3D Printed Force-Torque Sensor (FTS) Calibration

Jon Urcelay and Aleksander Skrede

NTNU

Kunnskap for en bedre verden

Index

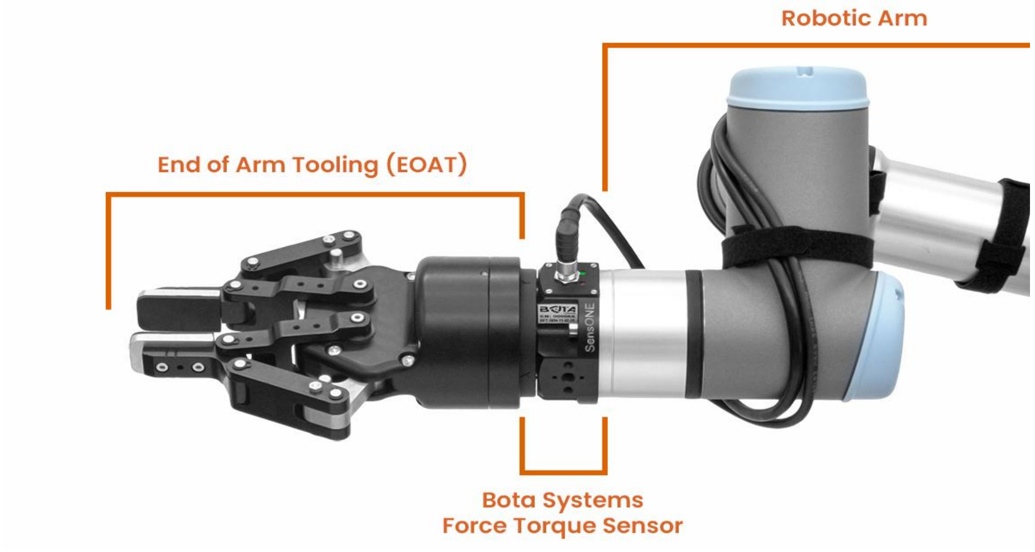
1. What is a FTS?
2. 3D Printed FTS: Fork-sensors, arduino nano, and 3D printed parts.
3. Initial setup of this 3D Printed FTS.
4. Calibration theory.
5. How to find C, L and Q?
    a) Collect S and W.
    b) Compute C, L and Q with OLS, Ridge, and LASSO.
6. Validation.
7. Real time use/inference.

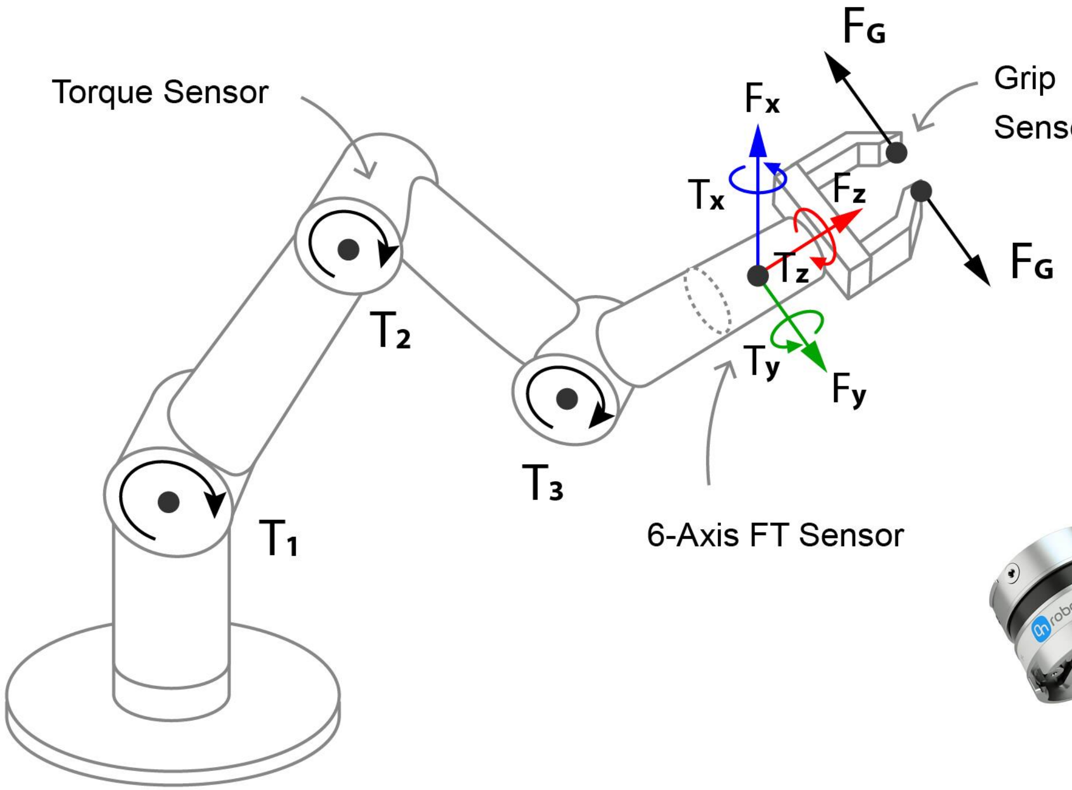
What is a 6 Axis Force-Torque Sensor (FTS)?

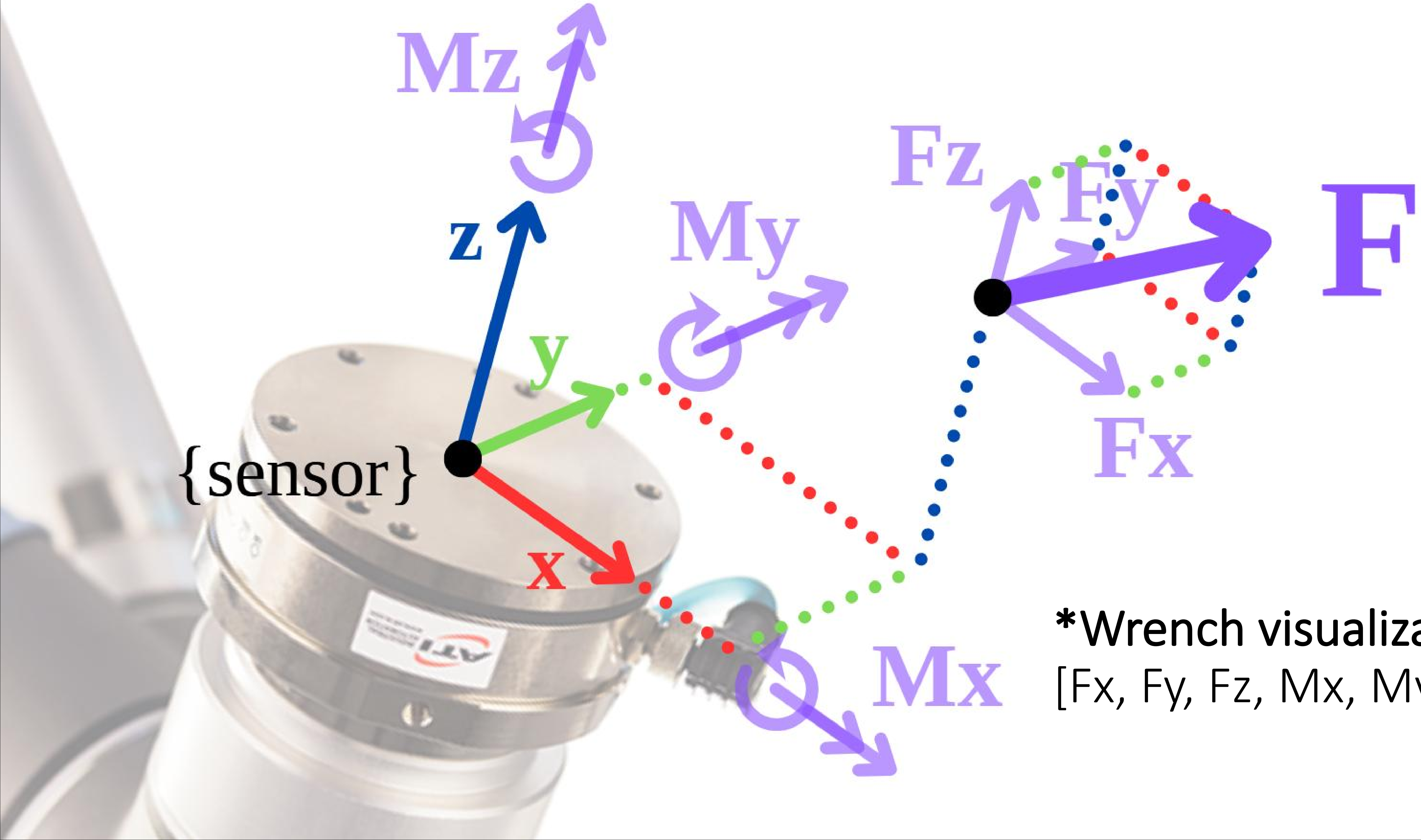
- Device to measure Force and Torque/Moment in 6 axis:

  Wrench = [Fx, Fy, Fz, Mx, My, Mz]

