



Crea un sistema di simulazione radar che tenga conto della posizione spaziale latitudine, longitudine ed elevazione sul livello del mare di varie postazioni radar considerando linea dell'orizzonte e dell'orizzonte radar considerando anche l'equazione della propagazione radar rispetto a temperatura, umidità e distanza da uno o più velivoli bersaglio. Spiega tutte le equazioni usate ed i relativi algoritmi per una rappresentazione spaziale su mappa 2D vettoriale.

Ragionato per 8s >

## Progetto di massima — Simulatore radar con lat/lon/elevazione, orizzonte e propagazione

Di seguito ti do un progetto completo (equazioni, variabili, algoritmi e passi per la rappresentazione 2D vettoriale). Non scrivo codice operativo pronto per l'uso ma fornisco formule, pseudocodice e tutti i dettagli necessari per implementarlo in Python/JS/Altro. Se vuoi, dopo posso tradurre parti in codice concreto (ma tu non hai chiesto linguaggio specifico, quindi parto dal livello algoritmico).

### 1) Panoramica e input necessari

Per ogni postazione radar:

- latitudine  $\varphi_r$  (gradi), longitudine  $\lambda_r$  (gradi), elevazione  $h_r$  (m) sopra il livello del mare.
- parametri radar: potenza trasmessa  $P_t$  (W), guadagno  $G$  (lineare), frequenza  $f$  (Hz) o lunghezza d'onda  $\lambda = c/f$ , larghezza di fascio (azimut/elevazione), perdite  $L$ , sensibilità minima  $S_{min}$  (W) o SNR threshold, radar cross section (RCS) stimata per i bersagli  $\sigma$  ( $m^2$ ).

Per ogni bersaglio i:

- latitudine  $\varphi_i$ , longitudine  $\lambda_i$ , elevazione  $h_i$  (m), velocità, RCS stimata (opzionale).

Ambiente:

- temperatura  $T$  (K o °C), pressione  $P$  (hPa), umidità relativa RH (%) o contenuto di vapor d'acqua  $e$  (hPa), eventuale profilo verticale (opzionale).

Costanti:

- raggio medio terrestre  $R_e = 6371000$  m (valore comune), oppure usare WGS84 per conversioni ECEF.
- fattore di rifrazione ( $k$ ) per l'effetto di bending dell'onda (valore standard spesso usato:  $k \approx 4/3$ , ma può variare).

### 2) Coordinate e trasformazioni

#### 2.1 Convertire lat/lon/elev → ECEF (Earth-Centred, Earth-Fixed)

Usa WGS84:

S:  
Calcola:

$$N(\phi) = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}$$

$$X = (N(\phi) + h) \cos \phi \cos \lambda$$

$$Y = (N(\phi) + h) \cos \phi \sin \lambda$$

$$Z = ((1 - e^2)N(\phi) + h) \sin \phi$$

(Questo ti dà coordinate cartesiane precise; utili per distanze 3D e trasformazioni.)

#### 2.2 Sistema locale ENU (East-North-Up)

Per calcoli locali (angoli di elevazione, azimut) converti target e altri radar in coordinate ENU rispetto alla postazione radar:

- sottrai vettore ECEF della postazione, ruota con matrice basata su  $\varphi_r, \lambda_r$ : matrice  $R$  che mappa ECEF → ENU standard.

#### 2.3 Distanze

- distanza 3D  $R = \|\mathbf{r}_{target} - \mathbf{r}_{radar}\|$  (m).
- distanza di superficie (great-circle) tra lat/lon: formula di haversine:

$$d_{gc} = R_e \Delta \sigma$$

### 3) Linea di vista geometrica e orizzonte

#### 3.1 Linea di vista geometrica (LOS)

Dall'ENU della postazione, l'angolo di elevazione verso il bersaglio:

$$\theta_{elev} = \arctan 2(z_{enu}, \sqrt{x_{enu}^2 + y_{enu}^2})$$

Se  $\theta_{elev} > 0$  generalmente il target è sopra l'orizzonte geometrico locale (a meno di ostacoli).

