Inverse Kinematics of SMA soft limb using Long Short-Term Memory RNNs

Richard Desatnik
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
rdesatni@andrew.cmu.edu

Avadha Patel
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
avadhp@andrew.cmu.edu

Saurabh Borse
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
sborse@andrew.cmu.edu

Abstract

Inverse Kinematics of soft robotic limbs powered by Shape Memory Alloys (SMAs) remains a challenge. Soft robots do not rely on traditional kinematics assumptions as they are made from compliant and deformable materials. These structures have infinite degrees of freedom, making them challenging to control. SMAs as a means of actuation induce further challenges due to their electrical-thermal-mechanical properties. In this work, we propose using Long Short Term Memory (LSTM) architecture to learn the inverse kinematics of an SMA soft limb. The model is trained using data from a capacitive bendsensor and demonstrates that a soft robot could be controlled with a Recurrent Neural Network.

1. Introduction

Soft robotics is a rich field whose focus is on robots created from flexible materials. Due to their compliant nature, soft robots offer new opportunities apart from their rigid counterparts. Soft robots can more closely mimic soft biological structures and can make automated work safer. Although there are a plethora of benefits, soft robots come with a host of challenges that stifle their use. Soft robots require compliant materials throughout their structure which includes the electronics that power them and the sensors that provide feedback. As a result of these machines being compliant, their motion is governed by continuum mechanics and therefore have infinite-degree-of-freedom (DOF). [4] The property of infinite DOF makes controlling such machines with traditional closed-loop architectures unreliable. Soft robots are nonlinear by nature, and their high-dimensional dynamics make them a potential candidate for control with neural networks.

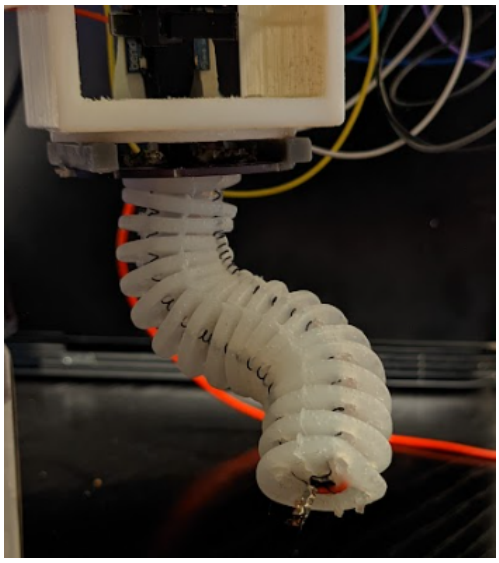

Fig1. The figure of the soft limb was used in the experiment during the data acquisition process.

A compact means of inducing actuation for soft robots are SMA actuators. They have been used in soft robots to mimic the muscles of echinoderms[11] and create small walking

robots.[10] SMAs actuators use Joule heating to induce force from their hysteresis properties.[5] They have high work densities and can be actuated with very little additional hardware which is why they are oftentimes used in untethered soft systems. However while easy to install they are complex to model. They have time-dependent hysteretic behavior that is dictated by interactions at an atomic level.[6] Although there are some models that characterize their behavior they come with a high degree of variability after manufacturing.

With the idiosyncratic challenges of SMA soft limbs, we used capacitive bend sensing and an RNN framework to determine their inverse kinematics. There are alternative methods of sensing such as resistive flex sensors and optical fiber. Resistive flex sensing suffers from issues in repeatability and accuracy with long-term use making them sub-par for machine learning applications. [7] Optical fibers are resistant to electrical noise but struggle with twisting. [8] BendLab's capacitive bend sensor measures bending with two compliant capacitive plates. As the sensor bends the difference in the strain at the top and bottom of the sensor leads to a measurable difference in capacitance which the sensor uses to generate an angular reading.[3]

RNNs are a neural network architecture designed to capture meaningful information from time-dependent and sequential data.[9] Recurrent Neural Networks learn time-dependent data by sharing information between layers in the form of hidden layers. Although this captures time-dependent information RNNs are prone to a phenomenon known as exploding and vanishing gradient. Vanishing gradient occurs when information between shared weights is lost as the output of weights approaches zero. Exploding gradients occur when the outputs of the weights increase exponentially and dominate the overall input signal. To combat these issues our team used LSTMs. LSTMs share information with hidden layers with a hidden state and a cell state. Within the LSTM there is a forget gate, input gate, and output gate. The LSTM structure allows for the RNN to retain important information from a time sequence without the pitfalls of vanishing and exploding gradients. A figure of an LSTM cell can be seen in figure 2.