- Used in robotic arms







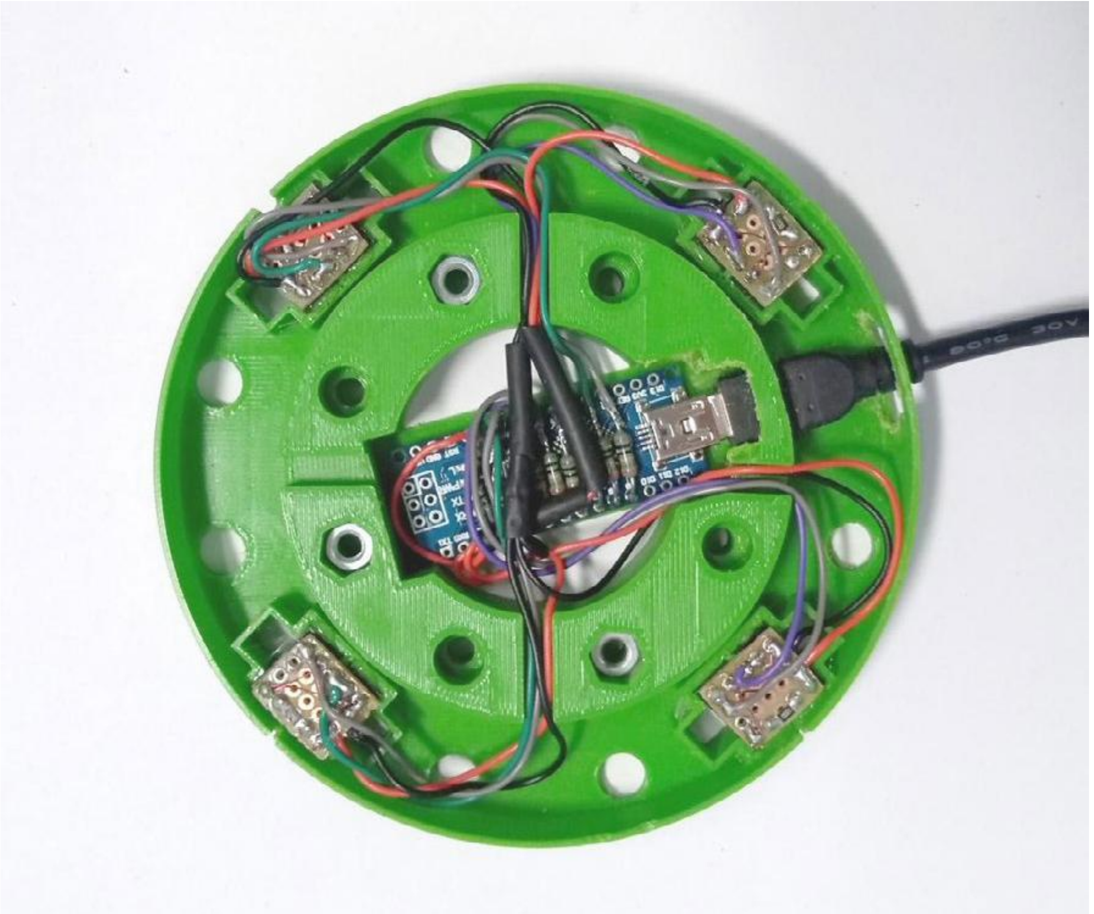
***Wrench visualization**

[Fx, Fy, Fz, Mx, My, Mz]

3D Printed 6-Axis FTS

Components:

- x8 Fork-type sensors
- Arduino Nano Vishay 1103
- 3D Printed parts

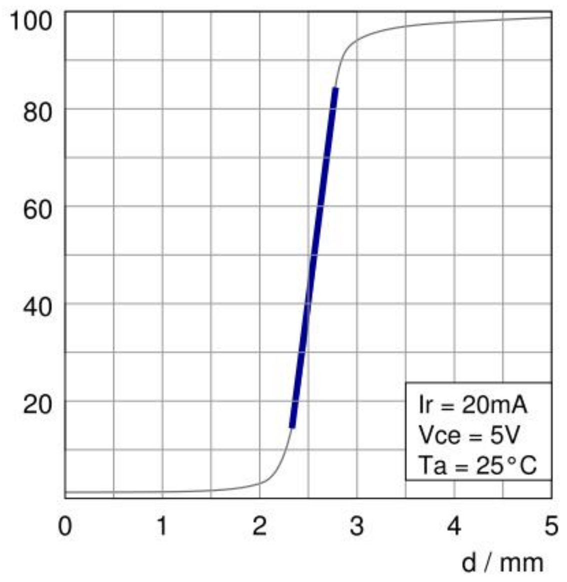


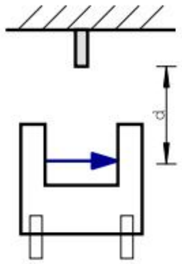
Original paper: [3D Printed Low-Cost Force-Torque Sensors | IEEE Journals & Magazine | IEEE Xplore]

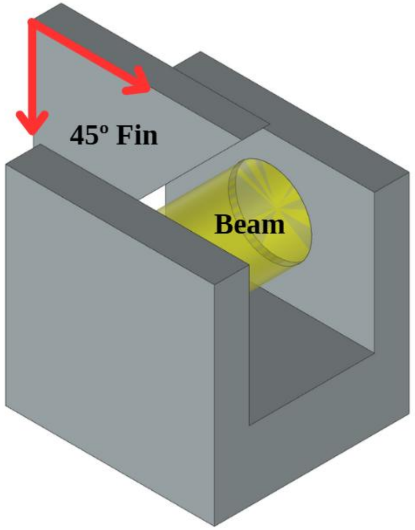
Original code: [TAMS-Group/tams_printed_ft: Low-cost 3D printed force-torque sensors, including ROS drivers, CAD designs, and Arduino firmware.]

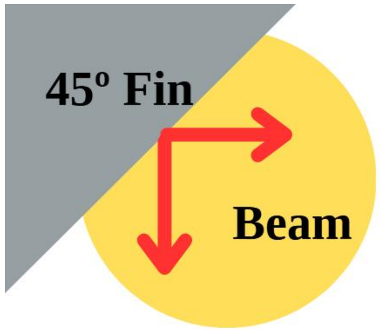
Fork-type sensors

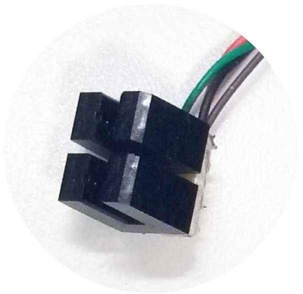
- Emiter: infrared LED
- Receiver: phototransistor
- Light beam is blocked when an opaque object (fin) moves into the slit, changing the receiver's current
- 45º fin: measurement in two directions = 2 Degrees Of Freedom (DOF)









