The Open University of Israel
Department of Mathematics and Computer Science

Error signals for adaptive neuro-robotics

Research proposal for thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
Department of Mathematics and Computer Science
The Open University of Israel

by

Elad Chamilevsky

Prepared under the supervision of
Dr. Elishai Ezra Tsur, The Open University of Israel

1 Project Abstract

Adaptive control schemes are an effective way to deal with uncertainties in all robot dynamic models in which the controllers are designed to be adjustable so as to automatically compensate for these uncertainties. Here, we propose a neuromorphic algorithm which calculates the error signals for adaptive neuro-robotics in 6 Degrees of Freedom (DOF). Our implementation will be integrated with two greater tasks: 1. neuromorphic adaptive control in a wheelchair-mounted 6-DOF robotic arm, which will be clinically evaluated at ALYN Hospital, Israel's most advanced pediatric and adolescent rehabilitation center. 2. stabilize a spider-bot walking on diverse terrain patterns.

2 Background

Robotics technology influences every aspect of our everyday life. Robotics has the potential to raise efficiency and safety levels and provide enhanced levels of service. Even more, robotics is set to become the driving technology underpinning a whole new generation of autonomous devices and cognitive artefacts that, through their learning capabilities, interact seamlessly with the world around them, and hence, provide the missing link between the digital and physical world. Robotics can be seen in a wide variety of fields, e.g. industrial machines applications[1], medical and surgical robotics[2], military robotics[3, 4], space robotics[5] and even spare time gadget robotics[6, 7].

One of the interesting issues that robotics deal with is the adaptive stabilization issue. As mentioned before, the use of robotics is diverse and stabilization is a critical attribute to all applications, e.g. surgical robot must be stable or it can cause harm to the patient on the table, and humanoid robot must stabilize itself to prevent from falling when walking or standing on harsh ground. This task can be classified as adaptive control task since the adaptation to a controlled system handles parameters which vary, or are initially uncertain[8].

The adaptive control branch of control theory offers solutions to deal with robotic manipulators subject to parameters variation and abrupt changes in the dynamics. Within adaptive controllers, two main categories can be identified: the model reference adaptive systems, and the self-tuning regulators. The first technique being studied for robot manipulators was the model reference adaptive control (MRAC)[9]. The idea behind this technique is to derive a control signal to be applied to the robot actuators which will force the system to behave as specified by a chosen reference model. Furthermore, the adaptation law is designed to guarantee stability using either Lyapunov theory or hyperstability theory[10]. The other most common approach for robot control is the self-tuning adaptive control[11], in which the matrices of mass, centrifugal, Coriolis and gravity are used to compute the control torque. The parameters of these matrices are then adapted to achieve trajectory tracking. The main difference between this technique and the MRAC is that the self-tuning approach represents the robot as a linear discrete time model and it estimates online the unknown parameters, substituting them in the control law. The literature for adaptive control of robot manipulators shows the ability of these techniques to perform well in presence of uncertain dynamics and varying payloads. Having said that, the complexity of the controller increases usually with increasing number of DOF.

Various methods have been proposed for designing an adaptive control controller such as fuzzy logic[12], neural network[13], internal model control (IMC)[14] and center of mass (COM) control[15]. However, these methods are quite complicated and/or difficult in the actual implementation. Among various control techniques, the proportional-integral-derivative (PID) control scheme is the most employed in the industries due to its simplicity and many industrial PID controller products are readily available in the market. The PID controller parameters can be tuned by both analytical designs and experimental tuning rules. However, in several practical cases, the controller should be able to take into account kinematic changes in the system, such as object manipulation of an unknown dimension or at an unknown gripping point.

The conventional way to attack this issue is by using additional data. One way to get this data is by using an inertial measurement unit (**IMU**) sensor to reconstruct the robots orientation based on gravity and inertial effects[16, 17]. Most IMUs have two main types of motion sensors: Accelerometer and Gyroscope. The acceleration measured by the accelerometer describes the exact direction of gravity vector (on each plane x, y and z, hence 3-DOF). The acceleration samples are given along time and by integrating them it is possible to point the exact location of the object. The angular velocity measured from gyroscope defines exact changes in orientation (roll, pitch and yaw, hence 3-DOF). By receiving the integrated acceleration, the current orientation and a target state a correction signal can be calculated as the inverse calculation between the target and the current state in each dimension. Another approaches are using of an approximate Jacobian matrix (the prominent

part of the control scheme may be the approximate transpose Jacobian control with/without a kinematic parameter adaptation law)[18], or applying neural network methodology to infer those parameters[19].

