ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Semester project, Fall 2019

Estimation of a Contact Surface Normal

Alexis Philip George-Georganopoulos

In the Laboratory of:
Professor Aude Billard

Supervised by:
Kunpeng Yao, Farshad Khadivar

January 10^{th}, 2020

Contents

1 Introduction **1**
1.1 Context 1
1.2 The Robotic Finger 2
1.3 Machine Learning 3

2 Data Capture & Set Up **4**
2.1 Data Acquisition 4
2.2 The ROS Interfaces 5
2.3 The ROS custom Package 6

3 Data Processing **8**
3.1 Rectification & pre-processing 8
3.2 Support Vector Regression 10
3.3 Multi-Layered Perceptron Neural Network 12
3.4 Choice of method 14

4 Implementation **15**
4.1 The Biotac_NN ROS package 15

5 Conclusion **16**

1 Introduction

1.1 Context

One aspect of Robotics that is of increasing importance is that of tactile feedback, a machine must have an idea of the geometry it is encountering. With multiple applications that extend to the fields of Cybernetics, the ability to deduce shape from sensation is something humans and animals take for granted, however the hardware to extend that power to Robots has only become feasible in the past decade.
Consider the following scenario: One is trying to open a door, and in doing so, searches for their keys in their pockets. Within the pockets, there could also be other items(phone, wallet, etc...) that must be ignored. Out of instinct, we search our pockets without needing to visually look inside them, relying purely on the tactile sensations we experience on our fingers/fingertips. As such, we can deduce very complex geometry from touch.
Another similar situation would be when one is in a dark room, looking for a light-switch. We can't know what to look at since we are blind, and so rely entirely on touch to locate the switches' position.
In a similar vein, this Semester Project seeks to extend this tactile power to a Robotic finger(The Syntouch Biotac finger). To emphasize the importance, consider figure 1.

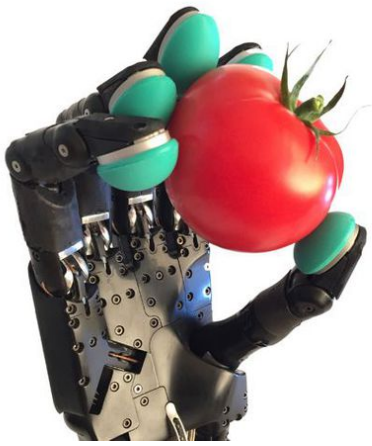

Figure 1: A Robotic hand holding a spherical object; but how can it deduce this?

One possible approach to estimate the form is to roughly estimate some geometry from the positions of the fingers needed to maintain to object static. However, what if the geometry is very complex? What if the surface of the object has some texture? This information would be lost using indirect methods of estimation. By providing the robot with direct sensory data, it could more accurately deduce geometry and vastly increase the type of information it detects: is the object warm? Is it sharp? Is it slippery?
Between these questions, there exists one(of many) fundamental properties the robot must be able to estimate: the normal direction of the surface it is touching.
A flat surface would have just one normal direction, a cylinder many along its axis. But at the core, the normal direction is required to begin any sort of deduction. And so, this paper is concerned with deducing geometry from the estimated normal direction of the Robots' sensors.

1.2 The Robotic Finger

The Syntouch Biotac sensor is a highly sophisticated artificial finger, capable of detecting a variety of tactile sensations. With generous software support, it is able to transmit real-time data from its sensors with a ROS interface, allowing the user to use it for both data acquisition, prediction, or any other need that may arise.
Among its sensors, it has 19 electrodes, a thermistor, and a static sensor pressure as indicated in 2

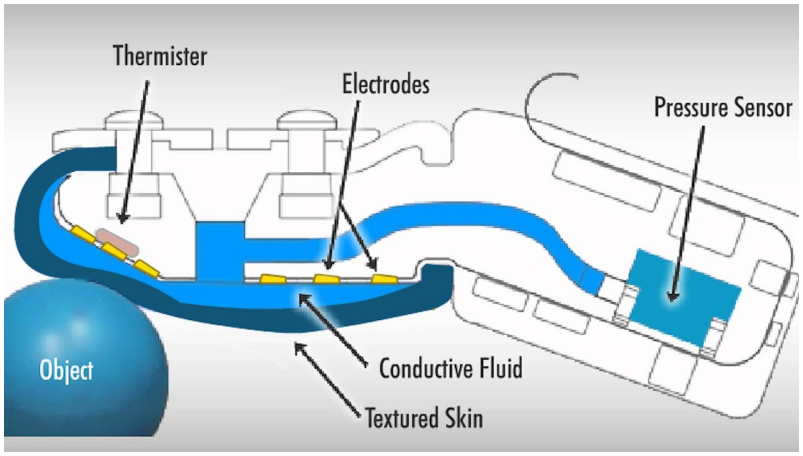

Figure 2: Biotac Sensor Architecture

To aid in this, this finger is equipped with a textured polymer skin containing a conductive fluid, with the latter mediating changes in pressure and temperature. The combination of these sensors and mechanical choices allows the finger to detect simple changes external touch, all the way to slipage thanks the textured skin.
For this paper, our interest lies in the electrodes and the pressure sensor. The latter for detecting if anything is even touching the finger, and the former for sensing changes in the fluid conductivity in its immediate vicinity. A detailed layout of the electrodes is shown in figure 3

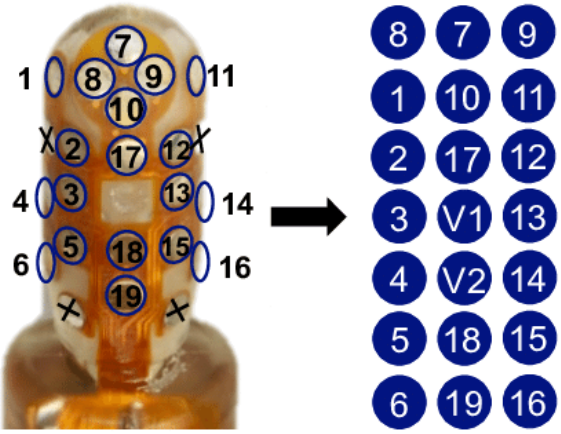

Figure 3: Electrode Layout of the Biotac(V1/V2 are empty)

We can see the finger has very distributed pattern of sensors, allowing it to detect

changes from most directions. The sensitivity of the electrodes allows them so sense differences in the milliVolt range, resulting with noise being the most significant limiting factor in the best case scenario. This bodes well since tactile changes are far larger than 1[mV] ranges, allowing the finger to detect subtle changes.

1.3 Machine Learning

To model the normal direction, supervised machine learning will be used to train a regressive model. This is preferred over an analytical approach, as the machine learning models tend to execute faster in real time. Furthermore, after discussion with colleagues, modeling the dynamics of the fluid and the electrode/temperature parasitic interactions is highly complex and infeasible.
Two models were to be used for Machine Learning: Support Vector Regression, and a Neural Network.
For the latter, a multi-layered perceptron model was used, set up in such a way as to perform regression(rather than classification, a typical use of perceptrons). To illustrate this, figure 4 accurately shows the used architecture.

