Penn
UNIVERSITY of PENNSYLVANIA

SINGH
Center for Nanotechnology

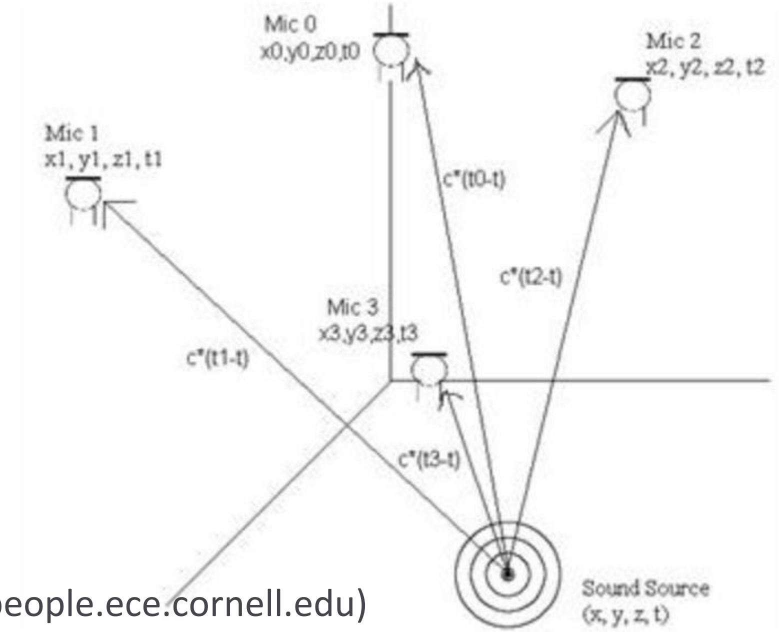
Group L
On the 3D Trilateration of Point-Source Sounds Via SPH0645LM4H-B MEMS Microphones

