Data Driven Actuator Selection

Aryan Dua, Atharv Dabli, Hetal Priyadarshi, Tushar Gujral

April 2022

Nomenclature

- **SVM:** Support Vector Machine
- **NN:** Neural Network
- **RBF:** Radial basis Function
- **GANN:** Generative Adversarial Neural Network

Contents

1 Introduction

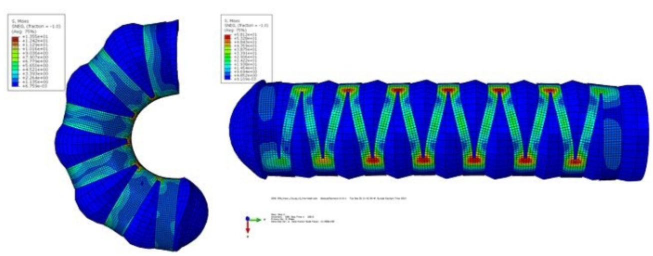
1.1 Conventional Robots and need of artificial muscles:

Conventional robots are based on rigid materials for actuation and load bearing. They certainly were not designed to be low cost, safe near people and adaptable to unpredictable challenges. Hence, human-robot and robot–environment interactions must be monitored and precisely controlled to avoid safety hazards to humans or the robot.

We've made good progress with robots' brains, but their bodies are still primitive. In contrast to conventional robots, human bodies extensively use soft and deform-able materials such as muscle and skin. We need a new generation of robot bodies inspired by the elegance, efficiency, and soft materials of the designs found in nature. Soft robotics is a new field of research in this idea. Biological muscles have evolved to become what we see today - they can contract fast enough to power the high-speed wings of birds, are strong enough to move elephants and are highly versatile as in the arms of an octopus.

1.2 What are actuators:

Artificial muscle actuators are for robots what biological muscles are for animals. There are wide variety of materials and configurations, which includes shape-memory alloys (SMAs), dielectric elastomers (DEAs), super-coiled polymers (SCPs), piezoelectric actuators (PZTs), and soft fluidic actuators (SFAs). Though they belong to a unifying class of artificial muscle actuators, these biomimetic actuators have historically been considered dissimilar and distinct technologies as they vary widely in terms of material, configuration, and scale; this induces significant challenges for robot designers to overcome. For one, when presented with the broad class of artificial muscle actuators, it is unclear to the typical robot designer how to select or even compare muscle actuators for use in their specific application.



Soft Pneumatic Actuator | Soft Robotics Toolkit

We have the following parameters to compare the actuator that should be used according to designer's need:

- **Bandwidth:** The range of frequencies that the actuator can be excited continuously.
- **Efficiency:** The ratio of output work over input energy (the input energy can be in the form of electricity, heat, radiation, etc.).
- **Power Density:** The energy (work) density normalized to the actuation period.
- **Strain:** deformation required for the application
- **Stress:** force required for the task

1.3 Support Vector Machines

SVM is a machine learning algorithm used both for classification and regression. In this case, we are using it for multi label classification. The objective of the SVM is to find a hyper-plane in N-dimensional space that distinctly classifies data points. What this does to achieve this is that it finds a hyper-plane with the maximum marginal distance from both the support vectors. The support vectors are the closest points to the hyper-plane on either side of it.

2 Problem Statement

Given the task parameters for a certain task, we have to select the optimal soft actuator. This is governed by the parameters listed above.

2.1 Dataset

A list of actuators suitable for certain tasks with given parameters.

Actuator Type	Bandwidth	Strain	Stress	Efficiency	Power Density
PZT	10000.0	0.2	110.0	90.0	0.2
PZT	100.0	NaN	NaN	90.0	NaN
DEA	100.0	200.0	0.8	85.0	0.2
IPMC	10.0	40.0	0.3	NaN	0.02
SMA	3.0	5.0	200.0	1.3	50.0

There is a sparse data problem at hand and needs to be dealt with. We tried using Generative models for data generation and data normalization {log normalization} for better learning.