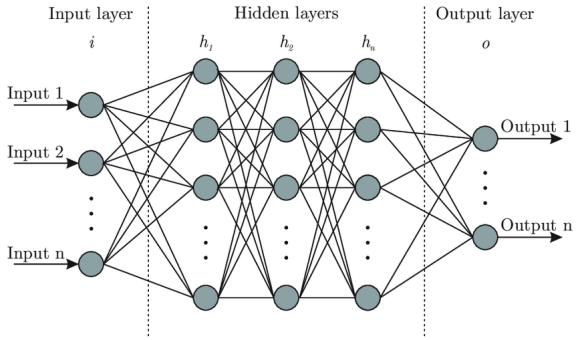

Figure 4: Multi-layered perceptron architecture

Notice how there is no recursion, nor cross-communication between non-adjacent layers of the network.

For the former, ϵ-SVR was chosen over v-SVR, since the size of the ϵ-tube was important. Likewise, an RBF model for the kernel was chosen. To show this, consider the transformation of the data in figure 5

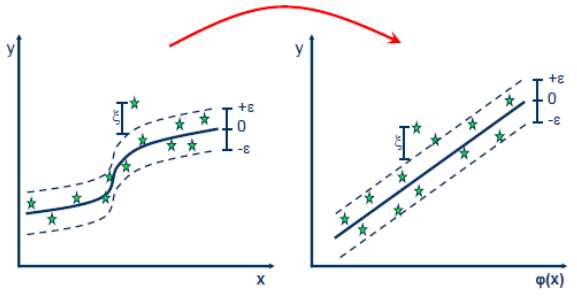

Figure 5: RBF kernel data transformation

For an arbitrary dataset, RBF is the safest choice. Similarly, the $\pm\epsilon$ tolerance was important on the scales of the outputs, thus the choice over v-SVR.

2 Data Capture & Set Up

2.1 Data Acquisition

To use supervised machine learning, a true direction of the normal is required in conjunction with the data from the Biotac finger. This was done using three ROS nodes working in tandem:

1. Orientation data from the room camera-capture system
2. Electrode data from the Biotac
3. A custom node capturing and interpreting the corresponding data at synchronous time

Two components were used for point 1: the Biotac itself, and a flat, reference surface for the true normal. The setup is described in 6.

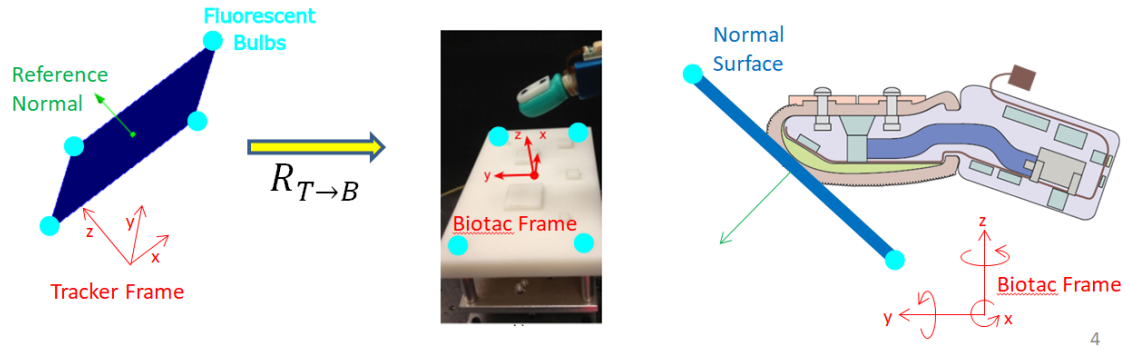

Figure 6: Via the bulbs, we can express the orientation of the reference normal in the Biotac frame. The reference normal pressed onto the surface of the finger is therefore the true normal.

Both these components were attached with fluorescent bulbs that the camera-tracking software could use to determine their location and orientation. By knowing the orientation of the reference surface relative to the Biotac, this provided the direction of the true normal. To mediate this transformation, the rotation matrix $R_{T\to B}$ is used to transform from the local reference frame of the reference normal to the Biotac frame. From here, the orientation data serves as the output of the machine learning regression, i.e. the data we wish to regress against.

Point 2. provides the corresponding data. Whenever the surface of the Biotac is pressed by the reference normal, the corresponding electrode activations are provided, alongside static pressure data, temperature, etc... This data will serve as the input data for the machine learning regression.

Point 3. serves to combine both sets of information such that the registered data is corresponding and synchronous(whenever a tactile change occurs, the data from the Biotac and camera system are saved at that same time for the same event).

2.2 The ROS Interfaces

The camera system and the Biotac had their own ROS packages and functions provided for data acquisition.
The node for the Biotac offers several services and (ROS)publications, the one of interest here is:

- */biotac_pub*

The latter contains all the sensory information detected in real-time, at a sample rate of 100 Hz. The information of interest are the 19 electrodes, and the static pressure sensor. The former was saved within an array of the */biotac_pub* data structure, and the latter as a floating value. Specifically, the electrodes were registered under *electrode_data* and the static pressure under *pdc_data*. Here, *pdc_data* measures the current actual pressure, as opposed to (and not to be confused with) *pac_data*, which measures the change in pressure.

The node for the camera system was instantiated twice: first for the physical placement for the Biotac, and second for the reference normal surface. As such, the data structure for both objects was identical. The cameras would sample data at 120 Hz. This is not the same as the Biotac which can cause oversampling of the data. This is dealt with in the next section(ROS custom Package).
For these nodes, the (ROS)publications of interest are:

- */vrpn_client_node/normalSurface/pose*
- */vrpn_client_node/baseHand/pose*

Each corresponds to the normal reference surface and the Biotac(respectively).
Withing the indicated */pose* data, there are two important aspects:

Firstly, the orientation information was given as 4 quaternions rather than angles (this avoids singularities and fully describes the local frame). These are found within the $pose.orientation.x, ..., pose.orientation.w$ variables, with x, y, z, w referring to the conventional quaternion constants.

Secondly, the orientations of the local frame are expressed in terms of the camera systems global frame(figure 7 shows this). This means that if we wanted to represent the vector of a single local axis, it would be composed of 3 coordinates ((x, y, z) directions) in the global frame. Therefore, a given local 3D frame has 9 coordinates corresponding to it in the global frame.
This is important as we wish to express the reference surfacs' frame in the Biotacs' frame. This means the relative rotations need to be transformed from the local reference surface frame twice: once to the global, and again to arrive at the local of the Biotac.

