A Multilevel, Posture-based Model for Motor Control in Simulation and Robotic Applications

Divya A. Gunasekaran
Dartmouth College

Follow this and additional works at: https://digitalcommons.dartmouth.edu/senior_theses

Part of the Computer Sciences Commons

Recommended Citation
https://digitalcommons.dartmouth.edu/senior_theses/76

This Thesis (Undergraduate) is brought to you for free and open access by the Theses and Dissertations at Dartmouth Digital Commons. It has been accepted for inclusion in Dartmouth College Undergraduate Theses by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.
A Multilevel, Posture-based Model for Motor Control in Simulation and Robotic Applications

Divya Gunasekaran
Dartmouth College
June 3, 2011
Abstract

This paper presents a multilevel, posture-based motor control model intended to plan collision-free movements in a 3D environment while maintaining computationally efficiency and accurately imitating human and primate motor function. Our model is a comprehensive approach that addresses the storage and lookup of postures and movements, path planning and the generation of new movements, and learning with experience. We demonstrate the functionality and computational advantages of the model through preliminary testing on a humanoid robot.

1. Introduction

Robustness, speed, and smoothness of execution are features of human and primate motor control faculties that many in the fields of robotics, computer simulation, and digital animation are attempting to replicate. Yet, how animal motor control is represented, organized, and carried out at the cortical and even neuronal levels is still an area of research and debate in neuroscience. Thus, there is an ongoing effort to both imitate human and animal motor control and to better understand those systems.

In this paper, we present a holistic approach to motor control in the form of a multilevel, posture-based artificial intelligence system that can plan and execute collision-free movements for novel reach tasks and quickly complete previously encountered tasks in 3-dimensional space. The primary goals we strive to achieve with this model are to closely reflect human and primate motor control – in the way it is organized in the primary motor cortex, how tasks are solved, and aesthetically how the execution of movements appears – and to achieve a reasonable degree of computational efficiency.
We also discuss the implementation of the model and results of initial testing on a humanoid robot (with a tracked base, however, rather than 2 legs). The current implementation and testing are restricted to one arm of the robot in a stationary position. By implementing the model, we hope to gain greater insight into motor function in the brain as well as work towards a robust, fast motor system that can be used in other applications.

In Section 2, we briefly summarize some of the relevant work that has been completed in the field. Section 3 details the model’s framework and the neurological evidence that motivates its design, and section 4 gives an overview of the implementation. We discuss the results of testing and the computational advantages of our model in Section 5 and possible improvements to the model and areas for future work in Section 6.

Before moving on, we will first define terms that will be used throughout this paper. A posture refers to a set of joint angles that determine the orientation of a limb or set of limbs. That is, a posture is an element of the configuration space (C-space). The workspace is the set of all points in space that are reachable by the actor, while the task space is the set of points in space in the neighborhood of a given task. Searches in the workspace are considered global, whereas searches in the task space are local. A position is a point in the workspace or task space. We use the term movement to mean a sequence of postures and path to indicate a sequence of positions.
2. Related Work

The development of motor control systems in robotics and simulation has applications as diverse as gaming and computer animation, factory automation, physical therapy and robotic prosthetics, and autonomous cars and vehicles, to name a few. Because we strive to develop a comprehensive model that touches on several areas in the development of motor control, the body of related work is vast. We, therefore, only focus on those that are most relevant to our model or most prominently used.

Our model is most closely related to [10] and [12], as parts of our model’s memory framework and learning system are derived from ideas and equations presented there (see sections 3.1 and 3.3.6, respectively). A key idea used in [10] that can be found in our model is the discretization of the workspace – which is commonly done in 2-D planning algorithms (e.g. [6], [3]) – combined with a mapping of postures to the resulting discrete nodes.

While our motor control system is posture-based, we have decided against the use of random posture generation, which is at the heart of the Rapidly-Exploring Random Trees algorithm [7] and a significant component of [10] and object manipulation for a humanoid robot in [4]. One of our goals is the generation of smooth and natural movements, which random generation does not necessarily guarantee. Although, smoothing techniques can be applied and have demonstrably produced good results for complex robots (e.g. [18]), random generation is too incongruent to animal motor processes. In our model, we largely depend on the use of a standard gradient descent method to generate new postures, where each iteration of the method brings the robot’s end effector closer to its goal (see section 3.3.3).

