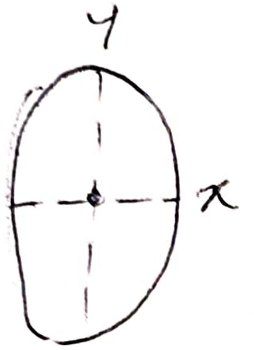
Circle_calibration.py:

→ Read file and extract mag_x & mag_y.
→ Stack raw data into (N, 2) matrix.
→ Convert T → MG (10^7)
→ Hard iron correction:
  → x = (min + max)/2
  → y = (min + max)/2
  → put (x, y) = hard_iron_offset.
  → raw_data - hard_iron_offset = hard_correcte
→ Soft iron correction:
  → Fit the ellipse using least squares
  → get correction matrix
  → create D, matrix with $[x^2, xy, y^2]$ and apply SVD to get V.
  → convert scalar to matrix form to be symmetric.
  → Find ellipse's principal axes using eigenvalue decomp. (eigh is for symmetric, much faster)
  → find semi major/semi minor axes from eigenval.
  → calculate scaling factors.
  → make diagonal matrix from above

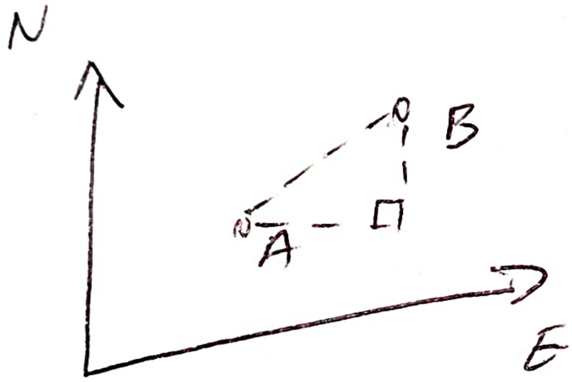
→ Build final transformation matrix.

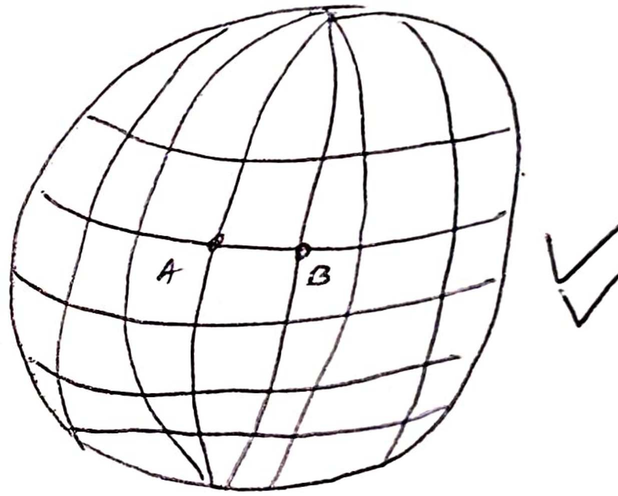
→ Return final 2x2 matrix ✓

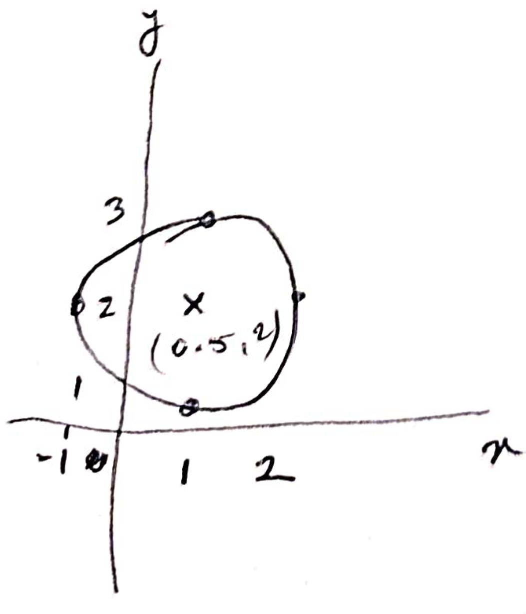
→ Apply soft iron correction using fit_ellipse().



$$\frac{x+y}{2} = rad$$







$$y\ (3+1) = 4/2 = 2$$

$$x\ (-1+2) = 1/2 = 0.5$$

yaw_angle_magnetometer.py :

→ Read driving data, extract mag_x, mag_y and timestamps for them.