2.3 The ROS custom Package

This serves to combine all the previously mentioned information in a usefull data package. The aim is to capture the electrode sensor and static pressure data synchronously with with the orientation data from the Biotac and the reference normal surface. The node is automatically instantiated and executes its function by calling the following script:

- *biochip_listener.py*

The clock of the camera system and the Biotac are not the same (120 v.s. 100 Hz respectively). This was resolved via the approximate synchroniser functionality of ROS:

- $ApproximateTimeSynchronizer([samples], buffer\_length, time\_window)$

Any set of unique samples received within 2ms of eachother are saved as a data sample. Since the sources of new data are received asynchronously from 3 sources, consistency can only be guaranteed within a window of time. This may have led to some data collisions(periodic oversampling). In practice, the latter occurred very infrequently(on the order of once every 200-300 samples), and was deemed negligible. Since the incoming data was unique, the apparent clock frequency of the saved data was about 100Hz.
The electrode and static pressure data required no manipulation, and so was immediately saved once captured by the synchronisation window.

As mentioned in the previous section, the quaternion orientations are represented from a given local frame with respect to the global frame. To transform them into their rotation matrix equivalent, we use the following formula:

$$q = x\mathbf{i} + y\mathbf{j} + z\mathbf{k} + w, \;\; s = \|q\|^{-2}$$

$$R = \begin{pmatrix} 1 - 2s(y^2 + z^2) & 2s(xy - zw) & 2s(xz + yw) \\ 2s(xy + zw) & 1 - 2s(x^2 + z^2) & 2s(yz - xw) \\ 2s(xz - yw) & 2s(yz + xw) & 1 - 2s(x^2 + y^2) \end{pmatrix}$$

Note this rotation matrix is normalized. To interpret what this means, consider figure 7. The quaternions from the tracker frame are expressed in the global frame. Therefore, the rotation matrix equivalent is the matrix T_n. This means the first column represents the X-axis of the tracker frame expressed in the global frame, hence it has (x, y, z) components. By analogy, the same holds for the Y and Z axis of the tracker frame. We can further extend this reasoning for the Biotac frame, this gives us the matrix T_b.
Now we must refine what we mean by the true normal direction of the Biotac. When the reference normal surface is pressed against the finger, the direction perpendicular to that surface is the true direction of the normal. Considering figure 7, this true direction is the same as the Z-axis of the *Tracker* frame, since the X and Y axes are in the plane of surface. So we seek to represent this Z-axis in the *Biotac* frame. In order to do so, we must consider the rotation matrices between the 3 frames: T_n and T_b. Since the inverse of a rotation matrix is its' transpose, we can go from the tracker frame to the Biotac frame by multiplying the rotation matrices:

$$R_{T \to B} = (T_b)^T T_n$$

The latter matrix represents the X,Y, and Z axes of the tracker frame expressed in the Biotac frame. Since we only want the Z-direction(the true normal), we only want the 3^{rd}

column of this matrix. The custom ROS package performs these calculation in real time with the quternion information, and saves the 3^{rd} column of $R_{T\rightarrow B}$ in the data package. Overall, we save the electrode and static pressure information with the corresponding true normal.
The full architecture of the custom ROS node is shown in figure 8.

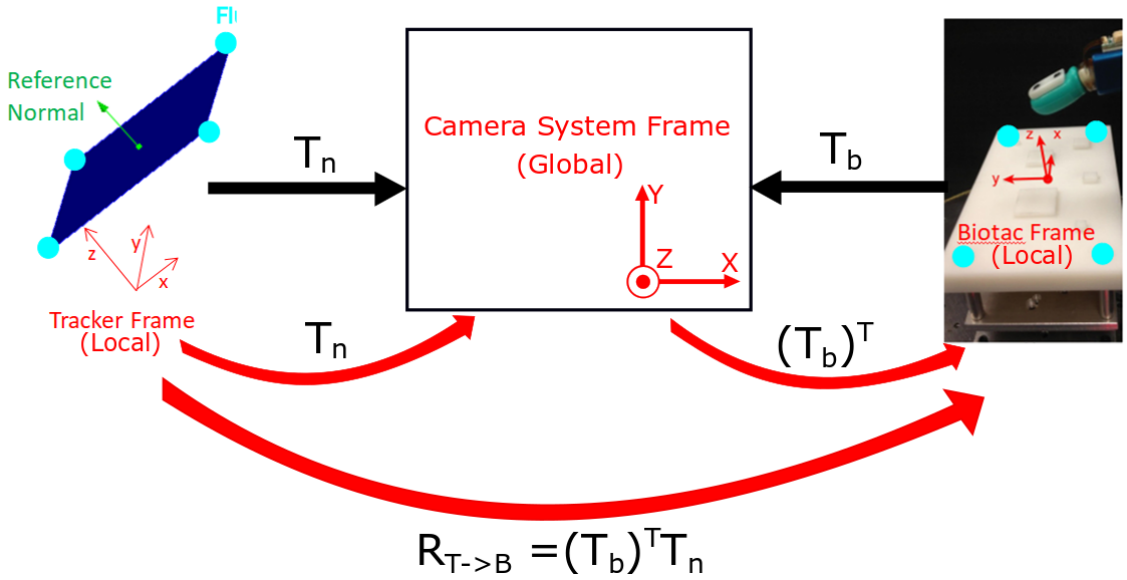

Figure 7: T_n rotates the tracker frame to the global frame, and similarly with T_b from the Biotac frame. Overall, $R_{T\rightarrow B}$ goes from the tracker frame to the Biotac frame.

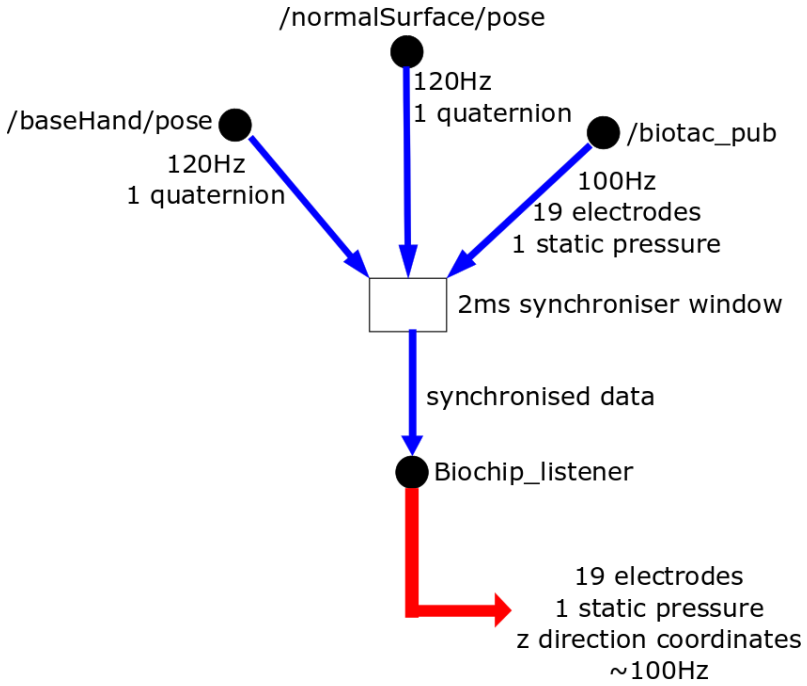

Figure 8: The global architecture of the custom ROS package. Note that the synchronisation window is part of the package, but is separated in this image for clarity. From 3 asynchronous sources, the node extracts and saves electrode data with the corresponding true normal(written as z direction, referring to the z-axis of the tracker frame in figure 7).