Inverse kinematics functions are also widely used in conjunction with other methods in planning and generating postures and movements [4], [2], [7], [16]. It is a method, however, that
we have consciously stayed away from due to the complexity that arises for multi-joint systems in a 3-dimensional environment; the problem of redundancy (i.e. for a given position, there exist an infinite number of postures that can reach that position); and the potential of the solution to converge to a local minimum. Instead, we focus on using forward kinematics in conjunction with other methods.

3. Framework

In this section, we first describe the memory framework and the motivation behind its design. The memory framework is layered and consists of posture-based, movement, and ethological memory layers. We then describe each of the levels of our multilevel motor control model. There are four motor control levels, which interact with the memory layers to varying degrees.

3.1. Memory Framework

The three layers of memory are posture-based, movement, and ethological memory. The posture-based memory is the lowest-level memory in our framework and is a store of postures and their mappings to positions in the workspace. We use the memory structure prescribed by the memory-based posture planning (MBPP) model in [10] for our posture-based memory. This structure divides the workspace into regions of space called *cells*. In our (3-dimensional) case, cells are cubic volumes of space. Each cell represents both the volume of space it contains as well as a memory store of postures, where each posture in a cell’s memory store is such that the position of the end effector given by that posture lies within the volume of space contained by the cell.
In our use of the MBPP cell structure, we store postures for the right and left arms of the robot separately because each task is only carried out by one arm and the positions of that arm are not affected by the joint angles of the other arm. Additionally, each cell has an obstacle flag that indicates whether the volume of space enclosed by the cell is occupied by an obstruction.

The movement layer is a store of movements, where – as previously defined – each movement is a sequence of postures. Each movement maps to two positions in the workspace – the position of the end effector at the start of the movement and the position of the end effector at the end of the movement – and each of those positions maps to a cell in the MBPP cell structure. Movements also carry a rating indicating how useful they are. The rating of movements will be discussed in greater detail in section 3.3.6.

The ethological memory layer represents a higher level memory that consists of ethological function categories (e.g. reaching, eating, defense). Each category contains a set of movements that perform that ethological function in some way. For example, the eating category might include hand-to-mouth movements. Furthermore, each ethological function has its own weights that are given to the costs used to rate movements and select postures. We use a spatial error cost and a travel cost, as defined in [12], which measure how accurately a target position is attained and the cost of joint movements from a start to end posture, respectively. The rating given to a movement is therefore dependent on the ethological function that

Figure 2. Each MBPP cell stores a set of postures that map to the workspace points enclosed by that cell. Reprinted from [10].
movement carries out, as the weights associated with an ethological function reflect its requirements. For example, spatial accuracy would be more highly weighted for reaching in order to successfully attain a reach target, while speed (which could be represented as minimum joint displacement) would be more highly weighted for defense in order to block an attack in time. Movement rating and posture selection will be discussed in greater detail in section 3.3.6.

3.2. Motivation for Memory Framework

Studies of animal motor cortices support the concept of a layered memory framework. Graziano details stimulation results that suggest regions of the motor cortex have multiple types of organization, specifically by somatotopy, spatial location of the hand, and ethological function [1]. Somatotopic organization of the motor cortex has been observed in numerous studies and is a generally accepted finding [9]. Graziano has also found cortical mapping of spatial location of the hand for monkeys, with lateral sites of the cortex corresponding to hand locations in upper space (close to the subject’s head), sites that are medial along the central sulcus corresponding to mid-level space, and the most medial sites corresponding to lower space (close to the ground) [1].

Furthermore, movements associated with different ethological functions have also been observed from stimulation on a behavioral timescale (i.e. 500 ms) of rat, cat, and monkey cortices. Rats exhibited exploratory, oscillatory whisker movement, and cats exhibited reaching movements.

Figure 3. The end postures observed when sites in the primary motor cortex of monkeys were stimulated on a behavioral timescale (i.e. 500 ms). The different movements elicited from the stimulation can be categorized by ethological function. Reprinted from [1].
with the forepaw. In monkeys, a range of movements were observed when different cortical sites were stimulated. These movements were categorized by ethological function as defense, hand-to-mouth, object manipulation, grasping, or climbing and leaping. Even when the monkey’s hand had different starting positions or had weights attached to it, stimulation of the same cortical site resulted in the same final position. Thus, our layered memory framework reflects the multiple types of organization found in animal motor cortices.