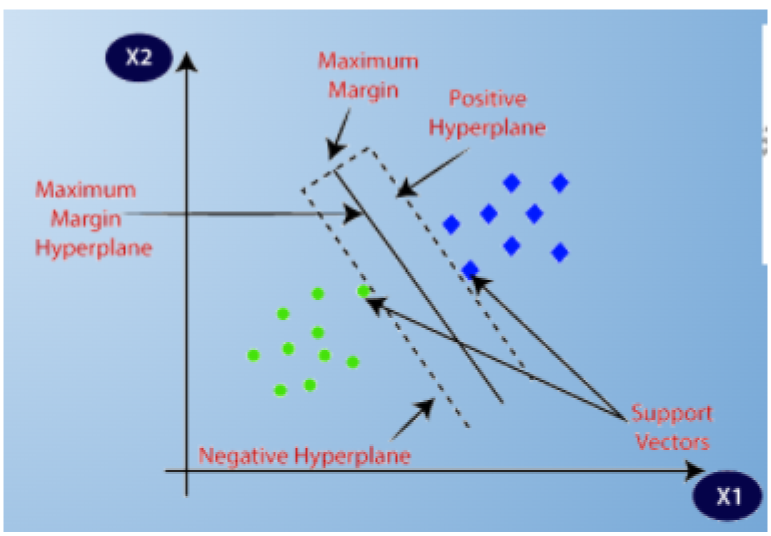
3 Methodology

3.1 Support Vector Machine

- Equation of Hyperplane: $wT(x) + b = 0$, where: b = Intercept and bias term of the hyperplane equation.
- In D dimensional space, the hyperplane would always be $D - 1$ dimensional.
- The goal of the support vector method approach is to maximize the minimum distance d between the hyperplane and the support vectors.
- For perfectly separable datasets, there are no issues, but for non perfectly separable datasets we can introduce a penalty for every misclassified datapoint to smoothen the learning curve,

$$d_H(\phi(x_0)) = \frac{|w^T(\phi(x_0)) + b|}{||w||_2} \qquad ||w||_2 =: \sqrt{w_1^2 + w_2^2 + w_3^2 + \ldots w_n^2}$$

- Sparse Data leads to network stagnancy during training, i.e. network learns working with zeroes and hence unable to perform with data in the sparse fields.
- The paper[1] implements 10-bivariate-SVMs for classification. It selects two features at a time and classifies the appropriate actuator. For data points with multiple features an ensemble of the models is used to arrive at the required actuator.



SVM Classifier

- **SVM Confidence Scores**

  If more than 1 model can be used, i.e. 3/5 features are available $[a, b, c]$, then we need to select the better model: $a + bora + corb + c$
  Confidence score maps the performance of the models, giving the more reliable model for selection.

Classifier

Label

3	3	0	2	13	13	8	14	5	6
10	3	0	7	12	12	7	12	4	0
0	0	0	0	0	3	0	3	0	4
9	8	8	0	188	13	10	12	6	10
2	0	4	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0	0
0	4	1	1	0	1	8	0	8	0
0	0	0	0	0	0	0	0	0	0

Predicted

Classifier

Label

3	3	3	2	18	15	9	16	9	9
11	6	3	7	15	13	10	12	8	5
3	2	1	2	11	6	3	5	4	4
11	10	8	3	194	14	12	13	9	12
6	3	4	2	6	6	7	2	5	4
2	2	1	1	1	1	1	1	1	1
2	2	1	1	5	4	9	1	2	4
8	7	3	4	24	10	11	9	12	5
0	0	0	0	26	0	0	0	0	0

Actual

Accuracies

[0.522 0.514 0.542 0.455 0.71 0.609 0.629 0.695 0.46 0.455]

[0.43, 0.38, 0.12, 0.44, 0.07, 0. , 0.04, 0.15, 0.]