3 Data Processing

3.1 Rectification & pre-processing

Two experiments were conducted in total, with the data saved in standard ROS .log files. The raw data itself itself is exemplified in figure 9.

```
[rosout][INFO] 2019-10-16 18:16:16,116:
pdc:
2853
electrode:
(3059, 3368, 3422, 3313, 3406, 3189, 3013, 2975, 3127, 3065, 3154, 3490, 3484, 3360, 3426, 3196, 3237, 3413, 3200)
z_dir:
[[-0.27342523]
 [-0.14213093]
 [ 0.95133456]]
```

Figure 9: Example of the raw data format of the information gathered by the custom ROS package.

The first line, $[rosout][INFO]$..., is from the ROS environment and serves as a usefull indicator between samples. Likewise, *pdc* :,*electrode* :, and z_dir : allows a script to distinguish between the static pressure sensor, the electrode data, and the true normal direction, respectively. In total, about 14,000 samples were taken for each experiment.
After running it through a script, it selectively extracts the data in a convenient form, the format to represent it is shown in table 1.

PDC	Electrode_1	...	Electrode_19	z_dir_x	...	z_dir_z
1,1	1,2	...	1,20	1,21	...	1,23
...	...	...	...	...	...	...
n,1	n,2	...	n,20	n,21	...	n,23

Table 1: Transformation of the raw data into a usable form. The numbers in each cell are the indices of a sample, with n being the total number of samples($\approx 14,000$).

To interpret the table, consider some arbitrary row: it is the sample. The first 20 columns are the static pressure and electrode information corresponding to the true normal of the last 3. This difference is important, as the last 3 columns serve as the output variables of machine learning, with the first 20 as the inputs.

During the experiment, it was agreed that at the beginning and end, the true normal surface would be completely removed as to make it distinguishable in the data. As such, the extrema rows(first few and last few) would be completely removed. This was done visually, by observing the plot of PDC(static pressure sensor data) in table 1.
Even so, there still existed rows of no contact, however these were intentional. By knowing when the Biotac was not in contact, an intrinsic offset to the electrodes could be removed. This makes sense, as there is still a default value that the sensors will measure when nothing is in contact. This is important as it centers the data for machine learning. Since all the inputs were in the same scale(electrodes in mV), it was not necessary to normalize them. The components of z_dir had no inherent offset since they represent a direction. Contrary to the electrodes, the quaternion transformation step already normalised the direction, so $\|\vec{z}_{dir}\| = 1$. The choice for the latter will be explained towards the end of this section.
At this point, the electrodes and static pressure(PDC) have been centered and the null extremas removed. Before the last step, it is important to remember that we want the

machine learning algorithm to start predicting $\vec{z}_{dir}$ only when something is physically in contact with the Biotac. When it is not in contact(the static pressure sensor is equal to its offset), this is a separate case where we do not want to use our algorithm at all. This case separation will be done in the final ROS package rather than in the machine learning algorithm.
Finaly, we use the static pressure values to determine where there was no contact, and these gaps are removed from the data.
The remaining rows therefore represent moments in time when the Biotac is in physical contact with the true normal.
To prepare the data for machine learning, the cleaned data from both experiments was concatenated, the inputs were separated from the outputs, and two sets of data were removed: the static pressure data(the first column in table 1) and the Y-component of $\vec{z}_{dir}(22^{nd}$ column). The former isn't relevant for learning, since we want to predict using only the electrodes. The latter is a consequence of $\|\vec{z}_{dir}\| = \sqrt{x^2_{z\_dir} + y^2_{z\_dir} + z^2_{z\_dir}} = 1$: by knowing two of the three (x,y,z) components, the third can be deduced. Therefore we only need two out of the three components to keep the outputs deterministic. We can choose any of the three, and it was arbitrarily chosen to remove the y-component.
The full process of pre-processing is shown in figure 10, and the formated data for machine learning in figure 11. In total, there are 9000 samples of concatenated data after pre-processing.

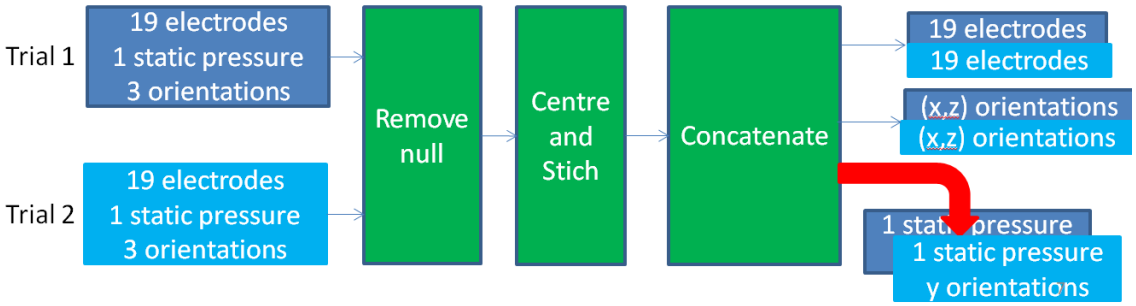

Figure 10: The full pre-processing of the data to prepare it for machine learning. The red arrow indicates discarded data in the final step.

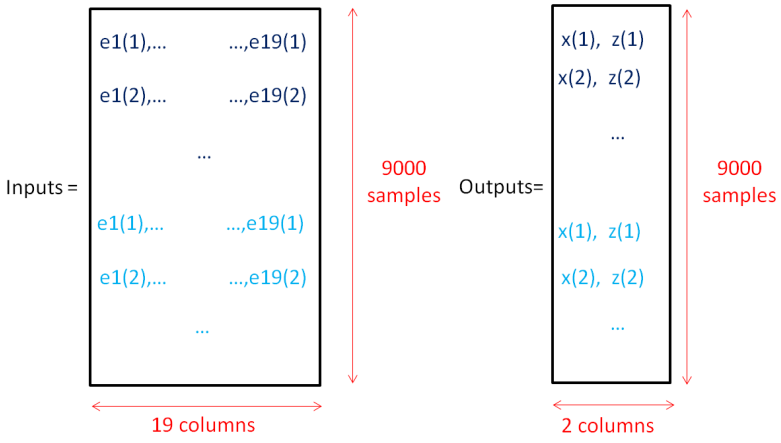

Figure 11: The concatenated data format: in dark blue is the data from experiment one, and light blue from experiment 2.