The use of postures as the basic unit of our model is supported by experiments at the neuronal level as well as human subject experiments. Hand position, joint angles of the arm, and grip aperture and recorded the neuronal activity of a monkey while the monkey was allowed to freely move its limb and perform natural behaviors [1]. Results showed that each neuron had a preferred posture and neurons fired more during movements that terminated near their preferred end postures and less during movements that terminated far from their preferred end postures in C-space. After recording from a neuron, the same cortical site was then electrically stimulated through the same electrode used for recording, and the monkey’s arm adopted a posture that closely matched the neuron’s preferred posture. These findings show that neurons are significantly tuned to posture, although neurons may be influenced by other factors as well since the preferred-end-posture model did not explain all of the variance in neuronal activity.

Observations and experiments with human subjects provide further support for the use of postures as a fundamental unit in movement planning. In [11], Rosenbaum argues that to achieve the smoothness and bell-shaped velocity curves characteristic of the hand and reaching movements of humans and primates, knowledge of the final position and time needed to get there are required before executing the movement. In [14], Rosenbaum reports human subjects demonstrating the end-state comfort effect in a variety of reaching and grasping tasks. The end-
state comfort effect is the sacrifice of initial comfort when first grasping an object in order to achieve comfort or better control when bringing the object to its final position, which suggests that humans have already planned a goal posture before initiating a given task.

3.3. Overview of Level Framework

Our multilevel model consists of four levels. Level 1 is considered the actor level. It is primarily responsible for searching across movements stored in memory, executing movements, and storing new movements. Level 2 generates new movements (by constructing sequences of intermediate postures) given a path in the workspace. Level 3 performs extremely simple path planning for the end effector in the workspace, while Level 4 executes a more robust and computationally expensive path planning algorithm for the end effector in the workspace. Control switches between levels according to the flow chart in Figure 4.

3.3.1 Initialization

Tasks are given in the form of a reach target in the workspace, the arm (right or left) that should carry out the task, and the ethological function associated with the task. However, before any tasks are received, the MBPP cell structure is first generated. Empty sets representing ethological function categories may also be created before any tasks are received, but these sets can also be generated on the fly when tasks of new ethological functions are received. For each task, we query the robot’s servo positions in order to obtain its starting posture. We then determine the starting position of the end effector using forward kinematics.
Figure 4. Flow chart describing how control switches between each of the 4 levels.

Target, [Reaching=ethological function], [Right or left] arm

Look up movement according to type of task

Level 1

Level 2

Level 3

Level 4

Initial, final positions

D* Lite planning for end effector

Collision free?

No solution

Collision free?

Collision free?

Initial, final positions

Generate a movement

Plan straight-line path for end effector

Collision free?

no

yes

no

yes

no

yes

no

yes

Level 4

Level 3

Level 2

Level 1
3.3.2 Level 1

Level 1 first searches across stored movements that belong to the same ethological function category as the given task. In order to be considered usable, movements must satisfy two requirements. First, the Euclidean distance between the current starting position of the end effector and initial position of the end effector dictated by the movement and the distance between the reach target and the final position of the end effector dictated by the movement must be less than a user-defined threshold $\sigma$. This allows us to reuse movements for which the initial and final positions do not exactly match the current starting position and reach target, respectively, but are “close enough.” Second, the movement’s rating must be above another user-defined threshold $\lambda$. If a movement’s rating falls below the threshold, then we do not use that movement in the hopes of developing a new movement with a better rating. Movement rating is described in section 3.3.6.

If a stored movement is found such that the first requirement is failed but the current starting position and the starting position of the movement lie within the same cell and the reach target and final position of the movement also lie within the same cell in the MBPP structure, then we adjust the movement by calculating a new final posture. This is done by treating the second to last posture in the movement as the initial starting posture, keeping the reach target the same as that given by the task, and generating a new movement. The procedure for movement generation is outlined in the next section.