Our lab is interested in neuro-robotics i.e. we wish to understand how real neural systems deals with a certain task and use this knowledge to implement a neuromorphic algorithms which can applied to robotics using the neuromorphic hardware. the neuromorphic methodology is highly suitable for the development of intelligent robots because it is better able to address the power consumption, which is especially important for mobile robots, and the practical implementation of perceptive and cognitive capabilities, which currently often requires a large, non-mobile computer cluster. Neuromorphic robots carry out perceptual tasks on raw data with a much higher degree of power efficiency than that of current computers. This reduced energy consumption is one of the most important driving forces behind the development of neuromorphic robots, and can be attributed to following: 1.Memory and computing are integrated, which avoids the huge energy consumption associated with data transfer. 2.Sparse spatiotemporal and spike-based coding minimizes communication overhead. 3.computation is only performed when necessary due to an event-driven mode of operation, and 4.novel passive devices store synaptic weights and memory with almost no power consumption[20]. In addition, neuromorphic systems have been shown to outperform PID-based implementations of the required non-linear adaptation, particularly, in their ability to handle a high DOF systems.

In the current work we wish to implement an adaptive algorithm to calculated error signals which enables adaptive control tasks for neuro-robotics. The algorithm integrates the acceleration of the gravity vectors and calculates the correction signal in both gravity and orientation axes (6-DOF). To do so we use the Neural Engineering Framework.

The Neural Engineering Framework (NEF [21]) is a general methodology that enable building large scale, biologically plausible, neural models of cognition. This can be thought of as a "neural compiler" where algorithms written in a high-level language are converted into neurons with connections between them. When using the NEF, you do not get an exact implementation of whatever algorithm you specify. Instead, the neurons approximate that algorithm, and the accuracy of that approximation depends not only on the neural properties but also on the functions being computed. That is, instead of using any computations at all in your model, the NEF forces you to use the basic operations that are available to neurons. This allows us to make strong claims about the classes of algorithms that could not be implemented in the human brain (given the constraints on timing, robustness, and numbers of neurons involved)[22].

NEF proposes three quantitatively specified principles that enable the construction of large-scale neural models:
(1) **Representation**: The NEF uses distributed representations. It makes a distinction between the activity of a group of neurons and the value being represented, which is usually thought of as a time-varying vector x. For example, you may use 100 neurons to represent a two-dimensional vector. Different vector values correspond to different patterns of activity across those neurons. To map between x and neuron activity, every neuron has an encoding vector. This can be thought of as the preferred direction vector for that neuron: the vector for which that neuron will fire most strongly. This fits with the general neuroscience methodology of establishing tuning curves for neurons, where the activity of a neuron peaks for some stimulus or condition. NEF makes the strong claim that the input current to a neuron is a linear function of the value being represented. The neural non-linearity can be any neural model, including simple rate-based sigmoidal neurons, spiking Leaky-Integrate-and-Fire neurons, or more complex biologically detailed models as long as there be some mapping between input current and neuron activity, which can include complex spiking behaviour. It is also important to go the other way around, e.i. given some neural activity, to decode the represented value. The simplest method is to find a linear decoder, which is a set of weights that maps the activity back into an estimate of x. Finding this set of decoding weights is a least-squares minimization problem, as we want to find the set of weights that minimizes the difference between x and its estimate. Having this decoder allows to determine how accurately a group of neurons is representing some value, and provides a high-level interpretation of the spiking activity of the group. Importantly, it also turns out that these decoders also allows to directly solve for the neural connection weights that will compute some desired transformation.
(2) **Transformation**: To do anything useful, neurons need to be connected together. One can not just connect the first neuron in population A to the first neuron in population B, because not only there might be different numbers of neurons in the two groups, with different parameters, but the neural non-linearity makes this naïve approach highly inaccurate. To create this connection in an accurate and robust manner, let us first assume that we have an intermediate group of perfectly ideal linear neurons, connected to both groups, and we have one of these for each dimension being represented. Given the activity in A, the