3.2 Support Vector Regression

The important hyperparameters for this type of regression are the tolerance tube ϵ, the penalty cost C, and the Kernel width σ. It should be noted that the python library used γ rather than σ with $\gamma = \frac{1}{2\sigma^2}$.
Since SVR is multi-input-single-output, two sets hyperparameters for 2 models had to be determined: one for predicting the x components, and one for the z components.
To find and verify them, a grid searched 5-fold cross-validation approach was used. 75% of the data was used for training and 25% for final validation. This is presented in figure 12.

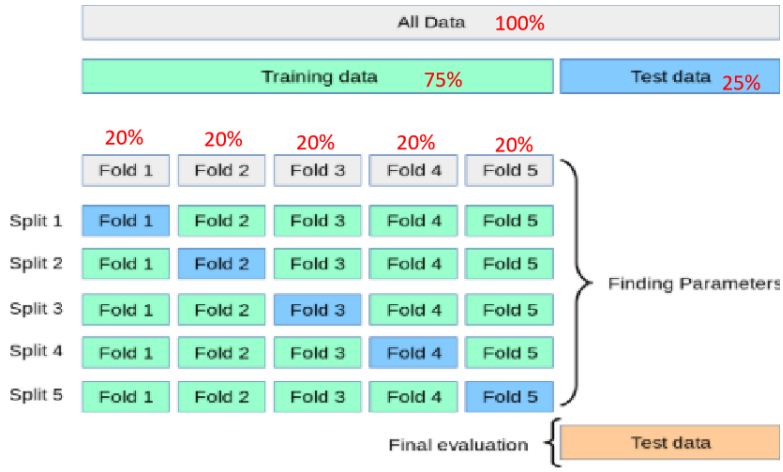

Figure 12: The 5-fold cross-validation technique used to find and verify a given combination of hypoerparameters.

This method was repeated for all combinations of the hyperpearameters, and depending on the AIC values, the combination that minimized it was tested against the Final evaluation 25% test data. In that, AIC is defined here as follows:

$$AIC = -\alpha * ln(R^2) + P * N^2$$

Where R^2 is the corelation coefficient, P is the number of support vectors, and N is the dimensions of the input data(19). α was manually tuned to find a satisfactory tradeoff.
Its important know that this definition of AIC is not intended to be compared to that for Neural Networks, but rather as an indicator to how useful an optimal model is, we seek to minimize it as much as reasonably possible. It is to be understood in a relative fashion.
The hyperparameters were gridsearched between:

- $\epsilon \in (0.05, 0.2)$
- $C \in (1, 5000)$
- $\gamma \in (1 * 10^{-5}, 1 * 10^{-4})$

Before the results are presented, it's important to mention that the number of support vectors should be no more than 30-40. This choice was imposed by the supervisors. For the x direction, the best model had $\epsilon = 0.3, C = 5, \gamma = 1 * 10^{-5}$ with $R^2 = 0.71, P = 45$ on the total training data, and AIC = 21,312. In the final validation data this became $R^2 = 0.68, P = 51$. For greater accuracy, the number of support vectors increases dramatically,

as shown in figure 13. Immediately we can see that this is almost double the desired number of support vectors, for a mediocre accuracy of 0.68. Furthermore, a tolerance of 0.3 is far too large for a normalised x-axis with $x \in (-1, 1)$, at most it should be 0.1. Similar results are in the z direction, $\epsilon = 0.15, C = 70, \gamma = 1 * 10^{-5}$ with $R^2 = 0.69, P = 80$ on the total training data, AIC = 35,290, and $R^2 = 0.68, P = 81$ on final validation. Now the ϵ is improved, but not enough, with more than double the imposed value of P. The full visualisation of the performance is shown in figure 14 for the x direction, and 15 for the z direction.

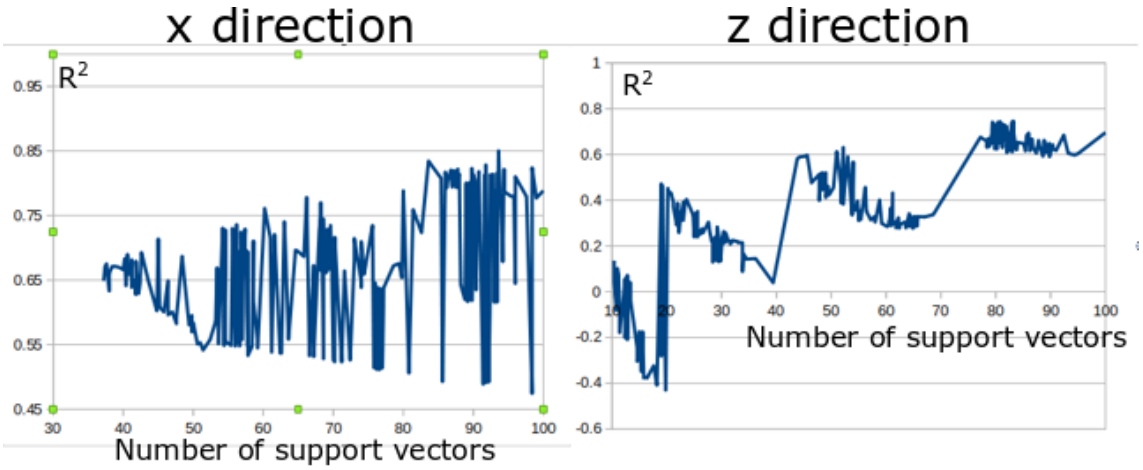

Figure 13: The plots of R^2 and P for the x and z direction for all combinations of the hyperparameters.

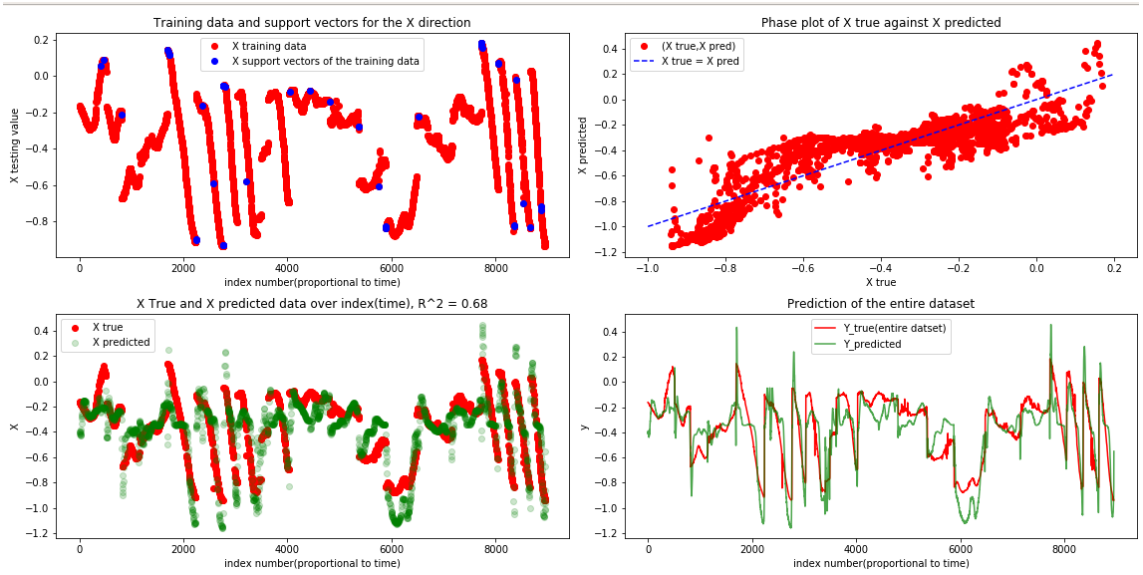

Figure 14: Performance plots for SVR along the x direction(note the number of support vectors appear few, but along the dimensions of the electrodes they are much more, this visual serves to summarize).