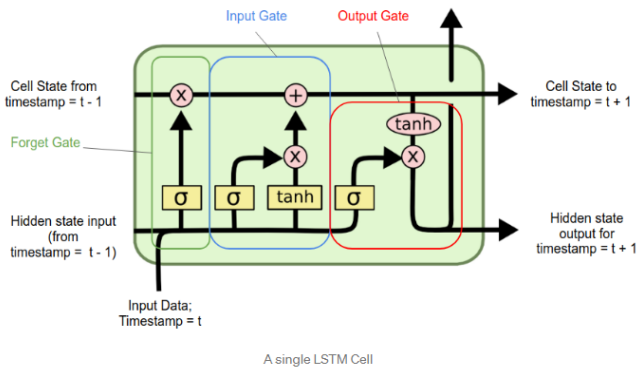

Fig 2. Displays a single LSTM cell with forget, input, and output gates. To the right are the hidden state and cell state.

2. Related Work

Control of an SMA soft limb with an RNN on a single axis has been demonstrated in [2]. The team used a single axis bend sensor to measure the bend angle of a soft limb on an XY plane. An attempt to control an SMA soft limb by more traditional means can be seen in X where the author used a Singular Value Decomposition (SVD) with Multi-Input Multi-Output (MIMO) anti-windup to control a multi-axis soft limb. [1] Our contribution is controlling a multi-axis soft limb using an LSTM.

3. Mathematical Model

The goal of our project is to map two Bend angles over time to the duty cycle of the SMAs actuators seen in equation 1. P1(t), P2(t), P3(t), and P4(t) represent the duty cycle of the 4 SMA actuators, and BX(t) and BY(t) represents the bend angles in the X and Y direction respectively.

$$[P1(t), P2(t), P3(t), P4(t)\ ] = [BX(t), BY(t)] \quad (1)$$

3.1 LSTMs

A single LSTM node uses the equations below to moderate information from within a cell:

$$\text{Forget gate } f_t = \sigma_g(W_f \times x_t + U_f \times h_{t-1} + b_f) \quad (2)$$

$$\text{Input gate } i_t = \sigma_g(W_i \times x_t + U_i \times h_{t-1} + b_i) \quad (3)$$

Output gate $o_t = \sigma_g (W_o \times x_t + U_o \times h_{t-1} + b_o)$ (4)

$$c_t^{'} = \sigma_c (W_c \times x_t + U_c \times h_{t-1} + b_c) \quad (5)$$

Cell state $c_t = f_t \cdot c_{t-1} + i_t \cdot c_t^{'}$ (6)

Hidden state $h_t = o_t \cdot \sigma_c (c_t)$ (7)

3.3 Back Propagation

To update the weights while the LSTM is being trained we used Adam Optimizer for backpropagation. The equations for Adam and the backpropagation of LSTM can be seen below.

	Back Propagation	Update
Output	$\frac{\delta J}{\delta b_\upsilon} = \frac{\delta J}{\delta \upsilon_t}$	$W_\upsilon \mathrel{+}= \alpha * \frac{\delta J}{\delta W_\upsilon}$
Hidden state	$\frac{\delta J}{\delta h_t} \mathrel{+}= \frac{\delta J}{\delta h_{next}}$	
Output Gate	$\frac{\delta J}{\delta b_o} = \frac{\delta J}{\delta a_o}$	$W_o \mathrel{+}= \alpha * \frac{\delta J}{\delta W_o}$
Cell state	$\frac{\delta J}{\delta b_c} = \frac{\delta J}{\delta a_c}$	$W_c \mathrel{+}= \alpha * \frac{\delta J}{\delta W_c}$
Input Gate	$\frac{\delta J}{\delta b_i} = \frac{\delta J}{\delta a_i}$	$W_i \mathrel{+}= \alpha * \frac{\delta J}{\delta W_i}$
Forget Gate	$\frac{\delta J}{\delta b_f} = \frac{\delta J}{\delta a_f}$	$W_f \mathrel{+}= \alpha * \frac{\delta J}{\delta W_f}$
Input	$\frac{\delta J}{\delta c_{t-1}} = \frac{\delta J}{\delta c_t} \odot f_t$	

Table 1 illustrates the update rule and backpropagation for LSTM

Adam Optimizer $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (8)

Adam Optimizer $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (9)

3.2 Mean Square Error (MSE)

To train the LSTM model we used MSE loss seen in equation X. between the real PWM signals and output PWM signals of the LSTM model.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \quad (10)$$

4. Data

For our experiments, our data has a continuous numerical time-dependent input and continuous numerical time-dependent output. The input data are sensor values from the BendLabs 2 axis capacitive bend sensor. This sensor is capable of measuring the bend angle on the X and Y-axis of the end effector. The X and Y inputs are continuous values from -120 degrees to +120 degrees for both X and Y. The capacitive Bend Sensor has a life of over 1 million cycles and repeatability of 0.18degress. Figure 3 illustrates the orthogonal planes and angles the bend sensor is measuring during operation.

