

CONSENSUS ON COMPACT RIEMANNIAN MANIFOLDS

ITALO CERVIGNI MATTIA IPPOLITI CLAUDIO MENOTTA MICHELE PIERMARINO

Sommario. Partendo dal documento "Consensus on compact Riemannian manifolds" [1] abbiamo studiato il problema di come garantire che le informazioni degli individui in rete ottengano consenso.

Key words. Compact Riemannian Manifold; Spazio tangente; SO(3); Mappa Logaritmica; Mappa Esponenziale; Compressione/Decompressione; IVP; Metodo di Eulero; Metodo di Heun; Metodo di Runge (RK2)

1. Introduzione. Dato $\mathbb{G}^n = G_{(i \in \tau)}$, l'insieme dei grafi orientati ponderati $G_i = (V_i, A_i, W_i)$, dove:

• $V_i = \bar{n} = 1, \dots, n$: è l'insieme dei vertici, che assumiamo essere i nostri corpi rigidi, considerati puntiformi, ovvero coincidenti con il proprio centro di massa.

• $A_i = (j, k) : j, k \in V_i$: l'insieme degli archi che collegano tra di loro i nostri corpi rigidi. Questi archi orientati possono essere considerati come dei canali di comunicazione tra i singoli corpi, tramite cui essi possono acquisire informazioni sulla configurazione reciproca. In particolare, sono archi orientati, ovvero canali unidirezionali di informazione. Inoltre si può avere che tra due vertici ci sia solo un canale di comunicazione in un certo verso, che permette quindi solo ad uno dei due di acquisire informazioni sulla configurazione dell'altro, e non viceversa. Se, invece, tra due vertici non vi è alcun arco, ciò indica che non c'è alcun canale di comunicazione diretto tra i due (ma essendo una rete di archi, ci saranno altri percorsi indiretti che collegano i due, pur non essendo connessi direttamente).

• $W_i = w_{jk} > 0 : (j, k) \in A_i$: l'insieme dei pesi. Quello che stiamo considerando è un insieme di grafi orientati ponderati, ovvero "pesati", in cui ogni arco avrà una valenza maggiore o minore. Questa è data da dei pesi, ovvero valori maggiori o uguali a zero (non negativi, perché altrimenti non avrebbero alcun senso fisico), che indicano la capacità di comunicazione reciproca della coppia di vertici nel verso descritto dall'arco. Il caso in cui il peso corrispondente abbia valore nullo corrisponde all'assenza di un canale comunicativo diretto tra i due vertici attraverso quella particolare direzione. Il loro valore, un caso reale come quello di un insieme di droni in movimento che considereremo noi, può dipendere da vari fattori, come la posizione reciproca dei droni, il mezzo fisico in cui si trovano, il danneggiamento dei singoli droni, ecc.

Chiamiamo una qualsiasi funzione continua $\sigma : [0, \infty] \rightarrow \mathfrak{I}$, funzione di scambio. Data una funzione di scambio $\sigma(t)$, chiamiamo $G_{\sigma(t)}$ grafico topologico di comunicazione. Inoltre definiamo $N_i(G_{\sigma(t)})$ l'insieme dei vertici vicini all' i -esimo vertice in $G_{\sigma(t)}$ all'istante t . Questo insieme, corrisponde a tutti i vertici con cui il vertice i -esimo può comunicare direttamente accedendo al loro stato. Dato M un Manifold Riemanniano compatto, consideriamo $q_1(t), \dots, q_n(t)$ un insieme di punti dinamici appartenenti a M , con t appartenente a $[0, \infty)$ e n maggiore o uguale a 2. Siano $q_1(0) = q_1^0, \dots, q_n(0)$ le rispettive posizioni iniziali dei singoli punti, prese casualmente da un certo sottoinsieme non vuoto S di M . Siano questi punti dinamici interconnessi da una rete di collegamenti descritta da un grafico topologico di comunicazione $G_{\sigma(t)}$.

$I_i(t) = (q_j, w_{ij}) : j \in N_i(G_{\sigma(t)})$ è l'informazione accessibile del punto $q_i(t)$ dalla rete di connessioni all'istante t . Come notiamo l'informazione a cui il singolo vertice accede corrisponde allo stato dei suoi vicini, in modo da modificare il proprio di conseguenza. Tuttavia tale informazione è acquisita in base ai limiti dei canali trasmissivi, rappresentati dai pesi. Il problema su cui ci

concentreremo in questo progetto è come realizzare una legge che governi questi punti dinamici affinché essi raggiungano il consenso, cioè esista un punto $c \in \mathbb{M}$ tale che tutti i $q_i(t) = c$ per $t \rightarrow \infty$. Supponiamo che questi punti dinamici raggiungano il consenso. L'algoritmo che governa il loro andamento viene detto di consenso globale per \mathbb{M} se $S = M$. In particolare, se il comportamento dei punti dinamici è descritto dal sistema dinamico:

$$\dot{q}_i = u_i(q_i, I_{i(t)}) \in T_{q_i} M, i \in \bar{n} \quad (1.1)$$

dove T_{q_i} è il vettore spazio tangente in q_i e ogni traiettoria $q_i(t) : t \in [0, \infty] \subset S$. Chiameremo tale l'algoritmo di consenso locale per S . Dato c il punto comune a cui tendono i punti dinamici nel raggiungimento del consenso, se esistono due numeri $\lambda, \gamma > 0$ tale che:

$$\sum_{i \in \bar{n}} d^2(c, q_i(t)) \leq \theta e^{-\lambda t}, \forall t \in [0, \infty]$$

diremo che tutti i punti dinamici raggiungono un consenso quasi-esponenzialmente. In particolare, se $\theta = \sum_{i \in \bar{n}} d^2(c, q_i(0))$, diremo che raggiungono il consenso esponenzialmente.

Per raggiungere un consenso globale, ovvero per ogni punto appartenente al Manifold \mathbb{M} (nel nostro caso $SO(3)$), è necessario per prima cosa introdurre il consenso locale, ovvero all'interno di un sottoinsieme di \mathbb{M} .

1.1. Corollario. Sia $(\mathbb{R}^M, |||)$ lo spazio normalizzato, e $\hat{B}_r = z \in \mathbb{R}^M : |z| < r$ lo spazio non vuoto. Consideriamo allora il seguente sistema dinamico:

$$\dot{z}_i = \sum_{j \in N_i(G_{\sigma(t)})} w_{ij}(z_j - z_i), i \in \hat{n}$$

dove ciascun $z_i(t) \in \hat{B}_r$, e $G_{\sigma(t)} \in \mathbb{G}^n$ è il grafo topologico di comunicazione. Se $g_{\sigma(t)}$ è debolmente connesso e bilanciato, allora tutti i punti $z_i(t)$ raggiungono il consenso globale esponenzialmente su \hat{B}_r .

1.2. Teorema 1. Consideriamo il sistema dinamico sul Manifold Reimanniano \mathbb{M} come segue:

$$\dot{p}_i = \sum_{j \in N_i(G_{\sigma(t)})} w_{ij} \exp_{p_i}^{-1}(p_j), i \in \bar{n} \quad (1.2)$$

dove ciascun $p_i(0) \in Br$. Se $G_{\sigma(t)}$ ha bassa connessione e basso bilanciamento, allora tutti i punti dinamici $p_i(t)$ raggiungono il consenso esponenzialmente sulla calotta Br .