Clara, Dexter, Derek

Intro

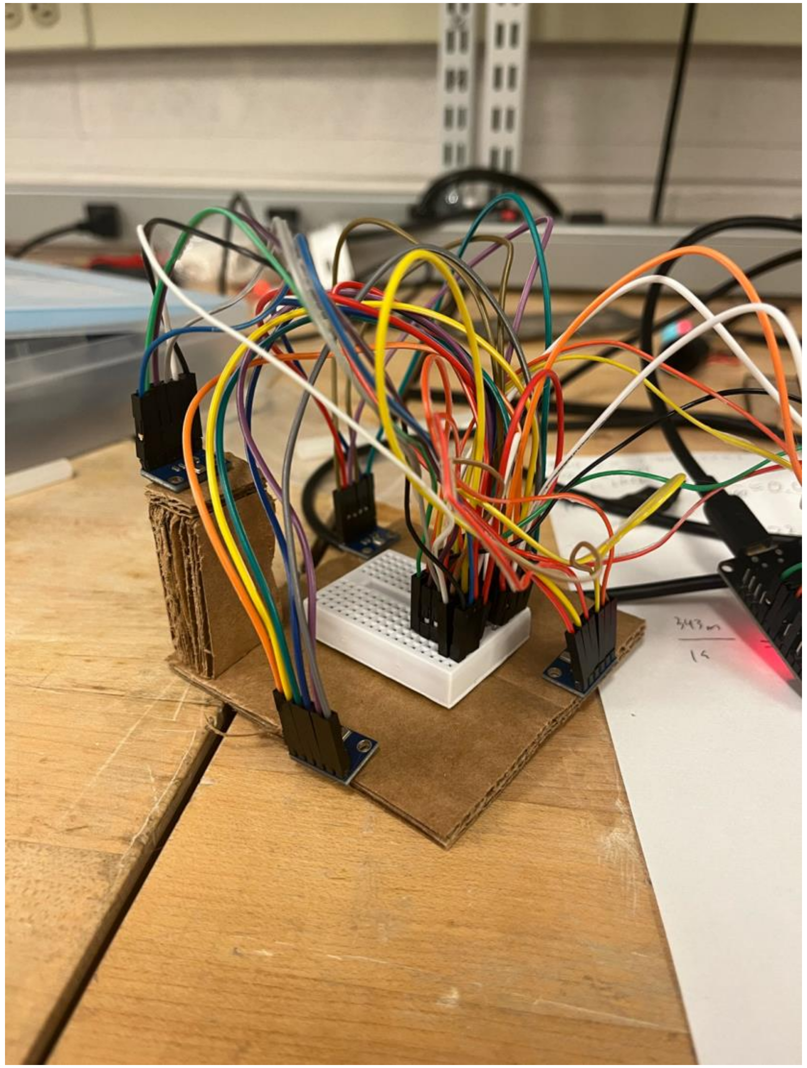
3D trilateration: triangulation but 3D

four mics in a non-degenerate pattern

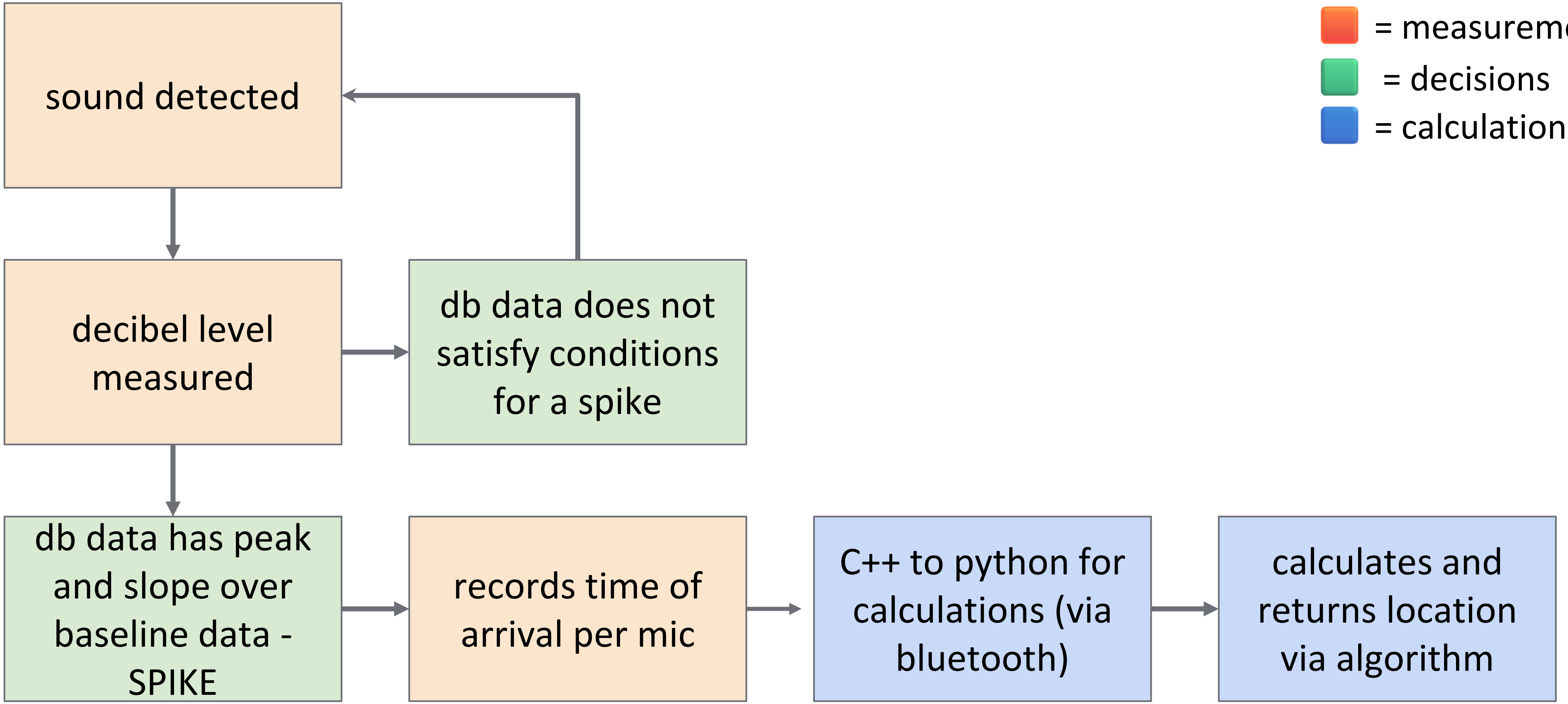


(people.ece.cornell.edu)