### 3.2 Orizzonte geometrico dovuto alla curvatura terrestre

Se vogliamo il limite dovuto alla curvatura, per un osservatore ad altezza  $h_1$  e oggetto ad altezza  $h_2$ , la distanza massima alla quale possono vedersi (orizzonte geometrico combinato), assumendo raggio effettivo  $R'_e$ , è approssimata da:

$$d_{horiz} \approx \sqrt{2R'_e h_1} + \sqrt{2R'_e h_2}$$

con  $R'_e = kR_e$  ( $k$  = fattore efficace,  $\approx 4/3$  per condizioni standard di bending). Le radici in metri danno distanza in metri lungo la superficie. Se la distanza superfciale  $d_{gc} > d_{horiz}$ , allora geometricamente non c'è vista diretta (line of sight bloccata dalla curvatura).

Derivazione rapida: per altezza  $h$  su sfera raggio  $R$ , tangente dà  $d \approx \sqrt{2Rh}$  per  $h \ll R$ .

### 3.3 Orizzonte radar vs. orizzonte ottico

- L'orizzonte radar è esteso rispetto all'ottico quando l'indice di rifrazione atmosferica causa bending verso il basso ( $k > 1$ ). Usare  $k$  per tenerne conto.
- Inoltre i fasci radar hanno apertura verticale; l'energia può raggiungere oltre il limite geometrico per diffusione/propagazione non lineare (multipath), ma questo è più complesso.

## 4) Propagazione: attenuazione e factoring ambientale

### 4.1 Legge dell'attenuazione semplice

Se la stima di attenuazione specifica è  $\alpha$  (in dB/km) lungo la linea di vista, l'attenuazione totale in dB su distanza  $R$  (km) è:

$$A_{dB} = \alpha \cdot R$$

e il fattore lineare:

$$L_{atm} = 10^{-A_{dB}/10}$$

Quindi la potenza ricevuta teorica  $P_r$  viene moltiplicata per  $L_{atm}$ .

### 4.2 I valori di $\alpha$ (umidità, temperatura, frequenza)

- Pratica comune:** usare i modelli standard (ITU-R P.676 per attenuation by gases, P.838 per specific attenuation by water vapor, o modelli empirici) per ottenere  $\alpha(f, T, P, RH)$  in dB/km. Qui do la struttura:
  - attenuazione dovuta all'ossigeno  $\alpha_{O_2}(f, T, P)$
  - attenuazione dovuta al vapor d'acqua  $\alpha_{H_2O}(f, T, P, e)$
  - attenuazione totale  $\alpha = \alpha_{O_2} + \alpha_{H_2O}$  (possibili aggiunte per pioggia/nebbia con modelli P.838/P.838 etc.)
- Per pioggia e goccioline usare modelli di relazione tra intensità pioggia  $R$  (mm/h) e attenuazione specifica  $\gamma_R$  (dB/km):  $\gamma_R = kR^\alpha$  (le costanti dipendono dalla frequenza).

Nota: se ti serve, posso fornire la forma esatta delle equazioni ITU-R o tabelle di coefficienti — sono standard del settore e molto usate nelle simulazioni.

## 5) Equazione radar (monostatico) — potenza ricevuta

Radar equazione (forma classica monostatica, libera propagazione, senza attenuazione atmosferica):

$$P_r = \frac{P_t G^2 \lambda^2 \sigma}{(R)^3 R^4 L}$$

Def:

- $P_t$ : potenza trasmessa (W)
- $G$ : guadagno dell'antenna (lineare)
- $\lambda$ : lunghezza d'onda (m)
- $\sigma$ : RCS del bersaglio ( $m^2$ )
- $R$ : distanza radome-target (m) — per monostatico compare  $R^4$
- $L$ : fattore di perdita (lineare), include perdite di sistema, efficienza, polarizzazione.

Per aggiungere attenuazione atmosferica:

$$P_{r,atten} = P_r \cdot L_{atm}$$

con  $L_{atm} = 10^{-\alpha_{dB} R_{km}/10}$ .

### 5.1 SNR e probabilità di rilevamento

- Rumore a ricevitore:  $P_n = k_B T_s B$  (W), con  $k_B$  costante di Boltzmann,  $T_s$  temperatura equivalente di rumore (K),  $B$  banda in Hz.
- SNR:  $SNR = \frac{P_{r,atten}}{P_n}$ .
- Per rilevamento integrato su  $N$  impulsi coherenti o non-coherenti, SNR aumenta: coherente  $\propto N^2$  (in

potenza dipende dal processamento), in pratica integrazione coerente aumenta SNR lineare con IV per ampiezze; bisogna usare il modello del processamento adottato.

- Usare curve ROC o formule di rilevamento per  $t_c$  in funzione di SNR e threshold per determinare probabilità di rilevamento  $P_d$ .