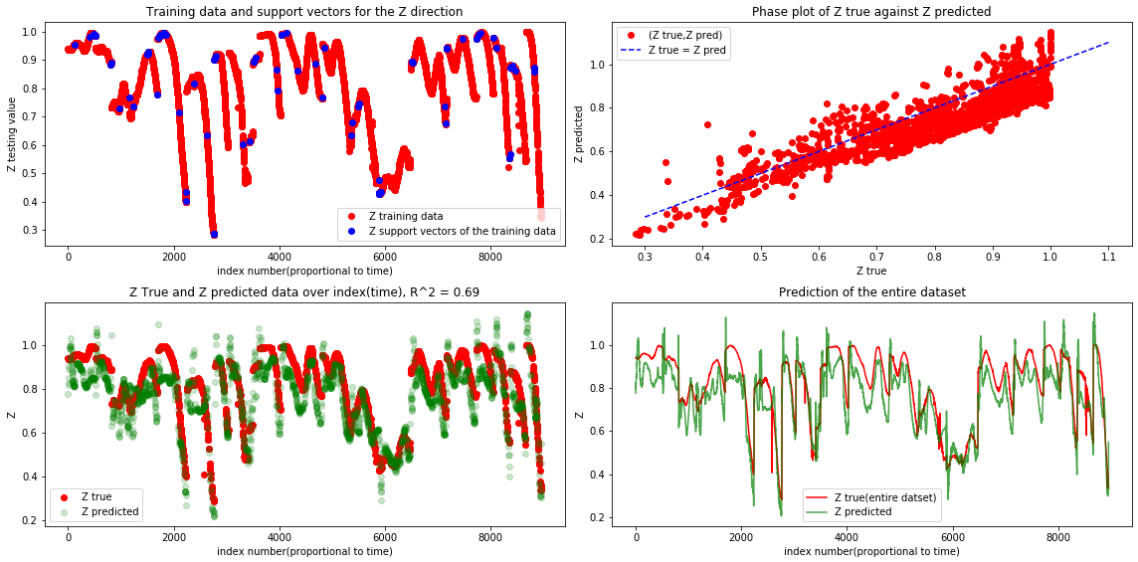

Figure 15: Performance plots for SVR along the z direction(note the number of support vectors appear few, but along the dimensions of the electrodes they are much more, this visual serves to summarize).

3.3 Multi-Layered Perceptron Neural Network

In this case, the hyperparameters are the number of the number of layers L, the number of nodes per layer N, and the learning rate η. It should be noted that the backpropagation procedure was directed to use an adaptive learning rate, meaning that the learning rate is divided by 5 if the loss does not change between 2 epochs. This allowed for faster convergence. The neural network was set up with 19 inputs(one for each electrode), and 2 outputs(one for the x component, and the other for z). In doing so, a single machine learning setup can be run in ROS rather than the 2 for SVR(since each model can only give one output, so 2 are required for x and z). The same 5-fold cross validation technique was also used here as mentioned in section 3.2, with the reasoning of figure 12.
The definition of AIC is:

$$AIC = -\alpha * ln(R^2) + L * N$$

Similar with SVR, this definition is to be interpreted relatively, and to be minimized as much as possible, not as a comparison to the SVR definition. Again, α was tuned manually. The hyperparameters were gridsearched between:

- $L \in (1, 10)$
- $N \in (5, 100)$
- $\eta \in (0.0001, 0.05)$

There were no particular imposed restrictions in this model, only that the total number of neurons should remain under 100. The best model had $L = 9$, $N = 10$, $\eta = 0.015$ with $R^2 = 0.95$(average across x and z directions), and AIC $= 90$ on the total training set. On final validation, $R^2 = 0.91$. To check the stability of the neuron weights, this model was retrained over 100 times to see how R^2 varies. The boxplot in figure 16 shows that in the worst case, the average R^2 is well above mediocre. Despite being within the restriction, 90 neurons is uncomfortably close to the limit. The performance visualisations are shown in 17 for the x direction, and 18 for the z direction.

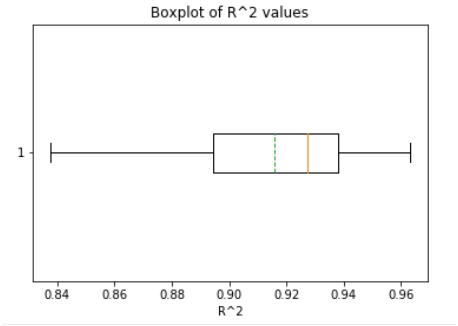

Figure 16: Stability of R^2(average of x and z directions) for the convergence of the weight of the neural network. Green is the mean, Yellow the mode.

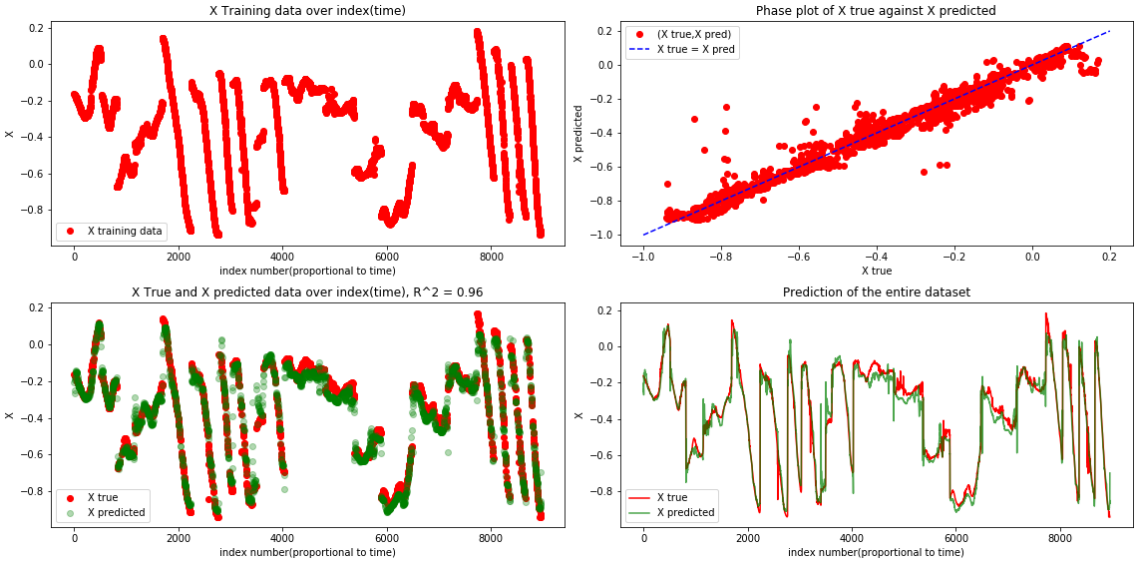

Figure 17: Performance visualisation of the neural net along the x direction.