The output data of our algorithm are the (Pulse Width Modulation) PWM signals to the SMAs. The Arduino Uno was sent to our computer through the Excel Data Analyzer which takes Serial Monitor Data generated from the Arduino to create CSV files. Data was collected every tenth of a second and placed in a chart seen in Figure 4.

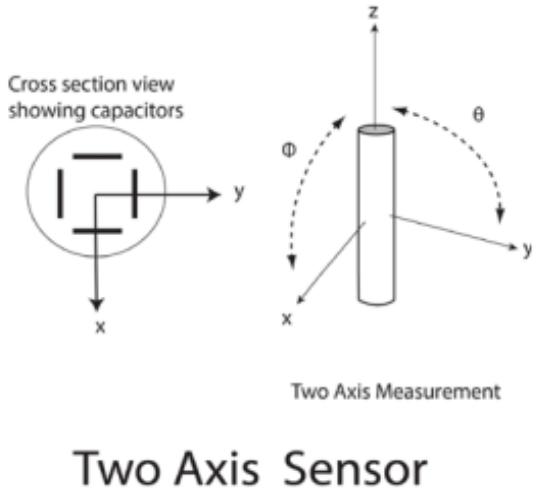

Fig. 3 Illustrated the planes and the angles the bend lab sensor is measuring for the state-space of the soft limb. [3]

After the data was collected our team cleaned the data with Pandas. In our CSV files from excel, we had to remove unnecessary cells that Excel Data Analyzer adds to the CSV file. Next, the excel analyzer saves values in the CSV as strings which we had to convert to floats to perform calculations. Errors in the data acquisition came from three main sources. One source of error was cells double counting values. This rarely occurred, happening at a rate of roughly 1 per every 10,000 values. The second source of error was the robot stepping out of the state space in values slightly or highly above 120 degrees on an axis or below -120 degrees. This happened more frequently, roughly 10-20 times per 10,000 values. The third source of error was null values. At first, we did not realize the null values were in the data. However, when we placed the data in the NN we realized our results would all become all null values. We noticed that although the code would still run during backpropagation, the null values would be distributed through the network. This would result in the NN not learning anything and losing all training once the batch with null values was fed through. Null values were rare, making up about ~20 data points per 100,000 but still critical to remove nonetheless. These errors happened so infrequently that we felt

	BX	BY	PWM5	PWM9	PWM10	PWM11
4	22.76	4.77	0.0	114.0	0.0	0.0
5	22.76	4.77	0.0	114.0	0.0	0.0
6	22.06	4.77	0.0	114.0	0.0	0.0
7	22.06	4.77	0.0	114.0	0.0	0.0
8	21.18	4.77	0.0	0.0	0.0	208.0
...	...	...	...	...	...	...
15125	32.14	-32.49	140.0	0.0	0.0	0.0
15126	32.14	-32.49	140.0	0.0	0.0	0.0
15127	3.05	-31.01	140.0	0.0	0.0	0.0
15128	-36.81	-29.97	140.0	0.0	0.0	0.0
15129	31.25	-29.97	140.0	0.0	0.0	0.0

Fig 4. Structure of pandas data frame after the data is thoroughly cleaned. BX and BY represent the input bend angles. PWM5, PWM9, PWM10, and PWM11 represent the output duty cycles and Arduino Pin designation.

removing these values would not make any significant difference in our results. Before inputting the pickled data in the LSTM, we messaged the data using the sklearn library. We used the standard scaler function to change the training and test data from zero to one. After the neural network was fully trained, we would use the inverse of this function to obtain the final PWM signals to place back into the soft limb. In the end, our team collected 268,314 data points after data cleaning. Over 7 hours of data collection for the robot.

5. Methods

5.1 Approach

Our soft limb utilizes an LSTM Recurrent Neural Network to learn inverse kinematics of a physical soft limb. We decided on this structure because the data we are mapping is time-dependent and continuous. We did not want to use simulated data to avoid the inherent issues of simulation to real

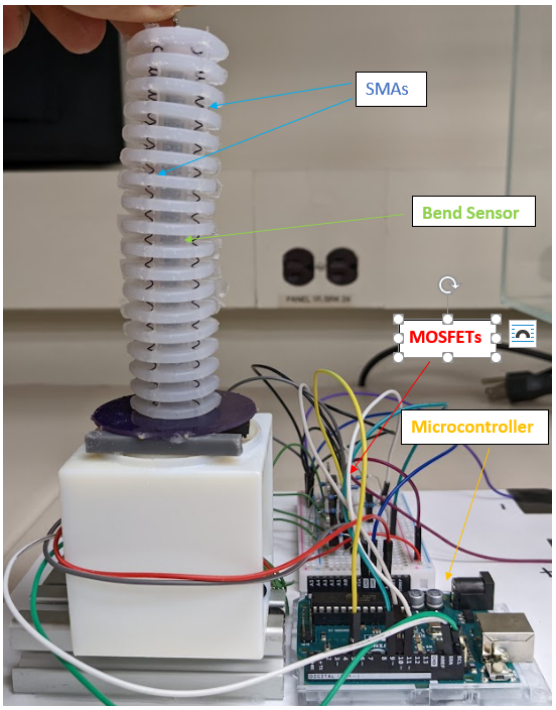

Fig 5. This figure lays out the main components of the soft limb. Pointing to the location of the bendsensor, SMA, and MOSFETS

