)+n3(:,4)];
DT_3t = mean(D4(:,4));

DT_est = [DT_0t, DT_1t, DT_2t, DT_3t];
```

Command Window

```
DT =

    0.0800    0.1000    0.0900    0.0750


DT_est =

    0.0789    0.0985    0.0928    0.0774

P_real =

         0         0         0
         0   12.0000         0
    4.7220   -9.7210         0
    3.1430   -4.6420    2.2240


Pr_CMD =

         0         0         0
         0   12.0076    0.0000
    4.7471   -9.7081   -0.0000
    3.1540   -4.6325    2.2229


P_alg =

         0         0         0
         0   12.0076         0
    4.7471   -9.7081         0
    3.1540   -4.6325    2.2229
```

Algoritmo ML con rumore e clock offset Risultati

[3/4]

Questi risultati fanno riferimento allo script Matlab `SimulazioneCal.m`

La minimizzazione vincolata del funzionale di costo è stata anche condotta utilizzando il comando Matlab `fmincon(myfun,A,b,Aeq,beq)`.

In `fun` è stato inserito il risultato di:

```
J1= 0;
for r = 1 : 4
    for m = 1 : 4
        if m ~= r
            J1 = J1 + 1/(2*(S(m))^2)*[(x(m) + sqrt((x(4 + 1 + 3*(m-1)) - x(4 + 1 + 3*(r-1)))^2 + (x(4 + 1 + 3*(m-1) + 1)
            - (x(4 + 1 + 3*(r-1) + 1)))^2 + (x(4 + 1 + 3*(m-1) + 2) - x(4 + 1 + 3*(r-1) + 2))^2) - R_m(m,r))^2];
        end
    end
end
```

I vincoli immessi sono lineari: le matrici `A,b` descrivono le diseguaglianze, `Aeq,beq` le equazioni.

Avendo assunto come vettore: $x = [\Delta t\ \Delta\tau_0\ \Delta\tau_1\ \Delta\tau_2\ \Delta\tau_3\ x_1\ y_1\ z_1\ x_1\ y_1\ z_1\ \ x_2\ \ y_2\ z_2\ \ x_3\ y_3\ z_3]$

```
A = [];

b = [];

Aeq = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
       0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0;
       0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0;
       0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0;
       0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
       0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
       0 0 0 0 0 0 0 0 0 0 0 1 0 0 0];

beq = zeros(7,1);

x0 = [DT_est, 0, 0, 0, 0, R_new(1), 0, a2_real(1)/abs(a2_real(1))*R_new(2), R_new(2), 0, R_new(3), R_new(3), ...
    a3_real(3)/abs(a3_real(3))*R_new(3)];

x = fmincon(myfun, x0, A, b, Aeq, beq)
```

Algoritmo ML con rumore e clock offset Risultati

[4/4]

Questi risultati fanno riferimento allo script Matlab `SimulazioneCal.m`

```
Command Window

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

P_real =

         0         0         0
         0   12.0000         0
    4.7220   -9.7210         0
    3.1430   -4.6420    2.2240


P_est =

   -0.0000    0.0000   -0.0000
   -0.0000   12.0830   -0.0000
    5.0472   -9.6500   -0.0000
    3.2626   -4.6033   -2.3550
```

Questi risultati sono stati ottenuti avendo fornito come condizioni iniziali x_0:

- i contributi di errore di range dovuti ai delay delle antenne incluso il termine di delay del Tag Δt.
- I range relativi compensati dell'errore stimato sui range.