→ Apply calibration in Tesla.

→ Calculate yaw angles :

* use atan2(y, x) for heading angle.
* find for raw and corrected yaw.
* atan2() handles all 4 quadrants.

→ unwrap raw & corrected yaw to prevent discontinuity. [3.14, -3.14, -3.13] → [3.14, 3.14, 3.15]

→ Normalise/zero the time t.

→ print stats & plot.

gyro-magnenometer-yaw.py

→ Load csv, read & extract mag-x, mag-y, ~~gyro~~ gyro-z and timestamps for all.

→ Find average of gyro-z and subtract from gyro-z to get ~~cor~~ bias corrected gyro bias.

→ Apply magnetometer calibration.

→ Find magnetometer yaw :

* atan2(y,x)
* unwrap()

→ Integrate gyroscope to get yaw :

* using cumulative trapezoid function

→ Align starting points

→ print stats

→ plot.

complementary_filter.py

→ Import signal module.

→ Read csv and extract mag_x, mag_y, gyro_z and timestamp t.

→ Normalise timestamp.

→ Remove gyro bias by subtracting mean from value.

→ Apply magnetometer calibration

→ Get calibrated magnetometer yaw using arctan2(y,x) and unwrap()

→ Integrate gyro to get yaw using cumulative trapezoid fn.

→ complementary filter:

* sampling freq = 1/mean(diff(timestamps))
  order = 2
  nyq = sampling freq/2
  cutoff = 0.1Hz
* LPF for mag
  HPF for gyro
* fuse both.

velocity_estimate_imu.py :

→ First, extract acc-x and time t and normalise
→ Check values of acc-x during stationary period (first 10s, and subtract this bias from acc-x to give corrected acc-x
→ Integrate acceleration (corrected) to get velocity.
→ GPS velocity :
  → Extract lat and lon, convert to radian.
  → Extract timestamps and normalise.
  → find change in lat & long. b/w consec. values.
  → find change in time b/w consec values.
  → $\frac{\Delta d}{\Delta t}$ = velocity
  → Interpolate 40Hz IMU o/p and 1 Hz GPS o/p
→ Apply LPF to GPS velocity, HPF to IMU velocity
→ Fuse both
→ print stats & plot.

dead_reckon-comparison.py :

→ Read csv and extract acc_x, acc_y, gyro_z, t.
And normalise timestamps.

→ Use fused velocity from velocity estimate script.
to get vel_fused.

→ $\ddot{y}_{predicted} = \omega \cdot \dot{x}$ [yaw rate (rad/sec) · velocity (m/s)]

→ Filter acc_y @ 1 Hz

→ plot ~~actual~~ $\ddot{y}_{predicted}$ and filtered acc_y

Northeastern University
Khoury College of Computer Sciences

NE-Mapping.py

→ Extract acc-x for velocity, mag-x, mag-y for heading, and gyro-z for heading.

→ Obtain fused/filtered velocity vel-fused.

→ Extract UTM Northing & easting.

→ Obtain fused/filtered yaw, yaw-fused.

→ Decompose velocity into North/east vectors:

* $V_e = V_0 \times \cos 45$
* $V_N = V_0 \times \sin 45$

[ex. if youre going V m/s at 45° heading]

* Do this for every timestamp

→ Use cumulative trapezoid integration to get position

→ zero the GPS Northing and easting values

→ Find initial heading alignment from GPS

→ Find initial heading alignment from IMU

→ ~~Rotate~~ ~~IMU~~ Find how many degrees rotation needed

→ Rotate IMU trajectory to match GPS trajectory.

→ plot

quaternion_data.py

→ open .mcap file
→ look for all msgs in /imu topic.
→ print to terminal all attributes of IMU message.
→ check for quaternion data
→ Extract all quaternion data
→ create dict(), add quaternion data, and append timestamps to quaternion data.
→ convert dict() to df() → dataframe and store into (N, 4) matrix. [x, y, z, w]
→ convert quaternions to rotation objects.
→ convert rotation obj to euler angles, with x, y, z.
→ save as .csv.

4_subplot_comp.py

→ Load csv from back, extract t and IMU_yaw.

→ Adjust starting point of IMU heading.

→ plot 4 subplots for comparison :

* LPF magnetometer
* HPF gyro
* Complementary filter of above
* VN_Nav internal estimate.