Per passare dal consenso locale in Br , al consenso globale è necessario introdurre una funzione ausiliaria che ci permettano di comprimere gli stati iniziali dei punti dinamici all'interno di Br . In tal modo sarà possibile applicare il protocollo del consenso locale in Br , per poi tornare al manifold \mathbb{M} attraverso un'ulteriore funzione ausiliaria di decompressione.

1.3. Lemma 1. Per i Manifold Riemaniani \mathbb{M} , esiste una mappa iniettiva $\phi : \mathbb{M} \rightarrow Br$ e una mappa suriettiva $\psi : Br \rightarrow \mathbb{M}$ in modo che: $\psi(\phi(q)) = q$ per ogni $q \in \mathbb{M}$.

Questa funzione ϕ può mappare tutti i punti di M nella palla Br , mentre la funzione ψ può mappare

tutti i punti nella palla, al di fuori di essa, in \mathbb{M} . Di conseguenza corrispondono a una funzione di compressione:

$$\phi : \mathbb{M} \rightarrow Br$$

e decompressione di dati:

$$\psi : Br \rightarrow \mathbb{M}$$

La funzione di compressione non è necessariamente suriettiva, cioè la sua controimmagine non è necessariamente tutta la palla Br , così come dall'altro lato la funzione di decompressione non è sempre iniettiva. Tuttavia si può dimostrare che ϕ mappa punti distinti in \mathbb{M} in altrettanti punti distinti in Br e, viceversa, lo stesso vale per la funzione di decompressione. Per questo tali funzioni possono essere utilizzate per il nostro scopo, cioè realizzare un algoritmo di consenso.

1.4. Teorema 2. Siano q_1^0, \dots, q_n^0 n punti presi randomicamente dal Manifold \mathbb{M} , e sia $\phi : M \rightarrow Br$ la funzione di compressione mentre ψ la funzione di decompressione. Consideriamo allora i punti decompressi $q_1(t) = \psi(p_1(t)), \dots, q_n(t) = \psi(p_n(t))$ tutti i punti dinamici $p_i(t)$ che soddisfano:

$$\dot{p}_i = \sum_{j \in N_i(G_{\sigma(t)})} w_{ij} \exp_{p_i}^{-1}(p_j), i \in \bar{n} \quad (1.3)$$

Il raggiungimento del consenso dei punti $p_i(t)$, cioè compressi, è garantito dal Teorema 1 ovviamente. Per quanto riguarda invece il raggiungimento del consenso da parte dei punti $q_i(t)$, decompressi, questo si può dimostrare con il seguente risultato:

1.5. Lemma 2. Date le funzioni di compressione e decompressione ϕ e ψ , esiste una costante $C > 0$, tale che $d(\psi(p), \psi(q)) \leq Cd(p, q), \forall p, q \in Br$.

Questo è un risultato fondamentale, infatti, essendo il consenso dei punti compressi p e q esponenziale, la funzione $d(p, q)$ nel tempo ha un andamento di tipo esponenziale e decrescente, tendendo di conseguenza a zero asintoticamente per $t \rightarrow \infty$. Ovviamente se $d(p, q)$ tende a zero anche $Cd(p, q)$ lo farà. Allora considerata la funzione $d(\psi(p), \psi(q))$ della distanza reciproca dei punti decompressi nel tempo, se questa è limitata superiormente da $Cd(p, q)$, allora anch'essa tenderà a zero e lo farà sempre in modo esponenziale. Questo passo ci permette allora di passare alla parte implementativa, per verificare i risultati teorici.

2. IVP. Un problema ai valori iniziali (IVP) è composto da un'equazione differenziale ordinaria al quale viene associato un valore della funzione incognita in un certo punto del dominio della soluzione, chiamato condizione iniziale. Nel seguente progetto ci concentreremo su equazioni differenziali del 1 ordine. Ciò che noi troveremo sarà una soluzione approssimata e non la soluzione analitica, dato che quest'ultima non sempre è possibile trovarla. Al contrario mediante metodi numerici è sempre possibile calcolare la soluzione. Per trovare la soluzione analitica ci concentreremo su 3 particolari metodi numerici:

- 1) Metodi di Eulero
- 2) Metodi di Heun
- 3) Metodo di RK2

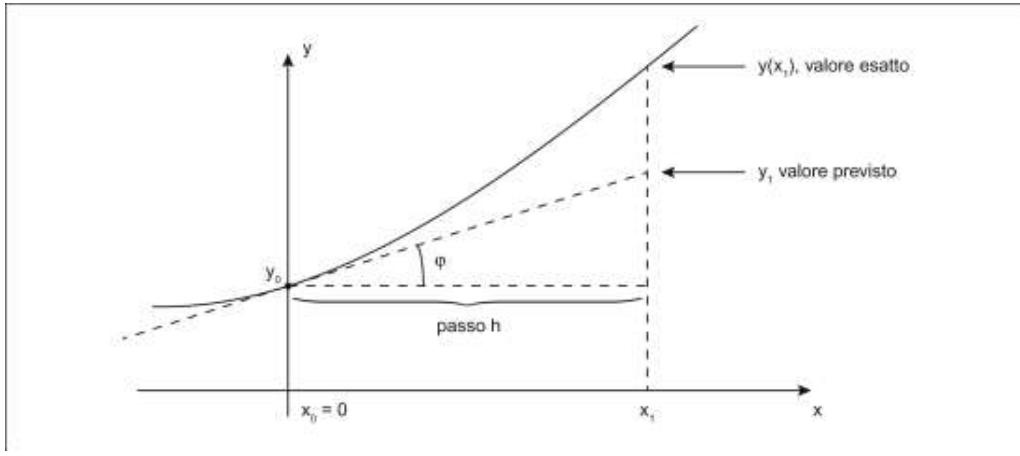


FIGURA 2.1. Esempio del metodo di Eulero

2.1. Metodo di Eulero. Considero $y \in A_n$. Noi vogliamo calcolare la soluzione $y(t)$ in un intervallo $t \in [t_0, t_f]$. Consideriamo inoltre un valore $h < 0$ il più piccolo possibile per ottenere una partizione uniforme dell'intervallo $[t_0, t_f]$.

$$\begin{aligned} t_1 &= t_0 + h \\ t_2 &= t_1 + h \\ t_3 &= t_2 + h \\ \dots \\ t_n + 1 &= t_n + h \\ \dots \\ t_N &= t_f \end{aligned}$$

In particolare h è definito passo di discretizzazione dell'intervallo continuo $[t_0, t_f]$ (è sempre una quantità positiva), e, come si intuisce più sarà piccolo quest'ultimo e minore sarà la differenza tra soluzione reale e soluzione approssimata con il metodo, cioè minore sarà l'errore e quindi maggiore sarà la precisione del nostro risultato. Ovviamente un valore di h piccolo comprometterà un numero di calcoli maggiori, visto che dovremo ripetere i vari procedimenti molte più volte. Data l' IVP:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases}$$

