SURE_REACH Optimal Parameter Set and Model Properties Investigation

Cezary Piotr Banaszek
c.p.banaszek@student.utwente.nl

January 20, 2023

Abstract

SURE_REACH (sensorimotor, unsupervised, redundancy-resolving control architecture) is an unsupervised motion planning model which is utilized for conducting reaching motions in space. The model allows for setting constraints on the mobility of the joints as well as perform obstacle avoidance. This paper explains the model, showcases its properties, and conducts an experiment optimizing for posture and space accuracy of the model. Additionally, this paper discusses the broader scope that the model and the findings of the study are contextualized in, and underlines the interplay between the scientific domains which contribute to, and can also benefit from, the study of the model.

1 Introduction

Throughout millions of years of evolution, humans have mastered the skill of moving around and performing reaching movements. These tasks are completed by solving the **motion planning** problem. This problem is defined in this paper as finding a set of motor commands generating movement in accordance with the physical structure (or simulation of a body) that eventually lead to a goal position in space (or goal posture) starting from an initial posture. Thus far, there has not been designed a robot that is able to be on par with the robust and flexible way in which humans solve motion planning problems. The introduction of robots with human-like command of interacting with its environment will undoubtedly benefit human society by allowing us to outsource more labor to machines. Naturally, this prompts the question why humans are so good at motion planning. In order to answer this question, one needs to try and analyze, or even replicate, the model behind this ability. This paper aims to contribute to solving the challenge of robot inflexibility by delving into and expanding on a model that has been inspired by humans and their learning process in regards to their motion planning skills.

The goal of this paper is to find optimal parameters that yield the most accurate results of the SURE_REACH model designed by Butz et al. (2007). This model solves the motion planning problem for a redundant simulated arm in order to reach a chosen point in space with as little prior knowledge as possible. Additionally, the paper's goal is to provide the reader with a clear explanation of the model as well as demonstrate some of the model's properties. Their approach assumes that the model learns by itself how to interact with its environment through sensory data combined with trial and error. This approach to the problem stands out due to the fact that the model was designed by adhering to research and findings in psychology and neuroscience investigating learning and goal directed movement of humans. Furthermore, it applies mathematical operations formulated in accordance with the way organic neurons work.

This paper will first introduce the reader to what has been already done within the field of models solving the motion planning problem. New important notions and definitions as well matrices will be shown in bold. Then, the SURE_REACH model will be introduced and explained through the use of a couple of illustrative examples. Furthermore, an experiment for finding an optimal coefficient set will be conducted. Lastly, the properties of the model will be discussed and all findings will be concluded.

2 Literature review

There are plenty of models solving the motion planning problem. A general distinction in the way these models deal with redundancy of a system can be made as outlined by Butz et al. (2007). A system is said to be **redundant** when there are more dimensions of the configuration space than the dimensions of its workspace, which is the number of independent variables that can be used to fully describe a certain configuration. Simply put, a system is redundant when it can approach a point in the workspace in infinitely many ways. For example, a planar (2D) arm with three joints is redundant since its configuration can be described with three variables, its joints, whereas its workspace is two dimensional. The configuration space is of higher dimensions (3D) than the workspace (2D) thus the system (the arm) is redundant.

There are two ways to resolve redundancy of a system. One is to resolve it before or during learning (provided that learning even takes place). The other is to store redundant solutions of the system all together. Primary examples of the former approach are *direct inverse modeling* (DIM), *feedback error learning* (FEL), *resolved motion rate control* (RMRC), *model-free reinforcement learning* (Model-free RL) and *model predictive control* (MPC). Instances of the latter are the *mean of multiple computations* (MMC) *network* and *posture-based motion planning* (PB)

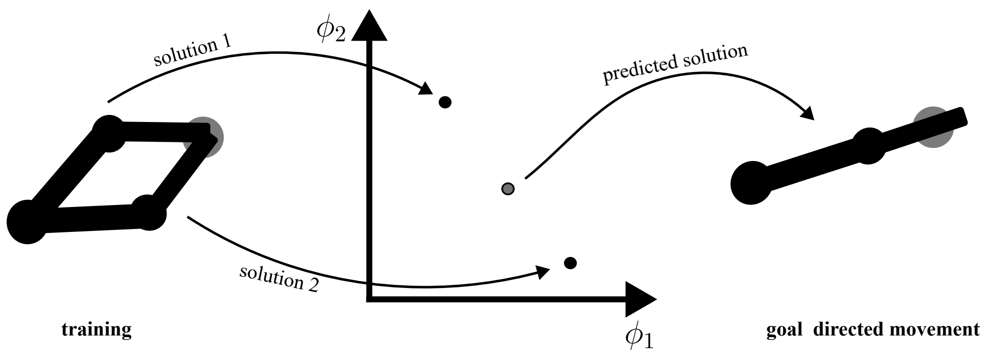


Figure 1: Example of an inaccuracy in finding a solution for non-convex action space for DIM approach.

theory. Some of the approaches outlined in this section are entirely model driven whereas others include a training and learning part. Further, a general overview of these methods is discussed.

Resolve redundancy of the system before or during learning

DIM

DIM (Jordan & Rumelhart, 1992) that is applied to an arm learns through performing a sequence of random actions similar to **motor babbling** or body babbling in infants (Meltzoff & Moore, 1997). This is a process of self-generating movements which then are associated with sensations experienced during the movement. This technique is used by many more models yet to be presented in this paper and by SURE_REACH itself. After such training, the arm is able to inversely solve the motion planning problem through actions associated with approaching a given goal. When the model calculates the solution to reaching the goal wrong it will adjust accordingly to do better the next time. The shortcomings of the model (Jordan & Rumelhart, 1992) are that it might not work well in a situation where there is one-to-many mappings between actions and sensations. Additionally, if the solution space is non-convex then the model is sure to produce a wrong outcome by taking two (non-convex) solutions. The model then averages the solutions into an arm posture that is not a solution to the motion planning problem. A visual example of such a situation can be seen in Figure 1 where two different postures encountered during training became a prediction that is outside of the set goal.

FEL

The FEL approach builds on the DIM method by making learning smoother and more efficient by implementing an explicit feedback controller which moderates the error passed to the inverse model during training (Kawato, 1990). The FEL approach can bypass the redundancy problem by multiplying the error with the Moore-Penrose pseudoinverse as done on PUMA, an industrial manipulator, by Kano et al. (1989). However, this method of dealing with redundancy forces the model to reject redundant solutions by choosing one and following a certain criterion such as the minimal norm of the solution in the aforementioned example. Both FEL and DIM are susceptible to an increase of complexity of an arm structure.

RMRC

The RMRC approach (Whitney, 1969) is entirely model driven and relies on using a Jacobian inverse matrix for generating motor controls which aim at reaching the goal. Redundancy of the model is solved by imposing additional constraints (D'Souza et al., 2001) making the approach fairly inflexible.

Model-free RL

This method utilizes the Bellman equation (Bellman, 1957) for training a model-free RL model which rewards actions that are expected to bring the arm closer to the goal. One strong point in this approach is that the RL approach is quite flexible as it depends merely on the reward scheme chosen to teach the arm how to behave which can account for various constraints and arm architectures. Furthermore, this method is also a case of resolving redundancy of the system before or during learning as the action is performed based on the learned rewards abiding the set reward scheme. However such model-free approach means that it requires more training. Lastly, once the goal or the constraints are changed, it might mean that the model needs to re-learn the rewards, often from scratch (Berthier et al., 2005).

MPC

The MPC model's (Bolognani et al., 2008; Guechi et al., 2018; Lenz et al., 2015) main feature is that it allows control over a system which is very hard to control and describe using a completely hand-designed controller. Instead, the model requires a well-designed dynamic model which then is used for creating an accurate enough cost function related to achieving a goal. The MPC approach tries to predict the future state and its behavior given its current state until reaching the goal state. The model's strong points are accurate linear models and ability to impose multiple constraints on the controller (Bolognani et al., 2008). MPC was used in creating a system control for a robotic arm cutting through materials of varying properties (Lenz et al., 2015) as well as a two degrees of freedom planar arm (Guechi et al., 2018).

Store redundant solutions of the system

The MMC network

MMC is another case of a fully model driving approach. In the MMC network (Cruse et al., 1998) the solution is found by iteratively calculating several geometrical relationships of the arm at the same time while solving for the same variable. The final result is the average of all of the calculated results, hence the name *the mean value of multiple computations*. The results are then fed forward to the next iteration of the algorithm further approximating the solution for reaching the goal. The model allows for adding constraints throughout the process, making it flexible. Furthermore, since the model consists of geometrical relationships it does not disregard any of solutions before or after goal directed movement starts.

The PB theory

Lastly, as pointed out by Butz et al. (2007), the PB theory (Rosenbaum et al., 1993; Rosenbaum et al., 1995; Rosenbaum et al., 2001) is the most similar to SURE_REACH when compared to all of the models for motion planning. Both models remain quite flexible by storing redundant solutions to a motion planning problem as well as allowing for adding constraints for arm's motion planning execution. Furthermore, in both cases the motions can be generated on the

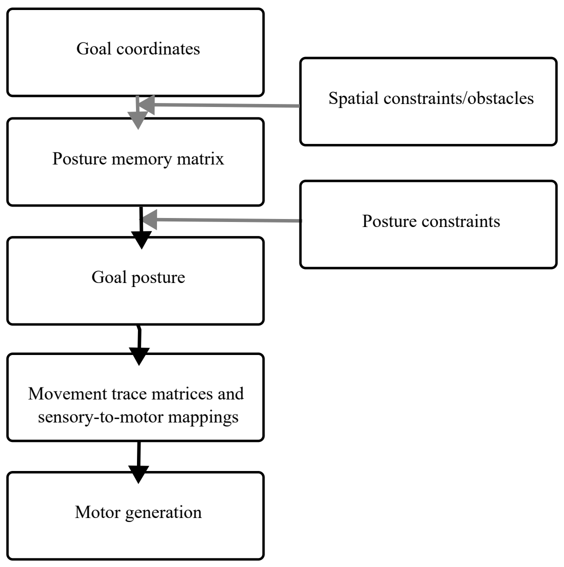


Figure 2: The overview of SURE_REACH model. Gray arrows signify parts of the model that can be utilized depending on the need but are not necessary for the most basic functioning of the model.

spot and both evaluate suitability of discretized postures by attaching weights to them which are trained through unsupervised learning. Despite similarities the models exhibit a great difference in their mathematical designs. As PB theory utilizes a Gaussian function for weight calculation whereas SURE_REACH uses Hebbian learning (Hebb, 1949) as the way of determining weights attached to postures.

To summarize, there are two ways to deal with redundancy of a system. One is to pick a single solution from an infinite number of them, the other is to find a way to save multiple solutions for the same problem. In the outlined approaches, DIM, FEL and RMRC are inflexible in finding alternative solutions after training. Model-free RL is in a similar situation, however depending on the change to the reward scheme it might be able to readjust to the new constraints "on the go". Nevertheless, it generally takes a long time to train. The MPC model allows for setting multiple constraints which allows it to be quite flexible compared to the other redundancy resolving type models. Lastly, there are two other models which store redundant solutions. Where the first one, the MMC network, relies on a detailed outline of geometrical relationships of the controlled arm as well as on a feedback loop which guides the arm step by step towards the solution. On the other hand, PB theory, despite its similarity to SURE_REACH, differs in its mathematical model assuming a more statistical Gaussian approach compared to a more neural based approach of SURE_REACH.

3 SURE_REACH model

Model overview

The sensorimotor, unsupervised, redundancy-resolving control architecture (SURE_REACH) model works by making connections between its posture neuron activation vector $\vec{p}$ (posture vector in short) and space activation neurons $\vec{h}$ (space vectors in short) that are activated while the tip of the arm moves through the workspace. The **inverse kinematics** problem is a problem regarding finding such angles of an arm so that its tip is placed in a given position in the workspace. The inverse kinematics problem is solved through training the posture memory matrix (PMM) $\boldsymbol{W}_{PM}$. This method allows to switch between hand space (coordinates in workspace) to posture space (angles of an arm) when specifying a goal. The motion planning problem is resolved by training movement trace matrices (MTMs) $\boldsymbol{W}_i$ which store information about how remote all posture neurons are in relation to all other posture neurons. All of the information about the arm's environment, as mentioned, is obtained through motor babbling. After the learning phase the arm is able to generate appropriate motor commands which directly determine the joints' movement. Motor commands can be generated thanks to sensory-to-motor mappings (STMMs) $\vec{a}_i$ which allow for evaluating how suitable different sets of motor commands are. The arm allows for goal directed movement in posture space by directly approaching the set goal posture vector $\vec{p}_g$ or in hand space by translating a goal space vector $\vec{h}_g$ into a goal posture $\vec{p}_g$ by using PMM $\boldsymbol{W}_{PM}$. The following sections aim to outline in detail the whole mathematical model behind SURE_REACH with a couple of illustrative examples. Matrices are denoted through bold text in equations whereas dimensions of variables are denoted as (rows,columns) at the end of every described element in equation description. The overview of the model is shown in Figure 2 which outlines the whole functioning of the model.