Confidence

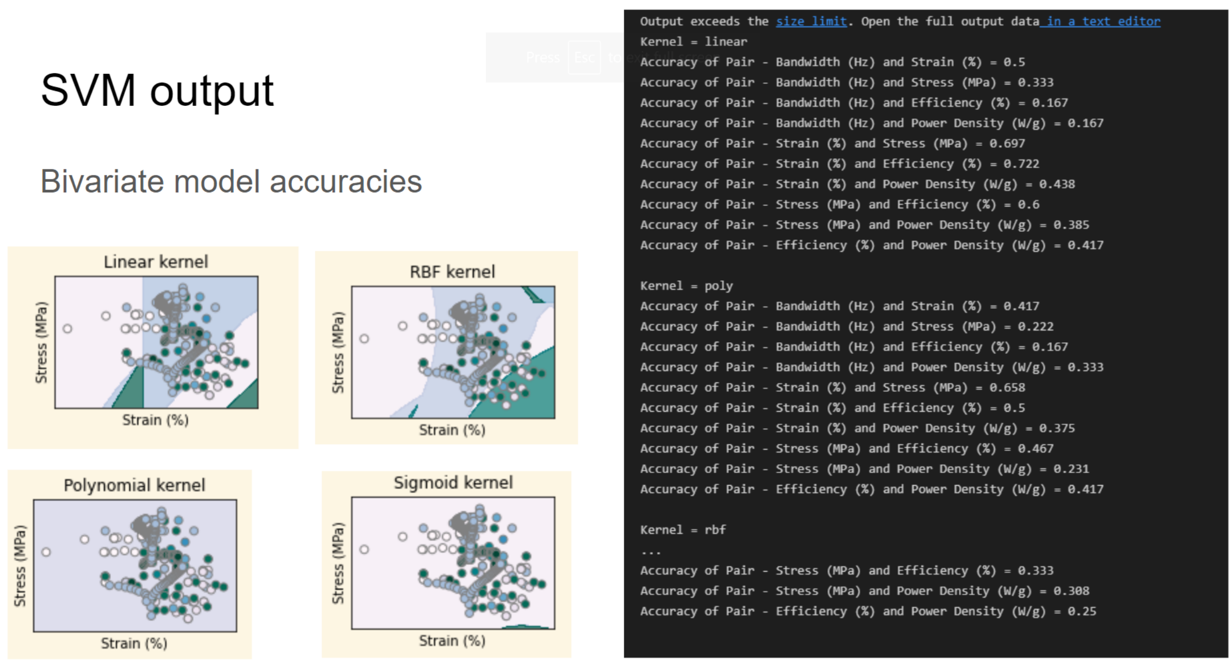
- **SVM Code: SVM architecture: Scikit Learn**

```python
data = pd.read_csv("data/original_data.csv")
cols = list(data)[2:]
column_pairs = []
for i in range(len(cols)):
    for j in range(len(cols)):
        if i<j:
            column_pairs.append((cols[i],cols[j]))
accuracies = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for j in range(len(kernels)):
    print("Kernel = {}".format(kernels[j]))
    accuracies.append([])
    for i in range(len(column_pairs)):
        column1,column2 = column_pairs[i]
        X,y = Functions.data_clean(data, column1, column2)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
        svc = SVC(C=0.1, random_state=1, kernel=kernels[j], degree= 3)
        svc.fit(X_train, y_train)
        y_predict = svc.predict(X_test)
        accuracies[j].append(np.round(metrics.accuracy_score(y_test, y_predict),3))
        print("Accuracy of Pair - {} and {} = {}".format(column_pairs[i][0],column_pairs[i][1],accuracies[j][i]))
    print()

```

- **SVM Code: PlotMultilabel Boundaries**

```python
def plot_multilabel_boundary(X,y,linear, rbf, poly, sig,column1,column2):
    h = .01
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
    titles = ['Linear kernel','RBF kernel','Polynomial kernel','Sigmoid kernel']

    for i, clf in enumerate((linear, rbf, poly, sig)):
        plt.subplot(2, 2, i + 1)
        #plt.subplots_adjust(wspace=0.1, hspace=0.1)
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        y_map = {'PZT':0, 'DEA':1, 'IPMC':2, 'SMA':3, 'SFA':4, 'TSA':5, 'SCP':6, 'EAP':7, 'SMP':8}
        for j in range(len(Z)):
            Z[j] = y_map[Z[j]]
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=plt.cm.PuBuGn, alpha=0.7)
        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.PuBuGn,      edgecolors='grey')
        plt.xlabel(column1)
        plt.ylabel(column2)
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.xticks(())
        plt.yticks(())
        plt.title(titles[i])
        plt.show()
```