Come illustrato nel grafico precedente, il punto di partenza della soluzione da approssimare è y_0 , ovvero $y(t_0)$. Da qui ci si muove orizzontalmente nel verso dell'asse dei tempi di un valore pari al passo h , cioè fino al secondo temine della partizione temporale (t_1). Invece verticalmente ci si muove di un valore pari a

$$hf(t_0, y(t_0))$$

(cioè $hf(t_0, y_0)$), in modo da formare un piccolo triangolo rettangolo che abbia pendenza dell'ipotenusa pari a $f(t_0, y_0)$. Il risultato, di fatto, è quello di muoversi lungo la direzione tangente alla soluzione nel punto y_0 , fino ad arrivare all'istante temporale t_1 . A questo punto si procede analogamente muovendosi orizzontalmente sempre dello stesso passo h (il campionamento è costante)

fino ad arrivare t_2 , mentre verticalmente ci si sposta di un valore:

$$hf(t_1, y_1)$$

Il risultato in analogia con il passo precedente sarà quello di muoversi nella direzione tangente alla soluzione da approssimare, fino ad arrivare all'istante temporale t_3 e così via. Continuando con questo procedimento per tutto l'intervallo campionato non si fa altro che approssimare la soluzione reale in una spezzata fatta di tanti piccoli segmenti tangentati alla soluzione. Tanto più piccolo sarà il passo h , allora, e tanto più piccoli saranno questi segmenti che approssimeranno, allora, in modo migliore la nostra soluzione reale. Si può dire che per $h \rightarrow 0$ la soluzione approssimata e quella analitica (che non sempre si può ricavare con le regole di calcolo differenziale) saranno coincidenti. Tuttavia, non potendo scegliere un h infinitamente piccolo, cercheremo di trovare un valore accettabilmente piccolo di h al di sotto di cui l'errore rispetto alla soluzione analitica sia accettabile. Dal punto di vista matematico il procedimento eseguito è stato il seguente:

$$\frac{dy}{dx} = f(t, y(t)) \quad (2.1)$$

e usando il rapporto incrementale avremo:

$$\frac{y_{n+1} - y_n}{y_{n+1} - y(t_n)} = f(t_n, y_n)$$

e portando $y_{(n+1)}$ a primo termine avremo:

$$y_{n+1} = y_n + hf(t_n, y_n) \quad (2.2)$$

dove (2.2) verrà chiamato: Metodo di Eulero in avanti. Il metodo di Eulero in avanti si dice esplicito, ovvero è una relazione in cui si hanno al secondo membro tutte grandezze che conosco, mentre al primo membro si ha la quantità da determinare. Per quanto riguarda il passo h esso, come già detto, deve avere ordine di grandezza molto inferiore rispetto a quello dei tempi e, volendo dividere l'intervallo $[t_0, t_f]$ in N parti uguali, h si può calcolare nel seguente modo:

$$h = \frac{t_f - t_0}{N}$$

2.2. Metodo di Heun. Il metodo di Heun è un metodo numerico per la ricerca della soluzione approssimata di una equazione differenziale ordinaria del tipo:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases}$$

dove la funzione $y = y(x)$ è definita in un intervallo chiuso e limitato $[a, b] \subset R$. Tale metodo può essere considerato come una variante del metodo di Runge-Kutta. La procedura del calcolo della soluzione avviene in due fasi distinte. Partiamo anzitutto l'espressione in Clark-Nicholson:

$$y_{n+1} = t_n + h \frac{f(t_n, y_n) + f(t_{n+1}, y_{n+1})}{2}$$

(questo è metodo implicito richiede infatti la conoscenza della funzione nel punto di arrivo che è ancora incognita, facendoci spostare un passo alla volta della media tra le 2 pendenze). In questo

caso al posto di y_{n+1} andremo ad utilizzare l'approssimazione: \tilde{y}_{n+1} , rendendo di fatto il metodo esplicito. Perciò avremo che:

$$y_{n+1} = y_n + \frac{f(t_n, y_n)h}{2} + \frac{f(t_{n+1}, y_n + h f(t_n, y_n))h}{2} \quad (2.3)$$

Chiamato k_1 il riferimento al punto temporale n:

$$k_1 = f(t_n, y_n)$$

e chiamato k_2 il riferimento al punto temporale $n + 1$:

$$k_2 = f(t_{n+1}, y_n + hk_1)$$

avremo che:

$$y_{n+1} = y_n + \frac{(k_1 + k_2)h}{2} \quad (2.4)$$

dove la formula (2.4) è il metodo di Heun, che appartiene appunto alla classe dei metodi del secondo ordine.

2.3. Metodo di Runge. Supponiamo di effettuare un passo con il metodo di Eulero esplicito fino a metà dell'intervallo $t_{n+1} = t_n + h/2$ per calcolare il valore della funzione $y_{n+\frac{1}{2}}$. Effettuiamo poi un passo completo in cui la derivata è calcolata in t_{n+1} (cioè a metà dell'intervallo). Si ottiene così il metodo di Runge:

$$y_{n+\frac{1}{2}} = y_n + \frac{hf(t - n, y_n)}{2}$$

da cui sostituendo:

$$y_{n+1} = y_n + hf(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$$

$$y_{n+1} = y_n + hf(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)) \quad (2.5)$$

Allora chiamati: $k_1 = f(t_n, y_n)$ e $k_2 = f(t_n + bh, y_n + ck_1)$ avremo che la formula di Runge sarà:

$$y_{n+1} = y_n + h(a_1 k_1 + a_2 k_2)$$

ponendo inoltre $a_1 = 0, a_1, b = c = \frac{1}{2}$, avremo che:

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

e riscriveremo la formula di Runge come:

$$y_{n+1} = y_n + hk_2 \quad (2.6)$$

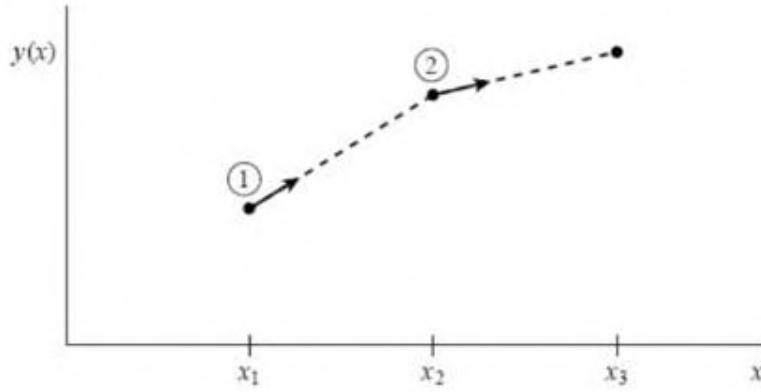


FIGURA 2.2. Nel metodo di Eulero la derivata nel punto iniziale di ogni intervallo è estrapolata per trovare il valore successivo della funzione. Questo metodo ha un'accuratezza del primo ordine

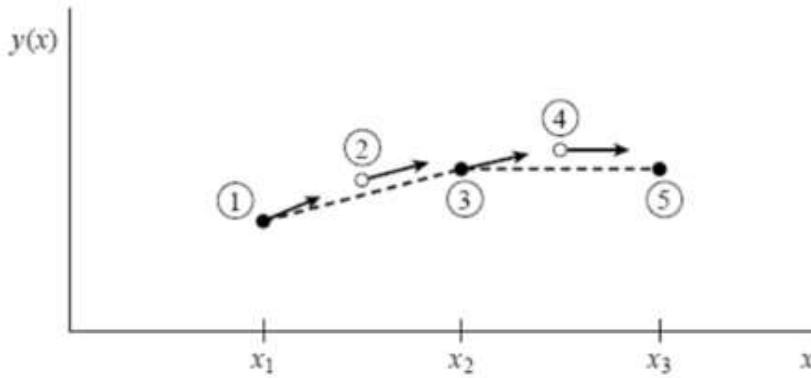


FIGURA 2.3. Nel metodo di Runge l'accuratezza del secondo ordine viene ottenuta utilizzando il valore della derivata nel punto iniziale per determinare il valore della funzione nel punto intermedio dell'intervallo e poi utilizzando questo valore per determinare la derivata da utilizzare per determinare il valore nel punto finale. Nella figura i cerchi pieni rappresentano i valori finali della funzione mentre i cerchi aperti rappresentano il valore della funzione che servono per calcolare la derivata da utilizzare nel singolo passo. Questi valori vengono scartati alla fine del passo.