four mics -> three vectors -> spans 3d space



Process Flow

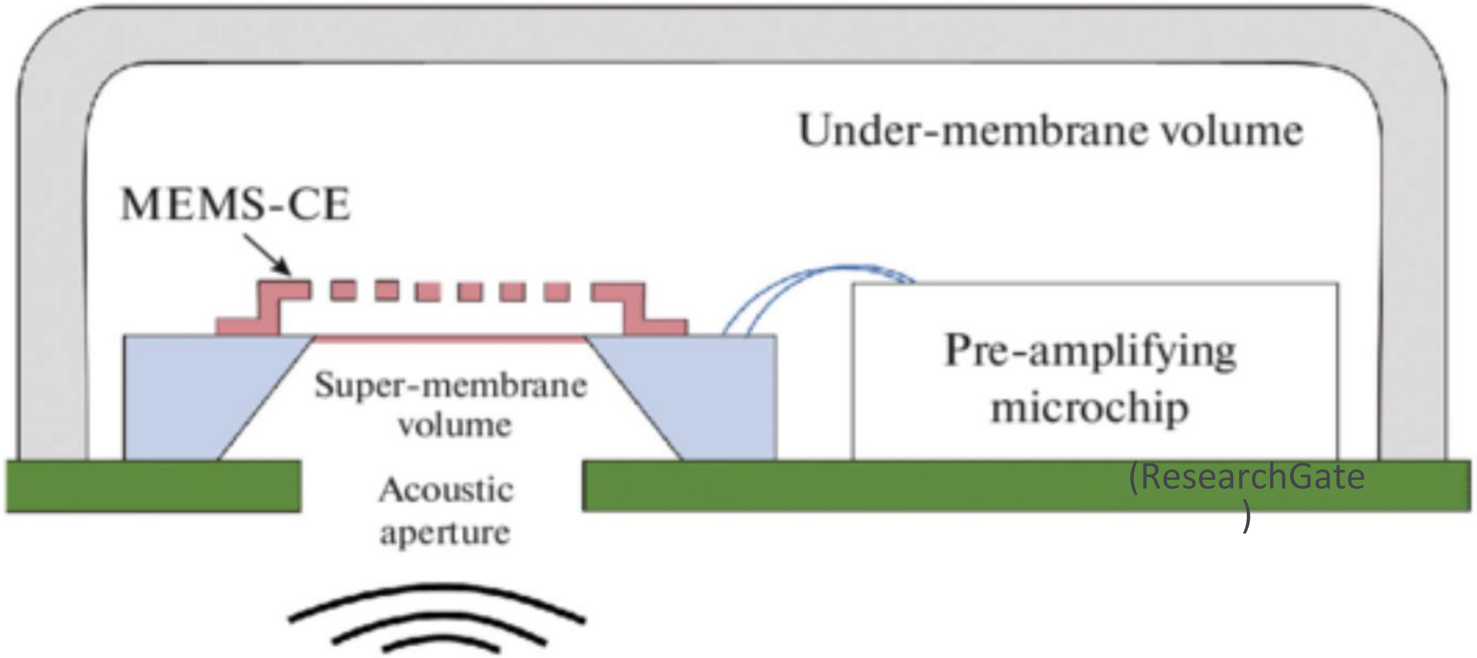


= measurements
= decisions
= calculations

MEMS Microphones

Micro-Electro-Mechanical Systems

SPH0645LM4H-B



(ResearchGate
)

Motivation

https://www.ece.ucf.edu/seniordesign/fa2009sp2010/g13/files/ATD%20NewUpdate.pdf

Size / Dimension	0.382" Dia (9.70mm)
Height (Max)	0.177" (4.50mm)