The arm and motor control

$$\phi_j(t) = \phi_j(t-1) + g(y_{2j} - y_{2j+1}) \text{ for } j \in \{0, 1, 2\} \tag{1}$$

where:	$\phi_j(t)$	– a joint angle at time t (scalar)
	$\phi_j(t-1)$	– a joint angle at time $t-1$ (scalar)
	g	– gain factor determining the maximum a joint can move in a single time step (scalar)
	y_{2j}, y_{2j+1}	– antagonistic motor commands, the larger one determines the movement direction of the joint (scalar)

The arm considered in this simulation has three joints: ϕ_0 (shoulder), ϕ_1 (elbow), ϕ_2 (wrist), and three links. A combination of all three joints in a form of (ϕ_0, ϕ_1, ϕ_2) defines a posture. The links are of lengths $l_1 = 140\,mm$, $l_2 = 70\,mm$ and $l_3 = 42\,mm$. The arm's full length is denoted as $l_{arm} = l_1 + l_2 + l_3$. The joints can move in the following ranges in radians: $\phi_0 \in [-\pi; \pi]$, $\phi_1 \in [-\pi; \pi]$ and $\phi_2 \in [0; \pi]$. The ranges are strict in a sense that the joint ϕ_1 or ϕ_2 can not jump from π to $-\pi$ and vice versa. For that, they need to turn all the way by 2π radians in order to reach the other boundary. The movement of the arm is represented by antagonistic motor commands, which is two per each joint. The motor commands are annotated as y_i, $i \in \{0, 1, 2, 3, 4, 5\}$. A detailed description of the arm's architecture is shown in Fig. 3. The movement defined as in Eq. 1. Now having described the arm treated in this paper as well as its dynamics, the next part of this section will introduce a method by which the arm can learn from its environment.

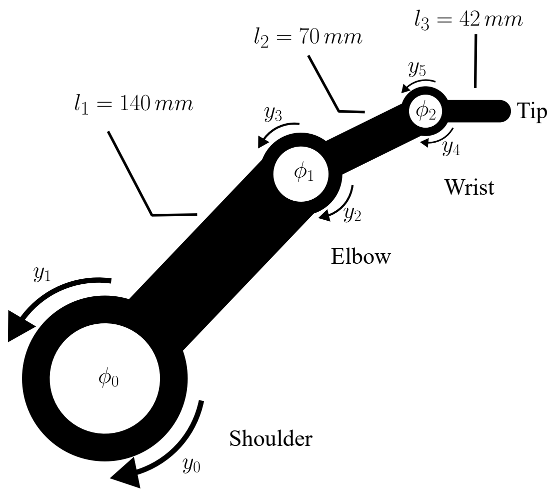


Figure 3: Outline of the arm architecture used for the model discussed in this paper.

Hebbian Learning

As mentioned, the model is inspired by the way humans move. This means that the Hebbian learning method is utilized. This method is inspired by the biological neural weight adjustment mechanism. It is commonly described by the phrase "what fires together, wires together" and was first described by Hebb (1949). Mathematically, it can be described as:

$$w_{new} = w_{old} + \beta\,\vec{x} \times \vec{y}^{\mathrm{T}} \tag{2}$$

where:	w_{new}	– updated neural weights
	w_{old}	– neural weights before the update
	β	– learning coefficient
	$\vec{x}, \vec{y}$	– activation vectors

Example 1

Let's say there is an arm that only has one joint (Figure 4). The arm can only assume four postures in a discrete form of p_1, p_2, p_3 and p_4. The arm can also sense four different discrete positions in space h_1, h_2, h_3 and h_4. An example of posture and space representation as a vector is shown below as $\vec{p}$ and $\vec{h}$. Each position in those vectors is associated with one of the four different positions h_i or postures p_i. In the example, the arm turned into a posture p_2, second entry in the posture vector, and consequently the position h_4 is also activated, fourth entry in the space vector. The nature of those vectors will be explained more in detail further in this section.

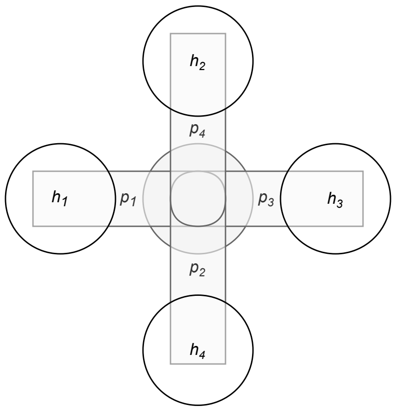


Figure 4: Simple arm architecture with four possible postures and positions that it can reach.

$$\vec{p} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{h} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Even though this simple arm has knowledge about its posture and position in space, it does not quite know how the two interconnect. Hence, the Hebbian learning method allows the arm to make such connections. Through using Eq. 2 and the defined posture and space vectors, the arm learns that p_2 approaches the position h_4. If the arm was to start rotating freely, going from h_1 all the way to h_4, it would encounter all of the posture and space neurons exactly once. In this case, the trained matrix would look as shown at the end of Example 1 and contain all discrete posture to space relations. In other words, $\boldsymbol{w}$ contains information about which posture reaches which position in space.

$$\boldsymbol{w}_{old} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \vec{p} \times \vec{h}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

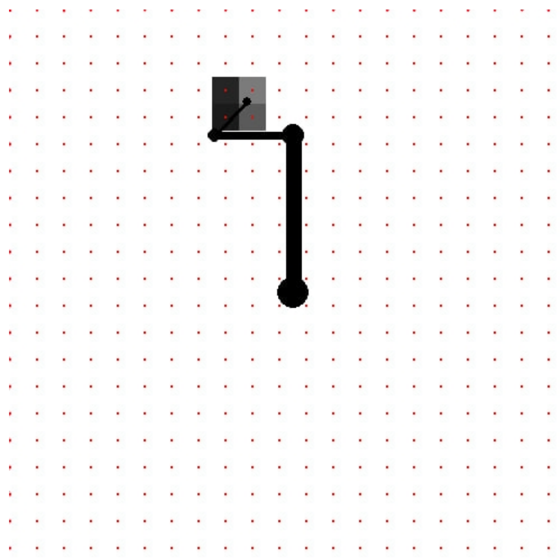
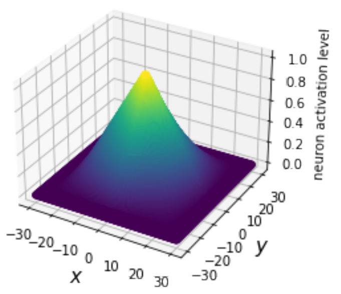


Figure 5: Representation of an arm and space neurons spread over the workspace. The grayscale tiles near the tip of the arm signify the activation level of the nearby space neurons. The lighter the higher the activation. Non-activated neurons are shown in plain white.

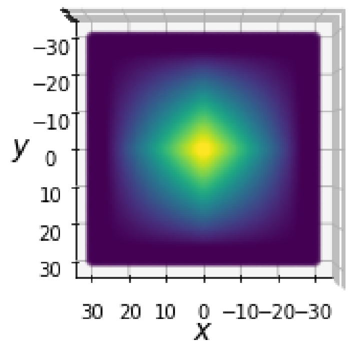
Space neurons

In order for the arm to learn from its environment, it needs to be able to sense its posture and its position in space. For that reason, the notion of posture and space neurons is introduced. An i-th **space neuron** is defined as a fixed position in space (h_{ix}, h_{iy}). The activation of such a single space neuron i is described in Eq. 3. The activation of a space neuron in a two dimensional Euclidean space is a pyramidal activation function (Figure 6). In practical terms, a space neuron is fired when the tip of the arm is close enough for it to fire. The max function ensures that a given space neuron is activated strictly within a set distance in space in x and y direction. In this paper this distance is set to be 10% of the arm's length. There is $21 \cdot 21 = 441$ different space neurons spread uniformly across the workspace (Figure 5). The distance between each space neuron is also $0.1 l_{arm}$. It is important to note that activation of neighboring (either posture or space) neurons can be simultaneous as the activation range of each neuron overlaps with its neighbors. The space neuron that is the closest to the tip of the arm yields the highest activation level. All of the 441 space neuron activation values are represented as a **space neuron vector** $\vec{h}$, where each position in the vector is associated with a unique space neuron. The order in which the neurons are arranged in the vector is not important as long as it can be easily accessed and identified given the index of a neuron in the vector. This process is called **encoding**.

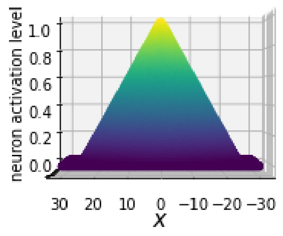
$$\begin{aligned} h_i &= \prod_{n=1}^{2} \max\left(1 - \frac{|h_n - h_{in}|}{0.1\, l_{arm}}; 0\right) \qquad (3) \\ &= \max\left(1 - \frac{|x - h_{ix}|}{0.1\, l_{arm}}; 0\right) \cdot \max\left(1 - \frac{|y - h_{iy}|}{0.1\, l_{arm}}; 0\right) \end{aligned}$$



(a) isometric view



(b) top view



(c) side view

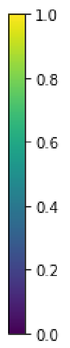


Figure 6: Activation of a single space neuron in two dimensional space.

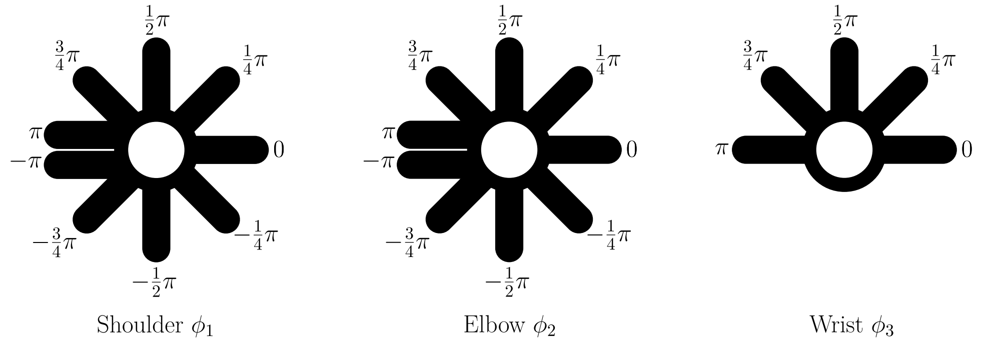


Figure 7: Possible configuration of each joint. A combination of the three makes a single unique posture neuron.

where: h_i – activation value of an i-th space neuron (scalar)
h_n – one of the arm's tip coordinates in space (x or y coordinate) (scalar)
h_{in} – one of the coordinates of the i-th encoded space neuron h_i (scalar)

Posture neurons

Posture neurons, unlike space neurons, are not defined by a fixed position in space but a fixed posture of the arm. Every posture neuron consists of a combination of three joint angles whose values are discretized as some multiples of 0.25π radians. An i-th posture is denoted as $(\phi_{i0}, \phi_{i1}, \phi_{i3})$. There are $9 \cdot 9 \cdot 5 = 405$ posture neurons (Figure 7) and each activation value can be calculated using Eq. 4. Similarly as in the case of space neurons, posture neurons also overlap with their neighbors. This allows for activation of multiple posture neurons at the same time. It is important to point out that although a joint is located in the same position in space whether it is at angle π or $-\pi$, they still need to be distinguished from each other as the joint can only move in the opposite direction from the joint limit. All of the 405 activation values are represented as a **posture neuron vector** $\vec{p}$, where each posture neuron is encoded in the vector in a similar way as explained for space neurons.

$$p_i = \prod_{n=1}^{3} \max\Big(1 - \frac{|\phi_n - \phi_{in}|}{0.25\pi}; 0\Big) \tag{4}$$

$$= \max\Big(1 - \frac{|\phi_1 - \phi_{i1}|}{0.25\pi}; 0\Big) \cdot \max\Big(1 - \frac{|\phi_2 - \phi_{i2}|}{0.25\pi}; 0\Big) \cdot \max\Big(1 - \frac{|\phi_3 - \phi_{i3}|}{0.25\pi}; 0\Big)$$

where: p_i – activation value of an i-th posture neuron (scalar)
ϕ_n – one of the arm's joints (shoulder, elbow or wrist) (scalar)
ϕ_{in} – one of the joint angles of the i-th encoded posture neuron p_i (scalar)

Example 2

Let's take an arm consisting of two joints and two links. The range of the first joint is $\phi_1 \in [0, \frac{\pi}{2}]$ and the range of the other is $\phi_2 \in [0, \pi]$. The posture neurons in this