3. Problema. Consideriamo una rete di oggetti in movimento. Ci concentreremo sul problema di trovare una legge di controllo che faccia raggiungere ai nostri droni un consenso nell'assetto di rotazione. Perciò lo stato che caratterizzerà i nostri droni sarà un punto sul manifold $\text{SO}(3)$, ovvero una matrice ortogonale speciale che rappresenterà la matrice di rotazione del nostro drone. Lavoreremo quindi in $\text{SO}(3)$, delle cui principali proprietà geometriche parleremo ora.

3.1. Algebra di Lie. In matematica, un'algebra di Lie è una struttura algebrica usata principalmente per lo studio di oggetti geometrico analitici come i gruppi di Lie e le varietà differenziabili.

Un'algebra di Lie è una struttura costituita da uno spazio vettoriale g su un centro campo \mathbb{F}

(per esempio i numeri reali, i numeri complessi, o un campo finito) e da un operatore binario $[\cdot, \cdot] : \mathfrak{g} \times \mathfrak{g} \rightarrow \mathfrak{g}$, detto prodotto di Lie, che soddisfa le seguenti proprietà:

1. è bilineare, cioè

$$[\alpha x + \beta y, z] = \alpha[x, z] + \beta[y, z]$$

e

$$[z, \alpha x + \beta y] = \alpha[z, x] + \beta[y, z]$$

per ogni x, y e $z \in \mathbb{F}$.

2. soddisfa l'identità di Jacobi, cioè

$$[[x, y]z] + [[z, x], y] + [[y, z], x] = 0$$

per ogni x, y e $z \in \mathfrak{g}$.

3. è nipotente, cioè $[x, x] = 0$ per ogni $x \in \mathfrak{g}$.

Notare che la prima e la terza proprietà insieme implicano $[x, y] = -[y, x]$ per ogni $x, y \in \mathfrak{g}$.

In particolare un gruppo di Lie è un gruppo G munito di una struttura di varietà differenziabile tale che le operazioni:

1. $G \times G \rightarrow G$

$$(a, b) \mapsto ab$$

2. $G \rightarrow G$

$$a \mapsto a^{-1}$$

sono entrambe differenziabili.

3.2. SO(3). Lo spazio delle matrici reali

$$A \in \mathbb{R}^{3 \times 3} : A^\top A = I, \det(A) = 1$$

rappresenta $\text{SO}(3)$, dove $I \in \mathbb{R}^{3 \times 3}$ è la matrice identità. Inoltre la sua algebra di Lie $\mathfrak{so}(3)$ è l'insieme delle matrici reali antisimmetriche 3×3 (vedi (3.1)). Se dotiamo lo spazio con la matrice indotta dal prodotto di Frobenius $\langle \cdot, \cdot \rangle_F$, diventa un Manifold Riemaniano.

3.3. Mappa Esponenziale. Come per ogni Manifold scriviamo la mappa Esponenziale $\exp(A)$ della matrice reale A come limite della serie convergente:

$$\exp(A) = \sum_{i=1}^n x_i \frac{A^k}{k!}$$

allora $\exp(A) \in \text{SO}(3)$ se $A \in \text{SO}(3)$.

Si definisce mappa Esponenziale il valore che la geodetica prende in $t=1$. Allora $\exp_p(v)$ è la quantità: $\sigma_{p,v}(1)$

$$\begin{cases} \exp_p(v) := \sigma_{p,v}(1) \\ \sigma_{p,v} : [0, 1] \rightarrow \mathbb{M} \end{cases}$$

Perciò la mappa esponenziale mi trasformerà un oggetto nello spazio tangente in uno sul Manifold: $\exp : T\mathbb{M} \rightarrow \mathbb{M}$.

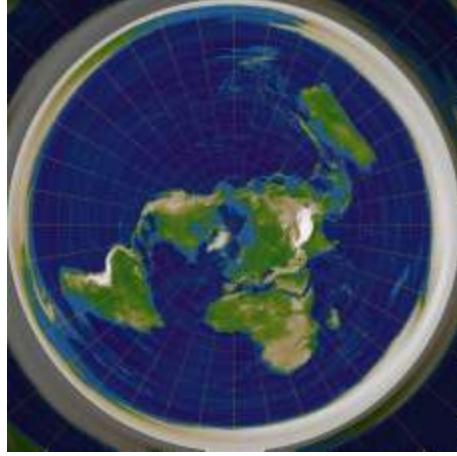


FIGURA 3.1. La mappa Esponenziale della Terra vista dal polo Nord è la proiezione equidistante azimutale polare in cartografia.

3.4. Mappa Logaritmica. Avendo definito la mappa esponenziale $\exp(A)$ andremo ora a definire la mappa logaritmica, cioè la sua inversa: $\mathbb{M}^2 \rightarrow T\mathbb{M}$. Data l'equazione di matrici reali $\exp(X)=A$, ogni soluzione reale $\log^+(A)$ è chiamata logaritmo reale di A. Anche in questo caso $\log^+(A) \in \text{SO}(3)$ se $A \in \text{SO}(3)$. Se la mappa logaritmica $\log(\cdot)$ è sostituita dalla mappa logaritmica speciale $\log^*(\cdot)$, definiamo $\log^*(A) = \log^+(A)$ per ogni matrice A reale, dove la matrice $\log^+(A)$ ha la minima norma di Frobenius tra tutti i logaritmi reali di A. Inoltre notiamo come $\log^*(A) = \log(A)$ dove A ed I non sono antipodali tra loro.

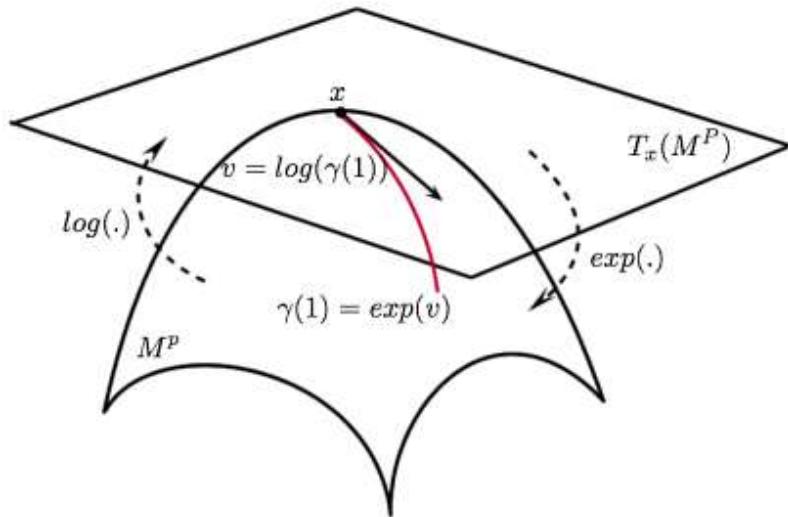


FIGURA 3.2. Mappe Logaritmiche ed Esponenziali

3.5. Compressione e Decompressione. Definiamo la funzione di compressione $\phi : SO(3) \rightarrow S$ come:

$$\Phi(p) = \text{expm}(l_{max}^{-1} \logm^*(p)) \quad (3.1)$$

Mentre definiamo la funzione di compressione $\Psi : S \rightarrow SO(3)$ come:

$$\Psi(p) = \text{expm}(l_{max} \logm^*(p)) \quad (3.2)$$

dove nel nostro caso tratteremo $l_{max} = 5$.