(Digikey)

device size

applications:

- automating a camera to face a speaker
- live transcription
- track moving objects using sound
    - footsteps, guns

(Seize the Day)

Steps to calculate point-source location

1. Find starting guess

2. Based on starting guess, calculate actual location

Equations Setup

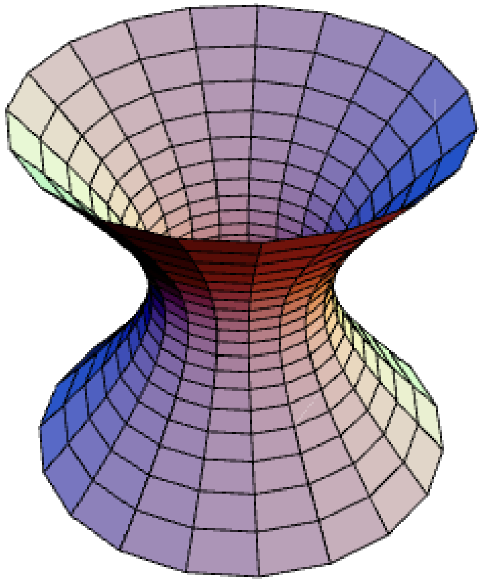
raw input: time of arrival, mic locations, temperature

calculated: time difference of arrival (TDOA), speed of sound

$$||p - r_i|| = c\,(t_i - t_{\text{emit}})$$

system of equations for source-mic distances:

$$||p - r_i|| - ||p - r_1|| = d_i \text{ for } i = 2, 3, 4$$



p = source location, r_i = mic location,

d_i = TDOA between microphones i and 1

$$d_i = c\Delta t_i$$

looking for intersection of hyperboloids

Finding our Starting Guess

$$p = y + k \cdot z$$

p = source location

y = point on line

z = direction vector

k = scalar

```
def closed_form_tdoa(rs, taus, c):
    p1, p2, p3, p4 = rs
    d = c * np.array([taus[(0, 1)], taus[(0, 2)], taus[(0, 3)]])
    A = np.vstack([p2 - p1, p3 - p1, p4 - p1])
    b = 0.5 * (np.sum(rs[1:] ** 2, axis=1) - np.sum(p1 ** 2) - d ** 2)
    #if coplanar - kinda unnecessary
    try:
        y = np.linalg.solve(A, b)
        z = np.linalg.solve(A, d)
    except np.linalg.LinAlgError:
        return np.full(3, np.nan)

    alpha = 1.0 - np.dot(z, z)
    beta  = -2.0 * np.dot(z, y - p1)
    gamma = -np.dot(y - p1, y - p1)
    disc  = beta ** 2 - 4.0 * alpha * gamma

    r1 = (-beta + np.sqrt(disc)) / (2.0 * alpha)
    return y + r1 * z
```

Finding y

$$||p - r_i|| - ||p - r_1|| = d_i \text{ for } i = 2, 3, 4$$

linearize (algebra) and take RHS of system:

$$b = \begin{pmatrix} \frac{1}{2}\left(||r_2||^2 - ||r_1||^2 - d_2^2\right) \\ \frac{1}{2}\left(||r_3||^2 - ||r_1||^2 - d_3^2\right) \\ \frac{1}{2}\left(||r_4||^2 - ||r_1||^2 - d_4^2\right) \end{pmatrix}$$

mic layout matrix:

$$A = \begin{bmatrix} \text{mic}_2 - \text{mic}_1 \\ \text{mic}_3 - \text{mic}_1 \\ \text{mic}_4 - \text{mic}_1 \end{bmatrix}$$

$$Ay = b$$

y is a point on the line of hyperboloid intersection

but where along the approximated line is p?