Once a usable movement is found, it is then checked to see if it results in any collisions with the environment. Collision detection is detailed in section 3.3.7. If the movement is collision-free, the movement is executed. After the task is completed, the movement – if it is an adjusted movement or newly generated movement – is rated and stored according to its
ethological function. Not only is the entire movement stored, but all subsequences of the movement, the reverse movement, and all subsequences of the reverse movement are also stored as separate movements. This allows us to avoid the computations involved in planning and generating movements for ones that have already been created as segments of a longer movement.

However, movements are only stored under the condition that a similar, better rated movement is not already stored. By similar, we mean the corresponding starting and final positions of the movements belong to the same cells and the Euclidean distance between the final positions are less than $\sigma$. If a similar and better rated movement exists, the new movement is not stored. If a similar movement with a worse rating exists, that movement is replaced by the new movement. Otherwise, the new movement is simply added.

If the usable movement results in a collision or no usable movements are found, control switches to Level 3 for path planning.

### 3.3.3 Level 2

Level 2 is responsible for generating new movements given a path in the workspace. For each point in the path, we try to find a posture that brings the end effector to that point. First, we search across the postures that are stored in the cell to which the point belongs. We calculate a total cost for each stored posture according to the same costs used for movement rating (see section 3.3.6) and choose the posture with the lowest total cost. If the Euclidean distance between the position of the end effector for the posture with the lowest cost and the path point is greater than $\sigma$ (the same user-defined threshold referred to in the previous section) if the posture results in a collision, or if the cell has no stored postures, then we generate a new posture.
To generate a new posture, we follow the gradient descent method outlined in [10] using the equation

$$\tilde{q}_{\text{new}} = \tilde{q}_{\text{curr}} + J_f^+ (\tilde{q}_{\text{curr}}) \delta^* (p_{\text{curr}} - p_{\text{target}})$$  \hspace{1cm} (1)$$

In equation (1), $p_{\text{curr}}$ is the previous path point and $q_{\text{curr}}$ is the corresponding posture for that point. $p_{\text{target}}$ is the path point for which we are currently trying to generate a posture, $\delta$ is the step size, and $J_f^+ (q_{\text{curr}})$ is the Moore-Penrose pseudoinverse of the Jacobian of the forward kinematics function evaluated at $q_{\text{curr}}$. We use the Moore-Penrose pseudoinverse because the Jacobian is not a square matrix in our case.

As in [10], we check whether any of the joint angle constraints of the robot have been violated after each iteration of the gradient descent method. The joint update for that iteration is undone at any joint whose constraints have been violated. We also calculate the position of the end effector given by the newly generated posture using forward kinematics. Our stopping criterion is that the Euclidean distance between the position of the end effector and the path point must be less than $\sigma$. Once the stopping criterion has been satisfied, we add the posture to the memory store of the appropriate cell.

If the posture generated via gradient descent results in a collision, we check every posture stored in the relevant cell and choose any collision-free posture. If no such posture exists and the cell does not contain the reach target, we change the obstacle flag for the cell to indicate an obstruction and then pass control to Level 3 to re-plan a path (which would necessarily avoid that cell).

If a collision-free posture has been determined, we add that posture to the sequence of movements and try to determine the posture for the next point in the path using the same procedure described above. When a collision-free posture has been determined for every point
in the given path, a complete movement has been generated and control switches to Level 1 for movement execution and storage.

Otherwise, the process halts and the target is declared unreachable once a collision-free posture cannot be generated for a point on the path.

3.3.4 Level 3

Level 3 does straight-line path planning for the end effector in the workspace by treating the end effector as a point mass in space. We obtain an equation for a straight line between the current starting position and reach target using

\[ p_i = p_{\text{start}} + (p_{\text{final}} - p_{\text{start}}) \times t \]

(2)

\( p_{\text{start}} \) is the current starting position, \( p_{\text{final}} \) is the reach target, \( p_i \) is an intermediate path point, and \( 0 \leq t \leq 1 \) is a real number. We choose a sequence of \( t \)'s such that each \( p_i \) lies in a cell that is adjacent to both the cell containing \( p_{i-1} \) and the cell containing \( p_{i+1} \) for \( 1 \leq i \leq k \), where \( p_0 = p_{\text{start}} \) and \( p_{k+1} = p_{\text{final}} \). If the last \( p_i \) generated lies in the same cell as \( p_{\text{final}} \), we replace \( p_i \) with \( p_{\text{final}} \). Otherwise, we add \( p_{\text{final}} \) to the end of the path.