3.6. Proprietà geometriche di SO(3). Le proprietà sono le seguenti: [3]

Mappa Esponenziale	$\exp_p(x) = p \expm(p^\tau x)$
Mappa Logaritmica	$\logm_p(q) = p \logm(p^\tau q)$
Distanza	$d(p, q) = \ \logm(p^\tau q)\ _F$

4. Simulazione. Assumiamo che ci sono $n=4$ corpi rigidi D_1, D_2, D_3, D_4 situati all'inizio in $q_1^0, q_2^0, q_3^0, q_4^0 \in SO(3)$. Usiamo i risultati ottenuti dalla teoria per portare questi droni al consenso. Per risolvere il problema dell'attitudine al consenso dovremo risolvere l'IVP:

$$\dot{p}_i = \sum_{j \in N_i(G_{\sigma(t)})} w_{ij} \exp_{p_i}^{-1}(p_j) \quad (4.1)$$

Prima di procedere dobbiamo definire le connessioni tra i vari corpi, cioè il grafico topologico di comunicazione. Questo non sarà altro che una matrice W , $4 \times 4 (n \times n)$, con valori ≥ 0 , in cui il generico valore $W(i, j)$ corrisponde al peso del canale di comunicazione che collega D_i a D_j , attraverso il quale D_i riesce ad accedere allo stato di D_j , per modificare il proprio di conseguenza. Si viene così a creare una rete di collegamenti tra i vari droni che sarà diversa in base alla struttura della matrice W . Per avere una più ampia variazione delle comunicazioni tra i singoli droni sceglieremo W a random tra 3 diverse matrici ad esempio. Ciò come vedremo sperimentalmente garantirà il raggiungimento del consenso in modo più rapido, perché permette ai droni di comunicare in modo diverso istante per istante. Attraverso le funzioni di compressione (3.5) ogni q_i^0 può essere compresso all'interno dell'insieme W , dove i stati iniziali dei dati compressi saranno $p_i(0) = \Phi(q_i^0)$. Tramite questi continueremo a calcolare i $p_i(t)$ fino all'istante $t = t_f$. Tuttavia dobbiamo decomprimere tutti i dati $p_i(t)$, attraverso la funzione Ψ utilizzando la formula (3.5), ottenendo il consenso in $q_i(5)$ a $t = t_f$. Ovviamente più sceglieremo un valore di t_f grande in cui valutare $p_i(t)$, più le matrici compresse avranno raggiunto il consenso locale su Br . A questo punto, decomprimendo i valori $p_i(t_0)$ ottenuti, avremo dei valori $qi(t_0) = \psi(p_i(t_0))$. Si può dunque verificare che anche le matrici decompresse hanno raggiunto il consenso, dimostrando in tal modo i risultati teorici. Per verificare l'effettivo raggiungimento del consenso locale su Br e globale su $SO(3)$ possiamo calcolare la somma delle relative distanze dei punti $p_i(t)$ e $q_i(t)$ su $SO(3)$ al variare del tempo.

$$cp(t) = \sum_{i \in \bar{n}} \sum_{j > i} d(p_i(t), p_j(t)) \quad (4.2)$$

e

$$cq(t) = \sum_{i \in \bar{n}} \sum_{j > i} d(q_i(t), q_j(t)) \quad (4.3)$$

Graficando queste due funzioni si verifica che hanno entrambe un'andamento esponenziale, e quindi tendono a zero per $t \rightarrow \infty$, come ci aspettavamo. Infine con Matlab è possibile graficare i droni durante il raggiungimento del consenso (sia locale, per i punti compressi, sia globale per i punti decompressi). Si osserva che i droni partono da un'assetto definito dalle condizioni iniziali date (che devono essere matrici in $\text{SO}(3)$ e che possiamo generare random all'inizio), e progressivamente si portano in un'assetto comune che varia dalla configurazione iniziale.

Nelle simulazioni successive andremo ad utilizzare come matrici di adiacenza:

i	matrice A					matrice B				matrice C			
1	0	1.5	0.5	0	0	0	1.0	0	0	0	0.5	0	0
2	0	1.5	1.5	1.5	1.0	0	0.5	0	0.5	0	1.0	0	0
3	1.0	1.5	0	0	0	0.5	0	1.5	0	0	1.5	1.0	0
4	1.0	1.5	0.5	0	0	1.5	1.5	0	0	1.0	0	0	0

4.1. Simuazione con matrici random. Possiamo considerare di scegliere solo una volta fuori dal for del ciclo principale, in modo random, tra le 3 proposte, la matrice dei pesi. In tal modo abbiamo che i canali comunicativi tra i vari droni non cambiano nel tempo, ma restano sempre gli stessi. Ciò comporta comunque il raggiungimento del consenso, ma in modo più lento rispetto al caso in cui sceglievamo la matrice dei pesi ad ogni iterazione del ciclo principale.

4.2. Simulazione con matrice dei pesi modificate. Un'altra prova che si può fare è modificare i valori della matrice dei pesi. Ad esempio scegliendole con valori molto alti e con il minor numero possibile di zeri, avremo una convergenza sempre maggiore. Ciò si spiega facilmente, dato che più il peso $W(i,j)$ è alto e maggiore è l'informazione che il drone i -esimo acquisirà sullo stato del drone j -esimo dato che vale:

$$\dot{p}_i = \sum_{j \in N_i(G_{\sigma(t)})} W(i,j) \exp^{-1}_{p_i}(p_j)$$

Invece ogni zero rappresenta l'assenza del canale comunicativo corrispondente, di conseguenza la situazione ottimale per avere una convergenza veloce sarebbe quella in cui si ha il numero minore possibile di zeri (oltre che pesi di valore alto). Se questi zeri li mettiamo in modo opportuno possiamo avere situazioni interessanti. Ad esempio scegliendo un valore di i tale che $1 < i < n$ e azzerando la colonna e la riga i -esime della matrice dei pesi W , avremo che non ci sarà alcuna connessione tra il drone i -esimo e i restanti droni, né da esso verso gli altri né il contrario. Di fatto, quindi, il drone i -esimo è isolato dai droni circostanti con cui non può scambiare informazioni in alcun modo. Graficando l'assetto dei droni con Matlab, si osserva che gli $n-1$ droni restanti raggiungono il consenso, ovvero un assetto comune, mentre il drone i -esimo rimane immobile, ovvero resta nello stato iniziale, perciò:

$$\frac{dp_i}{dt} = \frac{dq_i}{dt} = 0$$

Dunque lo stato del drone i -esimo non varia nel tempo.

4.3. Simulazione con metodo di Heun e Rounge(RK2). Possiamo risolvere l'IVP usando altri metodi come Heun o Rounge e notiamo come anche in questi casi riusciamo a raggiungere il consenso.