Finding z

$||p - r_i|| - ||p - r_1|| = d_i \text{ for } i = 2, 3, 4$, same 3x3 A

distance differences $d = \begin{bmatrix} ||p - r_1|| - ||p - r_0|| \\ ||p - r_2|| - ||p - r_0|| \\ ||p - r_3|| - ||p - r_0|| \end{bmatrix} = c \cdot \begin{bmatrix} \tau_{01} \\ \tau_{02} \\ \tau_{03} \end{bmatrix}$

linearize with taylor expansion around point y

simplified closed form: $Az = d$

Finding k

start with one equation from the system, WLOG:

$$||p - r_i|| - ||p - r_1|| = d_i \text{ for } i = 2$$

substitute $p = y + k \cdot z$

```
alpha = 1.0 - np.dot(z, z)
beta  = -2.0 * np.dot(z, y - p1)
gamma = -np.dot(y - p1, y - p1)
disc  = beta ** 2 - 4.0 * alpha * gamma

r1 = (-beta + np.sqrt(disc)) / (2.0 * alpha)
return y + r1 * z
```

quadratic in terms of *k* -> quadratic formula!

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Steps to calculate point-source location

1. Find starting guess

2. Based on starting guess, calculate actual location

Residuals and Fitting

Non-linear least squares method

- Levenberg–Marquardt algorithm
  - gradient descent
  - Gauss-Newton method

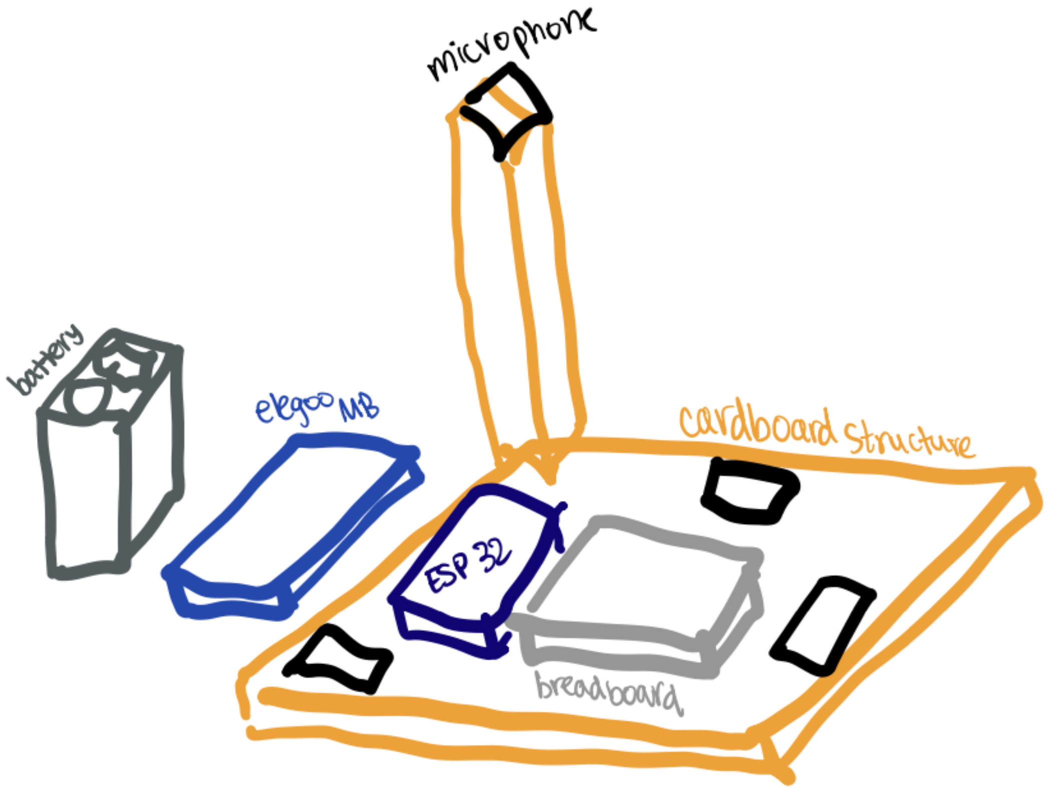
Jacobian for algo speed

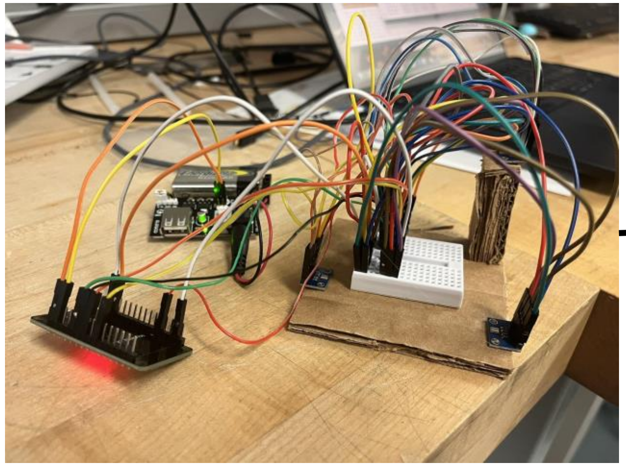
```
def residuals(p):
    D = np.linalg.norm(p - rs, axis=1)
    return w * np.array([(D[j] - D[i]) - c * taus[(i, j)]
                          for (i, j) in pairs])

def jacobian(p):
    D = np.linalg.norm(p - rs, axis=1)
    J = np.zeros((len(pairs), 3))
    for k, (i, j) in enumerate(pairs):
        J[k] = w * ((p - rs[j]) / D[j] - (p - rs[i]) / D[i])
    return J
```

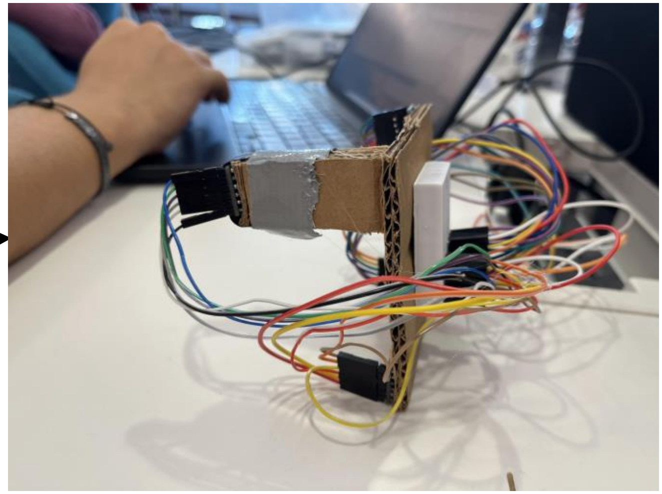
```
p0 = closed_form_tdoa(rs, taus, c)
if np.any(np.isnan(p0)):
    p0 = np.mean(rs, axis=0)
```

```
res = least_squares(residuals, p0, jac=jacobian,
                    method='lm',
                    ftol=1e-12, xtol=1e-12, gtol=1e-12,
                    max_nfev=20000)

pos_est = res.x
```