We then check whether any of the cells that the path goes through is obstructed by an obstacle (which is determined by checking the cell’s obstacle flag). If so, control switches to Level 4 to try a different path-planning algorithm. If, on the other hand, the path is valid, control switches to Level 2 for movement generation.

Though the planning in Level 3 is not robust, it is extremely simple, fast, and computationally inexpensive. We decided not to use a more robust path planner like hill climbing algorithm for Level 3 because of the likelihood for such an algorithm to find collision-free paths in an environment with obstacles, but ones that are suboptimal and result in unnatural movements.
3.3.5 Level 4

Level 4 also does path planning for the end effector in the workspace, but uses an algorithm that is robust and finds an optimal path if the goal is reachable. We use D* Lite, which was developed by Sven Koenig and Maxim Likhachev and presented in [3]. D* Lite is an incremental heuristic search algorithm that plans backwards starting from the goal vertex. It maintains an estimate of the cost to the goal and a one-step lookahead cost for each vertex (which are initially set to infinity for all vertices except the goal), and only expands vertices whose goal cost has changed and that are relevant for determining a shortest path. In our application of the algorithm, we use the center of cells as vertices, allow paths to cut across the diagonals of cells, and use straight-line distance as our heuristic.

If a path is found from the current starting position to the reach target, control switches to Level 2 for movement generation. Otherwise, the process halts and the target is declared unreachable.

3.3.6 Posture Selection and Movement Rating

Posture selection and movement rating are both based on a spatial error cost and travel cost, as defined in [12].

The spatial error cost is a measure of how accurately the end effector of a given posture attains a reach target. It is simply the Euclidean distance between the position \((x_p, y_p, z_p)\) of the end effector for the given posture \(p\) and the reach target \((x_{target}, y_{target}, z_{target})\):

\[
S_p = \sqrt{(x_{target} - x_p)^2 + (y_{target} - y_p)^2 + (z_{target} - z_p)^2}
\]  

(3)
The travel cost gives us the cost of angular joint displacement from a start to end posture. The travel cost $V_j$ for the $j^{th}$ joint to experience an absolute angular displacement of $\alpha_j$ in $T_j$ time is calculated by

$$V_j(\alpha_j, T_j) = \left( \frac{k_j \alpha_j}{r} \right) \left[ 1 + \frac{T_j - T_j^{*}(\alpha_j)}{s^2} \right]$$  \hspace{1cm} (4)

In equation (4), $r$ is the unit of absolute angular displacement, which is radians in our case, and $s$ is the unit of time, which is seconds. The $j^{th}$ joint has an expense factor $k_j$. Expense factors allow us to favor the use of certain joints over others. The travel cost for the $j^{th}$ joint with an expense factor of $k_j$ also involves calculating the optimal duration $T_j^{*}(\alpha_j)$ for absolute angular displacement $\alpha_j$ according to Fitt’s Law:

$$T_j^{*}(\alpha_j) = k_j \ln(\alpha_j + 1)$$  \hspace{1cm} (5)

[12] substitutes the common optimal time $T_p$ for the time $T_j$ in equation (4). The common optimal time given by equation (6) minimizes the entire travel cost to a posture under the constraint that all joints start and end their movements simultaneously. Thus, $T_p$ is the same for all joints.

$$T_p = \frac{\sum_j k_j \alpha_j T_j^{*}(\alpha_j)}{\sum_j k_j \alpha_j}$$  \hspace{1cm} (6)

In our model, we calculate the expense factor $k_i$ by taking the ratio of the length $L_j$ of the limb segment that joint $j$ moves to the sum of all limb segment lengths, as shown in equation (7). We weight the torso twice as much as the longest limb. This is intended to favor smaller limb movements.
\[ k_{ij} = \frac{L_j}{\sum_{i=1}^{n} L_i} \]  

(7)

Then the entire travel cost from a start posture to posture \( p \) is the sum of the joint travel costs across all joints:

\[ V_p = \sum_{j=1}^{n} V_j(\alpha_j, T_j) \]  

(8)