decoder is exactly the set of connection weights needed to compute x. Furthermore, using the encoder values for group B as connection weights from this x representation is exactly what would in turn cause group B to represent x. The intermediate group can be eliminated by directing group A directly to group B using the connection weights found by multiplying the two sets of weights (the decoder of group A and the encoder of group B). That is, the optimal weights to pass the information from A to B. This approach is not limited to simple functions, this means that nonlinear functions can be computed with a single layer of connections – no back-propagation of error is required. This includes not only the classic XOR problem, but also more complex functions such as multiplication, trigonometric functions, or even circular convolution. That said, it cannot compute any function, and in general, the more non-linear and discontinuous the function is, the lower the accuracy. This accuracy is also affected by the neuron properties and the encoding method used.
(3) **Dynamics**: NEF provides a direct method for computing dynamic functions. A particularly useful special case of this equation is $dx/dt = u$, an integrator. This is a neural system that, if given no input (u=0), will maintain its current state (since $dx/dt = 0$). Given a positive input, the stored value will increase, and given a negative input it will decrease. This sort of component appears in many models of working memory and in accumulator models of decision making. Building this system requires you to know the neurotransmitter time constant τ of a recurrent connection from the neurons in the group back to themselves. This time constant is a standard property of biological neuron models, and reflects how quickly the neurotransmitter released by a spike is reabsorbed. It varies widely across different types of neurons, from 2ms up to 200ms. For the special case of the integrator, the function can be computed in a feedback connection computing exactly the same identity function $f(x) = x$ as in the simple model that passed information from group A to group B, only here the neurons are passing that information back to themselves.
Figure 1 provides a graphical summary of these three principles.

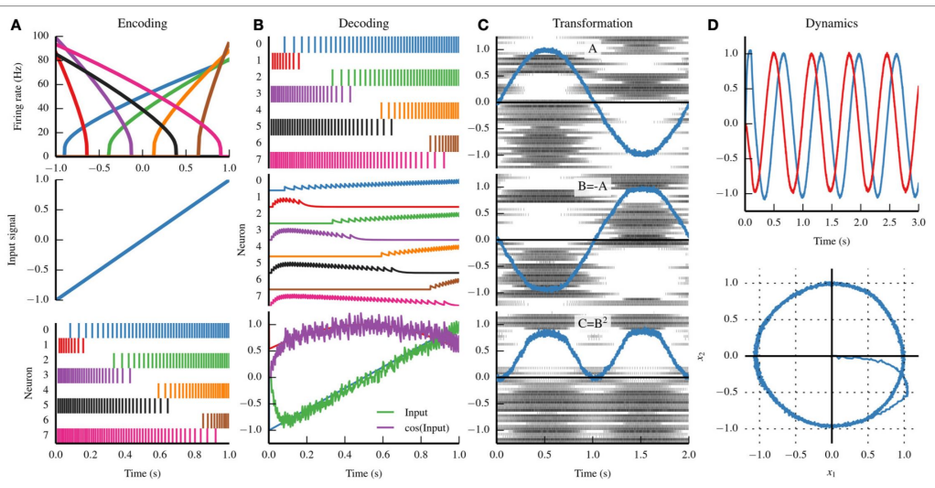

FIGURE 1 | Summary of the three principles of the Neural Engineering Framework (NEF). (A) By the representation principle, signals are encoded in neural populations based on the *tuning curve* of each neuron (top). The tuning curve describes how active a neuron is given some input signal. If we drive the eight neurons in the top panel with the input signal in the middle panel, we see the spike trains in the bottom panel. **(B)** By the representation principle, the spiking activity of a neural population can be decoded to recover the original input signal, or some transformation of that input signal. First, the firing pattern shown in the top panel is filtered with a decaying exponential filter (middle panel). The filtered activity is then summed together with a set of weights that approximates the input signal (bottom panel, green) and the cosine of the input signal (bottom panel, purple). **(C)** A sine wave is encoded by population A (top panel); the negative of that signal is projected to population B (middle panel) and the square of that signal is projected to population C (bottom panel). By the transformation principle, populations of neurons can send signals to another population by decoding the desired function from the first population and then encoding the decoded estimate into the second population. These two steps can be combined into a single step by calculating a set of weights that describe the strength of the connections between each neuron in the first population and each neuron in the second population. **(D)** A neurally implemented dynamical system has negative feedback across its two dimensions, resulting in a harmonic oscillator. The oscillator is plotted across time (top) and in state space (bottom). By the dynamics principle, signals being represented by population of neurons can be thought of as state variables in a dynamical system.