Physical Set-up

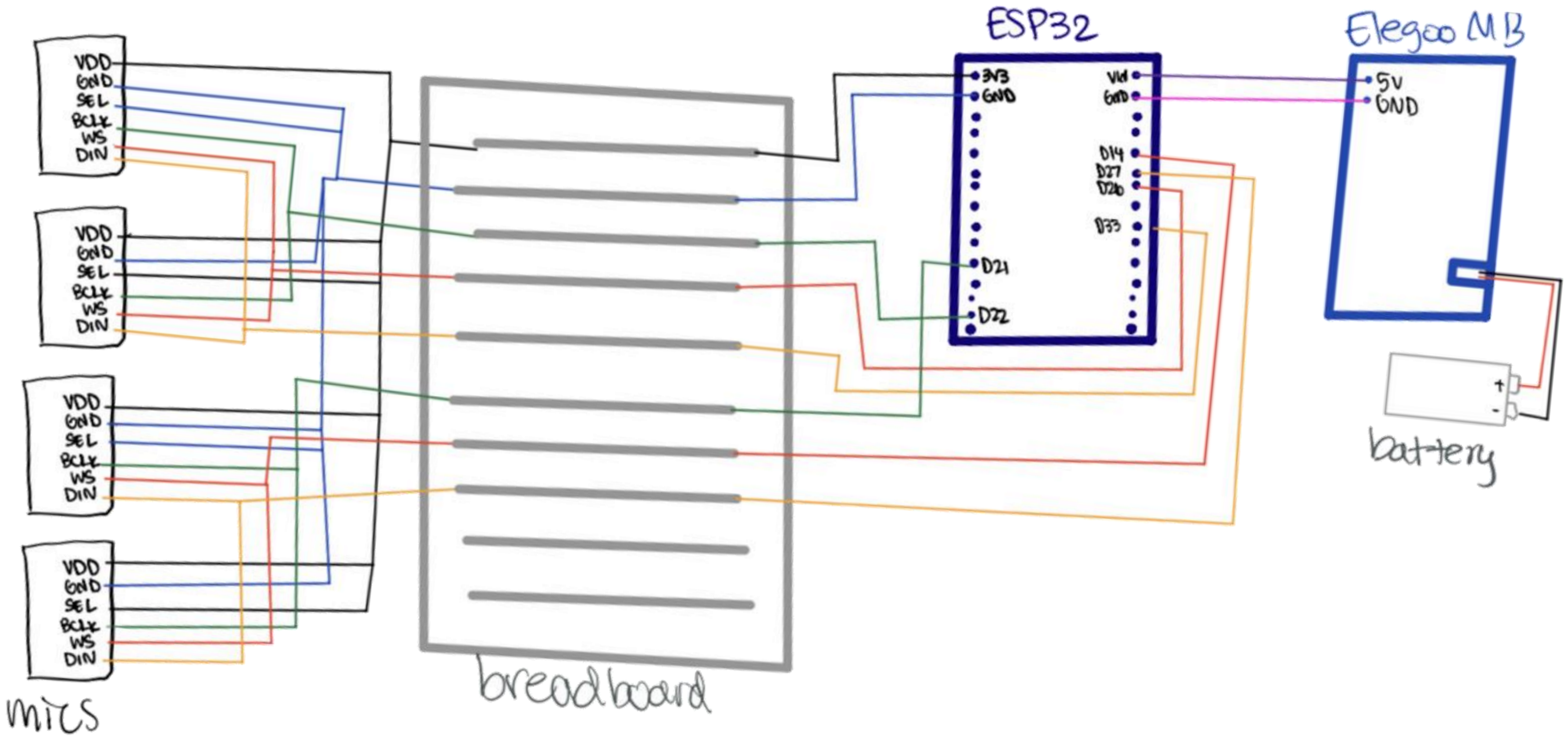






Moving wires around

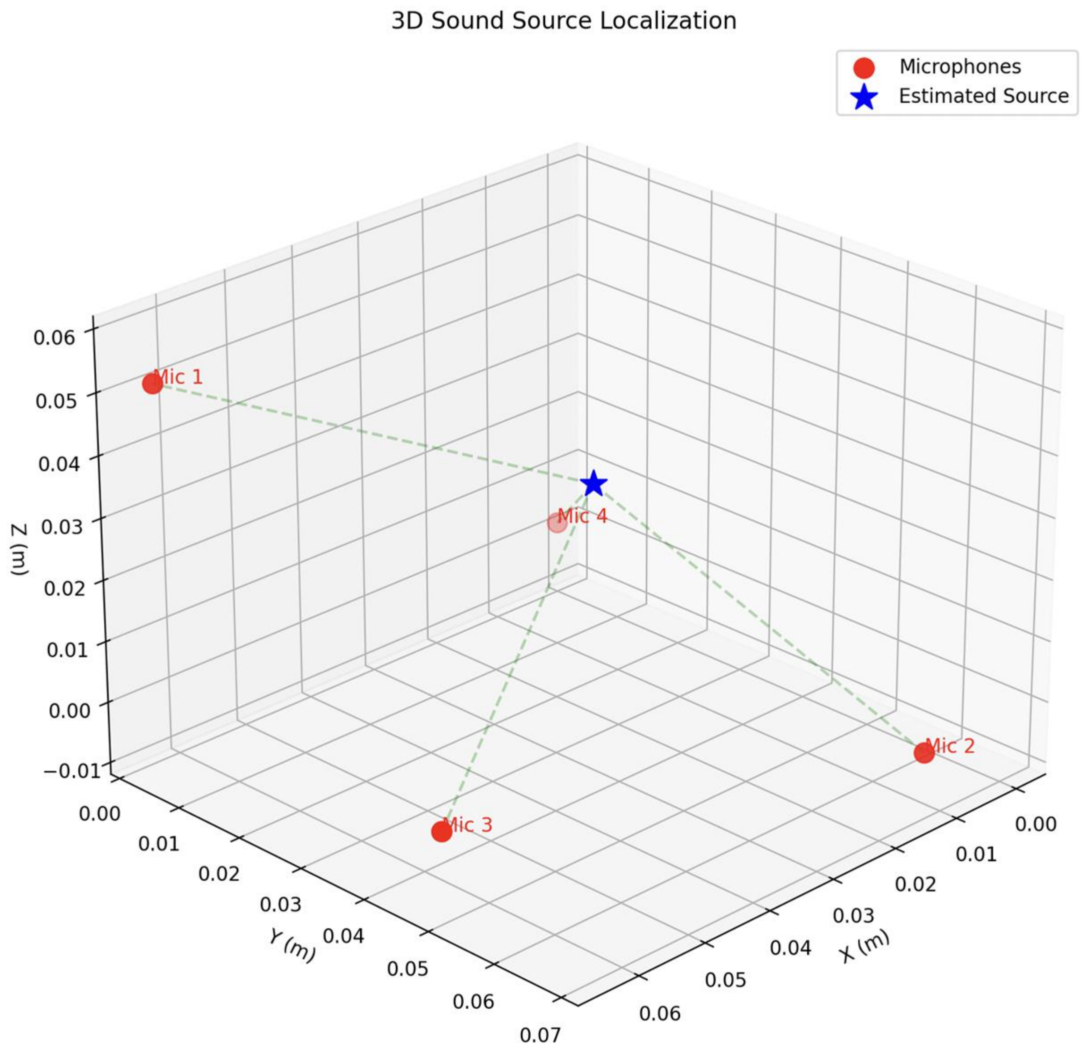
Wiring Diagram



Example of data

```
[11:57:20.001267] MIC4   -7.8 dBFS
[11:57:20.001402] MIC3  -12.8 dBFS
[11:57:20.001484] MIC1  -11.2 dBFS
[11:57:20.001587] MIC2  -11.0 dBFS
```