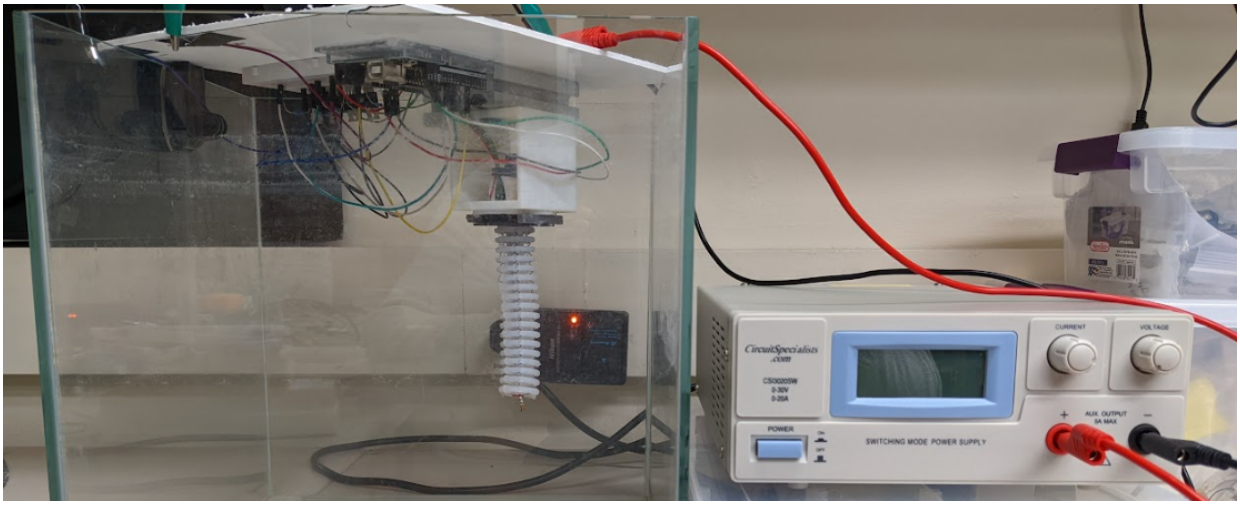

Fig. 6. Displays the hardware setup during data acquisition. The robot is placed upside down to ensure a stable end-effector position during the trials.

transfer and wanted to understand if the algorithm would work with data generated from the natural flaws in the machine. We tested the neural network under different conditions to create the best model we could. We wanted to understand how the number of timesteps, dropout, other activation functions, and the number of layers would impact the validation loss and accuracy. After the algorithm maps the input of bending angles in the X and Y positions over 30 timesteps to four duty cycles, we test our results on the soft limb itself.

5.2 Alternative Approaches

Our team felt that an LSTM RNN was the best choice for our application due to the nature of the input and output data, but there were other approaches worth considering. For example, we could have used a GRU or Attention model to handle the data instead of an LSTM. An alternative approach to controlling an SMA soft limb can be seen in [1]. Their team used singular value decomposition and multi-input multi-output anti-windup for SMA soft limb control. However, this model struggled to handle oscillations when the end effector was further from equilibrium, attributed to sensor noise. Using a data-driven model, our team wants to conduct motion tracking robust to the deviations caused by variability in SMA actuators or the silicone body.

5.3 Hardware Setup

The soft limb was created from Dragon Skin 10 Slow Silicone from Smooth-On. Through injection molding, we build the silicone shell that makes the body of the soft limb. The length of the soft limb sits at about 10cm tall. This is to fully encompass the capacitive bend sensor. Next, we stretch and insert four Nitinol Dynalloy SMA actuator coils into the limb and crimp them together at the end effector. To actuate the robot, the limb uses opposing actuators as the SMAs are only able to move in a uni-directional fashion and must be acted on by another actuator to return to a position. Finally, the capacitive sensor is inserted through the center of the limb. All of these are placed on a bracket connected to 80x20 aluminum. The Arduino Uno, the microcontroller used to control the soft limb, takes in sensor data through I2C and sends PWM signals through four IRFZ44N Mosfets. The soft limb is powered by a function generator set to 7.7V, and the SMAs are actuated with a current of 0.5A to 1.5A depending on the incoming duty cycle. Once fully assembled, the limb is placed upside down in an acrylic box. This is to ensure that the final stable position on the end effector is always facing down and ensures the safety of passers-by from the SMA coils as they can become quite hot. The entire test setup can be seen in figure 6.