4.4. Simulazione con matrici antipodali. Prese come matrici iniziali $q_i(0) = q_i^0$ avremo:

$q_1(0) = q_1^0$			$q_2(0) = q_2^0$		
1.000	0	0	-1.000	0	0
0	1.000	0	0	-1.000	0
0	0	1.000	0	0	1.000
$q_3(0) = q_3^0$			$q_4(0) = q_4^0$		
-1.000	0	0	1.000	0	0
0	1.000	0	0	-1.000	0
0	0	-1.000	0	0	-1.000



Perciò otterremo che i dati locali e globali convergeranno come:

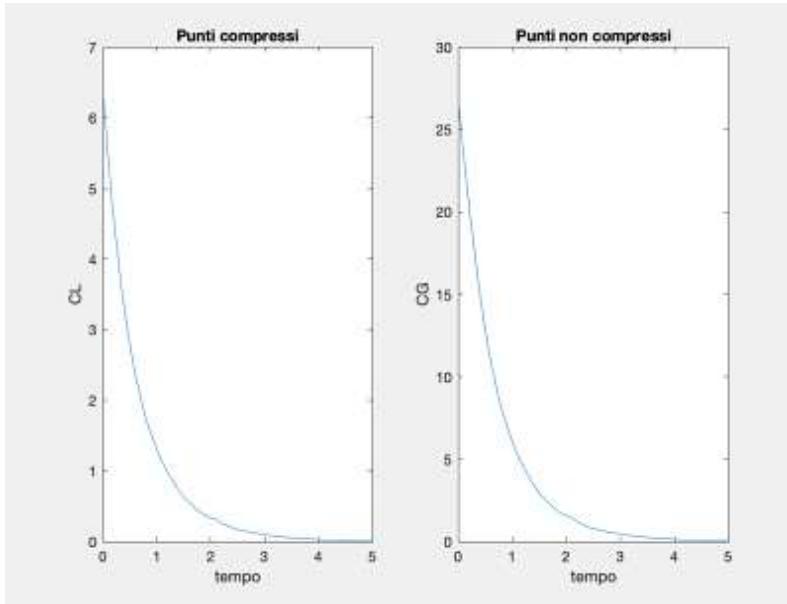


FIGURA 4.1. Vettori di prestazioni di convergenza per dati non compressi e compressi

In questo caso essendo le matrici antipodali il consenso verrà raggiunto, ma più lentamente. Allora raggiungendo il consenso nel caso peggiore, lo raggiungeremo per ogni altro caso.

4.5. Simulazione senza compresione. Infine abbiamo provato a considerare come stati di partenza dei droni dei punti in $\text{SO}(3)$ già appartenenti alla palla B_r . Quindi non è necessario effettuare una compressione usando la funzione ϕ . Quello che si ottiene è che il consenso viene raggiunto però più lentamente rispetto al caso in cui veniva usata la funzione di compressione.

4.6. Simulazione con drone leader. Osservando gli stormi di uccelli che volano si può notare che essi si muovono in gruppo come una nube e molti studiosi si sono chiesti come facciano a coordinarsi in questo modo. Addirittura alcune specie di uccelli migratori, come l'ibis eremita volano formando una "V". Da varie ricerche è risultato [2] che:

"Quante volte abbiamo osservato il cielo e visto uno stormo di uccelli migratori volare in formazione a V? Questo strano fenomeno è rimasto oscuro per molto tempo, nonostante diverse teorie matematiche abbiano tentato di spiegarlo. Analizzando i dati ottenuti, si è visto che lo stormo manteneva approssimativamente una formazione a delta, sebbene la posizione dei singoli elementi fosse dinamica, con cambiamenti del ritmo di battito delle ali. Più specificamente, gli animali lungo la V dietro e al lato dell'uccello in testa, o capobanda, battevano le ali in fase con esso. Questo permetteva loro di evitare le turbolenze e di beneficiare del flusso di aria verso l'alto (effetto upwash) causati dal movimento delle ali del capobanda, in modo da ridurre la fatica e aumentare la velocità".

Come emerge da questi studi durante il volo i componenti dello stormo comunicano in qualche modo tra loro, per avere le informazioni sul miglior assetto di volo. Ed in particolare emerge che vi è una sorta di leader dello stormo il cui assetto è seguito dagli altri componenti. Anche nel volo dei droni può essere utile la presenza di uno o più leader, ad esempio, se volessimo controllare l'andamento di un insieme di droni, sarebbe molto più comodo controllare un solo drone al cui assetto gli altri si conformano (attraverso il consenso), che controllarli uno per uno separatamente!



FIGURA 4.2. *Leader nello stormo di uccelli*

E nel nostro caso come possiamo ottenere la presenza di un leader che guidi i restanti droni? E' semplicissimo, è sufficiente azzerare l'intera riga della matrice W corrispondente al drone leader. In tal modo(il leader sia il drone i -esimo): -i valori $w(i, j) = 0, \forall j \in [1, n]$, cioè il drone i (leader) non modifica il proprio stato in funzione degli stati degli altri droni, mantenendo così il proprio stato invariato. -al contrario gli altri $n - 1$ droni potranno accedere allo stato del drone leader, modificando il loro stato di conseguenza e raggiungendo come assetto di consenso quello del drone leader. Graficando ciò si verifica che effettivamente il drone i rimane fermo allo stato iniziale mentre gli altri si allineano al suo assetto.



Notiamo inoltre come il Drone 1 (leader) non ruoti, bensì quest'ultimo cedendo solo informazioni fa sì che gli altri seguano la sua direzione. In questo caso ho che i vettori delle prestazioni di

convergenza saranno:

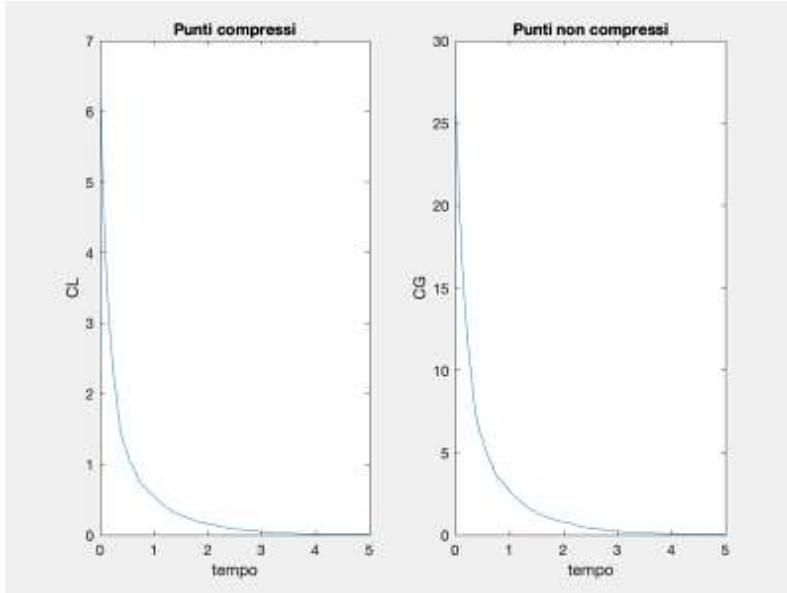


FIGURA 4.3. Vettori di prestazioni di convergenza per dati non compressi e compressi