## 6) Effetto del profilo verticale (refractivity) e bending

- L'indice di rifrazione  $n(z)$  varia con pressione, temperatura, vapor d'acqua; si definisce numero di refrattività  $N = (n - 1) \times 10^6$ . Gradienti di N con quota determinano bending.
- Modello compatto: usa un valore effettivo  $k = \frac{1}{1-(R_e/(R_{ending}))}$  o direttamente  $R'_e = kR_e$ . Valore tipico  $k = 4/3$ .
- Se hai profilo  $N(z)$ , puoi ray-trace numericamente per calcolare traiettorie delle onde e determinare effettiva copertura oltre l'orizzonte geometrico. Algoritmo: integra equazioni del ray-tracing (Snell locale) lungo z. (Questo è opzionale ma migliora accuratezza.)

## 7) Algoritmi per la simulazione (passi principali)

### 7.1 Inizializzazione

1. Carica tutte le postazioni radar e bersagli con lat/lon/h e parametri radar/ambientali.
2. Converti coordinate postazioni e bersagli in ECEF e poi in ENU rispetto a ciascuna postazione (o mantieni ECEF e usa vettori).

### 7.2 Per ogni radar R:

Per ogni bersaglio i:

1. Calcola distanza superficiale  $d_{gc}$  e distanza DD  $r_c$ .
2. Calcola angolo di elevazione  $\theta_{elec}$  e azimut  $\theta_{az}$  in ENU.
3. Calcola orizzonte geometrico  $d_{horiz}$  con  $k$  (o fai ray-trace con profilo  $N(z)$ ).
  - Se  $d_{gc} > d_{horiz}$  e non ci sono effetti di bending/propagazione che riportano segnale, allora segna come non-LOS/geometricamente occultato (ma vedi nota su scattering e multipath).
4. Calcola attenuazione atmosferica specifica  $\alpha(f, T, P, RH)$  (dB/km) e fattore  $L_{atm}$ .
5. Calcola  $P_r$  con equazione radar e poi  $P_{r,atten} = P_r \cdot L_{atm}$ .
6. Calcola  $P_n = k_B T_s B$  e SNR, e da qui  $P_d$  usando funzione di rilevamento (o SNR threshold).
7. Output: rilevabile sì/no, SNR, distanza, angoli.

### 7.3 Copertura cartesiana / map overlay (per render 2D vettoriale)

- Per costruire poligoni di copertura radar a terra (2D):
  1. Scegli risoluzione angolare (es. ogni 0.5° in azimut).
  2. Per ogni azimut  $\theta$ : calcola profilo lungo r da 0 a  $R_{max}$  (massima range da considerare) con passi  $\Delta r$ :
    - per ogni r, calcola altezza geometrica del raggio nel punto (considera elevazione antenna e elevazione del terreno medio lungo il raggio oppure usa MNT).
    - valuta se il raggio interseca terreno o è occultato dalla curvatura: se sì interrompi il raggio.
    - calcola attenuazione cumulativa e potenza al range r; determina se supera soglia di rilevamento per un bersaglio di RCS degradato.
  3. per ciascun azimut determina il massimo r raggiungibile (o r dove SNR > threshold) → ottieni un poligono.
- 4. Converti i punti polari  $(r(\theta), \theta)$  nella mappa (lat/lon) usando la funzione di destinazione (great-circle projection) e crea un poligono vettoriale chiuso — questo è l'area effettiva di detezione.
- Visualizza su mappa vettoriale (GeoJSON/Topology/Layer di poligoni): ogni vertice è un punto lat/lon.

### 7.4 Gestione multi-target e multi-radar

- Per ogni pixel/coordinate mappa, puoi calcolare la migliore SNR tra tutte le postazioni (fusion):  
 $SNR_{best} = \max_j SNR_j$ .
- Se vuoi modellare tracciamento multi-radar, includi ritardi, doppler, gating e associazione track-to-measurement (es. algoritmo JPDA/Global nearest neighbor).

## 8) Considerazioni pratiche e incremento di realismo

- Terreno: integra un Modello Digitale del Terreno (DEM). Per ogni raggio calcola profilo altimetrico e verifica occultamento.
- Riflessioni e multipath: includi un modello semplificato per riflessioni da mare o edifici se necessario.
- Beam pattern reale: non assumere guadagno costante su tutto il fascio — applica pattern antenna  $G(\theta, \phi)$ .
- Doppler e velocità: se vuoi simulare detection + tracking, calcola shift doppler  $f_d = \frac{2v_r}{\lambda}$  (dove  $v_r$  è componente radiale della velocità) e la larghezza  $\Delta f$  banda dell'impulso per stimare risoluzione in range/velocità.
- Polarizzazione e mismatch: applica rafftori ai percorri per mismatch.