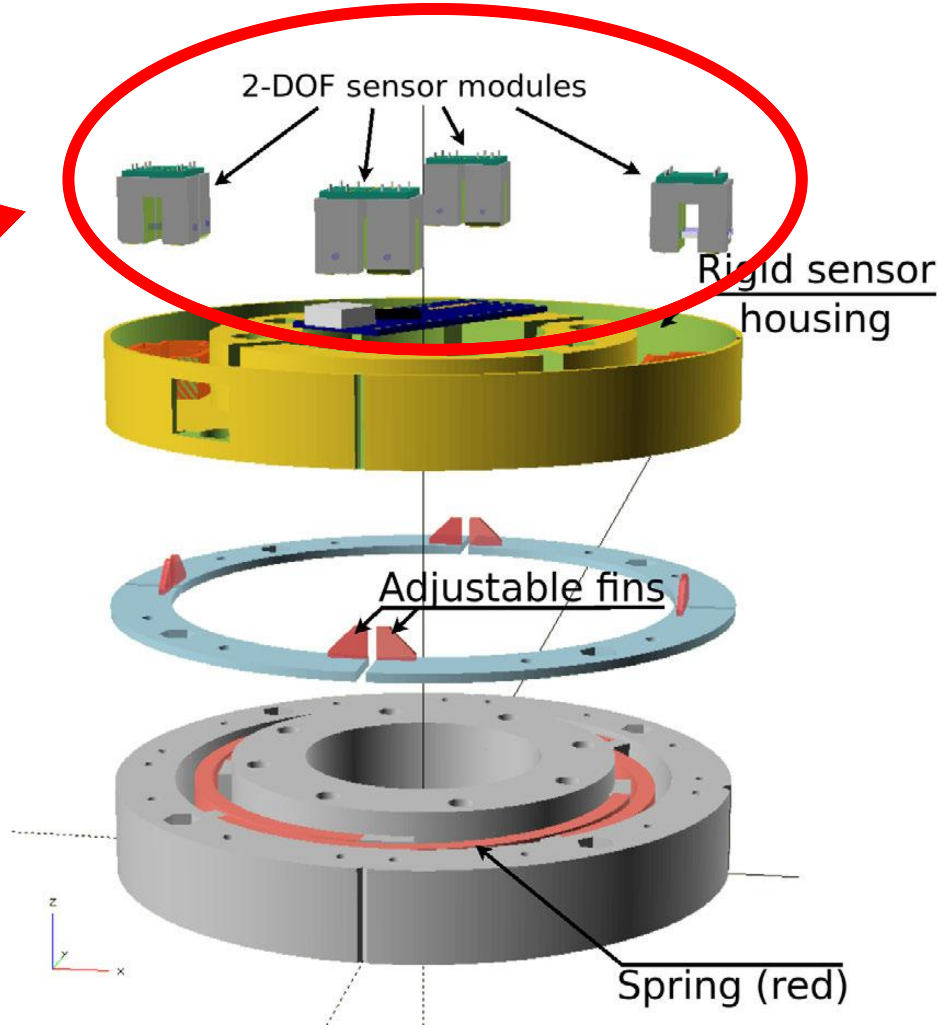


= 1 module

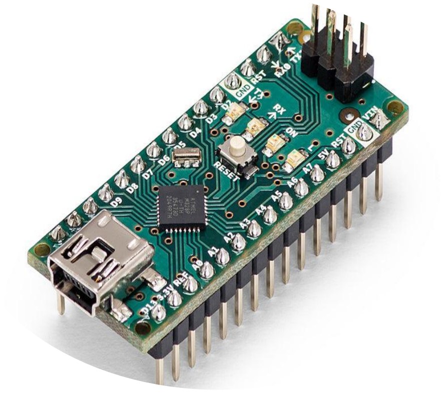
x4 modules with 2 sensors each =

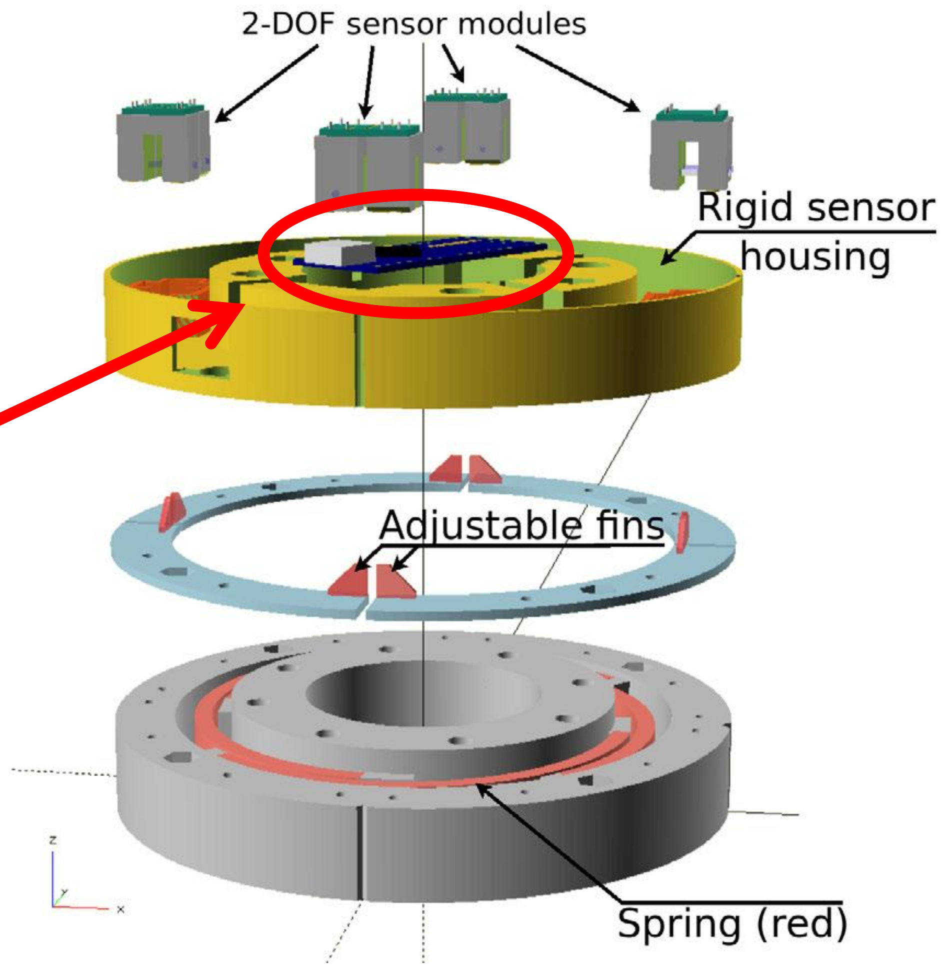
x8 Fork-type sensors

2 DOF each sensor



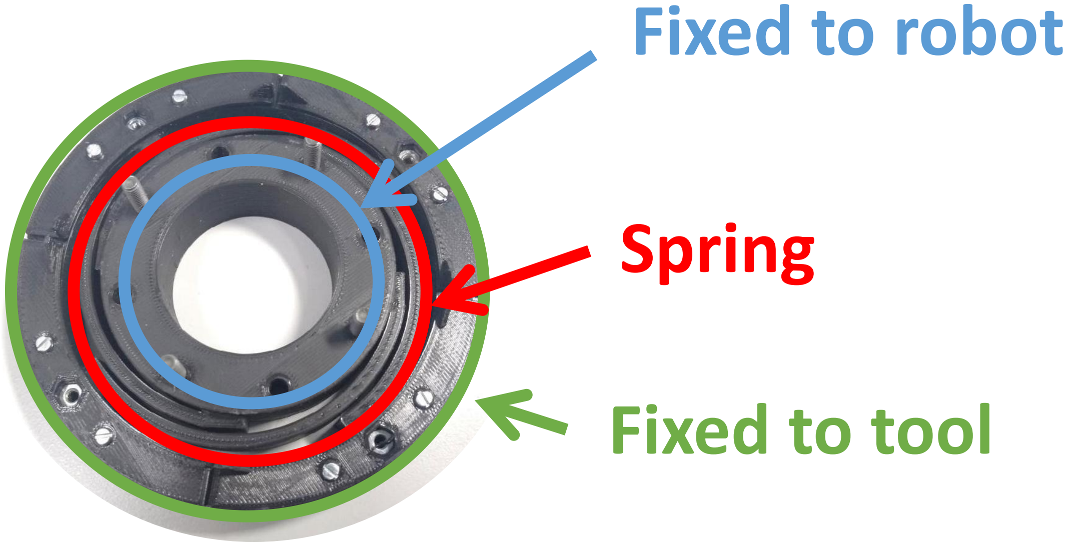
Arduino Nano Vishay 1103

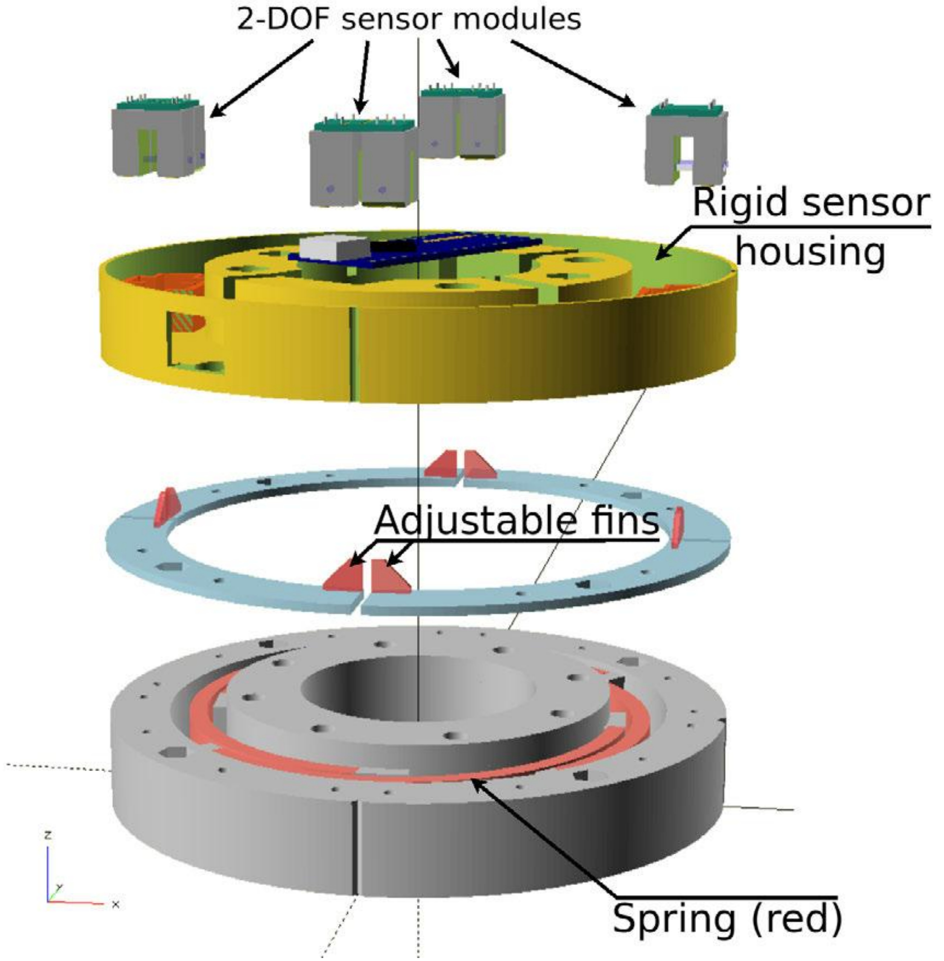




3D Printed parts

- Rigid sensor housing
- Adjustable 45º fins
- Spring part



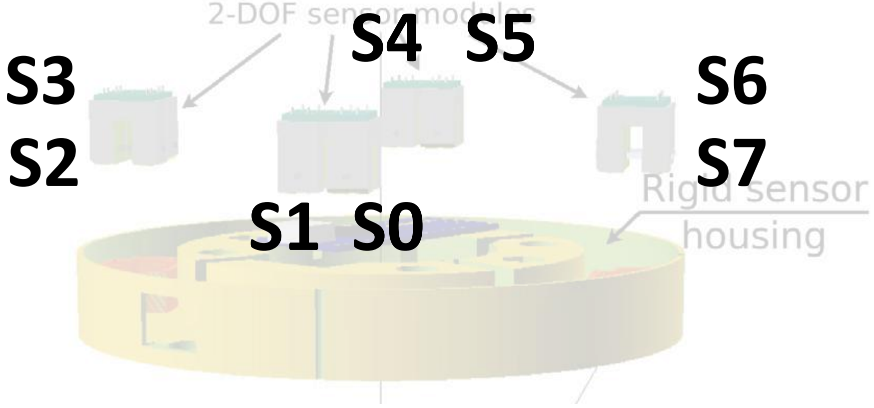


Setup from PC

- Upload “.ino” code to Arduino Nano using Arduino IDE

[tams_printed_ft/firmware/ft-big-octo-1103/ft-big-octo-1103.ino at master · TAMS-Group/tams_printed_ft]