Figure 1

According to those principles, given the properties of the neurons, the values to be represented and the functions to be computed, the NEF solves the connection weights between components such that the desired function will be performed best. Importantly, this works not only for feed-forward computations, but recurrent connections as well. In other words, The NEF is a large-scale modeling approach that can leverage single neuron models to build neural networks with demonstrable cognitive abilities. It provides principles to guide the construction of a neural model that incorporates anatomical constraints, functional objectives, and dynamical systems or control theory.

Studies discovered that even when a standard mathematical implementation of a theory does not match empirical results, the NEF model of that theory may do a much better job. Following those discoveries, one can use the NEF with the software package Nengo

[23], an open-source cross-platform Java application which implements the NEF. Nengo is both a teaching tool (with hands-on classroom demos) and a research tool. Neural groups can be created through a drag-and-drop interface or Python scripting. The functions to approximate are similarly specified, with Nengo automatically computing the optimized connection weights. Visualization interface is also included, with which viewing and interacting with running models is possible, including support for simulated environments and physical robotics.

3 Project Description

The design of the motion control for robot manipulators has attracted considerable attention. A number of either model-based or model-free control scheme have been proposed. In the first case, different control schemes have been often used. However, a robot manipulator is a complex nonlinear system, whose dynamic parameters are difficult to forecast precisely. In fact, it is almost impossible to obtain exact dynamic models because of such uncertainties as nonlinear frictions and flexibility of the joints and links of robot manipulator.

The proportional-integral-derivative (PID) control scheme is the most employed in the industries due to its simplicity and many industrial PID controller products are readily available in the market. The PID controller parameters can be tuned by both analytical designs and experimental tuning rules. However, in several practical cases, the controller should be able to take into account kinematic changes in the system, such as object manipulation of an unknown dimension or at an unknown gripping point. Neuromorphic systems have been shown to outperform PID-based implementations of the required non-linear adaptation, particularly, in their ability to handle a high DOF systems.

In the current work we wish to implement an adaptive algorithm to calculated error signals which enables adaptive control tasks for neuro-robotics. We will use an IMU sensor to get the current acceleration and orientation of the robot. The current location in space will be integrated from the acceleration of the gravity vectors and then, using the current state of location and orientation, the algorithm will calculate the correction signal in both gravity and orientation axes. Here we aim to implement our solution using the Neural Engineering Framework, which to our knowledge, is the first time to do so.
In this proposed research we aim to:

1. Deploy the error signal algorithm on Intel's Neuromorphic Research Cloud (NRC), enabling close to real-time execution. We aim to connect the NRC-deployed error signal algorithm to our custom designed blender/python-based simulation framework, in order to evaluate the performance of the algorithm in a virtual environment. We will consider real-time execution of control dynamics as a successful outcome for this aim.

2. Deploy our solution on Intel's Loihi chip and integrate our algorithm with a neuromorphic nonlinear adaptive control algorithm which will be deployed to a 6-DOF robotic arm. The robotic arm will be clinically evaluated at ALYN Hospital, Israel's most advanced pediatric and adolescent rehabilitation center. Being able to provide stable and precise object manipulation will be considered a successful outcome.

3. Simulate our solution on the NVIDIA Xavier board to stabilize a spider-bot when placed on diverse terrain patterns.

4 Research Plan

Milestone (total project timeline is 14 months):

- 08.2020-11.2020: Training.
- 11.2020-01.2021: Implementation of adaptive error signals algorithm over a model of a robotic arm on Intel's Neuromorphic Research Cloud (NRC).
- 01.2021-06.2021: Algorithm implementation on Intel's Loihi chip, robotic arm integration.
- 06.2021-08.2021: Simulate our implementation on NVIDIA Xavier board.
- 08.2021-10.2021: Data analysis, paper writing and submission.

Alternative plan

- We aim to deploy our model on a wheelchair mounted 6 DOF robotic system. For this, we will ask for a physical access to the Loihi chip for a limited period of time. However, if this will not be possible will deploy our solution to the SpiNNaker board. SpiNNaker (Spiking Neural Network Architecture) is a million-core computing engine whose flagship goal is to be able to simulate the behaviour of aggregates of up to a billion neurons in real time. A small SpiNNaker board makes it possible to simulate a network of tens of thousands of spiking neurons, process sensory input and generate motor output, all in real time and in a low power system.

