Calculating distance using GPS data

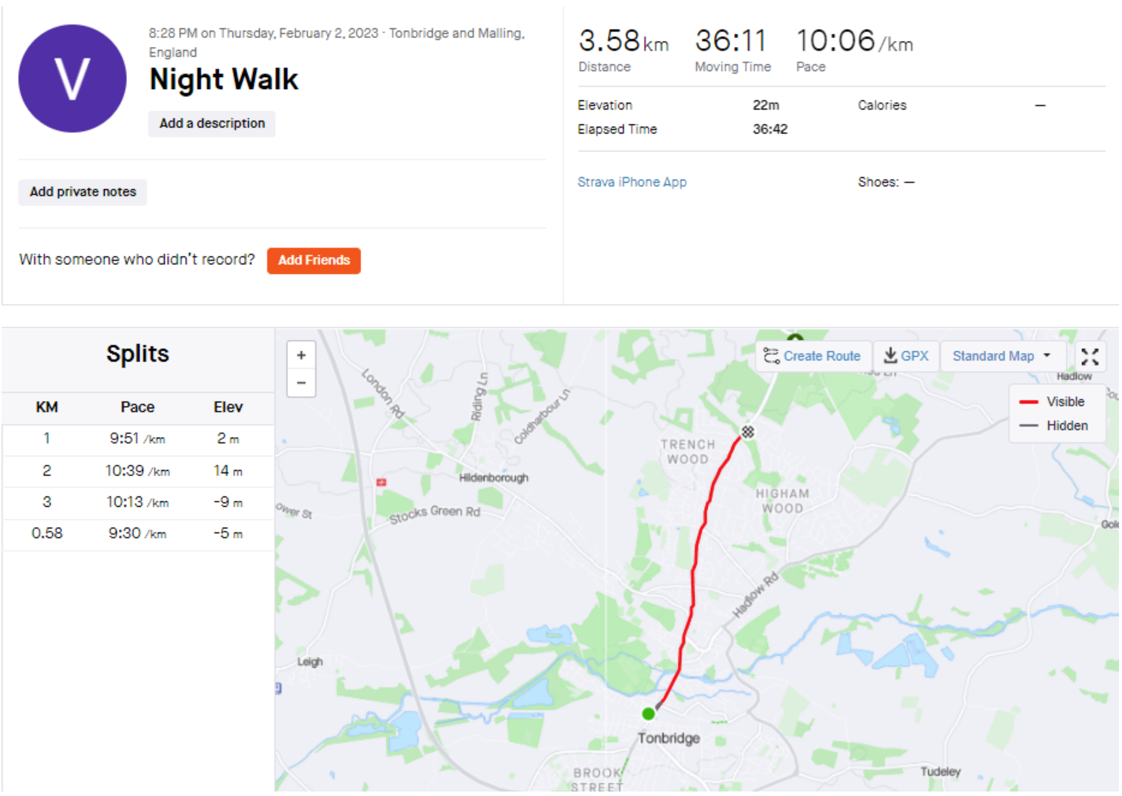
Using only elementary mathematics

VAMSHI JANDHYALA

February 6, 2023

Background

I recently started using the Strava app to keep track of my walking. A few "weird" Strava activity maps and WhatsApp discussions piqued my curiosity. I wanted to understand how Strava calculates the distance covered and if I could replicate the distance measurements using elementary mathematics. Here are the details of my run from last night:



Activity Data

Strava uses GPS on the phone to generate a **GPX** file for the route/track and allows users to download the data. You can find more details on the **gpx** format here. The activity data in its simplest form is essentially an ordered list of $(\mathtt{longitude}, \mathtt{latitude})$ tuples. With the activity data in hand, all I wanted was a simple, elegant yet accurate mathematical model for calculating the distance between two $(\mathrm{longitude}, \mathrm{latitude})$ tuples that doesn't require anything beyond high school mathematics.Here are a few trackpoints from the GPX file:

time	latitude	longitude	elevation
2023-02-02 20:28:18+00:00	51.191533	0.270437	28.4
2023-02-02 20:28:19+00:00	51.191454	0.270578	28.3
2023-02-02 20:28:20+00:00	51.191453	0.270605	28.3
2023-02-02 20:28:21+00:00	51.191451	0.270631	28.3
2023-02-02 20:28:22+00:00	51.191450	0.270658	28.3a

Mathematical Model

Assumptions

- Assume the Earth is **flat** between two consecutive trackpoints. This is a reasonable assumption as the distance between two consecutive trackpoints(less than a couple of meters) is much smaller compared to the radius of the Earth.
- Earth is a perfect sphere even though in reality it is an oblate spheroid.
- Elevation can be ignored as the route is on fairly flat ground.