- Read **raw sensor values** from the serial port:
  use “serial” in PyCharm



$$S = \begin{bmatrix} S0 \\ S1 \\ S2 \\ S3 \\ S4 \\ S5 \\ S6 \\ S7 \end{bmatrix}$$

```python
import serial
port = 'COM3'
baudrate = 115200
ser = serial.Serial(port, baudrate, parity=serial.PARITY_NONE)
data = ser.readline()
(str_D, seq_number, error_mask, s0, s1, s2, s3, s4, s5, s6, s7) = \
    [t(s) for t, s in zip((str, int, int, int, int, int, int, int, int, int, int), data.split())]
s = [s0, s1, s2, s3, s4, s5, s6, s7]
```

Calibration's objective: find C, L and Q

$$\boxed{W = C + LS + QS^2}$$

$$\underset{(6\text{x}1)}{\begin{bmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{bmatrix}} = \underset{(6\text{x}1)}{\begin{bmatrix} C_{Fx} \\ C_{Fy} \\ C_{Fz} \\ C_{Mx} \\ C_{My} \\ C_{Mz} \end{bmatrix}} + \underset{(6\text{x}8)}{\begin{bmatrix} L_{F_x 0} & L_{F_x 1} & L_{F_x 2} & L_{F_x 3} & L_{F_x 4} & L_{F_x 5} & L_{F_x 6} & L_{F_x 7} \\ L_{F_y 0} & L_{F_y 1} & L_{F_y 2} & L_{F_y 3} & L_{F_y 4} & L_{F_y 5} & L_{F_y 6} & L_{F_y 7} \\ L_{F_z 0} & L_{F_z 1} & L_{F_z 2} & L_{F_z 3} & L_{F_z 4} & L_{F_z 5} & L_{F_z 6} & L_{F_z 7} \\ L_{M_x 0} & L_{M_x 1} & L_{M_x 2} & L_{M_x 3} & L_{M_x 4} & L_{M_x 5} & L_{M_x 6} & L_{M_x 7} \\ L_{M_y 0} & L_{M_y 1} & L_{M_y 2} & L_{M_y 3} & L_{M_y 4} & L_{M_y 5} & L_{M_y 6} & L_{M_y 7} \\ L_{M_z 0} & L_{M_z 1} & L_{M_z 2} & L_{M_z 3} & L_{M_z 4} & L_{M_z 5} & L_{M_z 6} & L_{M_z 7} \end{bmatrix}} \underset{(8\text{x}1)}{\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix}}$$

Calibration's objective: find C, L and Q

$$\boxed{W = C + LS + QS^2}$$

$$+\begin{bmatrix} Q_{F_x00} & Q_{F_x01} & Q_{F_x02} & Q_{F_x03} & Q_{F_x04} & Q_{F_x05} & Q_{F_x06} & Q_{F_x07} & Q_{F_x11} & Q_{F_x12} & \cdots & Q_{F_x77} \\ Q_{F_y00} & Q_{F_y01} & Q_{F_y02} & Q_{F_y03} & Q_{F_y04} & Q_{F_y05} & Q_{F_y06} & Q_{F_y07} & Q_{F_y11} & Q_{F_y12} & \cdots & Q_{F_y77} \\ Q_{F_z00} & Q_{F_z01} & Q_{F_z02} & Q_{F_z03} & Q_{F_z04} & Q_{F_z05} & Q_{F_z06} & Q_{F_z07} & Q_{F_z11} & Q_{F_z12} & \cdots & Q_{F_z77} \\ Q_{M_x00} & Q_{M_x01} & Q_{M_x02} & Q_{M_x03} & Q_{M_x04} & Q_{M_x05} & Q_{M_x06} & Q_{M_x07} & Q_{M_x11} & Q_{M_x12} & \cdots & Q_{M_x77} \\ Q_{M_y00} & Q_{M_y01} & Q_{M_y02} & Q_{M_y03} & Q_{M_y04} & Q_{M_y05} & Q_{M_y06} & Q_{M_y07} & Q_{M_y11} & Q_{M_y12} & \cdots & Q_{M_y77} \\ Q_{M_z00} & Q_{M_z01} & Q_{M_z02} & Q_{M_z03} & Q_{M_z04} & Q_{M_z05} & Q_{M_z06} & Q_{M_z07} & Q_{M_z11} & Q_{M_z12} & \cdots & Q_{M_z77} \end{bmatrix}_{(6\text{x}36)} \begin{bmatrix} s_0^2 \\ s_0s_1 \\ s_0s_2 \\ s_0s_3 \\ s_0s_4 \\ s_0s_5 \\ s_0s_6 \\ s_0s_7 \\ s_1^2 \\ s_1s_2 \\ s_1s_3 \\ s_1s_4 \\ s_1s_5 \\ s_1s_6 \\ s_1s_7 \\ s_2^2 \\ s_2s_3 \\ s_2s_4 \\ s_2s_5 \\ s_2s_6 \\ s_2s_7 \\ s_3^2 \\ s_3s_4 \\ s_3s_5 \\ s_3s_6 \\ s_3s_7 \\ s_4^2 \\ s_4s_5 \\ s_4s_6 \\ s_4s_7 \\ s_5^2 \\ s_5s_6 \\ s_5s_7 \\ s_6^2 \\ s_6s_7 \\ s_7^2 \end{bmatrix}_{(36\text{x}1)}$$

How to find C, L and Q?

1. **Collect data** of known **W** (W_{ref}) and its corresponding **S**
2. **Fit** the **linear regression model**. $\boxed{W = C + LS + QS^2}$

- Find a solution for the **Ordinary Least Squares (OLS)** method:
  - Linear coefficients only: C and L, ignore Q
  - Combining linear and quadratic coefficients: C, L and Q

$$\min_{C,L,Q} \sum_{i \in Cal} \left(W_{\text{ref,i}} - W_{\text{est,i}} \right)^2 + \lambda \cdot ||C, L, Q||$$

1 Data collection method: Theory

Compute Wrench ($W_{s,est}$) with:

- Known **weight (m)** and **COG's position (r_s)**
- Known **orientations (RPY)** using UR3e

$$F_w = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \quad r_s = \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}_{COG}$$

$$R_{ws} = R_z(\psi) R_y(\theta) R_x(\phi)$$

*COG = Center Of Gravity
*RPY = Roll (ϕ), Pitch (θ), Yaw (ψ)

Knowns:

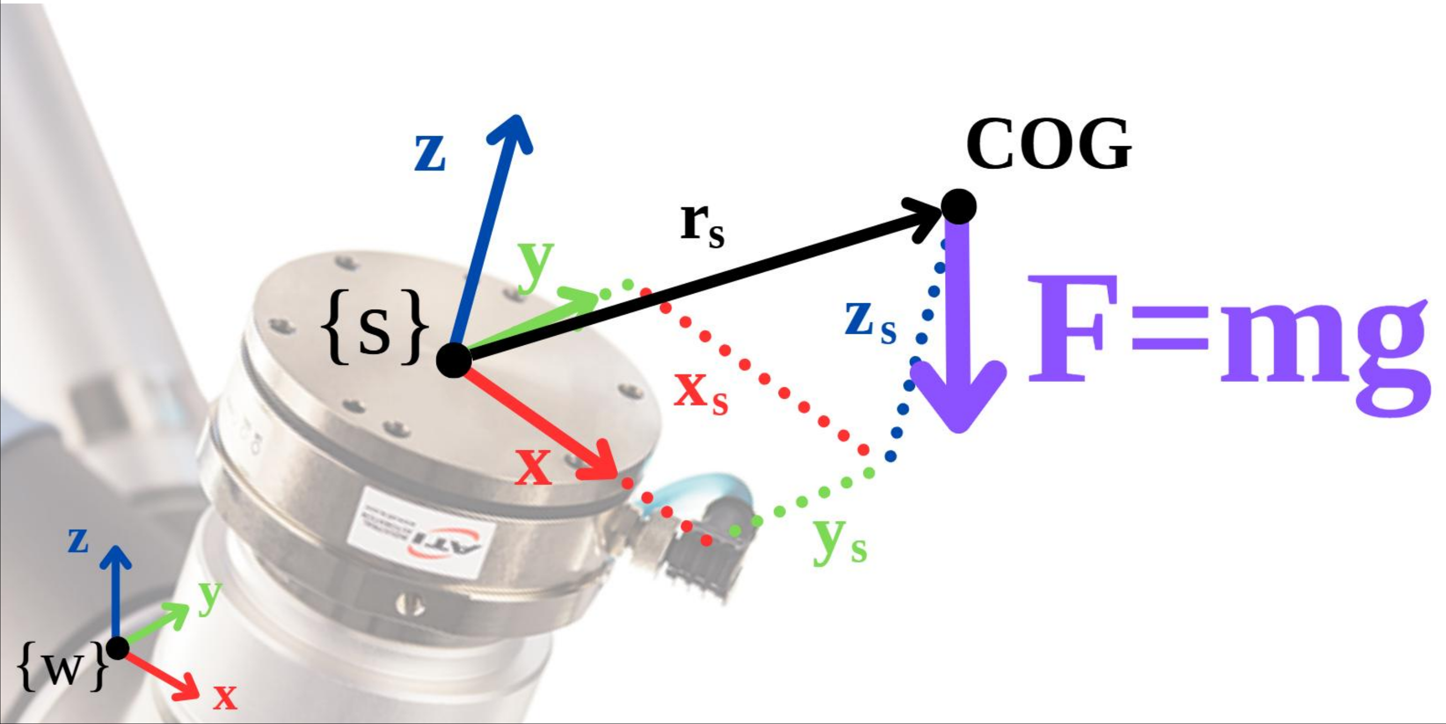
F_w, r_s, R_{ws}

$$F_s = R_{sw} F_w = R_{ws}^{-1} F_w$$

$$M_s = r_s \times F_s$$

$$\boxed{W_{s,est} = \begin{bmatrix} F_s \\ M_s \end{bmatrix}_{(6\times1)}}$$

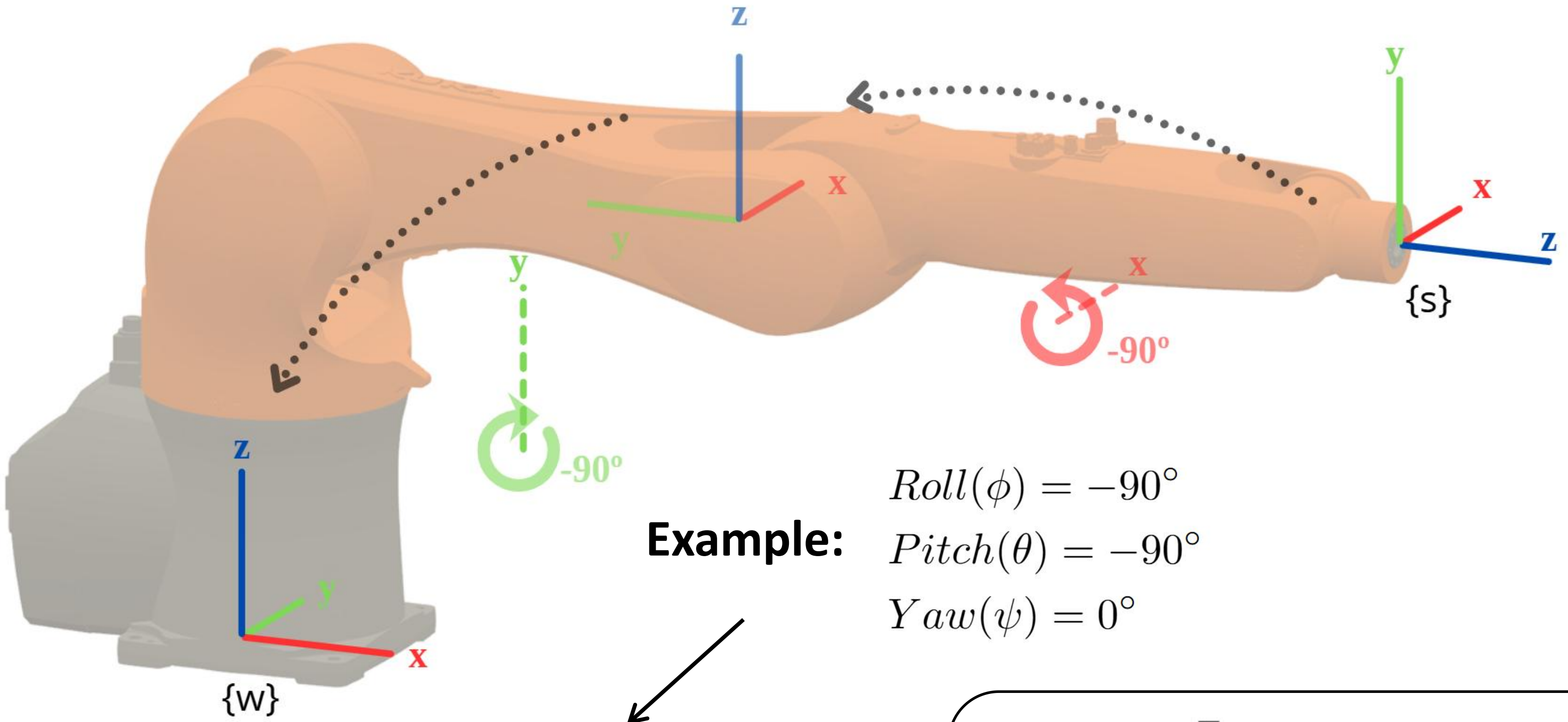
Knowns: $F_w = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$ $r_s = \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}_{COG}$



Knowns:

Rotation matrix from **{sensor} to {world}** frame

$$R_{ws} = R_z(\psi)R_y(\theta)R_x(\phi)$$



$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

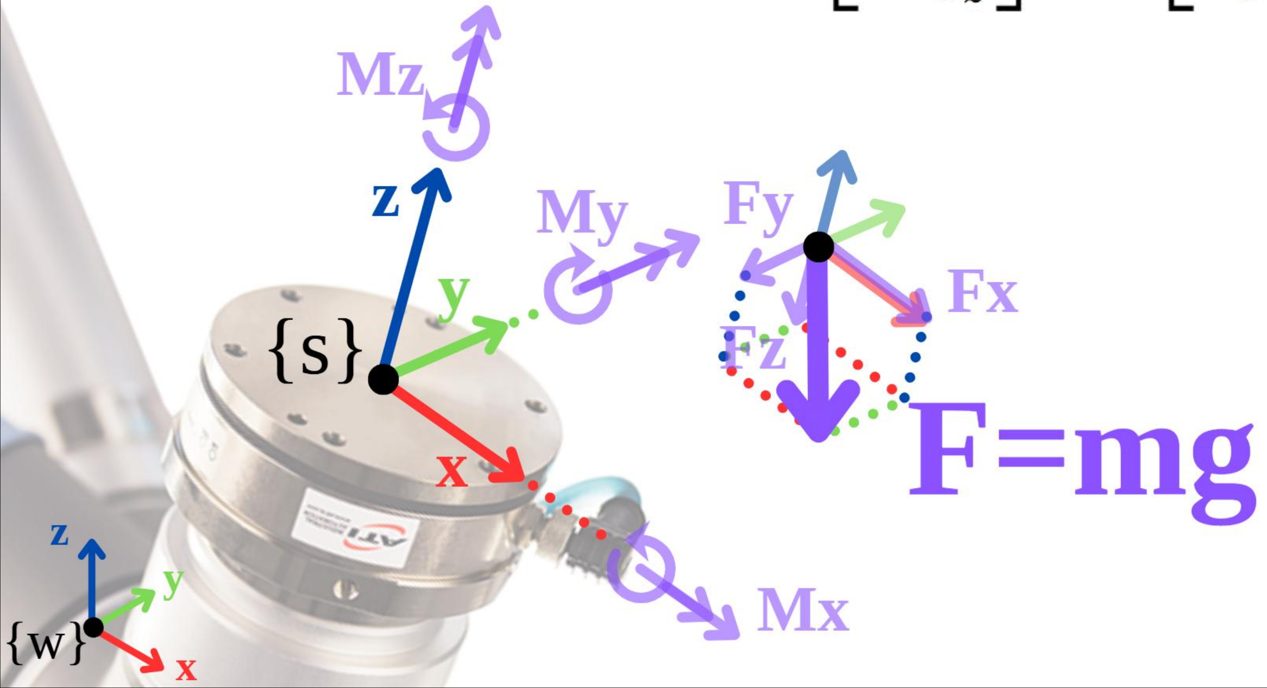
Example:

$$Roll(\phi) = -90^\circ$$
$$Pitch(\theta) = -90^\circ$$
$$Yaw(\psi) = 0^\circ$$

$$R_{ws} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} x_{w\{s\}} & y_{w\{s\}} & z_{w\{s\}} \end{bmatrix}$$

$$\boxed{R_{ws} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}}$$

$$\begin{cases} F_s = R_{ws}^{-1} F_w \\ M_s = r_s \times F_s \end{cases} \qquad \begin{cases} \begin{bmatrix} F_{s_x} \\ F_{s_y} \\ F_{s_z} \end{bmatrix} = R_{ws}^{-1} \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \\ \begin{bmatrix} M_{s_x} \\ M_{s_y} \\ M_{s_z} \end{bmatrix} = \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}_{COG} \times \begin{bmatrix} F_{s_x} \\ F_{s_y} \\ F_{s_z} \end{bmatrix} \end{cases}$$



$$\boxed{W_{s,est} = \begin{bmatrix} F_{s_x} \\ F_{s_y} \\ F_{s_z} \\ M_{s_x} \\ M_{s_y} \\ M_{s_z} \end{bmatrix}}$$