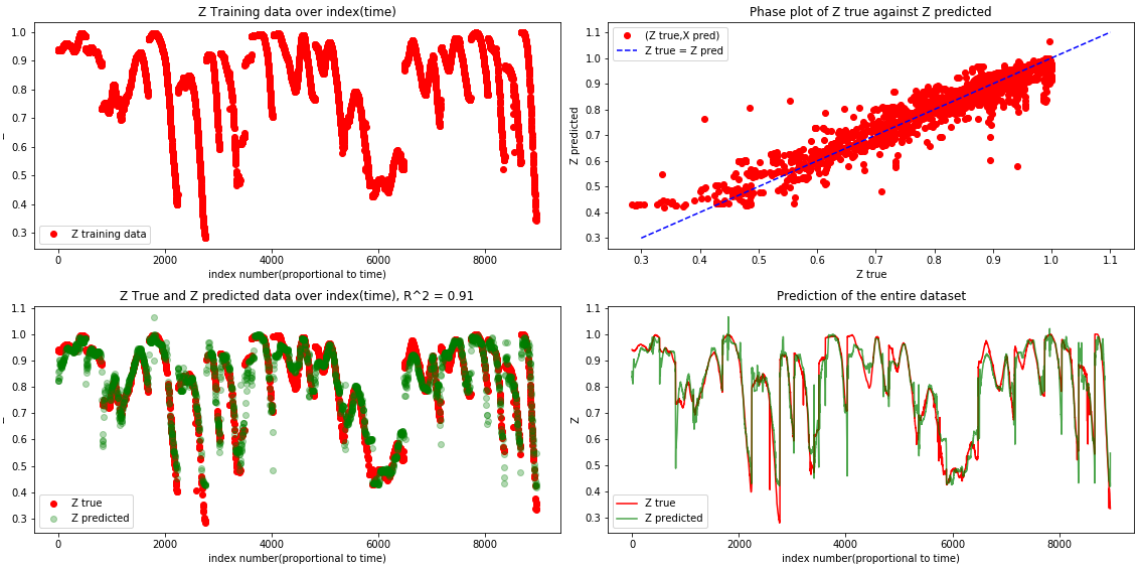

Figure 18: Performance visualisation of the neural net along the z direction.

3.4 Choice of method

Between the two methods, SVR simply does not satisfy the imposed constraints, so necessarily the neural network is the only available choice. However by this point, the situation becomes more complicated. This was overlooked during the training stages of the models, but the data was randomly shuffled during the 5-fold cross-validation. This is a problem since the model is suppose to be predictive with respect to time, so if it is training randomly, then the form of the training data will be almost identical to the final validation data. This means that it was impossible to tell if the models were learning or overfitting. To deal with this quickly, some assumptions were made:

- The data from both trials are equally as rich in their statistics.
- Any decrease in the train/test ratio would negatively impact model complexity.

As such, the model would train exclusively on one of the two trials, and would predict on the other. This means that the new train/test ratio would be 50% over unshuffled data, as opposed to the previous 75% on shuffled data.
Since SVR already couldn't satisfy the constraints, the decrease in the train/test ratio only serves to increase the number of support vectors to attain the same R^2 in section 3.2. Conversely, the neural network still had a margin in its restraints and allowed for another opportunity to further refine it. As such, a new grid search was run on the hyperparameters:

- $L \in (1, 10)$
- $N \in (1, 10)$
- $\eta \in (0.01, 0.02)$

This time, α of the AIC was pushed in favour of model simplicity.
The optimal results were for $L = 7$, $N = 5$, $\eta = 0.017$ with $R^2 = 0.86$(average across x and z directions), and AIC $= 35$ on the total training set. On final validation, $R^2 = 0.83$. The model was successfully improved in its complexity with an above mediocre performance. The performance visualisations are shown in 19 for the x direction, and 20 for the z direction. However we can see that both the x and z directions seem to saturate between two bounds, though this is unfortunate, this was the optimally determined model for 500 train/retrain loops for the set of optimal candidates.

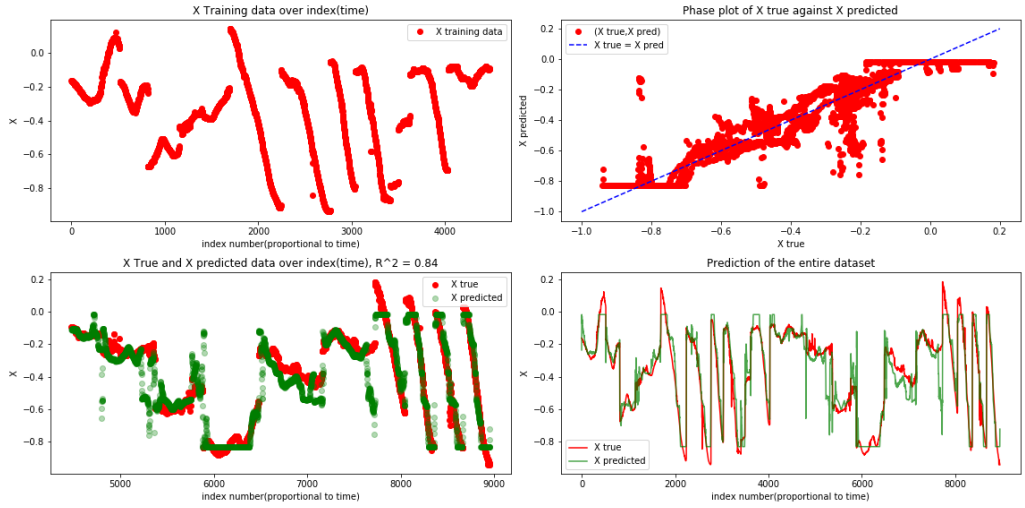

Figure 19: Performance visualisation of the new neural net along the x direction.