5.4 Algorithm

Our algorithm uses TensorFlow to map the input of Bending angles in the X and Y positions over 30 timesteps to duty cycles. To do this, we split our 268,314 data points into a training set of 80% of the data and a test set comprised of 20%. The input of the neural network takes in a 2 by 30 vector of Bend angle and time steps, respectively. The output is a 4 by 30 vector of duty cycles and time steps, respectively. To determine the error of our guessed duty cycles, we use MSE Loss to measure the difference between real PWM signals of the test set to the output PWM signals of the algorithm. This loss was used to update our weights along with Adam optimization.

Our algorithm uses TensorFlow to train and test the model. The model has 464,464 trainable parameters while using GELU as the activation function. GELU performed slightly better than the ReLU activation function (i.e., validation accuracy increased by 1%).

At first, we tried to train the dataset with discrete values for PWM (i.e., 0 and 1) to check if the model for if the model is able to be trained. Next, we used the actual values of PWM signals to train the model, but we found that it was not able to get high accuracy since the data was spread over a long range. To rectify this, we used the StandardScalar function from sklearn library to normalize the data. We were able to increase the accuracy significantly.

```
Model: "LSTM_Soft"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 30, 256)           265216

 activation_2 (Activation)   (None, 30, 256)           0

 lstm_4 (LSTM)               (None, 30, 128)           197120

 activation_3 (Activation)   (None, 30, 128)           0

 lstm_5 (LSTM)               (None, 30, 4)             2128

 dropout_3 (Dropout)         (None, 30, 4)             0

 reshape_1 (Reshape)         (None, 30, 4)             0

=================================================================
Total params: 464,464
Trainable params: 464,464
Non-trainable params: 0
_________________________________________________________________
```

Fig. 7 This image depicts the primary neural network used to train the robot.

Training of the data was conducted for 60 epochs with a batch size of 32. After getting 72% validation accuracy for training, we tested the model on a dummy dataset and used inverse transform to get reverse the normalization of the PWM values.

6. Experiment

The soft limb is operated upside down to ensure the stable position of the robot is at the Bend angles 0 degrees X and 0 degrees Y. Although, in practice, the robot is slightly off these angles when fully cooled. The robot is programmed to operate within a state-space of +120 to -120 degrees for the X and Y bend angles. This is achieved by a portion of the algorithm that stops data collection and SMA actuation for 15s if the robot exits this state space. This serves two purposes. One is to make sure that data collection stays within a specified state space for the NN. The other is to ensure the robot does not overheat and break itself during testing. If the robot does not have this limit added to the system, damage to the robot can occur. Most experiments with the robot were operated for about 10,000 timesteps or 16-20min of operation. There was a 30,000 timestep trial and 100,000 timestep trial to see if the robot could maintain operation for extended periods of time. During the data collection process, the robot receives a random duty cycle and a random pin for 1s of operation. This ensures the data is capturing the movement of the robot in response to the PWM signal as the robot receives 10 position locations in response to 1 PWM signal. The robot only allows for 1 SMA to be actuated at a time. This is to prevent scrunching from two opposing SMAs being actuated at the same time. This also prevents overheating of the SMAs as the other actuators can cool while others are operating. After the robot is left free to run, the data is collected directly into excel. Once in excel, the information is then cleaned and pickled with pandas to be sent to the RNN.

Once all the trials with the robot were conducted, we fed the data into our RNN. At this stage, we ran the algorithm. We wanted to test the algorithm in different conditions to maximize validation accuracy and minimize training loss. To do this, we wanted to run data through in 1s, 3s, and 9s batches and compare how the algorithm did with different time iterations. Next, we compared the

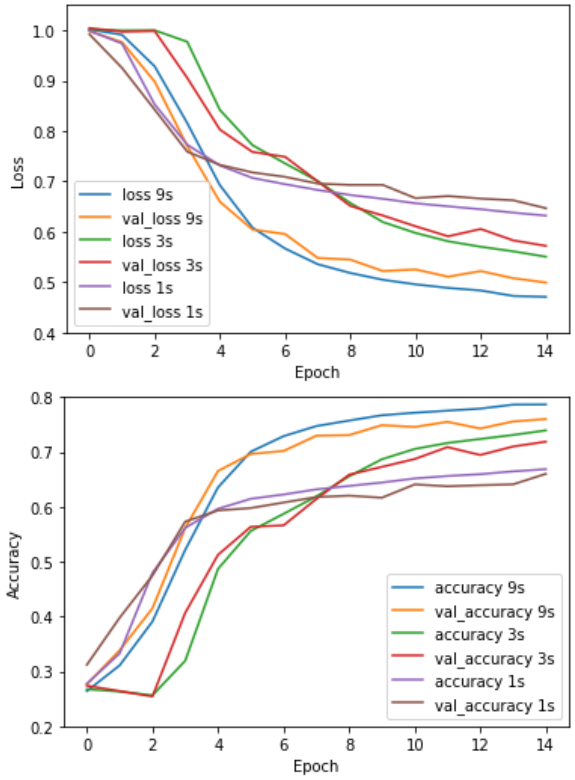

Fig 8 displays how changing the number of timesteps to train the model drastically increases the effectiveness of the LSTM in validation loss and accuracy.