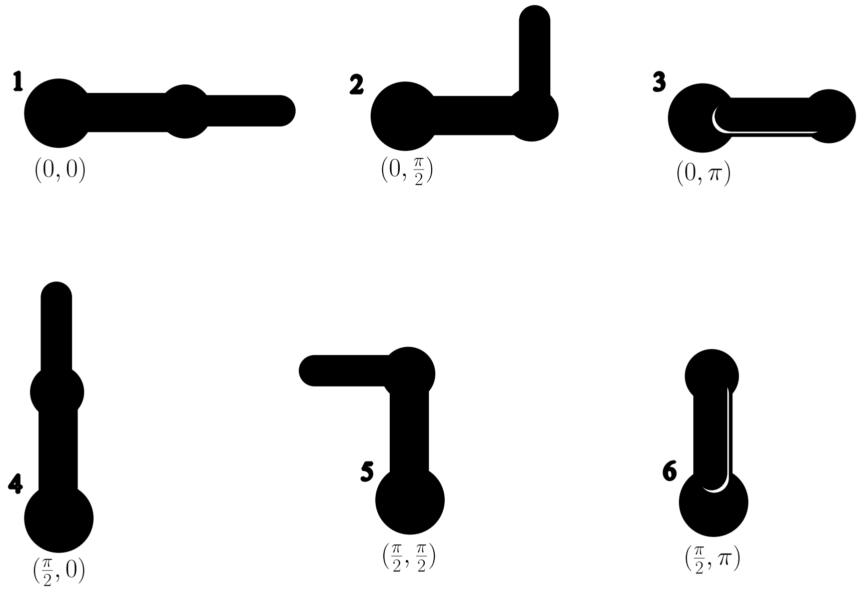


Figure 8: Six discretized posture neurons in Example 2.

example are discretized as a combination of joints with angles that are multiples of $\pi/2$. Accordingly, the distance after which a posture neuron turns zero is also set to be $\pi/2$, Eq. 5, instead of $\pi/4$ as in the model. In that case, there are only six posture neurons shown in Figure 8.

$$
\begin{aligned}
p_i &= \prod_{n=1}^{2} \max\Big(1 - \frac{|\phi_n - \phi_{in}|}{0.5\pi}; 0\Big) \\
&= \max\Big(1 - \frac{|\phi_1 - \phi_{i1}|}{0.5\pi}; 0\Big) \cdot \max\Big(1 - \frac{|\phi_2 - \phi_{i2}|}{0.5\pi}; 0\Big)
\end{aligned}
\tag{5}
$$

If the simple arm assumes posture $\phi_1 = 0$, $\phi_2 = \pi$, then the posture vector will be $\vec{p} = \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$. It is worth noticing that the numbers close to the postures in Figure 8 are corresponding to the entry in the posture vector that they are associated with. In this example the posture number three is assumed hence the third entry in the vector is set to one. The numbering is arbitrary and can be expressed in a different way than in this example. Alternatively, if $\phi_1 = \frac{\pi}{4}$, $\phi_2 = \frac{\pi}{4}$ then the posture vector is $\vec{p} = \begin{bmatrix}0.25 & 0.25 & 0 & 0.25 & 0.25 & 0\end{bmatrix}^{\mathrm{T}}$ meaning that such arm posture fires four different posture neurons (1, 2, 4 and 5) as they are quite close to it. The remaining two inactivated posture neurons are too distant from the arm posture to have a non-zero activation.

Posture memory matrix

One of the core features of the SURE_REACH model is the ability to solve inverse kinematics problems. For that purpose, the notion of PMM (posture memory matrix) W_{PM} is introduced. This matrix retains information about relations between posture and space neurons which are trained using Hebbian learning as shown in Eq. 6. The dimensions of the matrix are the result of the multiplication of the posture and space vectors which equals to $(405, 1) \times (1, 441) \rightarrow (405, 441)$. Intuitively speaking, each column in such a matrix can be regarded as one of the 441 different space neurons located in the workspace. The values in a given column in the matrix tell us how strongly associated different posture vectors are with the specific space neuron associated with the given column of PMM. The greater the value of an entry in the vector, the closer the posture associated with its encoding in the posture vector is to the point in space associated with its encoding in the space vector. Such hierarchy is created due to the Hebbian learning rule since only some postures can activate some specific space neurons.

$$\boldsymbol{W}_{PM}(t) = \boldsymbol{W}_{PM}(t-1) + \varepsilon\, \vec{p} \times \vec{h}^T \tag{6}$$

where: $W_{PM}(t)$ – updated PMM at time t (405,441)
$W_{PM}(t)$ – PMM at time $t-1$ (405,441)
ε – learning coefficient (scalar)
$\vec{p}$ – posture neuron vector at time $t-1$ (405,1)
$\vec{h}$ – space neuron vector at time $t-1$ (441,1)

As the arm moves in the workspace, it encounters different space-posture neuron combinations (alternatively, it excites different sets of posture and space neurons). In results, Hebbian learning reinforces the posture-space connections which can be later retrieved thus allowing for solving inverse kinematics problems. Given the fact the arm is redundant, it can reach the same point in multiple ways, hence the posture memory matrix connects all of the postures that can reach a certain point. This is provided that the postures were encountered before, e.g. in the motor babbling phase. Once PMM is trained it is possible to solve the inverse kinematics problem through Eq. 7. The equation translates a point in goal space represented as a space neuron vector $\vec{h}_g$ into a goal posture of the arm represented as a posture neuron vector $\vec{p}_g$.

$$\vec{p}_g = \boldsymbol{W}_{PM} \times \vec{h}_g \tag{7}$$

where: $\vec{p}_g$ – goal posture neuron vector (405,1)
$\boldsymbol{W}_{PM}$ – trained PMM (405,441)
$\vec{h}_g$ – goal space neuron vector (441,1)

Movement trace vectors and matrices

In order for the arm to move in a goal oriented manner it needs to possess information on how close every posture is to one another. This knowledge allows the arm to keep moving towards the postures that are increasingly closer to the goal posture. For this purpose, movement trace vectors and matrices are introduced. A movement trace vector (MTV) $\vec{r}_i(t)$ is calculated with Eq. 8. There are six MTVs with each one tied to one of the six mother commands $y_i(t)$. Each time the arm moves, it generates a set of motor commands which lead to posture $\vec{p}(t-1)$ (Figure 9). It is important to point out that even though MTV $\vec{r}_i(t)$ is at time-step t, it only contains information from the past to be used for the current time t. MTV is a vector with the same dimensions as the posture vector. All six MTVs contain information about the past actions and

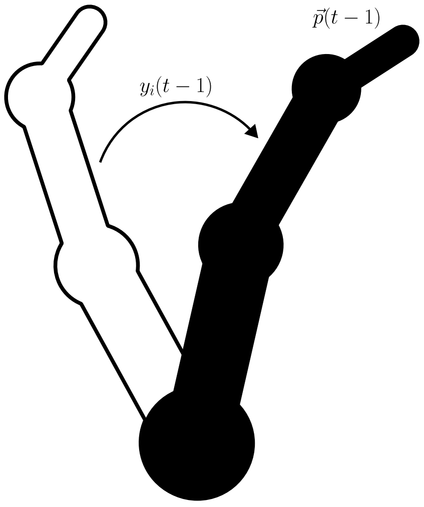


Figure 9: MTV uses the past posture and the motor command leading to that posture.

resulting postures leading up to the current posture $\vec{p_i}(t)$. However, due to the decay rate ρ, the weights of specific posture neurons exponentially decay with time. This emphasizes the more recent postures that are more immediately accessible from the current posture. A visualization of MTV is shown in Figure 10. MTVs alone serve merely an auxiliary purpose to MTMs hence their existence and way of working will become clearer in the following part of this subsection where MTMs are explained.

$$\vec{r_i}(t) = y_i(t-1)\vec{p}(t-1) + \rho\,\vec{r_i}(t-1) \tag{8}$$

where:	$\vec{r_i}(t)$	– MTV at time t (405,1)
	$y_i(t-1)$	– i-th motor command at time $t-1$ (scalar)
	$\vec{p}(t-1)$	– past posture neuron vector (405,1)
	ρ	– decay rate (< 1) (scalar)
	$\vec{r_i}(t-1)$	– MTV at time $t-1$ (405,1)

The movement trace matrix (MTM) is an extension of MTV $\vec{r_i}(t)$. There is six MTMs, with each one associated one of the motor commands. The weights of the matrices are trained, as shown in Eq. 9, using Hebbian learning. However, instead of correlating posture neurons with space neurons, MTMs $\boldsymbol{W}_i(t)$ store information about how remote all posture neurons are from any chosen posture neuron. The greater the weight in an entry of a column p in a matrix $\boldsymbol{W}_i(t)$ the easier it is to reach the posture $\vec{p}$ encoded by number p using the motor command y_i. In other words, MTVs store information on which postures can be immediately accessed from $\vec{p}(t-1)$ by looking at all the past postures it visited. Further, Eq. 9 connects MTVs $\vec{r_i}(t)$ with the arm's current posture neurons in the posture vector $\vec{p}(t)$ at time t. Consequently, connecting all accessible past postures to the current posture which later can be used to reverse the process and tell what postures are accessible from a given current posture. Additionally MTMs allow to compare how accessible certain postures are to $\vec{p}(t)$. It is important to point out that MTMs do not strictly contain information about postures with joint angles being multiples of $\frac{\pi}{4}$, as any other set of joint angles can be represented as an intermediate state of some postures (see Example 2).

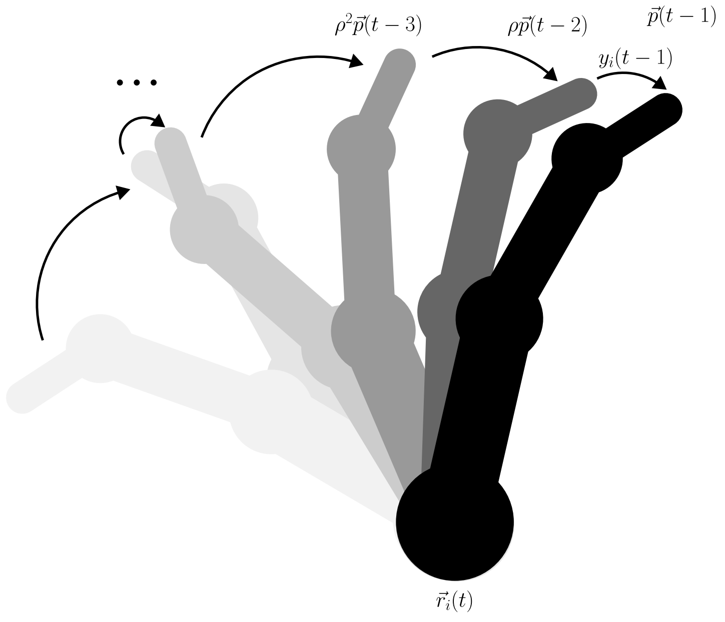


Figure 10: The movement trace vector acts as a brush which leaves a trace of the past postures which would use the motor command y_i in the past before time $t-1$. As the postures from the past are less and less connected with the posture at time $t-1$, they fade into obscurity.

Eq. 9 is another example of learning through Hebbian rule. The delta coefficient ensures that the weights do not explode. The delta coefficient decays exponentially from 0.1 to 0.01 throughout MTMs training process. Additionally, the theta coefficient is another mechanism moderating the matrices' growth rate by setting a ceiling for the maximal weight value. If a certain weight in the matrix exceeds the value of theta, the difference in theta matrix and MTM will become negative, effectively decreasing the weights' values in places that already exceeded the limit in the trained MTM. This happens by the means of matrix convolution with the newly learned weights.

$$\boldsymbol{W}_i(t) = \boldsymbol{W}_i(t-1) + \delta\, \vec{r}_i(t) \times \vec{p}(t)^{\mathrm{T}} * (\boldsymbol{\theta} - \boldsymbol{W}_i(t-1)) \tag{9}$$

where:	$\boldsymbol{W}_i(t)$	– MTM at time t (405,405)
	$\vec{r}_i(t)$	– MTV at time t (405,1)
	$\vec{p}(t)$	– posture neuron vector at time t (405,1)
	δ	– learning rate (scalar)
	$\boldsymbol{\theta}$	– matrix filled with theta coefficients (405,405)
	$\boldsymbol{W}_i(t-1)$	– MTM at time $t-1$ (405,405)

Sensory-to-motor mappings

Sensory-to-motor mappings $\vec{a}_i$, STMMs in short, allow the arm to perform goal directed movement towards a desired posture vector or space vector when turned into a posture vector with the use of posture memory matrix W_{PM}. The way STMs work is by creating six mappings each linked with one of the motor commands y_i. The mappings are updated as shown in Eq. 10 and 11. The max function returns the element-wise maximal value between the entries of two columnar vectors of the same shape. The mappings are updated by calculating an average of all mappings, aside from the one that is being updated, and performing a weighted sum with the previous version of a given $\vec{a}_i$. Then the resulting value is decayed by the beta coefficient so that the weights of STMs do not grow infinitely and allow for readjustment for when posture goal p_g is changed throughout the process. Lastly, the mappings are enriched by mixing the corresponding movement trace matrix W_i into it. The role of the first step of STMM update is to spread values across the mappings. Intuitively, if one of the mappings finds that a certain posture neuron might be suitable to achieve the goal posture, so might be the case for the rest of the mappings. However, the inhibited neuron will carry a smaller weight than the original mapping which inhibited such a posture neuron on its own. The maximum function ensures that despite the dynamic change in weights in STMMs, the goal posture always remains the greatest, thus the most important to achieve. The second step of STMM update is mixing in the MTMs which contain information on how remote postures are from each other. The mixing is performed by a cross product with $\vec{a}_i^*$. In turn, a weighted sum of columns of MTM is added to $\vec{a}_i^*$ with which weights are used to perform the weighted summation. With each update, STMMs contain more and more information about reaching the goal posture starting from inhibiting neurons corresponding to that goal posture in STMMs. STMMs are initialized with zero vectors.