- **Result**

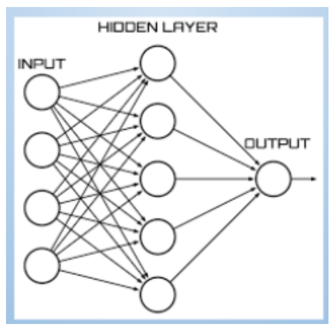


3.2 Neural Network

- **NN code: Used Tensorflow.keras()**

NN Architecture: L x 25 x 25 x 15 x 9
L = 2 for bivariate model;
L = 5 for multivariate model;

```python
# for i in range()

model = tensorflow.keras.Sequential([
    tensorflow.keras.layers.Flatten(input_shape=(2,1)),
    tensorflow.keras.layers.Dense(25, activation='relu'),
    tensorflow.keras.layers.Dense(25, activation='relu'),
    tensorflow.keras.layers.Dense(15, activation='relu'),
    tensorflow.keras.layers.Dense(9, activation='relu')
])

model.compile(optimizer='adam', loss=tensorflow.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

model.fit(X[:,1:3], labels, epochs=10)
```



Neural Network

4 Our Approach to counter Data sparsity

4.1 Generative Algorithm for generating similar data

- We tried using GANN for generating a tabular database to fill gaps in the training data.
- Issue: To generate feature “x1” for class “A”, some data of “x1” should be available. Due to the missing values in multiple classes, An All-feature classifier was not possible with this method.
- Result: Model ran partially; Output data showed negligible accuracy.

4.2 Data Imputation: Statistical Method of Data Generation

- There are seven types of muscle actuators : PZT, DEA, IPMC, SMA, SFA, SCP, TSA.
- Five physical properties : Bandwidth, Strain, Stress, Efficiency, Power Density.

Reference	Actuator T	Bandwidth	Strain (%)	Stress (MP	Efficiency	Power Density (W/g)
[1]	PZT	10000	0.2	110	90	0.2
	PZT	100			90	
	DEA	100	200	8	85	0.2
	IPMC	10	40	0.3		0.02
	SMA	3	5	200	1.3	50
	SMA	0.5	250	2	10	0.03
	SFA	130	50	10	30	11
	SFA	100	300		30	22
	SFA				30	10
	TSA	5	50	80	76	0.5
	SCP	3	49	11	1.02	27
[2]	SMA	3	10	200	3	4.3
	PZT	5000	0.2	35	50	

EAP	20223.17	10	34	41.12	0.15
EAP	20223.17	4	0.8	41.12	0.134947
IPMC	10	40	0.3	2.45	0.02
IPMC	49.37778	2	15	2.9	0.039594
IPMC	49.37778	40	11.91769	2.45	0.039594
IPMC	100	3	30	3	0.0026
IPMC	33	8.2	4.7	2.45	0.244
IPMC	49.37778	10	20	2.45	0.039594
IPMC	49.37778	0.5	3	1.5	0.00256
IPMC	1	5	11.91769	2.45	0.039594
IPMC	0.2	0.5	11.91769	2.45	0.039594
IPMC	0.2	2.1	11.91769	2.45	0.039594

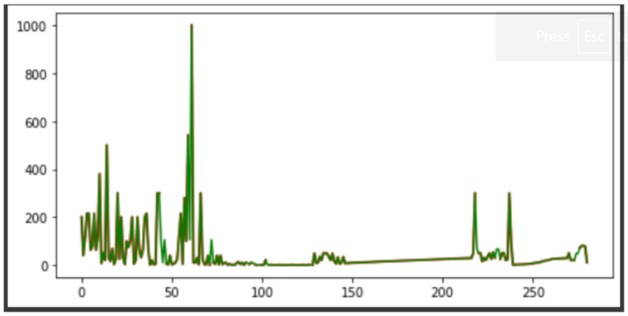
Original Data b. After statistic data imputation