algorithm at 3s time durations with varying amounts of dropout. We used none, 10%, and 20% in our experiments. Another aspect of the LSTM we wanted to test was the impact of different activation functions. For this, we tested Relu, Gelu, sigmoid, and Tanh for each layer. Our final test was on how many LSTM layers would be the best, comparing one layer, two layers, and three layers, all with 128 neurons. We only used two layers for all the experiments listed above, with the exception of the layer test, one with 256 neurons and the other with 128 neurons. For each experiment, we used 15 epochs and the adam optimizer. Once the algorithm was fully trained, we saved our LSTM model for testing on the physical robot. For testing the LSTM on the robot, we fed in a batch of 30 X and Y bend angles we wanted to reach. After that, we scaled the data with sklearn's standard scaler. Next, we fed the transformed data into the LSTM network and used an inverse standard scaler to output the real PWM signals for the robot. In this final test, we successfully ran PWM signals from the LSTM back on the original robot it was trained from.

7. Results

After all the tests, we found that 1s time steps performed the worst out of the 1s, 3s, and 9s comparisons. The 1s time duration only had a validation loss of 0.647 and a validation accuracy of 66%. The 3s timestep performed somewhere in the middle with a validation loss and accuracy of 0.57 and 71.8%. The 9s time step training performed the best with an MSE of 0.499 and a validation accuracy of 76%. Our team found that the LSTM performed better as time duration increased. We believe the reason the 1s time duration did so poorly was that the

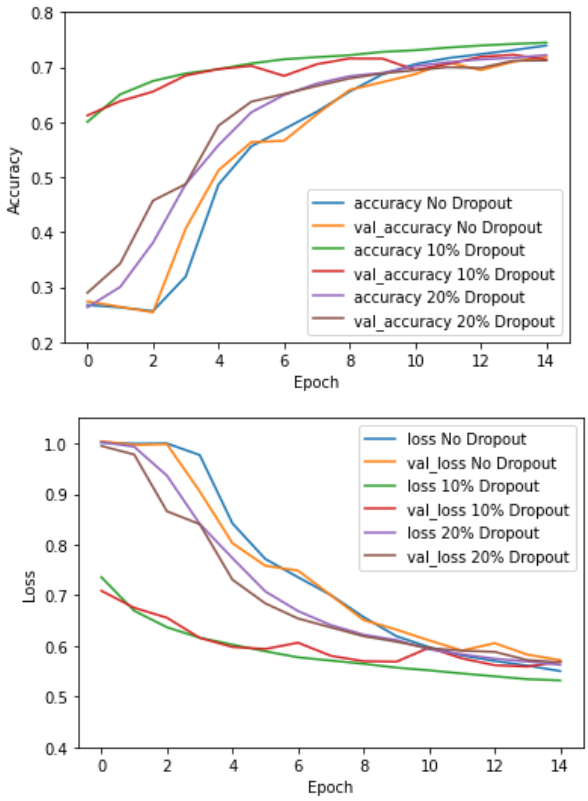

Fig. 9 displays the dropout trials; 10% of dropout by outperforms the others.

LSTM was not given enough timesteps to identify a whole transition as one PWM signal was being. switched to another. Because the PWM signal inputs were randomly generated over 1s intervals training the LSTM over 10-time steps was not enough to capture transitions. The fact that the 90 timestep trials resulted in a similar output is encouraging. This shows that the inverse kinematics learned by the LSTM model can and even perform better over longer durations. The results of changing the number of time steps per LSTM can be seen in Fig 8.

The results of dropout testing found that no dropout and 20% dropout performed the worst, resulting in the slowest and similar convergence that started to level off at eight epochs. The team found the sweet spot for dropout over 30 timestep batches was 10% resulting in the fastest convergence. The 10% dropout LSTM started to level off at four epochs. As for the validation accuracy and loss, they all performed about the same. However but the 10% still performed slightly the best. Fig. 9 depicts the dropout trials.