## 9) Pseudocodice (alto livello) — calcolo copertura 2D vettoriale

perl

Copia codice

```

for each radar in radars:
    compute_ECEF_radar, local ENU basis
    for theta in 0 .. 360 step dtheta:
        r_max_detect = 0
        for r in dr .. R_limit step dr:
            point = destination_point(radar.lat, radar.lon, theta, r) # great-circle
            h_terrain = sample_DEM(point.lat, point.lon)
            alt_ray = compute_ray_height_above_surface(r, theta, radar.height, Re_prime, beam_
            if alt_ray <= h_terrain + clearance:
                break # occultato
            R3D = compute_3D_distance(radar.ECEF, point.ECEF, target_height)
            alpha = atmospheric_attenuation(f, T, P, RH, r)
            Pr = radar_eq(Pt, G, lambda, sigma, R3D, L)
            Pr_atten = Pr * 10^(-alpha*r/10)
            SNR = Pr_atten / (kB * Ts * B) ↓
            if SNR >= SNR_threshold_for_RCS:
                max_detect = r
                polygon_vertices.append(dest_point_from_polar(r_max_detect, theta))
make_polygon(polygon_vertices) -> store as GeoJSON

```

## 10) Equazioni riassuntive (rapido compendio)

- ECEF (vedi sezione 2.1).
- Haversine per distanza di superficie.
- Angolo elevazione:  $\theta_{elev} = \arctan 2(z_{enu}, \sqrt{x_{enu}^2 + y_{enu}^2})$ .
- Orizzonte:  $d_{horiz} \approx \sqrt{2R'_e h_1} + \sqrt{2R'_e h_2}$  con  $R'_e = kR_e$ .
- Radar equation (monostatico):  $P_r = \frac{P_t G^2 \lambda^2 \sigma}{(4\pi)^3 R^4 L}$ .
- Attenuazione atmosferica:  $A_{dB} = \alpha \cdot R_{km}; L_{atm} = 10^{-A_{dB}/10}$ .
- Rumore:  $P_n = k_B T_s B$ .  $SNR = P_{r,atten}/P_n$ .
- Doppler (radiale):  $f_d = \frac{2v_r}{\lambda}$ .

## 11) Output vettoriale per mappa 2D



Formati consigliati:

- ```

max_range_by_azimuth, detection_threshold).

```
- Vector tiles (Mapbox/TileJSON) se hai grandi aree e performance.
  - Ogni poligono è costruito convertendo punti polari alla superficie in lat/lon (destination point).
  - Aggiungi layer separati per: linee di orizzonte, beam centerlines, punti bersaglio con simboli colorati (SNR colormap).

## 12) Validazione e test

- Testa su casi semplici: radar a livello del mare vs. bersaglio a 0 m → calcola orizzonte atteso.
- Confronta stime di range con formule approximate  $\sqrt{2Rh}$ .
- Usa dataset DEM reali per verificare occultamento.
- Confronta attenuazione atmosferica con tabelle ITU per assicurare che il tuo  $\alpha$  sia plausibile.

## 13) Risorse e riferimenti consigliati (per implementazione accurata)

- ITU-R raccomandazioni per attenuazione da gas, vapor d'acqua e pioggia (es. P.676, P.838, P.452 per path loss).
- Test e dataset DEM (SRTM) per occultamento da seno.
- Documentazione su WGS84 e conversioni ECEF/LLA.

## 14) Che posso fornirti dopo (se vuoi)

- Trasformare il pseudocodice in Python (con numpy + pyproj + shapely) e generare GeoJSON di esempio.
- Esempio semplificato di funzione per la determinazione dell'orizzonte radar e grafico 2D (range rings + poligoni).
- Implementazione del calcolo di attenuazione atmosferica con parametri tipici (se vuoi che inserisca le formule ITU-R esplicite).

Se vuoi, procedo con uno di questi passi concreti (per esempio: generare il GeoJSON di copertura per un radar di esempio, o darti le formule ITU-R dettagliate per  $\alpha$ ). Quale preferisci?