$$\vec{a}_i^*(t) \leftarrow \max\left\{\beta\Big(\gamma\frac{\sum_{j\neq i}^{5}\vec{a}_i(t-1)}{5} + (1-\gamma)\vec{a}_i(t-1)\Big);\, \vec{p_g}\right\} \tag{10}$$

$$\vec{a}_i(t) \leftarrow \vec{a}_i^*(t) + \boldsymbol{W}_i \times \vec{a}_i^*(t) \tag{11}$$

where: $\vec{a}_i^*(t)$ – an intermediate STMM before the mixing in of the MTM (405,1)
$\vec{p_g}$ – goal posture (405,1)
β, γ – coefficients (scalars)
$\boldsymbol{W}_i$ – MTM associated with the i-th motor command (405,405)
$\vec{a}_i(t)$ – updated STMM (405,1)

Example 3

Let's consider a simple model where there are only two STMMs containing four weights each. The mappings are initialized with zero vectors.

$$\vec{a}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ \vec{a}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The posture that we want to reach is $\vec{p_g} = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 \end{bmatrix}^T$ and $\beta = 0.5$, $\gamma = 0.6$. After the first step (Eq. 10) all of the mappings are simply a copy of $\vec{p_g}$.

$$\vec{a}_1 = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{bmatrix}, \ \vec{a}_2 = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{bmatrix}$$

Now that the mappings are not zero vectors, MTMs $\boldsymbol{W}_i$ start to come into play. For this example let's assume MTMs to be:

$$\boldsymbol{W}_1 = \begin{bmatrix} 0.1 & 0.0001 & 0.001 & 0.01 \\ 0.01 & 0.1 & 0.0001 & 0.001 \\ 0.001 & 0.01 & 0.1 & 0.0001 \\ 0.0001 & 0.001 & 0.01 & 0.1 \end{bmatrix}, \ \boldsymbol{W}_2 = \begin{bmatrix} 0.1 & 0.01 & 0.001 & 0.0001 \\ 0.0001 & 0.1 & 0.01 & 0.001 \\ 0.001 & 0.0001 & 0.1 & 0.01 \\ 0.01 & 0.001 & 0.0001 & 0.1 \end{bmatrix}$$

When STMMs are updated in the second step of the update process, the weights have MTM values mixed into them. The matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are asymmetric since they are responsible for movements that are antagonistic to each other. One is better at reaching the postures where the other struggles with and vice versa. After applying Eq. 11 the mappings are:

$$\vec{a}_1 = \begin{bmatrix} 0.00055 \\ 0.5005 \\ 0.55 \\ 0.0055 \end{bmatrix}, \ \vec{a}_2 = \begin{bmatrix} 0.0055 \\ 0.55 \\ 0.5005 \\ 0.00055 \end{bmatrix}$$

The updated mappings contain information about which motor command is more suitable for reaching the goal posture p_g. In this example, y_1 is better for reaching the goal when the arm activates the bottom two posture neurons. And reversely it is best to use y_2 when the top two posture neurons are fired.

Motor command generation

Having generated STMMs $\vec{a}_i$, it is possible to generate goal oriented motor commands y_i towards achieving $\vec{p}_g$. In order to assess how suitable motor commands are in achieving the goal posture from the current posture $\vec{p}(t)$, a cross product with an STMM associated with the given motor command $\vec{y_i}$ is taken (Eq. 12). This results in a scalar value for each of the preliminary motor commands $y_i^S(t)$. The remaining steps of motor command refinement after Eq. 12 will be shown without its elements described in detail. This choice has been made as they are quite similar and repetitive.

$$y_i^S = \vec{p}(t)^{\mathrm{T}} \times \vec{a}_i(t) \tag{12}$$

where: y_i^S – preliminary motor command (scalar)
$\vec{p}(t)^{\mathrm{T}}$ – transposed posture vector at time t (1,405)
$\vec{a}_i(t)$ – STMM associated with the i-th motor command at time t (405,1)

Then, the values are squared and normalized in order to emphasize the suitability of motor commands.

$$y_i^{net}(t) = \frac{y_i^S(t)^2}{\sum_{j=1}^{6} y_j^S(t)^2} \tag{13}$$

The resulting motor commands $y_i^{net}(t)$ are recalculated by taking a difference between antagonistic motor commands where the greater one becomes the difference of the two, and the smaller one becomes zero.

$$y_i^{net*}(t) = \begin{cases} y_i^{net}(t) - y_j^{net}(t) & \text{if } y_i^{net}(t) > y_j^{net}(t) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Finally, the motor commands are scaled by a gain factor g which accounts for the maximum speed of the arm. The set of six motor commands $y_i(t)$ for $i \in \{1, 2, 3, 4, 5, 6\}$ decides on the next best move that is to be taken by the arm to approach the goal.

$$y_i(t) = g \frac{y_i^{net*}(t)}{\sum_{j=1}^{6} y_j^{net*}(t)} \tag{15}$$

4 Optimum coefficients experiment

Model training

The training is conducted by training weights of PMM $\boldsymbol{W}_{PM}$ with a given coefficient ε and MTMs $\boldsymbol{W}_i$ with coefficients θ, δ, ρ. All weight sets are first initialized with zeros. The gain factor g which dictates how fast the arm can move influences both of the weight sets. The learning process happens by initializing the training with an initial posture at random. Then, random motor commands are generated with a probability of 30% that one of the antagonistic motor commands is set to 1 (and the other to 0). This process is called motor babbling. The arm posture is updated by moving it with the randomly generated motor commands. However, the choice was made to not train the weights on motor commands that try to exceed the set joint limits (when the random motor commands try to exceed the limit a new set of them is drawn). After each move, all weight sets are updated with new information gathered through motor babbling. The parameters used to train the weights from scratch are outlined in the

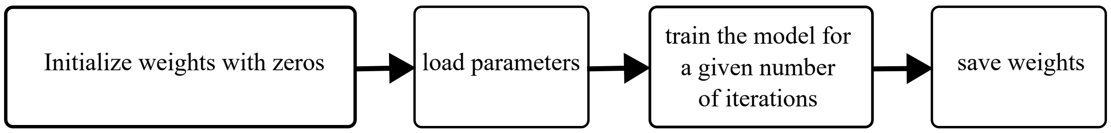

Initialize weights with zeros
load parameters
train the model for a given number of iterations
save weights

Figure 11: Outline of the training process.

appendix of the paper written by Butz et al. (2007). However, one of the parameters ε or θ is changed to a different one from a set of hand-picked values. The only training factor that is changed right away for all weight sets is the iteration batch. This decision was made because it turned out that the weights trained on a smaller number of iterations ($4 \cdot 10^5$) were no worse compared to 10^6 iteration batch in terms of model accuracy (Figure 13). This approach to training saves on computational time in the experiment. The gamma and beta coefficients used in STMMs generation are not a part of the training process hence it is tested directly in the testing procedure. The parameter values were chosen by considering a wide range of them in a pilot study which gave an initial idea of which values might be of interest. As for example, the iteration batch being 10^2, 10^3, 10^4, 10^5, $2 \cdot 10^5$, $4 \cdot 10^5$, $8 \cdot 10^5$ and 10^6, or beta coefficient and its hand-picked values being 0.1, 0.172, 0.2, 0.4, 0.8, 1.0. The outline of the training process is shown in the flowchart in Figure 11. In the testing part, each parameter is directly compared to its counterpart found by Butz et al. (2007).

Testing

In order to find a suitable set of variables for the SURE_REACH algorithm the following experiment was conducted. At first, the model was trained for a different number of iterations (also referred to as iteration batches) with coefficients found by Butz et al. (2007) as explained in the training subsection. The MTMs $\boldsymbol{W}_i^j$ and PMM $\boldsymbol{W}_{PM}^j$ are the weight sets that are used for the experiment. For example, for the iteration batch test there are eight different sets each trained on a batch j ($j \in \{10^2, 10^3, 10^4, 10^5, 2\cdot10^5, 4\cdot10^5, 8\cdot10^5, 10^6\}$). The different sets of weights are tested by utilizing them in the SURE_REACH model and calculating how accurately they allow the arm to approach the goal. The testing part of the experiment is outlined in the diagram (Figure 12). At first, a uniformly random starting posture is generated, as well as a random goal posture or goal coordinates. Then, the weight sets trained on some specific parameters are loaded into the model and the arm is tested for its accuracy of reaching the random goals generated before. The errors are calculated by taking the distance between the final posture/coordinate and goal posture/coordinate. The posture error is defined as $e_p = \sqrt{(\phi_1 - \phi_1^g)^2 + (\phi_2 - \phi_2^g)^2 + (\phi_3 - \phi_3^g)^2}$, whereas for space it is $e_s = \sqrt{(x - x^g)^2 + (y - y^g)^2}$. The final state, (ϕ_1, ϕ_2, ϕ_3) or (x, y), is considered to be the closest distance reached to the goal $(\phi_1^g, \phi_2^g, \phi_3^g)$ or (x^g, y^g) after 200 time steps as this was an amount of time deemed to be sufficient for the arm to reach even the most remote goal. The results are saved and next weight set is loaded. All of the sets are tested on the same initial postures and goals. After each weight set was tested 405 times, all of the results are saved and testing procedure is finished. Throughout the experiment, the best coefficient is considered the one that yields the most accurate results for posture since space is also heavily dependent on posture accuracy. In the case when accuracy is similar across coefficient values, the space accuracy is taken into account. Results are shown in a full version and in a zoomed-in version to make the results more readable. The best performing coefficients will be combined into a parameter set on which weights will be trained and their performance compared to the alternative weights trained on the parameter set found by Butz et al. (2007).

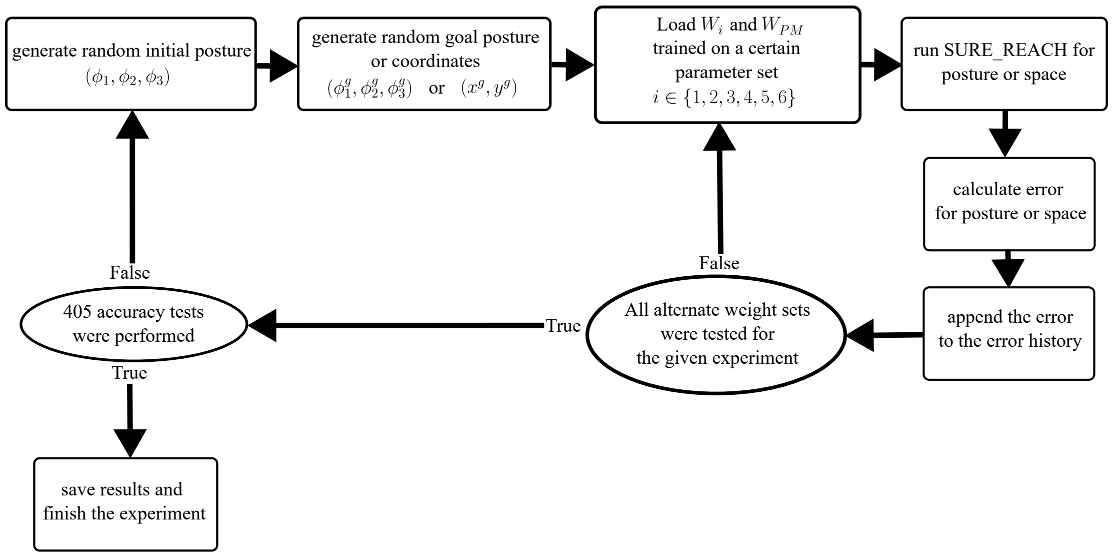

generate random initial posture (ϕ_1, ϕ_2, ϕ_3)
generate random goal posture or coordinates $(\phi_1^g, \phi_2^g, \phi_3^g)$ or (x^g, y^g)
Load W_i and W_{PM} trained on a certain parameter set $i \in \{1,2,3,4,5,6\}$
run SURE_REACH for posture or space
calculate error for posture or space
append the error to the error history
All alternate weight sets were tested for the given experiment
False
True
405 accuracy tests were performed
False
True
save results and finish the experiment

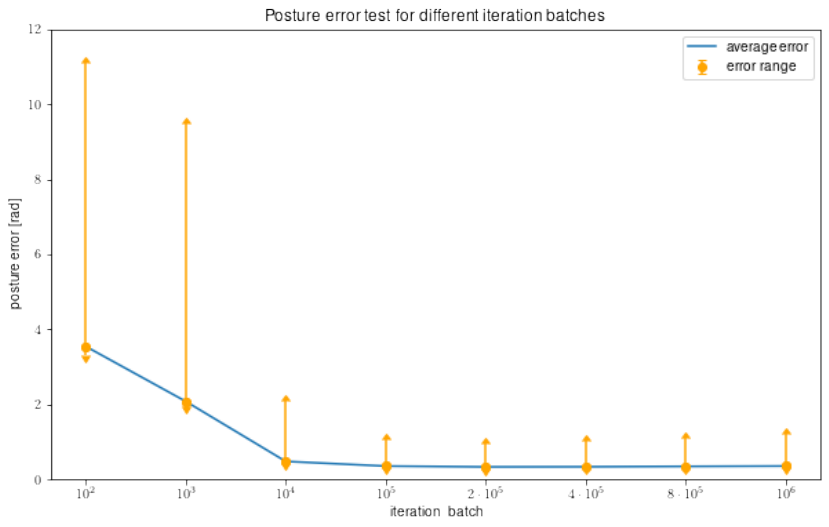
Figure 12: Outline of the testing procedure.