1 Data collection method: Hardware

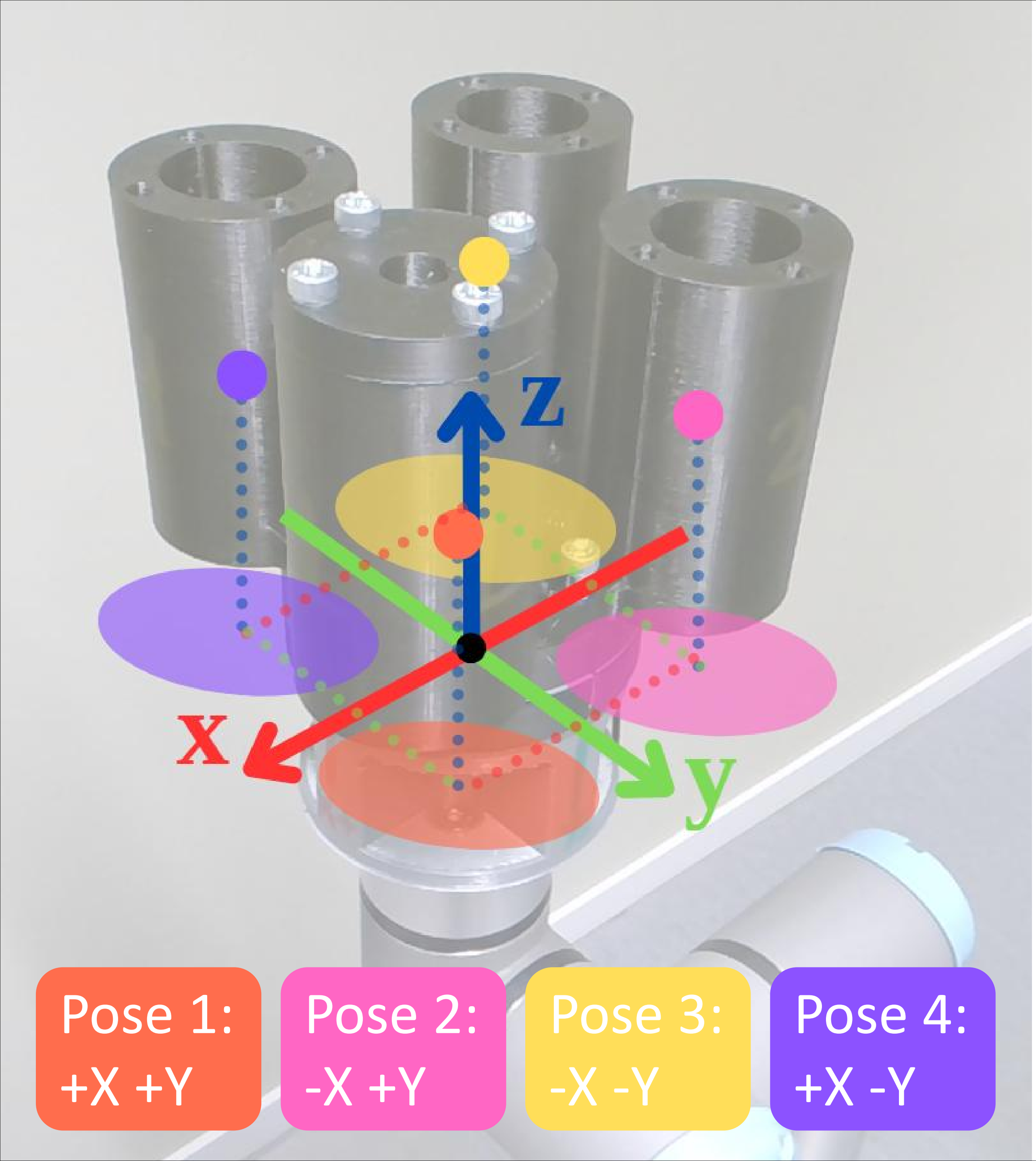
3D Printed FTS
assembled on UR3e

+

Jig to apply known
weight to FTS
(4 possible poses)

=

FTS with jig in UR3e
(weight in pose 1)



Values for **m** & $\mathbf{r}_s$

*Different r_s per pose

$\mathbf{r}_s$ = COG Coordinates in {S}

Object	Mass (kg)	X_s(m)	Y_s(m)	Z_s(m)
Jig	0.309	0	0	0.045
Cylinder	0.679	±0.042	±0.042	0.055
Top	0.040	±0.042	±0.042	0.105
TOTAL	**1.028**	**±0.029**	**±0.029**	**0.054**

$$\sum_{i=1}^{n} m_i \mathbf{r}_{COG} = \sum_{i=1}^{n} m_i \mathbf{r}_i$$

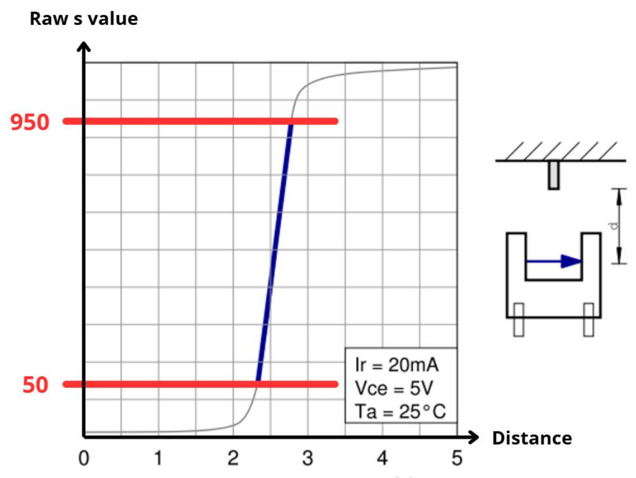
1 Data collection method: Code

- Full code in GitHub: [jonurce/FTS_Calibration: Code for calibrating a 3D Printed Force-Torque Sensor]
- For each timestamp, safe raw sensor values and estimated wrench in a CSV file in the next format:

  [Timestamp, Fx, Fy, Fz, Mx, My, Mz, s0, s1, s2, s3, s4, s5, s6, s7]