Press Esc to exit full screen

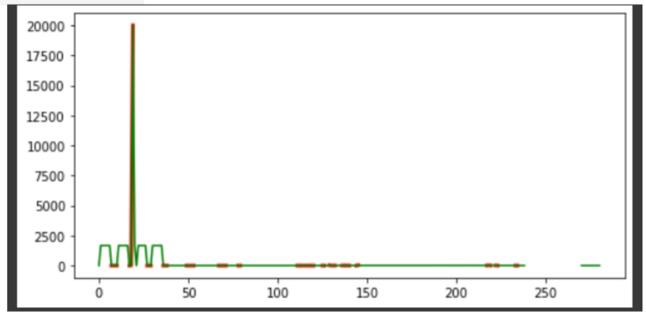
```
# Imputing Mean Values in PZT.csv
df = pd.read_csv('PZT.csv')
del df["Actuator Type"]

df.fillna(-999, inplace = True)
numpy_data_PZT = df.to_numpy()
imp = SimpleImputer(missing_values=-999, strategy='mean')
imp.fit(numpy_data_PZT)
imp = imp.transform(numpy_data_PZT)
new_column_values = ['PZT'] * 32

df = pd.DataFrame(imp, columns = [ 'Bandwidth (Hz)',  'Strain (%)', 'Stress (MPa)', 'Efficiency (%)', 'Powe
df["Actuator Type"] = new_column_values
df = df[['Actuator Type', 'Bandwidth (Hz)', 'Strain (%)', 'Stress (MPa)', 'Efficiency (%)', 'Power Density
df.to_csv('PZT_mean.csv', index=False)
display(df)
```

	Actuator Type	Bandwidth (Hz)	Strain (%)	Stress (MPa)	Efficiency (%)	Power Density (W/g)
0	PZT	10000.0	0.200000	110.000000	90.0	0.200000
1	PZT	100.0	1.242103	83.246154	90.0	0.230783
2	PZT	5000.0	0.200000	35.000000	50.0	0.230783
3	PZT	5800.0	1.700000	300.000000	90.0	0.230783
4	PZT	5650.0	0.100000	83.246154	72.6	0.230783

Data imputation using SciKit Learn

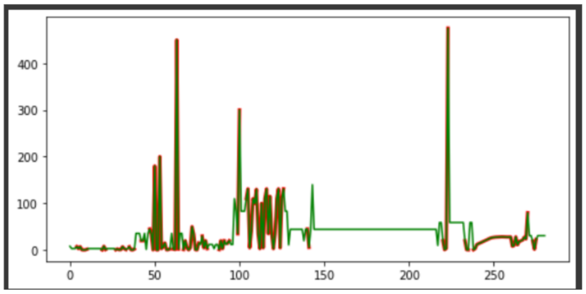


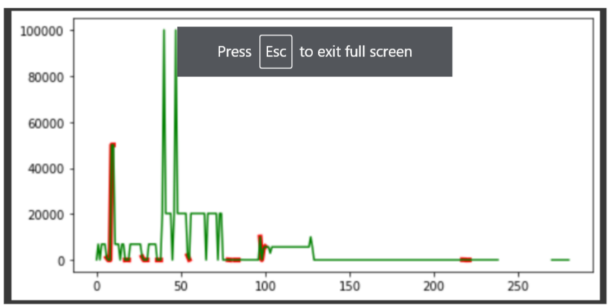
Strain (%)



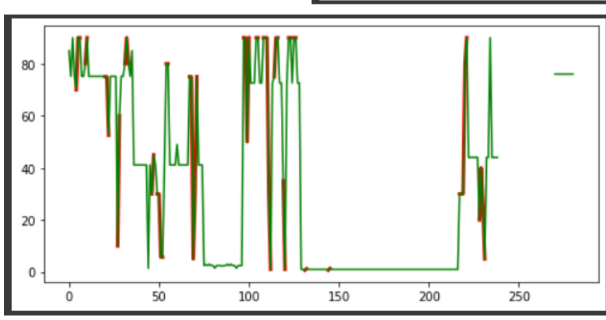
Power Density (W/g)

Stress (Mpa)





Bandwidth (Hz)



Red : Original
Green : Imputed

Efficiency (%)