Iteration batch

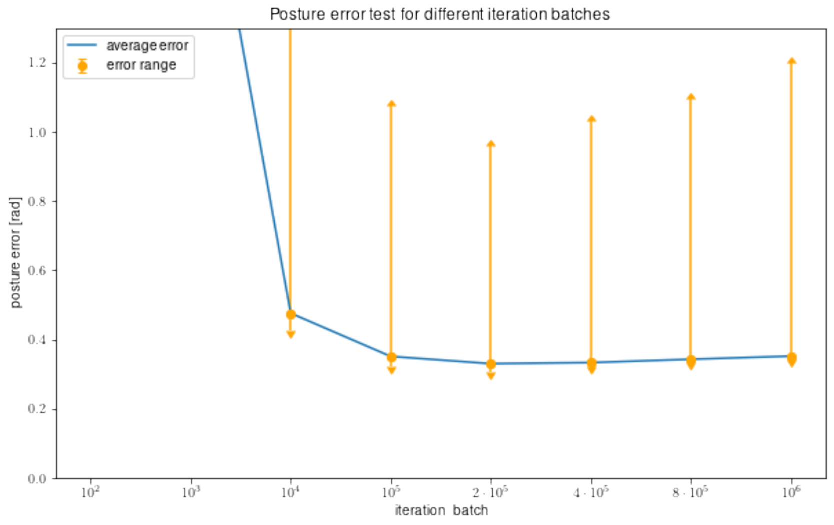
The results of the experiment for finding the most accurate iteration are shown in Figure 13 and 14. The reason why the range error is so high is because most weights are zero for small training iterations. This makes it impossible for the arm to move. After most of the weights are visited (so non-zero), the model can start moving towards its goal. It turns out that starting from the 10000 iteration batch, there are no drastic changes in accuracy of the model and the performance starts to converge. Even though there are certain weight sets that perform ever so slightly better than others, it is likely the result of the inherent randomness in the training and testing process stemming from reaching the aforementioned iteration threshold. Thus, the results are not clear cut and require a more thorough investigation to be able to conclude the optimal value of an iterations batch despite the element of chance within data. Nevertheless, in this experiment, the weight set of 400 000 iterations is the most accurate for achieving a goal posture (Figure 13b). This will be taken as the generally most accurate weight set instead of the original 10^6 for the next experiments in order to save time on computation. The space accuracy results are still a subject to further experimentation as separating the training of MTMs and PMM (i.e. testing them on a different number of iterations) might lead to better results.

Theta coefficient θ

The value of the theta coefficient does not increase the accuracy for posture (Figure 15). There is a considerable improvement in space accuracy (Figure 16) Moreover, the coefficient is necessary as its existence ensures that the weights in MTMs $\boldsymbol{W}_i$ do not increase infinitely. Its influence on the model might become more pronounced with a greater iteration batch as weights might become oversaturated with a lot of training. Even though the posture accuracy is not too different across theta value, the value $\theta = 1.0$ is taken as the most optimal as it provides the best space accuracy.

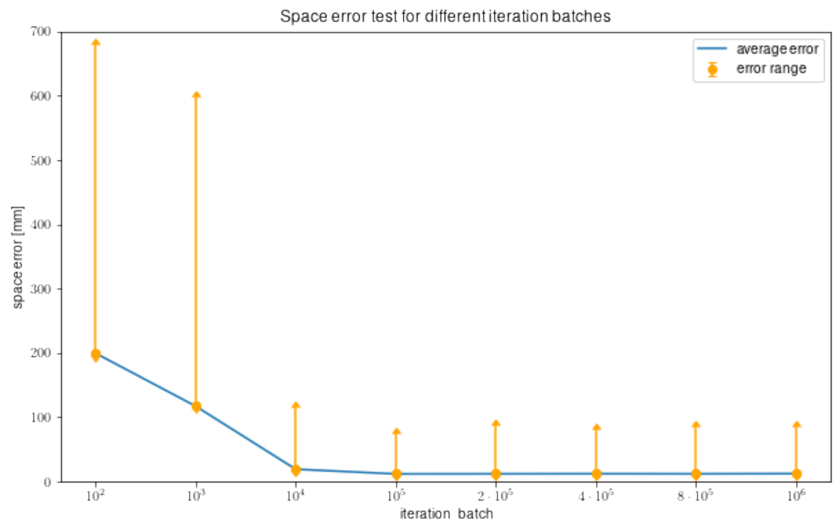


(a)

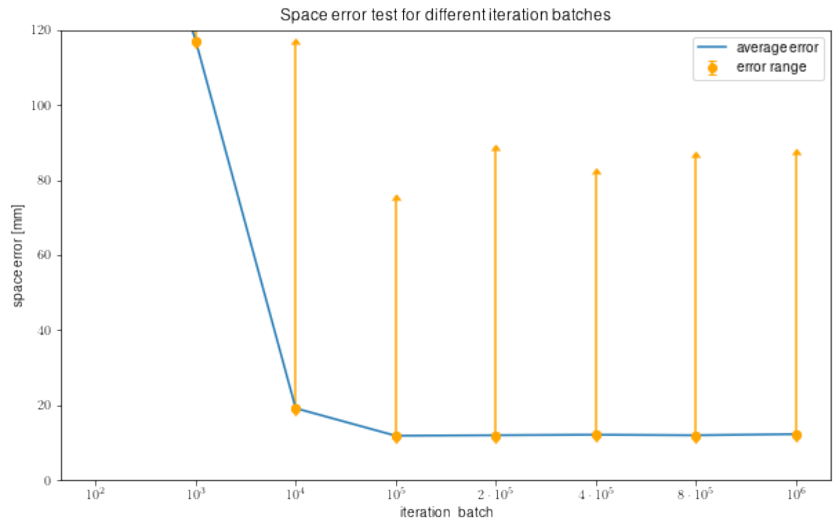


(b)

Figure 13: Posture error test for different iteration batch sizes.

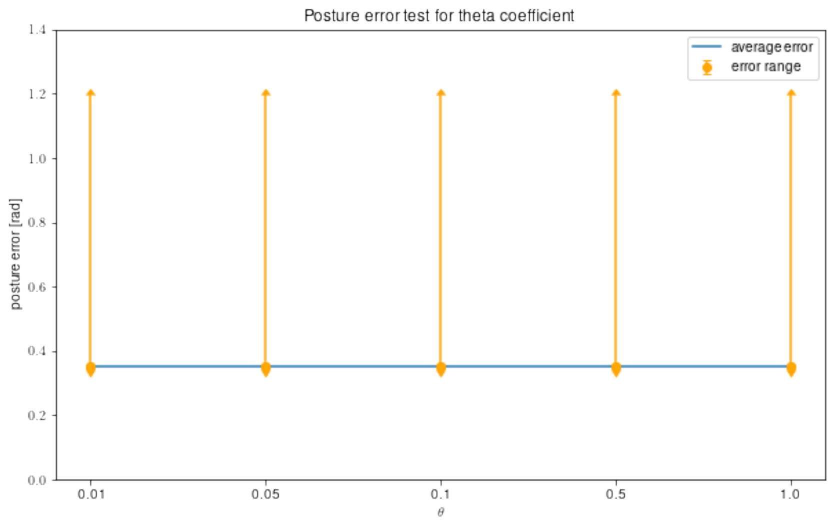


(a)

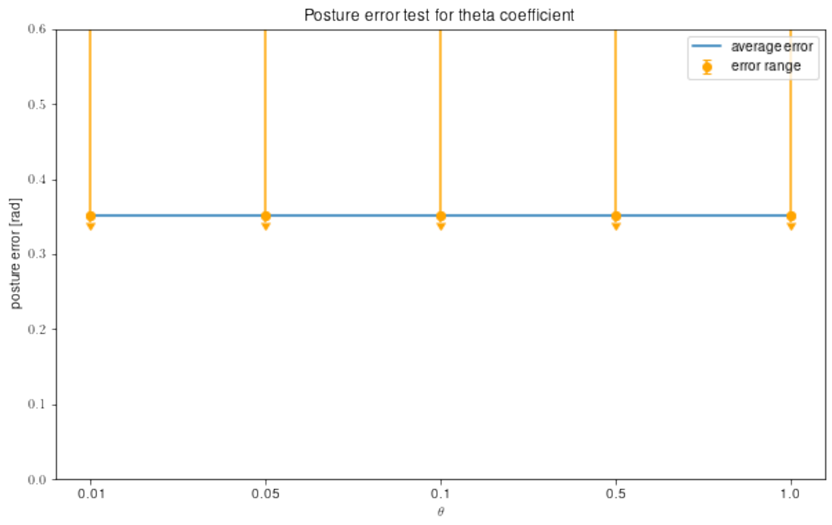


(b)

Figure 14: Space error test for different iteration batch sizes.

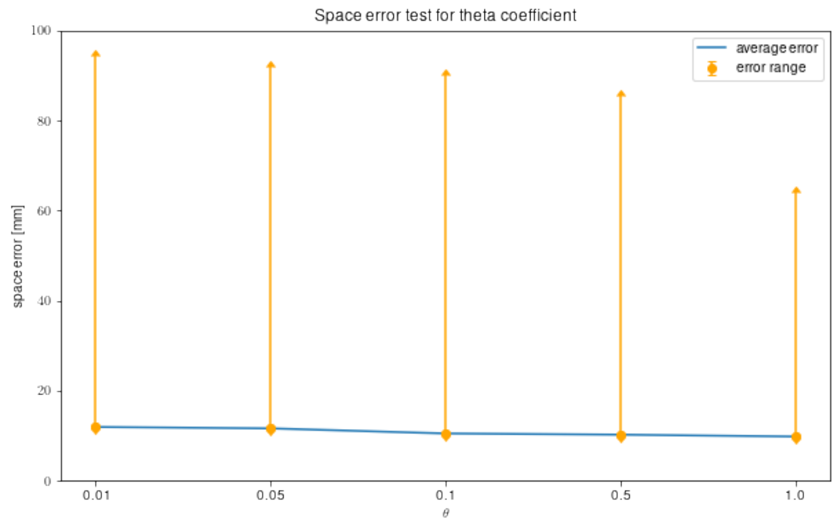


(a)

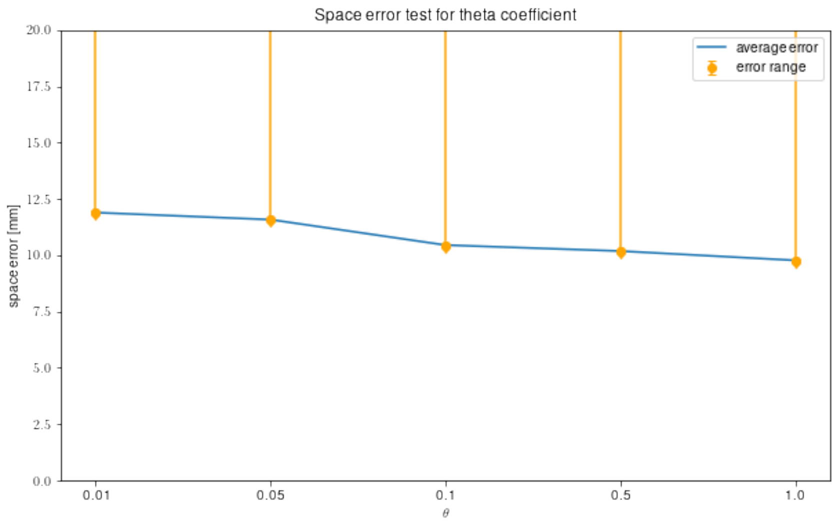


(b)

Figure 15: Posture error test for the theta coefficient.



(a)



(b)

Figure 16: Space error test for the theta coefficient.