4.7. Simulazione di un caso fisico. Possiamo inoltre dare alla distribuzione dei pesi all'interno della matrice W un significato di tipo fisico. Quello che noi abbiamo considerato in questo progetto è stato un consenso nell'assetto rotazionale e non traslazionale, ovvero gli stati che abbiamo considerato sono punti in $SO(3)$ (cioè matrici ortogonali speciali) e non posizioni nello spazio(R^3). Il consenso verso un unico punto dello spazio non è stato l'oggetto dei nostri studi, tuttavia è possibile rappresentare la situazione in cui i droni si avvicinano verso un'unica posizione dello spazio, per poi allontanarsene, realizzando una matrice dei pesi, i cui valori oscillano in maniera sinusoidale nel tempo. Avremo in questo modo i pesi della matrice W , che in un semiperiodo della sinusoide cresceranno, stando ad indicare la situazione in cui i droni avvicinandosi tra loro spazialmente riescono a comunicare in maniera migliore perché meno distanti l'uno dall'altro, mentre nell'altro semiperiodo decresceranno a causa del successivo allontanamento dei droni. Nella simulazione abbiamo utilizzato $T/2$ come periodo, rappresentando la situazione in cui dal picco negativo (maggiore distanza) arrivano al picco positivo della sinusoide (minore distanza tra loro), avvicinandosi quindi tra loro. Se, come abbiamo fatto, consideriamo una matrice de pesi W , con struttura “gaussiana”, cioè i cui elementi hanno i valori di una gaussiana centrata nel mezzo della matrice, ciò descriverà la situazione in cui i droni centrali, avendo pesi di valore più alto, saranno più vicini, mentre quelli più esterni saranno sempre più distanti. Quindi considerando un numero di droni elevato, ad esempio $n=100$, avremo una nube di droni che si addenseranno verso il centro. Se poi la matrice con struttura gaussiana ha un andamento nel tempo di tipo sinusoidale come quella descritta precedentemente, avremo la situazione in cui questa nube addensata di droni si addensa sempre di più verso il centro.



In questo caso i vettori di convergenza avranno un andamento del tipo:

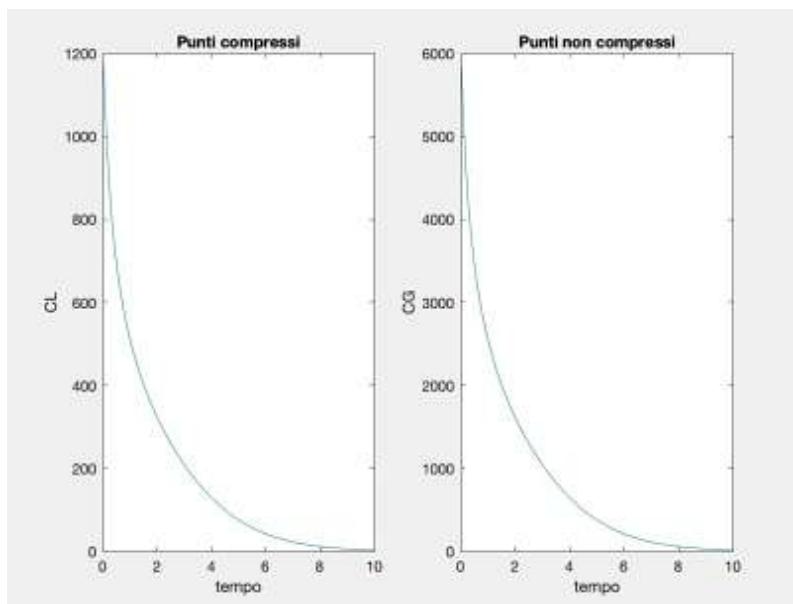


FIGURA 4.4. Vettori di prestazioni di convergenza per dati non compressi e compressi

5. Codici. In questa sezione vengono riportati alcuni dei codici utilizzati per il calcolo del consenso. Sono state realizzate diverse versioni del programma MATLAB, in accordo con le diverse specifiche di progetto per effettuare delle simulazioni di alcuni casi particolari come riportato sopra. Nei vari casi viene solamente cambiato il valore di alcuni parametri, lasciando comunque la struttura del programma immutata. Prima vengono memorizzate le variabili che verranno usate per il disegno dei droni e si carica il file contenente l'immagine del drone. Una volta deciso il numero di droni e la dimensione delle matrici, vengono inizializzate le matrici dei punti iniziali appartenenti a $\text{SO}(3)$ e le matrici dei pesi dei collegamenti, che vengono poi memorizzate rispettivamente in due vettori di matrici. A questo punto vengono calcolate le matrici compresse attraverso la funzione di compressione sopra riportata. Una volta inizializzate tutte le matrici viene effettuato il calcolo dell'equazione differenziale di stato, attraverso il metodo di Eulero (come già detto vi sono le varianti con RK2 ed Heun). Verranno perciò calcolate le matrici compresse e quelle non compresse ad ogni istante temporale, determinato dal passo di campionamento step. Queste matrici saranno matrici di rotazione, che verranno moltiplicate alle coordinate dei droni per ogni istante temporale, ottenendo così una visualizzazione in dell'intero movimento descritto dai droni prima di raggiungere il consenso (vengono visualizzate soltanto le matrici non compresse). Una volta completata l'intera visualizzazione vengono infine visualizzati i grafici della convergenza della distanza complessiva presente tra i droni in tutto l'arco temporale di calcolo, sia utilizzando i dati compressi che quelli non compresi.

Codice Main

```

1 %Main
2 load drone
3 close all;
4 figure;
5 Droni = [Xr';Yr';Zr']; %le coordinate dei punti che costituiscono la figura del
6 %drone
7 close all
8 Centrorot = [-0.0125 -0.0048 0]'; %centro di massa del nostro drone
9
10 %numero droni
11 n=4;
12
13 %la dimensione dei dati della matrice quadrata che rappresentano la
14 %posizione nel punto.
15
16 m=3;
17
18 %passo discretizzazione del tempo dei sistemi continui
19 step=0.01;
20
21 %inizializzazione matrici
22 InitData=init_data;

```

```

23 A(1)={[0,1.5,0.5,0;
24     0,0,0,1.5;
25     1,0,0,0;
26     1,0,0.5,0]};
27 A(2)={[0,1,0,0;
28     1,0,0.5,0;
29     0,0.5,0,1.5;
30     0,0,1.5,0]};
31 A(3)={[0,0.5,0,0;
32     0.5,0,1,0;
33     0,0,0,1;
34     0,1,0,0]};
35
36 %le matrici vengono compresse nel sottoinsieme Br.
37 for i=1:1:n
38     LData(:,:,:,i)=comp_fun(InitData(:,:,:,i),m);
39 end
40
41 %salvo i dati iniziali.
42 InitGData=InitData; %dati non compressi
43 InitLData=LData;    %dati compressi
44
45 % vettore di convergenza.
46 CL=[];
47 CG=[];
48 %posizione scritta centrale per il conteggio dei passi
49 dim = [.37 .205 .3 .3];
50
51 for i=1:1:500
52
53 %viene scelto casualmente una delle matrici con i pesi.
54 index=floor(rand*3)+1;
55 M=A{index}(:,:);
56
57
58 % Calcolo le nuove matrici all'istante successivo usando l'equazione di
59 % stato. (usiamo eulero in avanti)
60 for r=1:1:n
61     increment=zeros(m,m);
62     pr=LData(:,:,:,r); % prendo una alla volta le matrici compresse
63     for j=1:1:n
64         w=M(r,j);
65         if (w>0)
66             pj=LData(:,:,:,j); %prendo di nuovo, una alla volta, le matrici
                                compresse