Result: Results showed low accuracy on validation set: 43% accuracy

Reason deduced: The imputed data showed little to no coherence with the actual/original working parameters of the actuators.

```
9/9 [==============================] - 0s 2ms/step - loss: 4.1923 - accuracy: 0.2242
Epoch 9/10
9/9 [==============================] - 0s 2ms/step - loss: 3.8033 - accuracy: 0.2100
Epoch 10/10
9/9 [==============================] - 0s 3ms/step - loss: 3.5285 - accuracy: 0.1993
```

```
Epoch 9/10
9/9 [==============================] - 0s 3ms/step - loss: 2.2581 - accuracy: 0.4306
Epoch 10/10
9/9 [==============================] - 0s 416us/step - loss: 2.1953 - accuracy: 0.4270
```

accuracy(22%) before imputation, b. accuracy (43%) after imputation

4.3 Data augmentation with Real-life-parameters

Replaced missing data with actual work parameters of the given actuators.

Actuator type	ε_{max}	σ_{max} (MPa)	E (MPa)	ρ[W/kg]	Efficiency	f_{max} (Hz)
Muscle	0.3-0.4	0.1-0.4	5-20	50-284	0.2-0.4	50-500
DC Motors	0.4	0.1	--	100	0.6-0.9	---
Pneumatic	0.1-1	0.5-0.9	50-90	4000	0.4-0.5	50-300
Hydraulic	0.1-1	20-70	2 - 3 x 10^3	1600-2000	0.9-0.98	50-300
SMA	0.07	100-700	3 - 9 x 10^4	6400-6600	0.01-0.02	0.02-0.07
SMP	1.0	2-14	4 - 12 x 10^3	850-880	< 0.1	< 0.01
EAP (I)	0.02-0.4	5 - 34	0.2 - 3 x 10^3	150	<0.01	1-500
EAP (N/I)	0.2-3.8	5-6	1	1000-2500	0.15-0.9	1-2000
MREs	0.005	0.1-10	1-10	3 - 4 x 10^3	0.6-0.8	--
MRFs	0.002	0.1	--	3 - 4 x 10^3	--	--

G. Alici; *"Soft Robotics vs Hard Robotics",* 2018, Semantic Scholar; 2018.158

We replaced missing data with actual work parameters of the given actuators.

Result: 52% accuracy

Reason: Better Data distribution should be selected.

```
9/9 [==============================] - 0s 3ms/step - loss: 1.8207 - accuracy: 0.5231
Epoch 9/10
9/9 [==============================] - 0s 2ms/step - loss: 1.8044 - accuracy: 0.5196
Epoch 10/10
9/9 [==============================] - 0s 3ms/step - loss: 1.7951 - accuracy: 0.5267
```

5 Conclusion

- A bivariate model like SVM is really performant as the data sparsity problem is solved, removing stagnancy.
- The Augmented Dataset with NN has a comparable accuracy to the bivariate SVM avg. Thus it can be a potential method to do the classification.
- A more rigorous search for a better NN architecture can be done by using Grid Search CV to solve any NN-capacity problems.

6 References

[1] *Taylor West et al; "Data-driven Actuator Selection for Artificial Muscle- Powered Robots"; 2021; arxiv.2104.07168*

[2] *Curci. S. et al; "Truly Sparse Neural networks at Scale"; 2021; arxiv.2102.01732;*

[3] *Xu, L., Veeramachaneni, K.; "Synthesizing Tabular Data using Generative Adversarial Neural Networks"; 2018; arxiv.1811.11264;*

[4] *Goodfellow, I., et al; "Generative Adversarial Neural Networks"; 2014; arxiv.1406.2661*

[5] *Data Imputation; SciKit Learn https://scikit-learn.org/stable/modules/impute.html*

[6] *Khan, S., Hoque; "Improved Missing Data Imputation"; Journal of Big Data–Springer; 37(2020)*

[7] *G. Alici; "Soft Robotics vs Hard Robotics", 2018, Semantic Scholar; 2018.158*

[8] *Petro, L., Pavlo, L;"Grid Search, Random Search, Genetic Algorithm: Big Comparison for NAS"; 2019; arxiv.1912.06059*

[9] *Khan, S., Hoque; "Electro-active polymer actuators and devices"; 1999; SPIE Conference-California*