Epsilon coefficient ε

The model performed most accurately in reaching goal coordinates for $\varepsilon = 1.0$. However the difference in performance across different values of the epsilon coefficient is not particularly pronounced. This means that realistically any of the tested coefficients will perform similarly well.

Gamma coefficient γ

As for the gamma coefficient, it is shown in Figure 18 and 19 that the accuracy both for posture and space slightly improves as gamma increases (i.e. within the tested gamma range) with the most accurate value for posture being $\gamma = 1.0$. What is intriguing in this result is the fact that gamma being set to one essentially means that each STMM only takes into account the averaging part of the step in Eq. 10. This might be an indication for a change in the mathematical approach for STMM calculation.

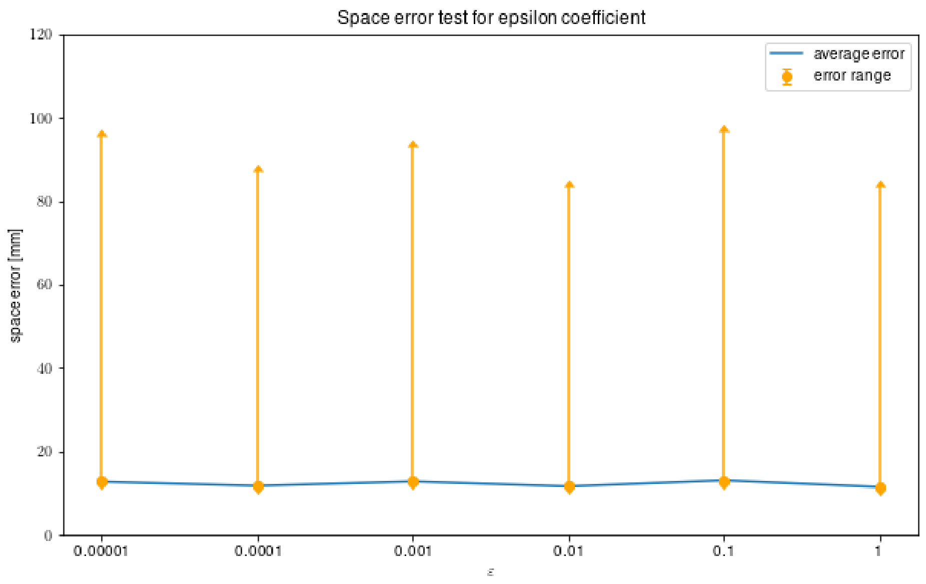
Beta coefficient β

The opposite to the gamma coefficient happens with beta. The smaller it is, the better the accuracy of the model (within the investigated range) for either posture or space (Figure 20 and 21) hence $\beta = 0.1$ is the most suitable for the model.

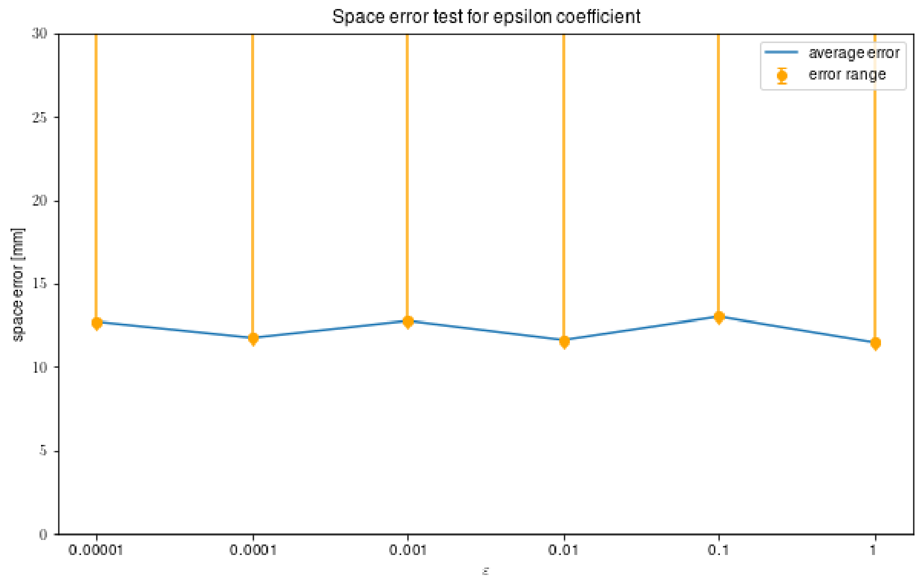
Overall comparison

Having identified the coefficients (i.e. new parameters) that yield the most accurate results separately, it is time to compare the performance of the parameter set found in the experiment and the ones found by Butz et al. (2007) (i.e. old parameters). As shown in Figure 22, the model performs with lower error for both posture (Figure 22a) and spatial tasks (Figure 22b). The orange bars show by how much one error is higher than the other. The greatest relative accuracy improvement is found in posture accuracy. The lowest and highest values during the tests for the new parameter set are 0.006798 rad and 0.588122 rad respectively for the posture tests, whereas for the old parameter set it is 0.008643 rad and 0.699021 rad. Thus, the old parameter set has a less accurate lower accuracy limit and a greater higher accuracy limit. This makes its range less accurate overall. As for space, similarly the error limits are 0.0 mm, 56.0 mm for the new set and 0.0, 69.6 for the old one. Both managed to achieve an error of value zero however in terms of the highest error encounter, the old parameter set scores higher making its space error range higher. Hence, it is concluded that a new alternative parameter set for SURE_REACH is $\theta = 1.0, \varepsilon = 1.0, \gamma = 1.0, \beta = 0.1$ with the iteration batch being 400 000.

As for the general performance of the model, it appears that the model has problems with achieving goal postures that are close to joint limits (for example $-\pi$ for the shoulder) in which case the arm will approach the goal however there would always be a notable distance from the arm's final posture and the set goal posture. This behavior most likely stems from the training approach undertaken for this study, where during the babbling phase it would be prohibited to train weights on motor commands which would not be possible to realize due to the joint limits. The author advises against such an approach in future research.

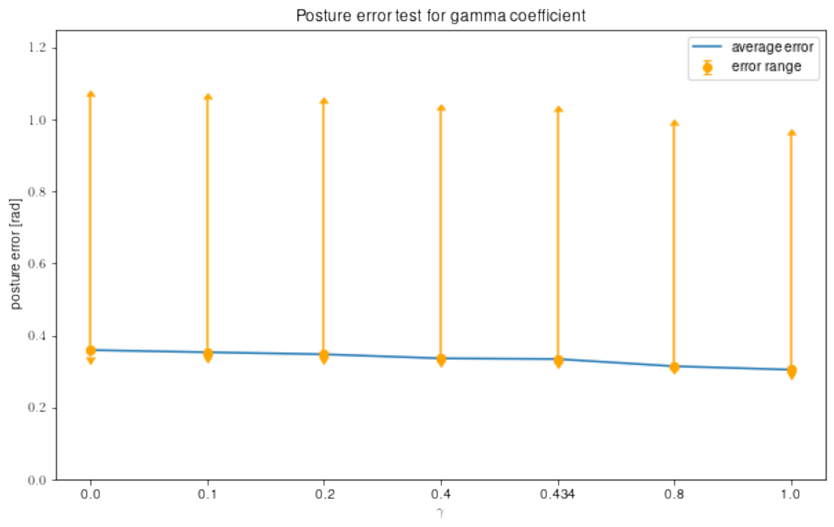


(a)

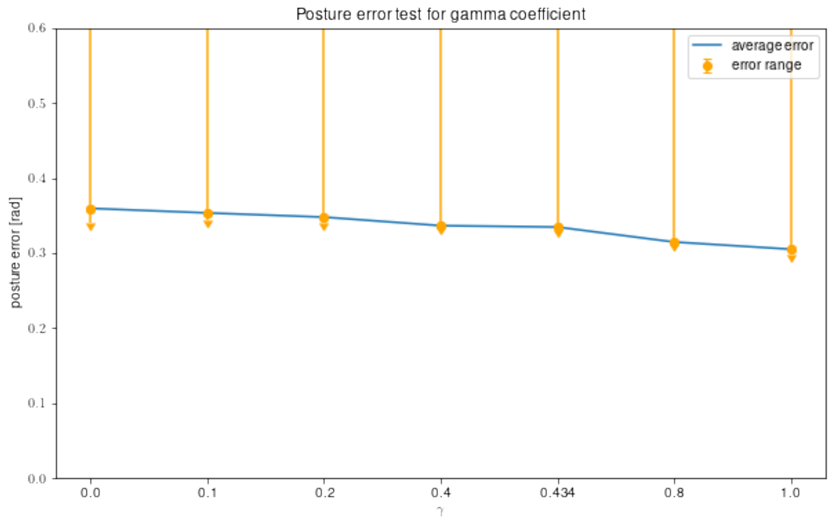


(b)

Figure 17: Space error test for the epsilon coefficient.

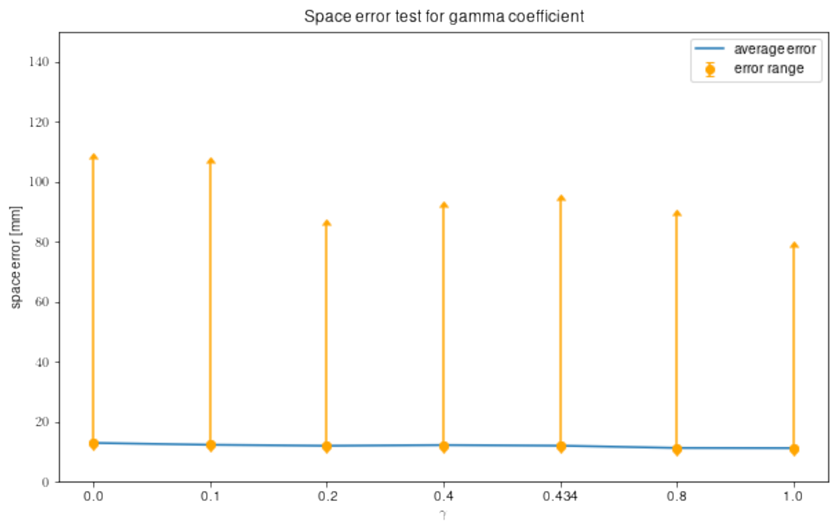


(a)

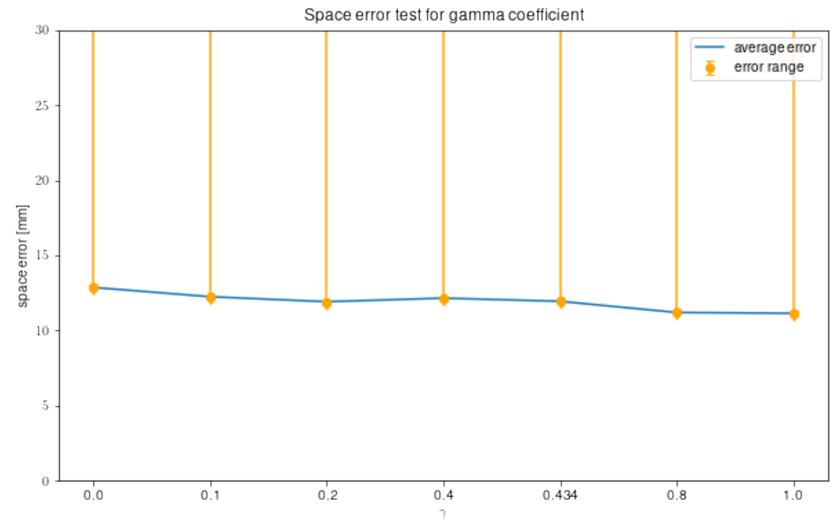


(b)

Figure 18: Posture error test for the gamma coefficient.