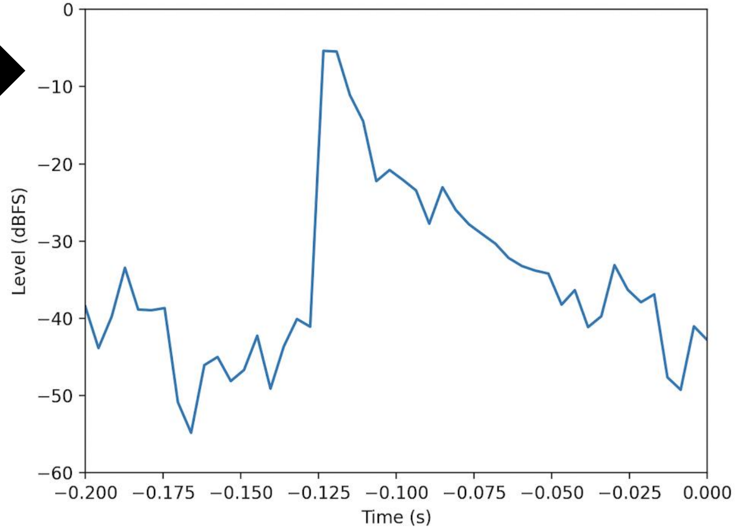
3D Graph with Sound Source



Demo!

Next Steps

- decreasing size further + precision measuring
- accounting for obstacles
- better signal processing - fourier transforms?
- better peak detection
- higher sampling rate
- **testing**
- noise filtering - AI?



(Actuate

Thank you!

Questions?