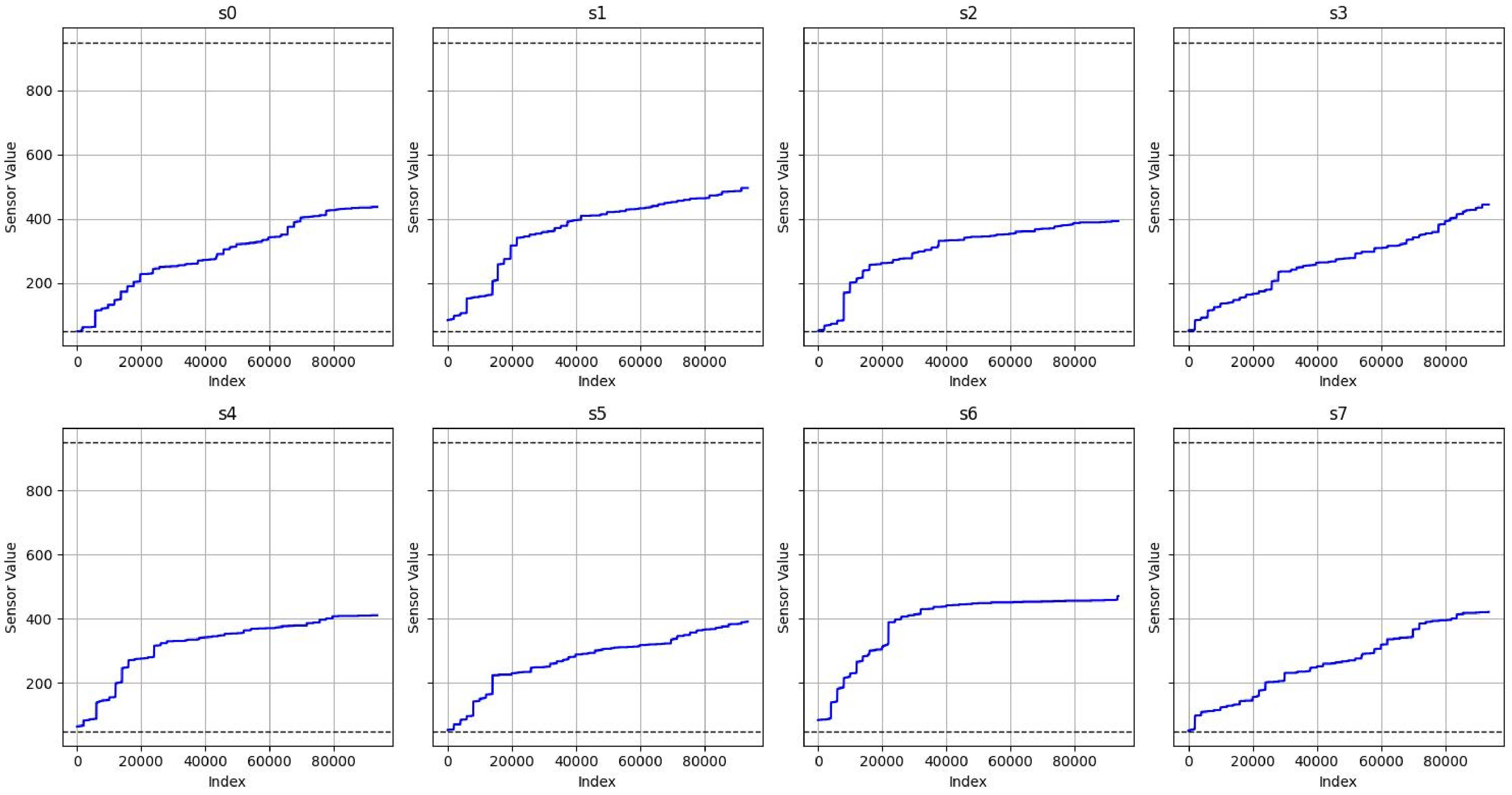
```
# Create CSV
filename = f'Datasets/11_final_extra/data/data_{pos}_R{roll}_P{pitch}_Y{yaw}.csv'
csvfile = open(filename, 'a', newline='')
fieldnames = ['Timestamp', 'Fx', 'Fy', 'Fz', 'Mx', 'My', 'Mz',
              's0', 's1', 's2', 's3', 's4', 's5', 's6', 's7']
writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

1 Collected data



- **x5** different mass **positions**:
  - 0: Empty jig, no cylinder and no top.
  - 1, 2, 3, 4: Positions for cylinder in the jig (with top).
- For each position, **x12** different **axis poses**:
  - x6 orthogonal, with gravity acting in: +X, -X, +Y, -Y, +Z, -Z.
  - x6 random, with gravity acting on different combinations of X, Y, Z.
- For each pose: 2.000 datapoints -> 5 x 12 x 2.000 = 120.000 datapoints
- Removing datapoints with raw sensor values out of range:

93.585 total datapoints



s0
s1
s2
s3
s4
s5
s6
s7
Sensor Value
Index
800
600
400
200
0
20000
40000
60000
80000

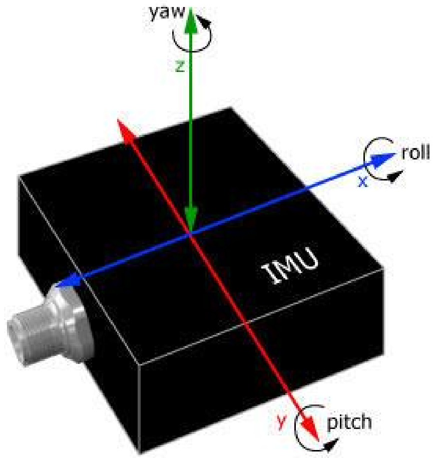
1 Other ways of collecting data [W and S]

[Timestamp, Fx, Fy, Fz, Mx, My, Mz, s0, s1, s2, s3, s4, s5, s6, s7]

- Using another calibrated sensor:
  - Calibrated sensor measures Wrench -> Save W
  - Sensor to be calibrated measures Raw Sensor Values -> Save S

- With IMU and known mass
  - IMU measures accelerations -> Estimate position and orientation of sensor
    + Knowing the mass -> Estimate and save W
  - Sensor to be calibrated measures Raw Sensor Values -> Save S

- Multiple ways in your imagination



2 Fit the linear regression model: Theory

- Linear regression: statistical method used to model the relationship between a dependent variable (W) and an independent variable (S) by fitting a linear equation to the observed data.
  - Observed data: collected dataset with W and S datapoints
  - Linear equation with quadratic values ->
  - “Model relationship” -> Find C, L and Q

$$\boxed{W = C + LS + QS^2}$$

- Use Ordinary Least Squares (OLS) to find best fitting C, L and Q.

- Regularization using Ridge Regression or LASSO.

Ordinary Least Squares explained

Stop when RSS is small / not changing more

Iterate with next

Training datapoint: $S, \ W_{ref}$

*RSS = Residual Sum of Squares

Start with random coefficients

C, L, Q Coefficients

$$\beta_j = \{C_k, L_{kl}, Q_{kq}\}$$

$$W_{est} = C + LS + QS^2$$

$$\boxed{RSS} = \sum_{k=1}^{6} (W_{ref,k} - W_{est,k})^2$$

$k \in (1 \ldots 6)$

$l \in (1 \ldots 8)$

$q \in (1 \ldots 36)$

$6 + 6 \times 8 + 6 \times 36 = 270$

$j \in (1 \ldots 270)$

$$\frac{\partial RSS}{\partial \beta_j} = -2 \sum_{k=1}^{6} (W_{ref,k} - W_{est,k}) \cdot \frac{\partial W_{est,k}}{\partial \beta_j}$$

- For $\beta_j = C_k$

$$\frac{\partial W_{est,k}}{\partial C_k} = 1$$

- For $\beta_j = L_{kl}$

$$\frac{\partial W_{est,k}}{\partial L_{kl}} = S_l$$

- For $\beta_j = Q_{kq}$

$$\frac{\partial W_{est,k}}{\partial Q_{kq}} = (S^2)_q$$

New coefficients

$$\beta_{j,next} = \beta_j - \frac{\partial RSS}{\partial \beta_j} \beta_j$$

Regularization techniques

- One of the differences with OLS is replacing RSS with:
  - Ridge: $$\sum_{k=1}^{6}(W_{ref,k} - W_{est,k})^2 + \lambda\sum_{j=1}^{270}\beta_j^2$$
  - LASSO (Least Absolute Shrinkage and Selection Operator):

$$\sum_{k=1}^{6}(W_{ref,k} - W_{est,k})^2 + \lambda\sum_{j=1}^{270}|\beta_j|$$

- More details: [Ridge Regression vs Lasso Regression - GeeksforGeeks]

2 Fit the linear regression model: code

- Full code in GitHub: [jonurce/FTS_Calibration: Code for calibrating a 3D Printed Force-Torque Sensor]
- Very easy importing sklearn:

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

```python
# Load dataset
directory = 'Datasets/12_final_extra_bounded'
df = pd.read_csv(f'{directory}/train/train_data.csv')
# Features (S: 8x1) and targets (W: 6x1)
S = df[['s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7']].values  # Shape: (datapoints, 8)
W = df[['Fx', 'Fy', 'Fz', 'Mx', 'My', 'Mz']].values  # Shape: (datapoints, 6)
```

```python
# Create pipeline with quadratic terms
poly = PolynomialFeatures(degree=2, include_bias=False)  # Linear + quadratic terms
estimator = 'lasso' #estimator = 'ridge' #estimator = 'linearregression'
if estimator == 'linearregression':
    model = make_pipeline( *steps: poly, LinearRegression())
elif estimator == 'ridge':
    model = make_pipeline( *steps: poly, Ridge())
elif estimator == 'lasso':
    model = make_pipeline( *steps: poly, Lasso())
else:
    print('Invalid estimator')
model.fit(S, W)