If $P(a_1, b_1)$ and $Q(a_2, b_2)$ are two consecutive track points, the key idea is to project Q onto the **tangent plane at** P, with axes parallel to the lines of latitude and longitude at P. We first set up a coordinate system (x, y) that puts P at the origin.

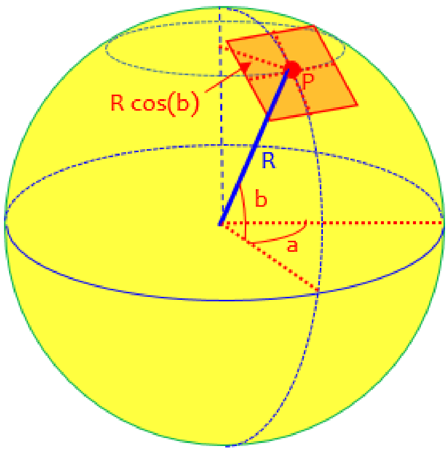


Figure 1: Tangent plane at P

For the x coordinate of Q, we can use the **the distance along a line of latitude** from one line of longitude to the other:

$$x = \frac{\pi R}{180}(a_2 - a_1)\cos(b_1)$$

Here we have an additional factor, the cosine of the latitude along which we are measuring. The line of latitude is a circle with a smaller radius than that of the equator; it is reduced by the factor $\cos(b_1)$.

For the y coordinate, we can use the north-south **distance between two lines of latitude**:

$$y = \frac{\pi R}{180}(b_2 - b_1)$$

The distance from the origin(P) to the other point $Q(x, y)$ is then given by the square root of $(x^2 + y^2)$.

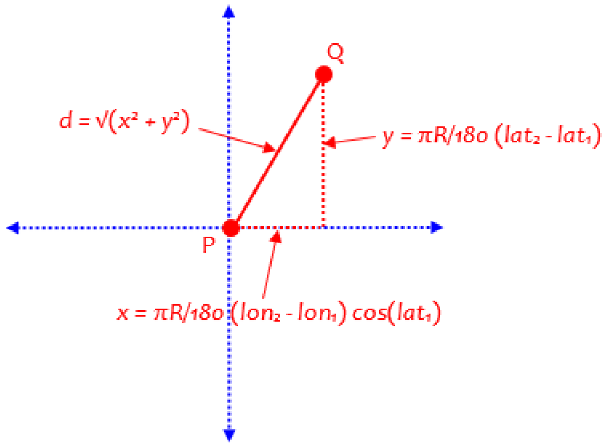


Figure 2: Coordinate system with origin at P

Python implemenation

Here is the code in Python which implements the above model.

```
def gpx_to_coords(gpx_path):
    import gpxpy
    with open(gpx_path) as f:
        gpx = gpxpy.parse(f)

    points = []
    for segment in gpx.tracks[0].segments:
        for p in segment.points:
            points.append({
                'time': p.time,
                'latitude': p.latitude,
                'longitude': p.longitude,
                'elevation': p.elevation,
            })
    return [(p["longitude"], p["latitude"]) for p in points]

def planar_distance(start, end, R=6367):
    from math import pi, cos, sqrt
    lon1, lat1, lon2, lat2 = start[0], start[1], end[0], end[1]
    x = pi*R*(lon2-lon1)*cos(lat1)/180
    y = pi*R*(lat2-lat1)/180
    return sqrt(x**2 + y**2)

def distance(coords, dist_fun):
    return sum([dist_fun(start, end) for start, end in zip(coords[:-1], coords[1:])])

print(distance(gpx_to_coords('Night_Walk.gpx'), planar_distance))
```

Using the simple model above, we get a distance of **3.574**km which is very close to the distance calculated by Strava - **3.58**km.