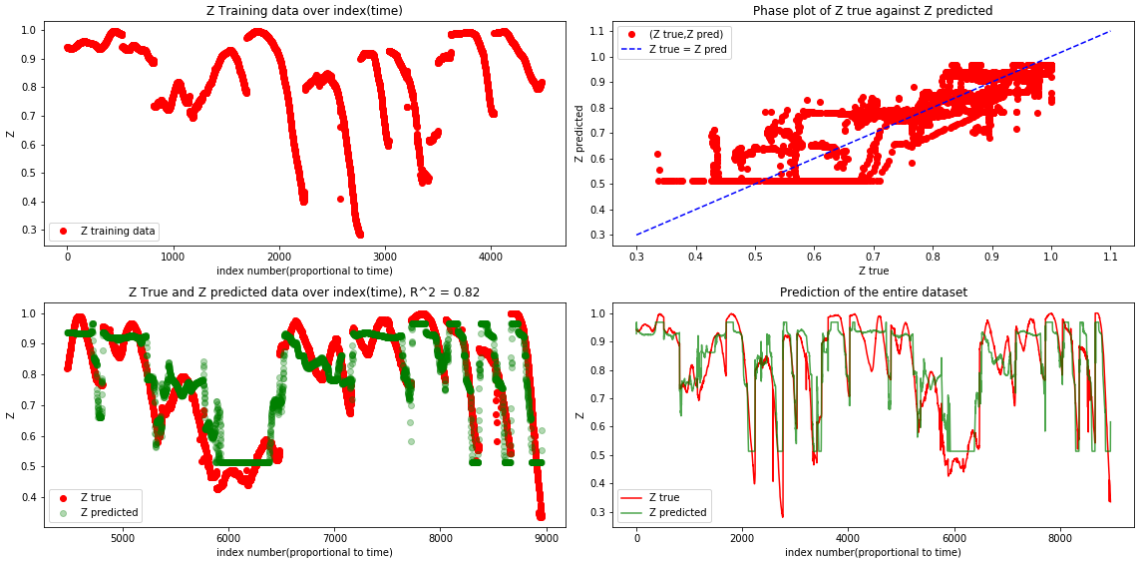

Figure 20: Performance visualisation of the new neural net along the z direction.

4 Implementation

4.1 The Biotac_NN ROS package

It was decided that the a service/client approach to be used for running the neural network, meaning that a new ROS package was developed. This was the Biotac_NN package which was the service node to the adapted Biochip_listener client node. The latter was originally the one used in section 2.3, but now has been added two extra functionalities:

1. A calibration mode to calculate the offset values of the electrodes.
2. A binary case function that determines if the Biotac is in contact with a surface.

Point 1. serves to continuously remove the offsets in real time so that the neural network can directly give a $\vec{z}_{dir}$. The moment the Biochip_listener node is instantiated, the first command in $\_\_main\_\_$(the first directive to be executed) creates a temporary subscriber node to */biotac_pub* that loops 500 times(5 seconds of calibration) and takes the average of all observed electrode and static pressure values as the offsets. These values are then stored in a global variable. As such, the Biotac must <u>not</u> be touched during the 5 five seconds this node is instantiated.
Point 2. serves to give a default value when nothing is touching the Biotac. For this, anything below the static pressure offset + 5% will automatically return $\vec{z}_{dir} = (0, 0, 0)^T$ rather than request the service of the Biotac_NN node.

When the Biochip_listener node is detecting contact, a service request can be made to the Biotac_NN with the offset-removed electrode data. When the latter node is first instantiated, it creates a neural network model with the weights and architecture described in section 3.4. Upon a service request, the node takes the sample of electrode information and runs it through the network to give the x and z directions. Finally, using $\|\vec{z}_{dir}\| = \sqrt{x^2_{z\_dir} + y^2_{z\_dir} + z^2_{z\_dir}} = 1$, the node calculates the y direction and services the Biotac_NN with $\vec{z}_{dir}$.
The full architecture of the final setup is shown in figure 21.

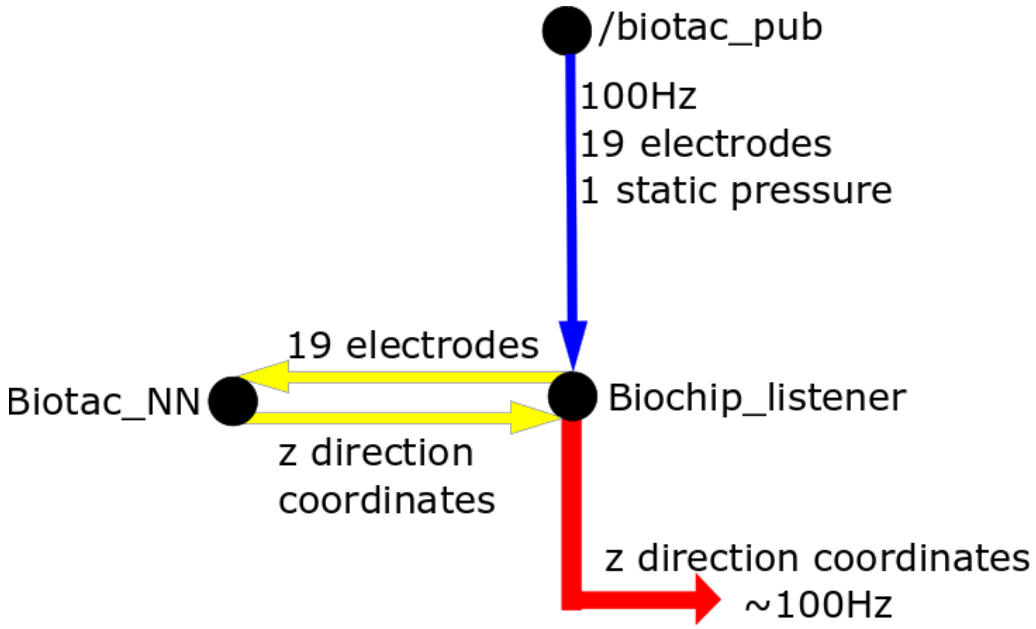

Figure 21: Complete architecture for the Biotac to predict the normal directions when a surface comes into contact with it.

5 Conclusion

The overall goal of this project was successfully achieved despite some setbacks. Furthermore, the neural network model was dramatically reduced in complexity along the way. Despite this, there are some notable issues: when running the completed ROS packages, the $\vec{z}_{dir}$ values would noticeably drift over the course of a few minutes. This happened in two circumstances:

1. If the nodes where instantiated soon after the Biotac was first turned on.
2. If the Biotac was pressed against a surface that conducted heat very well

This is a consequence of thermal drift, which was described in the Biotac documentation. It seemed the change in internal temperature would change the true offset values and thus produced skewed direction for $\vec{z}_{dir}$.
Point 1. produced the worst case scenario. When the Biotac isn't touching anything, the threshold of the static pressure + 5% would gradually cause $\vec{z}_{dir}$ to jump between $(0, 0, 0)$ and some other random value. What was happening is the thermal drift moving the offset above the threshold, giving useless values for $\vec{z}_{dir}$.
Point 2. somewhat returned to normal when removed from a thermally conductive surface, and the skew of $\vec{z}_{dir}$ would slowly disappear.
The most reliable way to use it was to allow the Biotac to warm up when it was powered on, 20 minutes to be sure, and to then press it against non-thermally conductive surfaces to minimize the effect of the $\vec{z}_{dir}$ skew over time.