5 Preliminary result

Here, we demonstrate the adaptive inverse calculation which calculate the adaptive error signal vectors: (x, y, z) in the gravitational space and $(roll, pitch, yaw)$ in the orientation space. This model is written in python, using Nengo GUI.

Since the calculation of one space is done independently of the another, for the demonstration we set a model (Figure 2) constructed of two sub-models – one for the gravitational space and one for the orientation space. Each sub-model contains three compute unites, since each dimension can be calculated independently. Each unit has 1. a node of input which represent the current state of the system, 2. a target node which represent the required target state, 3. an ensemble that represent the neuronal result, 4. a direct ensemble which is a reference for the expected result and 5. an ensemble which represent the adaptive error compensation signal.

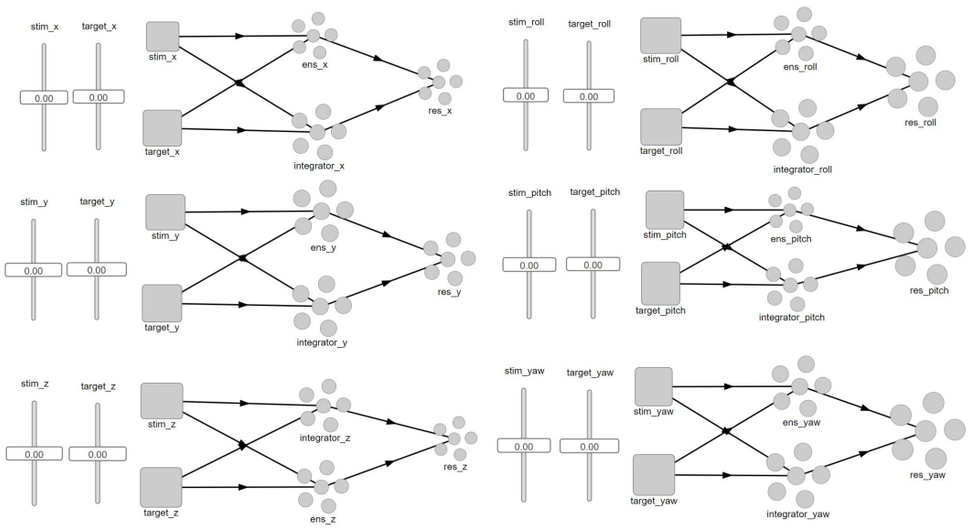

Figure 2: Nengo-GUI elements structure: on the left the gravitational space sub-model, on the right the orientation space sub-model. Each sub-model contains three compute unites, since each dimension can be calculated independently. Each unit has 1. a node of input which represent the current state of the system, 2. a target node which represent the required target state, 3. an ensemble that represent the neuronal result, 4. a direct ensemble which is a reference for the expected result and 5. an ensemble which represent the adaptive error compensation signal

We use pre-defined parameters, represented using a 1-dimension ensembles, to demonstrate the inverse calculation and the error correcting signals. We ran the simulator for four second while setting the target of both gravitational and orientations vector to be $(0, 0, 0)$ and in each second we changed the current state of the dimensions. The error compensation signal should adapt to the new state respectively. First, we set current state to be $x = 0, y = 0, z = 0, roll = 0, pitch = 0, yaw = 0$. In this configuration the error signal should be ≈ 0 on all dimensions since we reached the target state.
On the next second we set current state to be $x = 1, y = -1, z = 0, roll = \pi, pitch = -\pi, yaw = 0$, and on the third we change to $x = -1, y = 0, z = 1, roll = 2\pi, pitch = 0, yaw = -2\pi$. Since the target state is ≈ 0 the expected correction signal should be the same as the input signal on both seconds.
On the last second we set current state to be $x = 0, y = 0, z = -1, roll = 0, pitch = 0, yaw = -2\pi$ and the target state to be $x = 0, y = 0, z = 1, roll = 0, pitch = 0, yaw = -2\pi$. Now we expect that the correction signal will be $x = 0, y = 0, z = 2, roll = 0, pitch = 0, yaw = 4\pi$ since the current state and the target state are on both limits of the scale (note that on the orientation space we have modulo 2π, so the correction signal can be the supplemental angle instead of the actual calculated angel. Here we provided the actual angel for simplicity).
Figure 3 concludes the results.