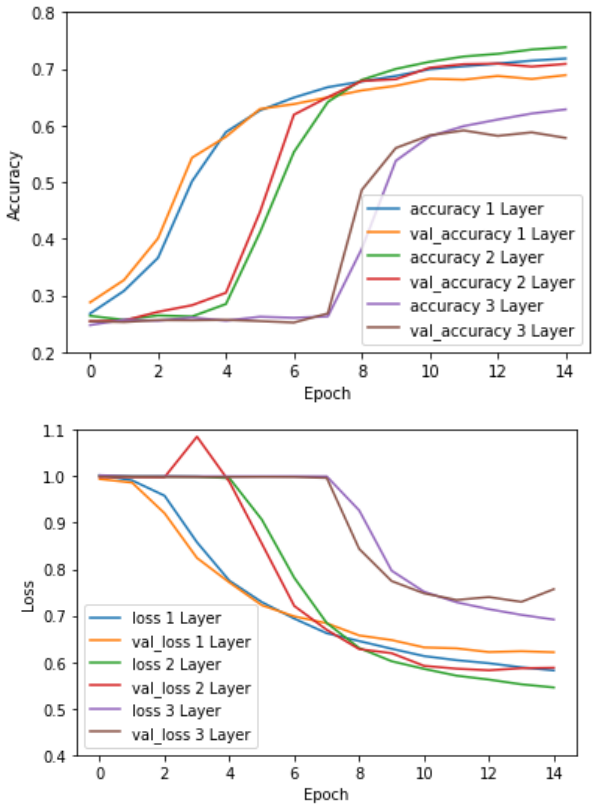

Fig.10 This figure shows the differences in performance between the number of layers for different LSTM models.

The number of layers made a large impact on the overall performance of the LSTM. The two-layer model seemed to perform slightly better in terms of validation and accuracy when compared to the one-layer model. However, using one layer required fewer epochs to converge. The model with three layers was both the slowest to converge, the highest final loss, and the lowest overall accuracy. This is important as increasing the depth of the LSTM did not boost and, in some cases, quite hindered the performance of the overall algorithm. Fig. 10 depicts how changing the number of layers changed the quality of the network.

The trials with the different activation functions can be seen in figure 11. The worst performing activation function by far was sigmoid. Sigmoid took the most epochs to start learning the data and was the least accurate. Our team believes the sigmoid activation function struggled so much from vanishing gradient. Since sigmoid can only output values from zero to one the gradients tend to head toward zero. Relu, Gelu, and Tanh performed roughly the same. Surprisingly Tanh produced the best values scoring slightly above Relu and Gelu. This finding was surprising as we anticipated that Tanh would suffer from vanishing gradient much as a sigmoid had.

8. Discussion

Performing LSTM RNNs on our data resulted in the model accuracy reaching 72% prediction to draw a successful conclusion of the inverse kinematic of the soft robotic actuation with signals that successfully result in a desired X and Y. Applying a RNN allowed the ability to extend the ability to track motion from a 2D to 3D model using a multi-segmented limbs system. With the initial application of teach and repeat based on physical manipulation. Our process resulted in a learning process from the random PWM signals given to a desired trajectory which allowed for creating PWM for the desired trajectory. This process extended the range of motion resulting in a more robust control with LSTMs to train and test the data yielding a 72% accuracy. The unexpected Result we received during the training process was accuracy of only 65% before the use of 30 timeslots because

the time required to warm up the SMA results in less control of reaching the desired trajectory.

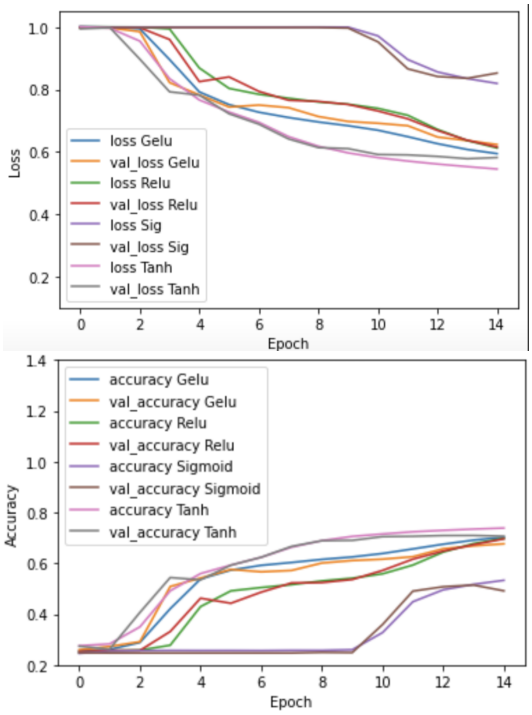

Fig.11 Depicts the loss and accuracy of different activation functions. Sigmoid performs the worst as it suffers from vanishing gradient**.**

The limitation occurring in our process was to prevent the soft limb from overstretching out of the regions of interest over 120 degrees in an axis or -120 degrees in an axis, and the minor occurrences of NaN or cell double-counting numbers disrupted the data collection process. Future work includes the implementation of a reinforcement learning policy to nudge the SMA to stay within the region of interest to improve trajectory tracing. Using a physics-based engine to perform using the LSTM model to gather data and implement a Policy to stay within the desired region to be translated into a physical system to understand the translation of the system between physical and simulations accuracy. The LSTM model to predict the PWM signals needed to get to the desired trajectory reached a 72% accuracy at being able to learn from the dataset of X and Y positions based on the input of the 4 PWMs, which displayed the adaptability of RNNs to learn inverse kinematic that occur in soft robotics.