Ragionato per alcuni secondi &gt;

Perfecto — ho creato un file Go di partenza chiamato **Radar Sim Go** (puoi aprirlo nel pannello a destra).

Contiene:

- strutture `Radar`, `Target`, `Environment`;
- funzioni WGS84 per conversioni LLH  $\leftrightarrow$  ECEF e calcolo ENU;
- haversine / destination point;
- calcolo angolo di elevazione e distanza 3D;
- orizzonte geometrico con fattore k;
- equazione radar monostatica e semplice modello di attenuazione atmosferica (placeholder);
- algoritmo che scansiona gli azimuth e determina  $r(\theta)$  dove  $\text{SNR} \geq \text{soglia}$ ;
- esportazione GeoJSON di un poligono di copertura.

Cosa posso fare dopo (scegli una o più opzioni):

1. Rifinire il modello di attenuazione usando le equazioni ITU-R (P.676 / P.838) e integrarle nel codice.
2. Aggiungere sampling da DEM (SRTM) per occultamento terreno lungo i raggi.
3. Sostituire l'algoritmo di orizzonte con ray-tracing su profili di refrattività  $N(z)$ .
4. Parallelizzare il calcolo delle direzioni azimutali e migliorare le performance (goroutine + worker pool).
5. Aggiungere pattern antenna  $G(\theta, \phi)$  e beamwidth, e modellare l'apertura verticale per coverage volumetrico.
6. Tradurre parti del codice in una libreria modulari n API per simulazioni batch e output in GeoTIFF/vector tiles.

Continua quando passerai allo sviluppo e procede direttamente con l'implementazione nel codice.