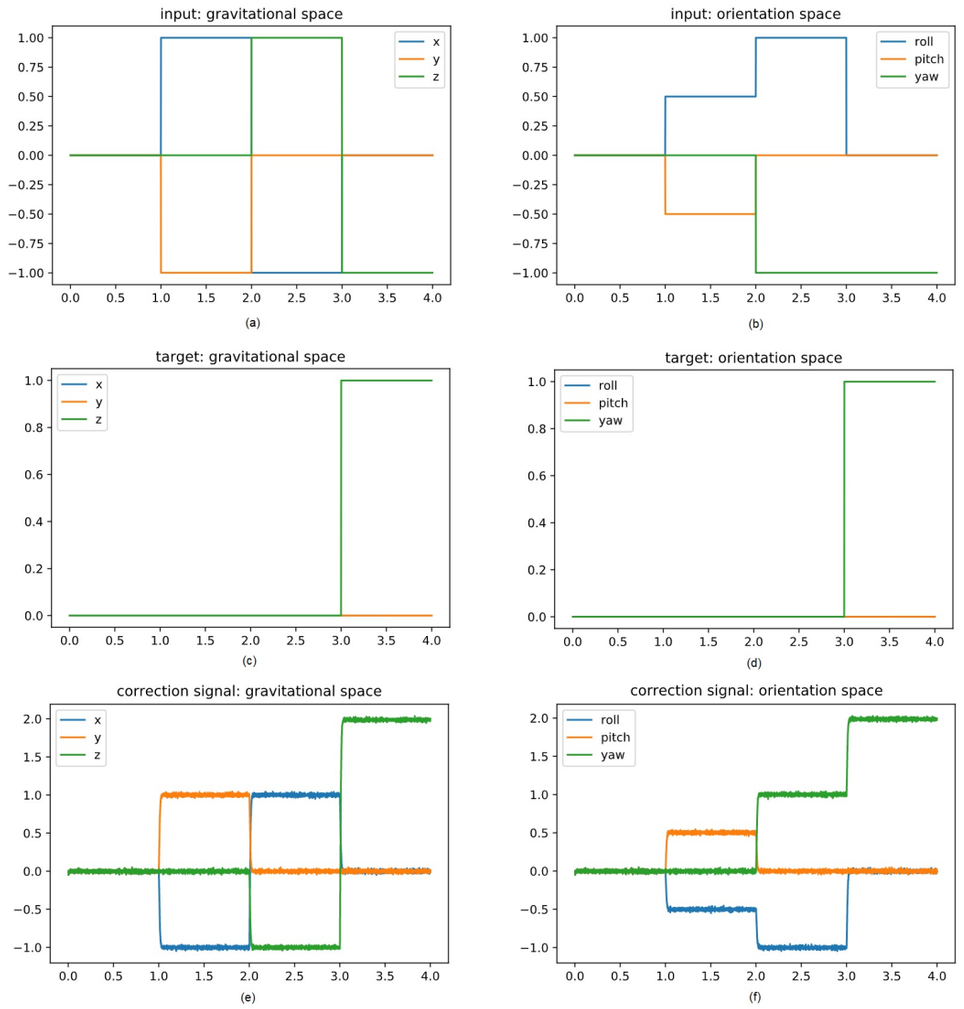

Figure 3: Preliminary results: (a)(b) Input stimulus of the gravitational and orientation spaces accordingly. (c)(d) Target state of the gravitational and orientation spaces accordingly. (e)(f) Error correction signals of the gravitational and orientation spaces accordingly. Note that the units of the gravitational space are normalized to the domain $[-1, 1]$, the units orientation space are radian and normalized to the domain $[-2\pi, 2\pi]$ and both experiments are presented along time with units of seconds.

6 Summary

Adaptive control schemes are an effective way to deal with uncertainties in all robot dynamic models in which the controllers are designed to be adjustable so as to automatically compensate for these uncertainties. Here, we propose a neuromorphic algorithm which calculates the error signals for adaptive neuro-robotics in 6 Degrees of Freedom (DOF). Our implementation will be integrated with two greater tasks: 1. neuromorphic adaptive control in a wheelchair-mounted 6-DOF robotic arm, which will be clinically evaluated at ALYN Hospital, Israel's most advanced pediatric and adolescent rehabilitation center. 2. stabilize a spider-bot walking on diverse terrain patterns.