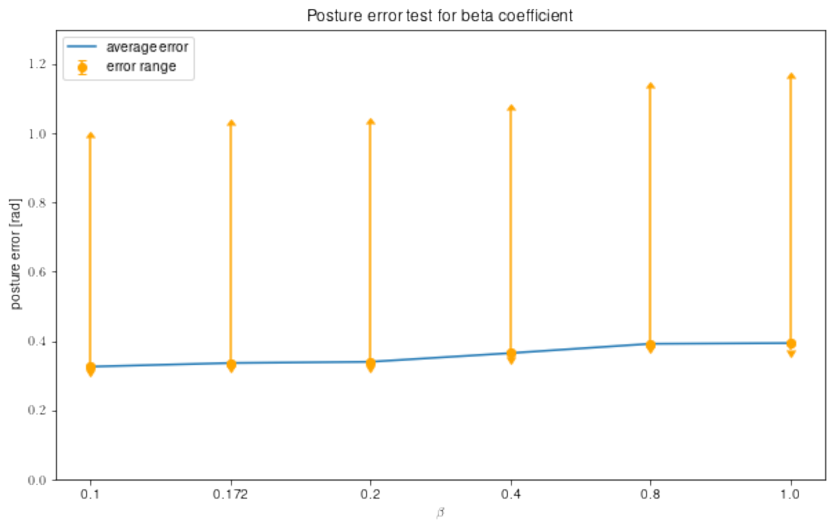


(a)

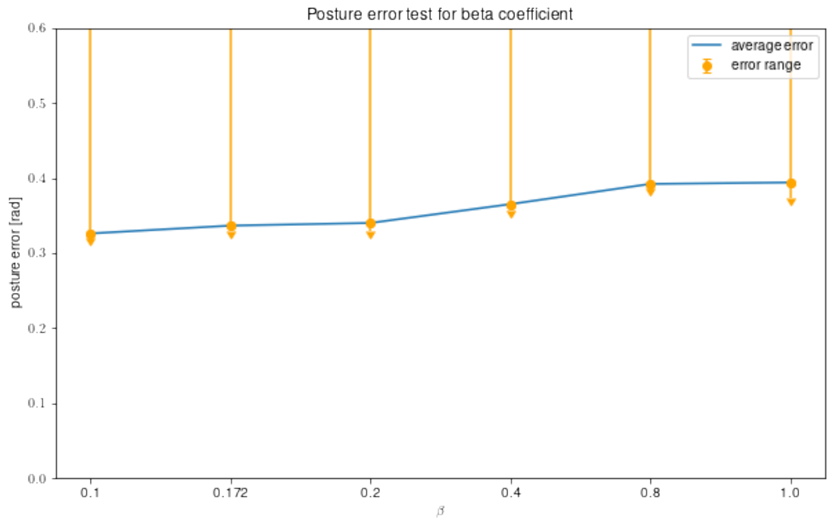


(b)

Figure 19: Space error test for the gamma coefficient.

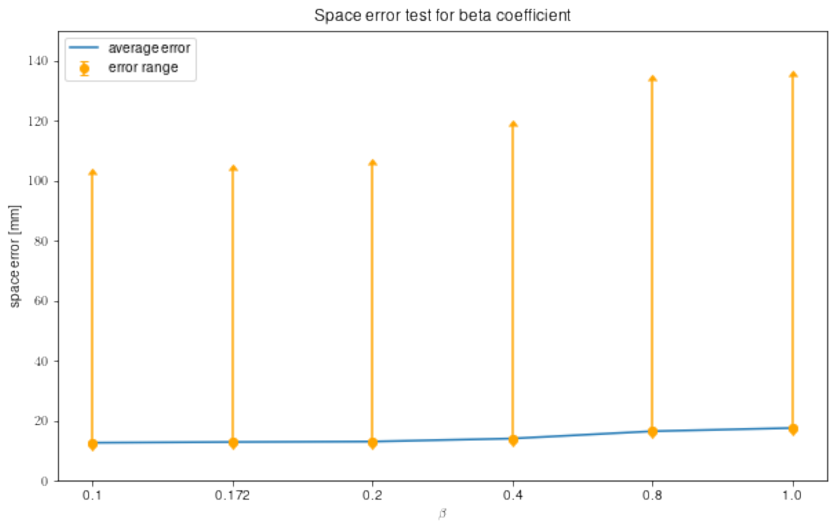


(a)

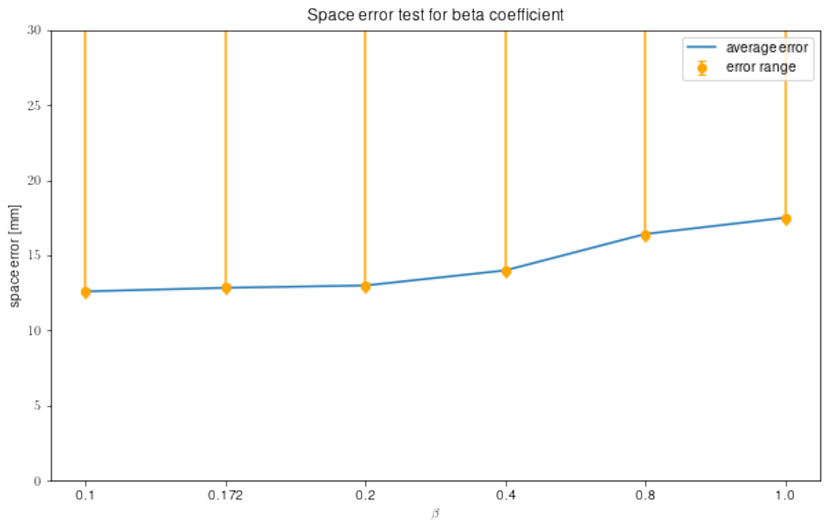


(b)

Figure 20: Posture error test for the beta coefficient.

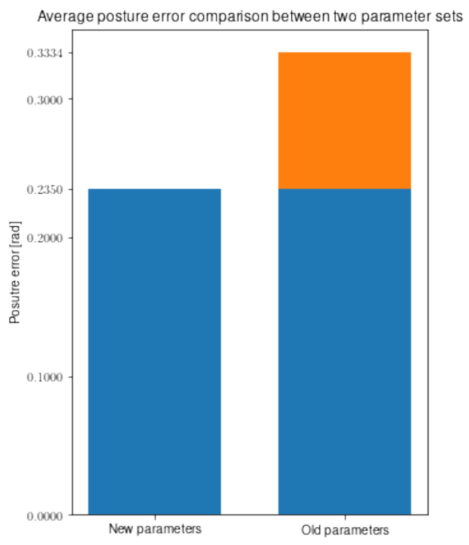


(a)

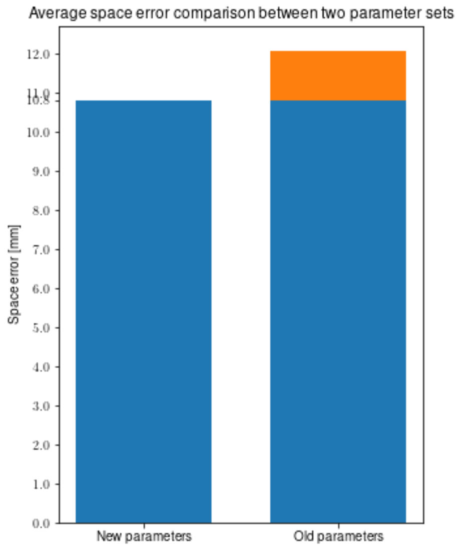


(b)

Figure 21: Space error test for the beta coefficient.



(a)



(b)

Figure 22: Posture and space error comparison between the new and old parameter set.

5 Model properties

In this section, the properties of SURE_REACH are briefly discussed. The following properties stem from the design of the model where various changes in the arm's behavior are obtained through manipulation of certain parts of the algorithm directing it. Such changes can be made either to the space or posture neurons as well as the STMs which will directly change the arm's actions.

Joint mobility reduction

One of the properties of the model is its ability to imitate the way humans move when one of their joints hurts when moving. As displayed in Figure 23, the blue dashed line arm is a simulation of an arm for when it can move freely with no limitations (aside the set joint ranges). However, the black arm's movement is simulated for when the elbow has its mobility reduced by 90%. The mobility of a given joint can be altered as shown in Eq. 16. What the operation does is that it decreases the activation of the STMM for chosen motors in the arm putting them lower in the hierarchy when choosing the path which changes the motor planning.

$$\vec{a}_i(t) \leftarrow \nu_i \vec{a}_i(t) \tag{16}$$

It is important for this operation to happen **after it was normalized** so that the scaling ν_i can actually make a difference in the model. What such scaling of a certain STMM does is it reduces the movement of a chosen joint since that specific STMM is smaller compared to the rest of STMMs. What it allows is control over which joint should be limited in movement. This is similar to how humans limit their movement when e.g. their elbow (or any other joint) hurts. Furthermore, such operation allows for control over not only which joint should have reduced mobility. More precisely, it is even possible to control mobility of a joint only for a clockwise or counter clockwise direction.

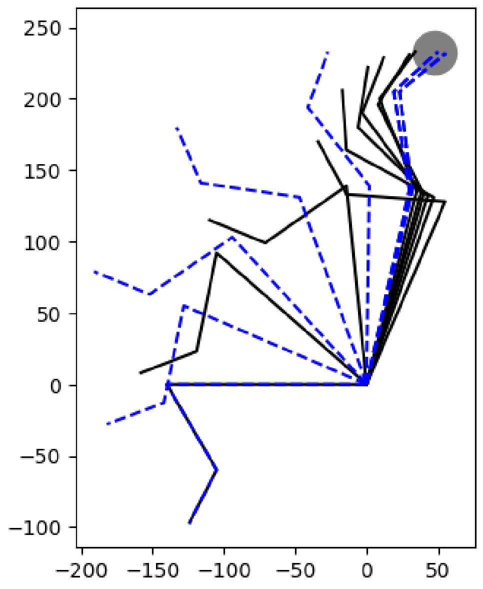


Figure 23: The two arms are trying to reach the set point in space however, the black one has its mobility reduced in its elbow.

Joint immobilization

A special case of joint mobility reduction is when $\nu = 0$ in Eq. 16. In such a case, the whole STMM becomes just a zero vector $\vec{a}_i = \vec{0}$. This completely immobilizes the joint which makes it unable to move. This is similar as in humans when their joint hurts so much it makes it impossible to move or when the arm is put in a cast. What the simulated arm will do in the case where one of the joints is immobilized is that it will try to reach the goal posture that is found by the Equation 7 even if it is not possible. The arm is able to reach most of the points in its workspace when the wrist is immobilized as the other two joints provide enough mobility and movement range. The difference in working between a constrained (black) and unconstrained (blue and dashed) arm is shown in Figure 24 where clearly the constrained arm had to assume a different posture in order to reach the same goal in space compared to the unconstrained one.

Obstacle avoidance

Sometimes, it is not possible to reach the goal by going directly for it as there might be obstacles in the way that need to be avoided. The SURE_REACH model can account for such situations and avoid obstacles by adjusting its trajectory. The model implicitly stores all possible solutions to the motion planning problem in its MTMs ($\boldsymbol{W}_i$) and STMMs ($\vec{a}_i$). First of all, what is an obstacle for the model? An **obstacle** is a set of (either space or posture) neurons which lay within a physical obstacle. The obstacle can be either a typical spatial obstacle such as a rectangle or an obstacle in a form of postures that can not be assumed by the arm.

In order to represent an obstacle in hand space as an obstacle in posture space the following process needs to be undertaken. First, the space neuron vector $\vec{h}_{obs}$ needs to be created by setting positions in the vector to one in places where the associated space neuron is covered by the obstacle. Second, the space neuron vector needs to be represented in the posture space. For

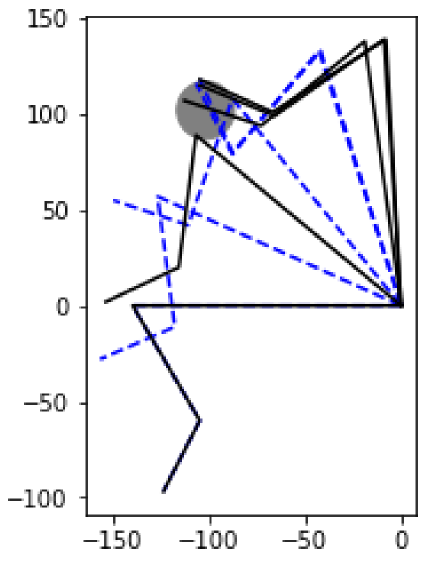


Figure 24: The two arms are trying to reach the set point in space. However, the black one has its wrist immobilized.

that purpose the posture memory matrix ($\boldsymbol{W}_{PM}$) should be used.

$$\vec{p}_{obs} = \boldsymbol{W}_{PM} \times \vec{h}_{obs} \tag{17}$$

where: $\vec{p}_{obs}$ – posture space obstacle vector (405,1)
$\boldsymbol{W}_{PM}$ – posture memory matrix (405,441)
$\vec{h}_{obs}$ – spatial obstacle vector representation (441,1)

Such an operation is described in Eq. 17. Applying the space neuron vector to the posture memory matrix crates a posture neuron vector ($\vec{p}_{obs}$) which activates all those posture neurons that coincide with the obstacle defined in hand space. In other words, the **posture space obstacle vector** $\vec{p}_{obs}$ is a representation of the obstacle in posture space. Lastly, having defined what postures the arm should not approach, we can use this information for motion planning. When STMMs $\vec{a}_i$ are being generated, they should have their activities set to zero in all entries of the vector where the posture space obstacle vector has a non-zero activity. The activation is set to zero when the activity in a STMM is higher than 0.1, otherwise it can be left as is. This allows for slight movements instead of hard stopping the algorithm in some cases. For obstacle avoidance implementation in this paper such operation is conducted both right after the step in Eq. 10 as well as Eq. 11 to ensure deactivation of postures coinciding with the obstacle.