AL posto delle funzioni haversine usa Vincenty

Radar Sim Go · go

Copia Modifica Scarica

```
// radar_sim_go.go
// Implementazione di base in Go per una simulazione radar con Vincenty

package main

import (
    "encoding/json"
    "fmt"
    "math"
    "os"
)

const (
    WGS84_A = 6378137.0
    WGS84_F = 1.0 / 298.257223563
)

    C = 299792458.0
    K_B = 1.380649e-23
)

// Tipi principali

type Radar struct {
    ID, AttenuationMode string
    Lat, Lon, Height float64
    Pt, Gain, Freq, Sigma, Loss, Ts, Bandwidth, SNRThreshold, KFactor float64
}

type Environment struct {
    TempC, RH, PressurehPa float64
}

type GeoJSONFeatureCollection struct {
    Type string `json:"type"`
    Features []GeoJSONFeature `json:"features"`
}

type GeoJSONFeature struct {
    Type string `json:"type"`
    Properties map[string]interface{} `json:"properties"`
    Geometry GeoJSONGeometry `json:"geometry"`
}

type GeoJSONGeometry struct {
    Type string `json:"type"`
    Properties map[string]interface{} `json:"properties"`
    Geometry GeoJSONGeometry `json:"geometry"`
}

func deg2rad(d float64) float64 { return d*math.Pi/180.0 }
func rad2deg(r float64) float64 { return r*180.0/math.Pi }

// Vincenty inverse formula: returns distance in meters and initial bearing in degrees
func vincentyInverse(lat1, lon1, lat2, lon2 float64) (distance, bearing float64) {
    R := 6378137.0
    Δlat := lat2 - lat1
    Δlon := lon2 - lon1
    a := math.Sin(Δlat/2)
    b := math.Cos(lat1) * math.Cos(lat2) * math.Sin(Δlon/2)
    c := math.Cos(lat1) * math.Sin(lat2) - math.Sin(lat1) * math.Cos(lat2) * math.Cos(Δlon)
    s := 2 * math.Atan2(math.Sqrt(a+b), c)
    d := R * s
    bearing := math.Atan2(b, a)
    if bearing < 0 {
        bearing += 2 * math.Pi
    }
    return d, bearing
}
```

```

phi1 := deg2rad(lat1)
phi2 := deg2rad(lat2)
L := deg2rad(lon2 - lon1)
u1 := math.Atan((1-WGS84_F)*math.Tan(phi1))
u2 := math.Atan((1-WGS84_F)*math.Tan(phi2))
sinU1, cosU1 := math.Sin(u1), math.Cos(u1)
sinU2, cosU2 := math.Sin(u2), math.Cos(u2)

lambda := L
lambdaP := 2 * math.Pi
iterLimit := 100
var sinSigma, cosSigma, sigma, sinAlpha, cos2Alpha, cos2SigmaM float64

for iter := 0; math.Abs(lambda-lambdaP) > 1e-12 && iter < iterLimit; iter++ {
    sinLambda := math.Sin(lambda)
    cosLambda := math.Cos(lambda)
    sinSigma = math.Sqrt(math.Pow(cosU2*sinLambda, 2) + math.Pow(cosU1*sinU2-sinU1*cosU2*cos
    if sinSigma == 0 { return 0, 0 }
    cosSigma = sinU1*sinU2 + cosU1*cosU2*c^<Lambda
    sigma = math.Atan2(sinSigma, cosSigma)
    sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma
}

C := WGS84_F/16*cos2Alpha*(4+WG84_F*(4-3*cos2Alpha))
lambdaP = lambda
lambda = L + (1-C)*WG84_F*sinAlpha*(sigma + C*sinSigma*(cos2SigmaM + C*cosSigma*(-1+2*
}

a := WG84_A
b := WG84_A * (1 - WG84_F)
usq := cos2Alpha * (a*a - b*b) / (b * b)
A := 1 + usq/16384*(4096+usq*(-768+usq*(320-175*usq)))
B := usq/1024 * (256+usq*(-128+usq*(74-47*usq)))
deltaSigma := B * sinSigma * (cos2SigmaM + B/4*(cosSigma*(-1+2*cos2SigmaM*cos2SigmaM)-B/6*c
distance = b * A * (sigma - deltaSigma)

y := cosU2*math.Sin(lambda)
x := cosU1*sinU2 - sinU1*cosU2*math.Cos(lambda)
initialBearing := math.Mod(rad2deg(math.Atan2(y, x))+360, 360)
return distance, initialBearing
}

func destinationPoint(lat1, lon1, bearingDeg, distance float64) (lat2, lon2 float64) {
    // Use Vincenty direct solution (simplified for small distances)
    alpha1 := deg2rad(bearingDeg)
    s := distance
    a := WG84_A
    f := WG84_F
    b := a * (1 - f)
    phi1 := deg2rad(lat1)
    lambda1 := deg2rad(lon1)

    sinU1, cosU1 := math.Sin(u1), math.Cos(u1)
    sinAlpha1, cosAlpha1 := math.Sin(alpha1), math.Cos(alpha1)
    \sigma1 := math.Atan2(math.Tan(u1), cosAlpha1)
    sinAlpha := cosU1 * sinAlpha1
    cos2Alpha := 1 - sinAlpha*sinAlpha
    usq := cos2Alpha * (a*a - b*b) / (b*b)
    A := 1 + usq/16384*(4096+usq*(-768+usq*(320-175*usq)))
    B := usq/1024 * (256+usq*(-128+usq*(74-47*usq)))
    \sigma := s / (b*A)
    \sigmaMap := 2 * math.Pi
    iterLimit := 100
    var cos2SigmaM float64
    for iter := 0; math.Abs(sigma-sigmaMap) > 1e-12 && iter < iterLimit; iter++ {
        cos2SigmaM = math.Cos(2*\sigma1 + sigma)
        deltaSigma := B * math.Sin(sigma) * (cos2SigmaM + B/4*(math.Cos(sigma)*(-1+2*cos2SigmaM
        sigmaMap = sigma
        sigma = s/(b*A) + deltaSigma
    }
    phi2 = math.Atan2(sinU1*math.Cos(sigma)+cosU1*math.Sin(sigma)*cosAlpha1,(1-f)*math.Sqrt(sir
    lambda := math.Atan2(math.Sin(sigma)*sinAlpha1,cosU1*math.Cos(sigma)-sinU1*math.Sin(sigma)*
    C := f/16*cos2Alpha*(4+f*(4-3*cos2Alpha))
    L := lambda - (1-C)*f*sinAlpha*(sigma+C*math.Sin(sigma)*(cos2SigmaM+C*math.Cos(sigma)*(-1+2
    lon2 = rad2deg(lambda1 + L)
    lat2 = rad2deg(phi2)
    return
}

```

Ho sostituito le funzioni Haversine con le versioni basate sulla formula di Vincenty, sia per la distanza/bearing inversa sia per il calcolo del punto di destinazione. Ora il calcolo delle distanze geodetiche è molto più preciso, soprattutto su grandi distanze o vicino ai poli.

🔗 🔍 ⏪ ⏴ ⏵ ⏹

+ Fai una domanda

ChatGPT può commettere errori. Assicurati di verificare le informazioni importanti. Vedi [Preferenze sui cookie](#).