References

[1] Y. Pi and X. Wang, "Trajectory tracking control of a 6-dof hydraulic parallel robot manipulator with uncertain load disturbances," *Control Engineering Practice*, vol. 19, no. 2, pp. 185–193, 2011.

[2] Y. Nakamura, K. Kishi, and H. Kawakami, "Heartbeat synchronization for robotic cardiac surgery," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, 2001, pp. 2014–2019 vol.2.

[3] J. Khurshid and Hong Bing-rong, "Military robots - a glimpse from today and tomorrow," in *ICARCV 2004 8th Control, Automation, Robotics and Vision Conference, 2004.*, vol. 1, 2004, pp. 771–777 Vol. 1.

[4] S. Young and A. Kott, "A survey of research on control of teams of small robots in military operations," *arXiv preprint arXiv:1606.01288*, 2016.

[5] A. Flores-Abad, O. Ma, K. Pham, and S. Ulrich, "A review of space robotics technologies for on-orbit servicing," *Progress in Aerospace Sciences*, vol. 68, pp. 1–26, 2014.

[6] S. Schön, M. Ebner, and S. Kumar, "The maker movement. implications of new digital gadgets, fabrication tools and spaces for creative learning and teaching," *eLearning Papers*, vol. 39, pp. 14–25, 2014.

[7] T. Karvinen and K. Karvinen, *Make: Arduino Bots and Gadgets: Six Embedded Projects with Open Source Hardware and Software.* " O'Reilly Media, Inc.", 2011.

[8] T. Hsia, "Adaptive control of robot manipulators - a review," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, 1986, pp. 183–189.

[9] D. Zhang and B. Wei, "A review on model reference adaptive control of robotic manipulators," *Annual Reviews in Control*, vol. 43, pp. 188–198, 2017.

[10] M. Tarokh, "Hyperstability approach to the synthesis of adaptive controllers for robot manipulators," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, 1991, pp. 2154–2159 vol.3.

[11] R. G. Walters and M. M. Bayoumi, "Application of a self-tuning pole-placement regulator to an industrial manipulator," in *1982 21st IEEE Conference on Decision and Control*, 1982, pp. 323–329.

[12] N. Goléa, A. Goléa, K. Barra, and T. Bouktir, "Observer-based adaptive control of robot manipulators: Fuzzy systems approach," *Applied soft computing*, vol. 8, no. 1, pp. 778–787, 2008.

[13] T. Sun, H. Pei, Y. Pan, H. Zhou, and C. Zhang, "Neural network-based sliding mode adaptive control for robot manipulators," *Neurocomputing*, vol. 74, no. 14-15, pp. 2377–2384, 2011.

[14] Q. Li, A. N. Poo, C. M. Lim, and M. Ang, "Neuro-based adaptive internal model control for robot manipulators," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 5, 1995, pp. 2353–2357 vol.5.

[15] M. Wasielica and M. Wasik, "Active stabilization of a humanoid robot base on inertial measurement unit data," in *Proceedings of the 16th International Conference on Mechatronics-Mechatronika 2014.* IEEE, 2014, pp. 364–369.

[16] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (imu) sensor applications," *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013.

[17] V. Malyavej, W. Kumkeaw, and M. Aorpimai, "Indoor robot localization by rssi/imu sensor fusion," in *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 2013, pp. 1–6.

[18] H. Wang, "Adaptive control of robot manipulators with uncertain kinematics and dynamics," *IEEE Transactions on Automatic Control*, vol. 62, no. 2, pp. 948–954, 2016.

[19] N. Jaisumroum, P. Chotiprayanakul, and S. Limnararat, "Self-tuning control with neural network for robot manipulator," in *2016 16th International Conference on Control, Automation and Systems (ICCAS).* IEEE, 2016, pp. 1073–1076.

[20] B. Zhang, L. Shi, and S. Song, "Creating more intelligent robots through brain-inspired computing," *in Science Robotics*, p. 3, 2016.

[21] T. C. Stewart and C. Eliasmith, "Large-scale synthesis of functional spiking neural circuits," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 881–898, 2014.

[22] T. C. Stewart, "A technical overview of the neural engineering framework."

[23] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, p. 48, 2014.