# Extract coefficients
L = model.named_steps[estimator].coef_[:, :8]  # Linear terms (6x8)
Q = model.named_steps[estimator].coef_[:, 8:]  # Quadratic terms (6x36 for 8 sensors)
C = model.named_steps[estimator].intercept_  # Bias (6x1)
```

2 Fit the linear regression model: Dataset

- Collected dataset: 93.585 total datapoints
  - Random 80% of the datapoints -> Used for training
  - Rest 20% of the datapoints -> Used for validation later

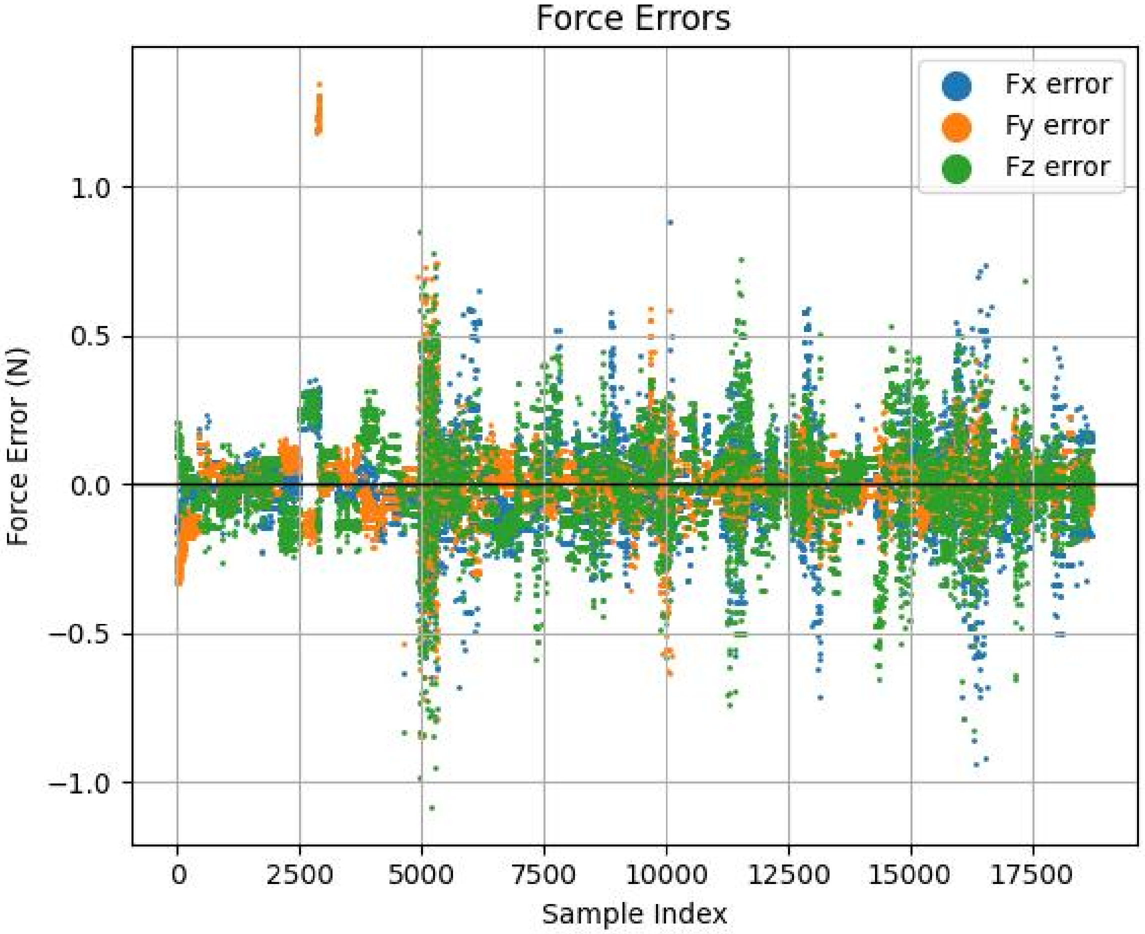
- Once the linear regression is completed C, L and Q coefficients are stored in a CSV file (or similar), so they can be used for validation and for inference (if validation is good).

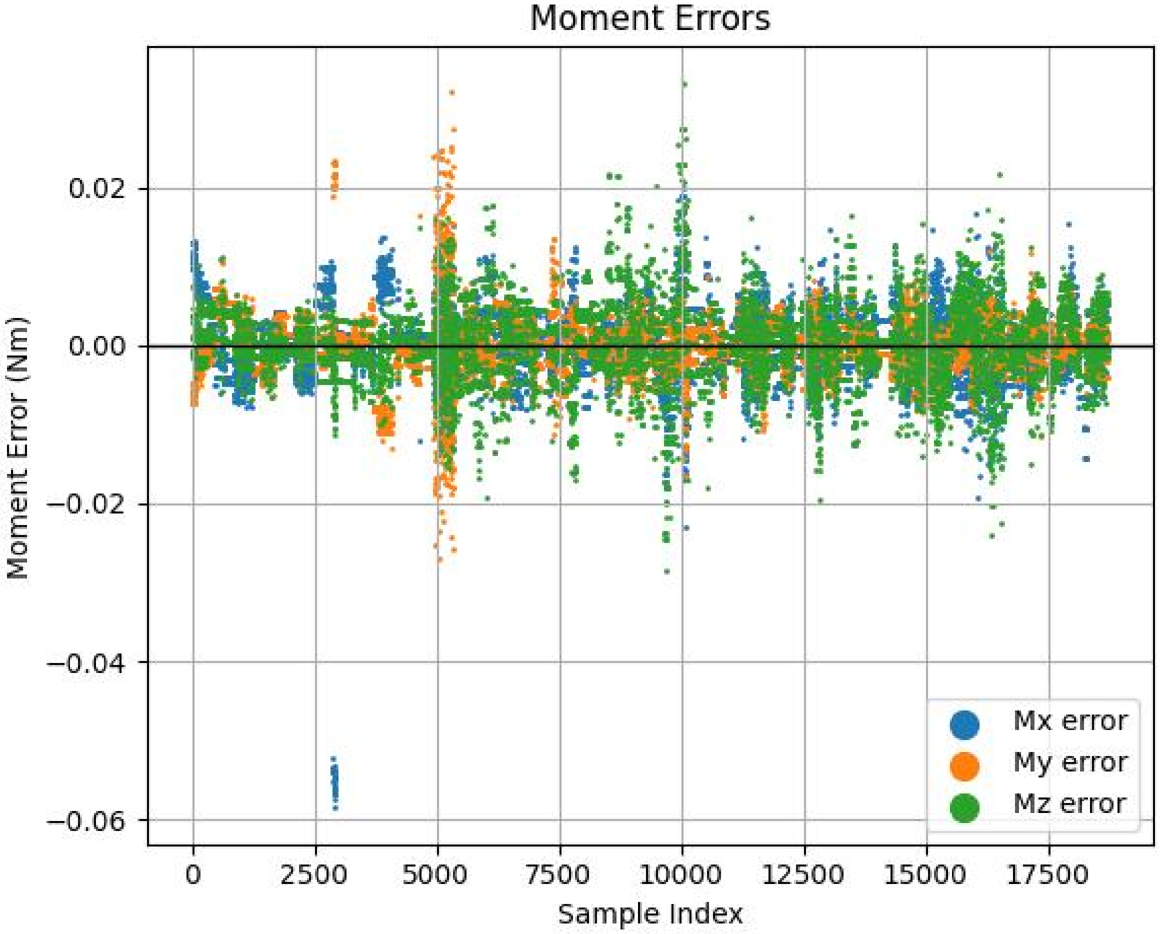
Validation

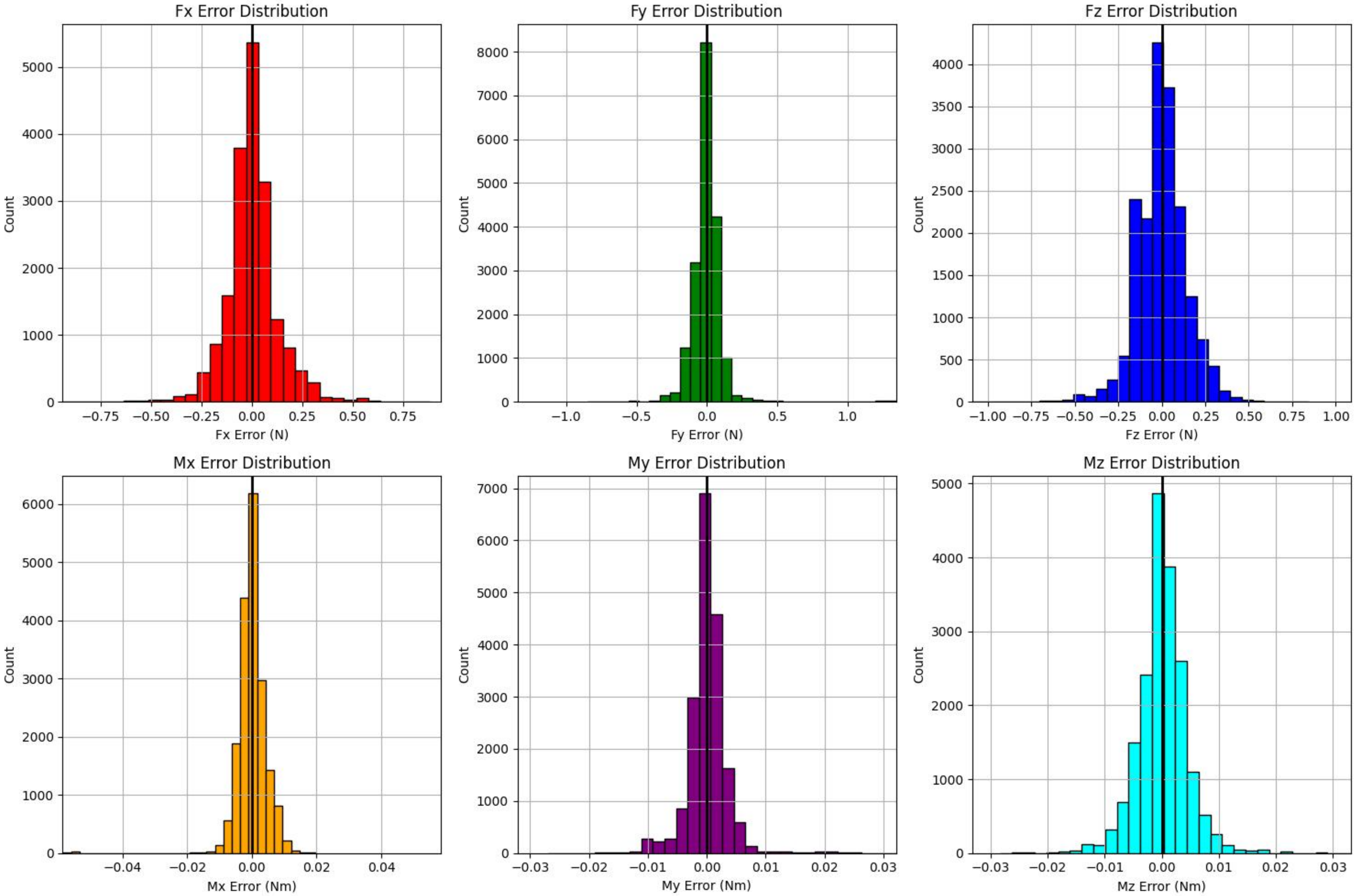
- Using the validation dataset:
  - Use raw sensor values (S) from data + calibrated coefficient (C, L, Q) to compute: $W_{est} = C + LS + QS^2$
  - Collect W_{ref} from data

- Compute the Error = W_{ref} - W_{est}

Validation results:






Fx Error Distribution
Count
Fx Error (N)
Fy Error Distribution
Count
Fy Error (N)
Fz Error Distribution
Count
Fz Error (N)
Mx Error Distribution
Count
Mx Error (Nm)
My Error Distribution
Count
My Error (Nm)
Mz Error Distribution
Count
Mz Error (Nm)

Real time use / inference

- Once C, L and Q have been computed and validation has been succesfull, they can be used for inference.
- Raw sensor values (S) collected at each timestand and introduced in the formula to obtain thw wrench W in real time:

$$W_{est} = C + LS + QS^2$$

Full code in GitHub: [jonurce/FTS_Calibration: Code for calibrating a 3D Printed Force-Torque Sensor]