The application of the described approach is shown in Figure 25. The figure shows trajectories of two arms moving from left to right to a goal in space (gray dot). The blue dashed-line arm is a simulated arm for when no changes are made to STMMs so it does not account for any obstacle and behave in a ”business-as-usual” way. The black arm however is aware of the obstacle (pink rectangle) and its STMMs are adjusted accordingly. Both arms are approaching the same goal. Even though the arm would usually take the shorter route and rotate counterclockwise since there is an obstacle in the way it accounts for it and chooses a different trajectory. This

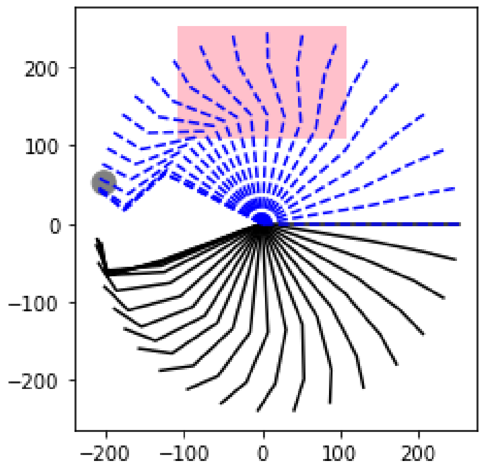


Figure 25: The two arms are trying to reach the set point in space. However, the black one an obstacle to avoid whereas the dashed blue one acts as if the obstacle did not exist.

alternative path ensures that the arm does not run into the obstacle by choosing trajectories that bring greater activation in STMMs since some of them are deactivated due to the obstacle. The black arm does not reach its goal fully as the implementation of the model is found to have problems with reaching postures near its joint limits which is most likely the result of the training approach. Nevertheless, this example demonstrates the obstacle avoidance property of SURE_REACH.

System redundancy

As shown in this section, the model displays a great deal of properties stemming from the fact that the system is redundant which allows for choosing different solutions to the motion planning problem whenever certain constraints are imposed on the model. SURE_REACH's approach to movement and motion planning introduces a versatile framework for reaching movements which can be altered in case the arm experiences reduction in its mobility, suffers complete immobilization of some of the joints and for when an obstacle needs to be accounted for. The analysis of the model proposes that similar problems and processes might be happening in human brains when conducting reaching movements whether free or with a combination of constraints mentioned in this section, given its heavy psychology and neuroscience based approach. The following section attempts to contextualize the paper's insights and findings in a broader scientific as well as societal context and explore the direction in which such technologies and concepts like SURE_REACH are likely to develop in the future.

6 Contextual exploration

The aim of this section is to reflect upon which disciplines the theory and findings of this paper feed into. Furthermore, it aims to explore in what ways these findings can contribute to these areas and society in general. First and foremost, the three domains most connected to SURE_REACH are neuroscience, psychology and computer science. Neuroscience played a crucial role in the creation of the model as it describes the low level mechanisms allowing humans to learn. The Hebbian learning rule (Hebb, 1949) used for learning new information in the model is a direct example of how neuroscience contributed to the paper. Furthermore, another science domain partaking in laying the foundations of the model is psychology. For example the training process was inspired by observations of how humans do it. Most notably, the motor babbling phase of the model was designed after the way infants were observed to move (Meltzoff & Moore, 1997) through self-generated movements which aim at learning how certain neural activities map over to body movements. Lastly, the domain of computer science allowed the model to be implemented and simulated on a computer. This permitted complete control over the way the model is trained, tested, and generally stimulated. Additionally it allows us to look directly into the values of simulated neural connections. This would prove to be a lot more challenging to monitor within an organic brain. Nevertheless, it is crucial to highlight that SURE_REACH is only a simulated model and not a one-to-one replica of the human brain's parts that are responsible for movements and sensory-to-motor mapping.

Starting from the 2000's up to this day, machines are becoming an increasingly greater part of human lives, as the technology improves exponentially. There are robot vacuum cleaners that autonomously can vacuum an appointed area as well as come back and dock to its charging station with no human intervention. Industrial robotic arms help in many assembly and manufacturing jobs. As of now, there are even functioning robots inhabiting the surface of Mars which help to push the boundaries of science by collecting earth samples and data of environment. Therefore, it is safe to assume that machines will be used more often and for more complex tasks. Such anticipation for ground breaking advancement in technology and robotics is reflected in the recently conceived notion of **the fourth industrial revolution** also commonly referred to as **industry 4.0**. Industry 4.0 refers to rapid changes in the way work and production across industries firmly shifts towards a high level of inter-connectivity, automation and complex **AI** (Artificial Intelligence) process optimization. However, for the time being, the problem with robots is their lack of autonomy. For instance, even the most automated and modern retail warehouses still require human hands for packing all sorts of products into boxes. It is because such tasks require a high level of flexibility and autonomy. Products come in many different shapes and sizes and packing them often requires a unique set of movements for placing the objects inside of the box as well as taping the box closed. Such problems can not be simply solved by a hand-modeled machine, at least not without compromising on efficiency. Robots are mostly used in simple and repetitive tasks. They are also specialized to a very narrow scope of jobs that it can perform which leaves it with little room for adaptation. What the society is in need of, yet lacks greatly, are robots that are more general in the work they do and can create tailor-made solutions to problems "on the go" such as in the packaging example. The hardware of robots is not the issue here as companies like *Boston Dynamics* or *Agility Robotics* have shown time and time again that robots are capable of doing very complex maneuvers such as a backflip. What all these flashy displays have in common is that they were predominantly hard coded into robots' actuators to perform a sequence of actions. What robots lack are the "brains" to be able to fully utilize the mechanical potential of their bodies.

Allowing robots to robustly and flexibly interact with the physical world will open a plethora of possibilities for the betterment of society. Robots can take up jobs that are too risky or remote for a human to do. This could be helping in rescue missions, nuclear reactor maintenance (even in a high radiation environment), all day care of people with disabilities or lunar settlement construction before any humans arrive with an intent to settle on the moon. However, before these advancements can become a reality, there is a certain cognitive gap that needs to be bridged in order for robots to achieve close to human-like agility. Such a feat can be achieved by creating AI models which are highly inspired by human anatomy, cognition and neural processes. This process is commonly referred to as **biomimicry**, a design method that is no stranger to science and resulted in creation of inventions such as velcro, aerodynamic trains or neuromorphic computing (an example of which is SURE_REACH itself). The goal of this paper in the broader scope is to bring the state of arts of robotics and AI closer to the situation described in this section. This can be done by tackling one of the cornerstones of flexible interaction with a physical environment which is motion planning that can respond to changes in environment. For example, through obstacle avoidance or adaptation to defects in the physical structure of an arm (for instance broken joint or actuator). This paper hopes to provide more insight into SURE_REACH by describing it from the domain of robotics point of view, as well as propose a different set of parameters suitable for work of the model. The model can contribute to increasing robot agility by allowing them to perform reaching tasks through solving the motion planning problem in an entirely self-learned manner. Furthermore, even though the model in this paper was simulated in a two dimensional environment, it can easily be used in a three dimensional space after a few adjustments. However, the current limitation of the model is its computational cost which increases exponentially with the number of degrees of freedom of the robotic arm. Thus, for SURE_REACH to be further improved, it needs to be optimized for a higher number of degrees of freedom, which for comparison, is six for a human arm and 27 in a human hand.

Lastly, the paper's findings are not only useful for the field of robotics and AI but also psychology and neuroscience. Given that the model is designed after observing how humans move, it can provide valuable insight into human anatomy as it can partly explain the reason and mechanisms behind why humans move the way they do in certain situations. For example, when mobility of one of the joints is reduced during the rehabilitation process or when humans need to reach for something behind an obstacle. This information can be interesting as these are not trivial problems. To conclude, given the current societal and technological landscapes, it is reasonably expected that the world is headed towards greater automation and more AI driven solutions. This paper contributes towards achieving this goal by proposing improvements in the field of robotics through utilization of biomimicry in humans and combination of the domains of robotics, computational science and AI, psychology and neuroscience on an example of SURE_REACH algorithm (Butz et al., 2007).

7 Conclusion

The goal set for this paper was to find optimal parameters for running the SURE_REACH model as accurately as possible. After a series of tests on model's main coefficients which compared their accuracy with the ones found by Butz et al. (2007), it was found that the new parameter set is indeed more accurate for spatial and posture tasks compared to the older parameters. The newly found parameters are $\theta = 1.0, \varepsilon = 1.0, \gamma = 1.0, \beta = 0.1$ with the iteration batch being 400 000. Additionally, the paper aimed to explain the model in a clear and digestible manner as well as showcase its properties. The paper provided a detailed explanation of the model with easy to digest examples which make the ideas presented more tangible. Furthermore, the properties of the model were described and visualized with an example per each of the model's properties.

Next, the model is presented in a larger scope as a little addition to the direction the domain of robotics, as well as world in general, is headed towards which is high automation and interconnectivity which is commonly described as the fourth industrial revolution or the industry 4.0. In the smaller scope, the model serves as a valuable addition towards bringing a closer attention to the interplay of psychology, neuroscience, robotics, AI and computer science and their possible impact on science.

The model implementation showed that it is not optimal to disregard motor commands which try to exceed joint limits from the training procedure. In future research it is advised to let motor commands generate even the invalid movements for possibly better model performance at joint limits. Next, the most optimal gamma coefficient value found in this study indicates a possible need for change of the way STMM calculation is designed mathematically. Furthermore, more investigation is needed for the model trained on, and designed for a different arm architecture, one that is more complex by introducing more degrees of freedom. The model can also be more densely populated with posture and hand space neurons. Additionally, the model can be modified to work in a three dimensional workspace. After said improvements and modifications, the performance as well as the properties of the model should be tested again. Additionally, more sophisticated statistical tools are expected to bring more insight into the performance, validity and accuracy of the model.

To conclude, the SURE_REACH model is an interesting case of an unsupervised learning model based on research of neural mechanisms observed in humans. The model can solve motion planning as well as inverse kinematics problems even when it has constraints set onto it. Lastly, the model can be a great way for further investigation of how humans move as well as how to allow robots to achieve greater agility of movement.

8 Appendix

Glossary

- PMM - posture memory matrix
- MTV - movement trace vector
- MTM - movement trace matrix
- STMM - sensory to motor mapping

Coefficients used in the model

Parameters found by Butz et al., 2007: $g = \frac{\pi}{12}$ (or $15°$), $\varepsilon = 0.001$, $\rho = 0.1$, $\theta = 0.1$, $\beta = 0.172$, $\gamma = 0.434$

References

Bellman, R. (1957). Dynamic programming. *Princeton University Press*.

Berthier, N. E., Rosenstein, M. T., & Barto, A. G. (2005). Approximate optimal control as a model for motor learning. *Psychological review*, *112*(2), 329.

Bolognani, S., Bolognani, S., Peretti, L., & Zigliotto, M. (2008). Design and implementation of model predictive control for electrical motor drives. *IEEE Transactions on industrial electronics*, *56*(6), 1925–1936.

Butz, M. V., Herbort, O., & Hoffmann, J. (2007). Exploiting redundancy for flexible behavior: Unsupervised learning in a modular sensorimotor control architecture. *Psychological Review*, *114*(4), 1015.

Cruse, H., Steinkühler, U., & Burkamp, C. (1998). Mmc-a recurrent neural network which can be used as manipulable body model. *From animals to animats*, *5*, 381–389.

D'Souza, A., Vijayakumar, S., & Schaal, S. (2001). Learning inverse kinematics. *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, *1*, 298–303.

Guechi, E.-H., Bouzoualegh, S., Zennir, Y., & Blažič, S. (2018). Mpc control and lq optimal control of a two-link robot arm: A comparative study. *Machines*, *6*(3), 37.

Hebb, D. O. (1949). The organization of behavior. *New York*.

Jordan, M. I., & Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive science*, *16*(3), 307–354.

Kano, M., Kawato, M., & Suzuki, R. (1989). *Acquisition of inversekinematics and inverse-dynamics model of a redundant arm by the feedback error learning* (tech. rep.). Jpn IEICE Tech Rep MBE 88-171.

Kawato, M. (1990). Feedback-error-learning neural network for supervised motor learning. In *Advanced neural computers* (pp. 365–372). Elsevier.

Lenz, I., Knepper, R. A., & Saxena, A. (2015). Deepmpc: Learning deep latent features for model predictive control. *Robotics: Science and Systems*, *10*.

Meltzoff, A. N., & Moore, M. K. (1997). Explaining facial imitation: A theoretical model. *Infant and child development*, *6*(3-4), 179–192.

Rosenbaum, D. A., Engelbrecht, S. E., Bushe, M. M., & Loukopoulos, L. D. (1993). Knowledge model for selecting and producing reaching movements. *Journal of motor behavior*, *25*(3), 217–227.

Rosenbaum, D. A., Loukopoulos, L. D., Meulenbroek, R. G., Vaughan, J., & Engelbrecht, S. E. (1995). Planning reaches by evaluating stored postures. *Psychological review*, *102*(1), 28.

Rosenbaum, D. A., Meulenbroek, R. J., Vaughan, J., & Jansen, C. (2001). Posture-based motion planning: Applications to grasping. *Psychological review*, *108*(4), 709.

Whitney, D. E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on man-machine systems*, *10*(2), 47–53.