```

```

67 % applico l'equazione di stato.
68 increment = increment+w.*log_S0(pr,pj); %increment mi indica la
69 % matrice velocit\`a con cui il punto cambia la sua posizione
70 end
71 end
72 % calcolo il nuovo valore della matrice, applicando l'esponenziale lo
73 % mappo su Br. devo moltiplicare increment (che \`e una velocit\`a) per
74 % step(che mi indica il tempo di discretizzazione), ottenendo la nuova
75 % posizione della matrice.
76 NewLData(:,:,r)=exp_S0(pr,step.*increment);
77
78 %memorizzo i nuovi dati nella vecchia matrice.
79 LData=NewLData;
80
81 %decomprimo i valori ottenuti e grafico il movimento dei droni fino al
82 %raggiungimento del consenso.
83 for j=1:1:n
84 GData(:,:,j)=decomp_fun(LData(:,:,j),m);
85
86 DroniRuotati = GData(:,:,j)*(Droni - Centrorot) + Centrorot; %droni ruotati
87 % usando la corrispondente matrice di rotazione
88 DR(j)=subplot(2,2,j); %ho 4 subplot, uno per ogni drone
89 plot3(DroniRuotati(1,:),DroniRuotati(2,:),DroniRuotati(3,:),'ro-'); %
90 % stampa i droni ruotati a punti usando le 3 coordinate
91 title(DR(j),sprintf('Drone %i',j));
92 grid on;
93 axis([-2 2 -2 2 -2 2]); %limito la finestra grafica alla porzione che mi
94 %interessa per vedere bene il drone
95 xlabel('x');
96 ylabel('y');
97 zlabel('z');
98 %visualizzazione conteggio dei passi
99 str=sprintf('Numero passi %i /500', i);
100 h=annotation('textbox', dim, 'string', str,'FitBoxToText','on');
101 pause(0.0001);
102 delete(h);
103
104 end
105 %inserimento visualizzazione a tempo reale del movimento dei droni
106 %attraverso le funzioni di visualizzazione drone e rotazione oggetti
107 tmp_L=0;
108 tmp_G=0;
109
110 for j=1:1:n

```

```

108     for k=(j+1):1:n
109         %calcolo la distanza tra due punti.
110         tmp_L=tmp_L+dist_S0(LData(:,:j),LData(:,:,k)); %compressi(locali)
111         tmp_G=tmp_G+dist_S0(GData(:,:j),GData(:,:,k)); %non compressi(non
112             locali)
113     end
114 end
115
116 % uso questi 2 vettori per visualizzare le prestazioni di convergenza
117 CL=[CL,[i;tmp_L]];
118 CG=[CG,[i;tmp_G]];
119
120
121 %visualizzazione grafica dell'andamento della distanza tra i punti
122 show_C(CL,CB,step);

```

Codice con dati iniziali

```

1 function Data = init_data
2
3 Data=zeros(3,3,4);
4 %Matrici antipodali tra loro
5 Data(:,:1)=[1,0,0;
6                 0,1,0;
7                 0,0,1];
8 Data(:,:2)=[-1,0,0;
9                 0,-1,0;
10                0,0,1];
11 Data(:,:3)=[-1,0,0;
12                0,1,0;
13                0,0,-1];
14 Data(:,:4)=[1,0,0;
15                0,-1,0;
16                0,0,-1];

```

Codice calcolo logaritmo

```

1 function L = realLogSO(R)
2 [V,T] = schur(R);
3 % R e' una matrice di rotazione quindi se e' di ordine dispari ha un
4 % autovalore pari a 1. Essendo una matrice "normale", la sua decomposizione
5 % di Schur e' diagonale a blocchi 1x1 e 2x2.

```

```

6 D = []; % Inizializza D ad una matrice vuota
7 [n,~]=size(R);
8 if mod(n,2) % Se n e0 dispari, T contiene un blocco "1" che occorre trovare
9   precisione = 10^(-10);
10  [~,pos] = min(abs(diag(T)-1-precisione)); % Posizione stimata del blocco "1":
11  % il valore non e' esattamente 1 quindi si cerca il valore piu' vicino;
12  % una precisione non adeguata potrebbe causare il malfunzionamento della
13  % funzione
14  T(pos,:)=[]; T(:,pos)=[]; % Elimina la riga e la colonna pos-ma
15  app = V(:,pos); V(:,pos)=[]; V=[app V]; % Porta la colonna pos-ma
16  in prima posizione
17  n = n-1; % Riduce la dimensione di 1
18  D(1,1) = 0; % Logaritmo del blocco 1 = 0
19 end
20 for b = 1:n/2
21  B = T(2*b-1:2*b,2*b-1:2*b); % Estrae i blocchi 2x2
22  ll = log(B(1,1) + B(1,2)*li); % real(ll) = 0 per una matrice di rotazione
23  [s,~] = size(D);
24  D(s+1:s+2,s+1:s+2) = [0 imag(ll);-imag(ll) 0]; % Aggiunge un
25  blocco-logaritmo a D
26 end
27 L = V*D*V'; % Risultato finale
28 return

```

codice calcolo Esponenziale nel gruppo Ortogonale Speciale

```

1 function h=exp_S0(p,x)
2 %dato il vettore tangente x del punto p su S0(3) calcolare il valore della
3 %funzione esponenziale corrispondente.
4 h=p*expm(p'*x);

```

codice calcolo Logaritmo nel gruppo Ortogonale Speciale

```

1 function v=log_S0(p,q)
2 %dati due punti p e q su S0(3) trova la funzione logaritmo
3 v=p*realLogS0(p'*q);

```

codice calcolo delle distanze nel gruppo Ortogonale Speciale

```

1 function d=dist_S0(p,q)
2 %dati due punti calcolo la distanza tra di essi
3 d=norm(realLogS0(p'*q), 'fro'); %definizione distanza su S0(3).

```

codice per la compressione dei punti della sfera SO in Br

```
1 function LData=comp_fun(GData,m)
2 %comprime tutti i punti della sfera S0 in Br.
3
4 e=eye(m);
5 v=log_S0(e,GData); %trovo il vettore tangente dal punto p al punto q.
6 LData=exp_S0(e,0.2.*v); % è mappato a Br attraverso la funzione esponenziale
```

codice per la compressione dei punti da Br sulla sfera SO

```
1 function GData=decomp_fun(LData,m)
2 %decomprimere i dati della sfera Br su S0
3
4 e=eye(m);
5 v=log_S0(e,LData);
6 GData=exp_S0(e,5.*v);
```

6. Conclusioni. In questo documento, forniamo principalmente una soluzione al consenso locale e un algoritmo di consenso globale attraverso l'utilizzo di strumenti come i Manifold Reimanniani e applichiamo queste tecniche proposte per risolvere il problema di consenso degli atteggiamenti di rotazione in rete, specializzandoci sul gruppo delle rotazioni $SO(3)$.

RIFERIMENTI BIBLIOGRAFICI

- [1] "Consensus on compact Riemannian manifolds" by Sheng Chen Lindu Zhao , Weigong Zhang , Peng Shi.
- [2] <https://www.wired.it/scienza/ecologia/2014/01/16/perche-stormi-uccelli-volo-v/>
- [3] Sherif, E. Morsy, Computing real logarithm of a real matrix, Int. J. Algebra 2 (3) (2008) 131–142.