9. Conclusion

The problem faced was the controllability of soft robotics with the complex inverse kinematics that is difficult to model using physics principles resulting in the use of RNNs to be trained on data collected from running a soft robotic concerning X and Y desired position and PWM signals imputed to generate the trajectory. Creating a Test setup with random PWM signals used to gather the final X and Y positions from the Bend sensor. using an Arduino microcontroller that printed the numerical data into excel for data collection. The data used as an input for the LSTMs model can be trained on time-dependent data, which can be output to the next cell or forgotten based on the weighting of the value. LSTM RNNs provided 72% accuracy in giving PWM signals for the desired X and Y positions, creating a solid foundation for LSTMs learning the inverse kinematics of soft robotics.

Future Work

LSTM models on soft robotic limbs can benefit from more studies into how adaptable trained models can be to variations in the soft limb. All of the tests in our experiments were conducted on the same soft limb. Future studies can include seeing how the algorithms perform on soft limbs that are built the same or varying in length and stiffness. Another issue is the response to external forces. Although in this study we focus on inverse kinematics future work can see how machine learning models can perform when external forces are involved.

Contributions

Richard worked on the hardware setup and data collection process and the LSTM implementation of the data. For the write-up, Richard worked on the Introduction, related work, data, and methods. Saurabh worked on LSTM implementation on the data, the equation section, and the data acquisition algorithm. For the write-up, Avadh worked on the Results Sections, Discussion, and Conclusion. Avadh worked on a safety mechanism for the robot to make sure it did not overheat. Avadh also worked on the data loader and transfer to the LSTM. For the

timeline of events, the robot hardware including the soft limb and wiring was created in February and mid-March. From late March to mid-April the algorithms used to control and gather data on the robot were created. From mid-April to early May the data was collected and the LSTM TensorFlow model was created. Fully trained models were completed in early May.

References

[1] Z. Patterson, A. Sabelhaus, and C. Majidi, "Robust Control of a Multi-axis Shape Memory Alloy-driven Soft Manipulator," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2210-2217, April 2022, doi: 10.1109/LRA.2022.3143256.

[2] A. Sabelhaus, R. Mehta, A. Wertz, and C. Majidi, "In-Situ Sensing and Dynamics Predictions for Electrothermally-Actuated Soft Robot Limbs," *arXiv e-prints,* arXiv-2111, 2021.

[3] *Soft Angular Displacement Sensor Theory Manual*, BendLabs, Salt Lake City, UT, pp.1-4, 2018.

[4] K. Chin, T. Hellebrekers, and C. Majidi, "Machine learning for soft robotic sensing and control," *Adv. Intell. Syst.,* 2: 1900171, 2020.

[5] D. Grant and V. Hayward, "Variable structure control of shape memory alloy actuators," *IEEE Systems and Control,* vol. 17, no. 3, pp. 80-87, 1997.

[6] M. Zakerzadeh, H. Salehi, and H. Sayyaadi, "Modeling of a nonlinear Euler-Bernoulli flexible beam actuated by two active shape memory alloy actuators," *J. Intell Material Sys Struct*, vol. 22, 2011, doi:10.1177/1045389X11414227

[7] L. K. Simone and D. G. Kamper, "Design considerations for a wearable monitor to measure finger posture," *J. Neuroeng. Rehabil.*, vol. 2, no. 1, p. 5, 2005.

[8] S. Sareh, Y. Noh, M. Li, T. Ranzani, H. Liu, and K. Althoefer, "Macrobend optical sensing for pose measurement in soft robot arms," *Smart Mater. Struct.*, vol. 24, no. 12, 2015, Art. no. 125024.

[9] S. Hochreiter, and J. Schmidhuber, "Long short-term memory" *Neural computation*, vol. 9, pp: 1735–1780, 1997.

[10] S. Rich, R. Wood, and C. Majidi, "Untethered soft robotics," *Nat Electron* vol. 1, p. 102, 2018, https://doi.org/10.1038/s41928-018-0024-1

[11]W. Huang, X. Huang, C. Majidi, and M. Jawed, "Dynamic simulation of articulated soft robots," *Nat Comm*, vol. 11, pp. 2233, 2020, doi:10.1038/s41467-020-15651-9. Number: 1