Although posture selection and movement rating are both based on the spatial error cost and travel cost, we use a different calculation for each. When selecting a posture from a cell’s memory store, we have an initial posture and a reach target as part of our initial conditions. The spatial error cost \( s_p \) and travel cost \( v_p \) is calculated for each posture \( p \) in the memory store. As stated previously, each ethological function category has its own set of weights \( w_s \) and \( w_v \) – which sum to 1 – that are given to the spatial error cost and travel cost, respectively. We keep track of the maximum spatial error cost \( \text{Max}S \) and maximum travel cost \( \text{Max}V \) across all postures stored in the cell and then calculate the total cost \( C_p \) for each posture \( p \) according to equation (9), which was developed in [12].

\[ C_p = w_s \frac{(s_p)}{\text{Max}S} + w_v \frac{(v_p)}{\text{Max}V} \]  

(9)

All postures have a score between 0 and 1 and we select the one with the lowest cost. Posture selection occurs during the generation of new movements in Level 2 (see section 3.3.3). We do not store a posture’s total cost because of its dependency on the reach target and initial posture, which will change depending on the given task.

For movement rating, we use a different calculation for two main reasons. First, movements consist of a sequence of postures, so our calculation of the travel cost must account for cases that have more than 2 postures. Second, it is more difficult to compare movements to
one another since they may differ in length, so we should avoid normalizing them using the maximum cost values of all stored movements. Instead, we rate movements independently of other stored movements according to equation (10).

\[
R_m = w_s \frac{1}{1 + S_{p_k}} + w_v \frac{v_{o,k}}{\sum_{i=0}^{k-1} v_{i,i+1}}
\]  

(10)

Let \( m \) be a movement consisting of postures \(<p_0, p_1, \ldots, p_k>\), where \( p_0 \) is the starting posture and \( p_k \) is the final posture. We calculate the spatial error cost \( S_{p_k} \) using the reach target and \( p_k \) in equation (3). For the travel cost of the movement, we divide the travel cost from \( p_0 \) to \( p_k \) by the sum of the travel costs from \( p_i \) to \( p_{i+1} \) for \( 0 \leq i \leq k-1 \). With this ratio, we are able to compare the travel cost of the given movement to the travel cost of a movement that goes straight from posture \( p_0 \) to posture \( p_k \). For example, a movement that follows a circuitous path in order to avoid obstacles will have more intermediate postures and therefore a larger travel cost than a movement that goes directly from the starting to final posture. Thus, larger ratios correspond to movements with better travel costs. A ratio of 1 is considered ideal, although in testing we observed movements that actually had ratios larger than 1.

Since larger values correspond to more desirable movements in the second part of equation (10), we adjust the spatial error cost so that higher rating values correspond to better movements. To do this, we add 1 to the spatial error cost and take the reciprocal of the sum. The addition of the constant is to avoid undefined values when the spatial error cost is 0. The weights \( w_s \) and \( w_v \) for the spatial error cost and travel cost, respectively, which must sum to 1 and are specified according to ethological function, are then applied.

Movement rating takes place in Level 1 (see section 3.3.2) before new or adjusted movements are stored. Higher ratings indicate more desirable movements.
3.3.7 Collision Detection

In a continuous environment, collision detection would theoretically require us to check every point on the arm to ensure it does not come into contact with any obstacles. However, because we discretize the workspace using the MBPP cell structure, we need only check every cell that the arm lies in.

For a given posture, we obtain the positions of each joint using forward kinematics. For each pair of adjacent joints, we use the straight line algorithm described in section 3.3.4 to determine a sequence of points between the joint positions. The exact points themselves are immaterial, but from them we are able to obtain the set of cells that the arm lies in. We check the obstacle flag of each of these cells to see if any of the cells are obstructed by an obstacle. If so, the posture results in a collision. Otherwise, the posture is deemed collision-free. Since movements are comprised of a sequence of postures, detecting collisions for a movement simply involves checking each posture for collisions.

Collision detection occurs in levels 1 and 2 when movements are being adjusted or generated (see sections 3.3.2 and 3.3.3, respectively).

4. Implementation Details

4.1. Robot

The robot used is a BrainBot, developed by Jon Hylands and HUV, Inc.. Each of its arms consists of four joints: 2 at the shoulder (for pitch and rotation), a revolute joint at the elbow, and a rotational joint at the wrist. There are also three torso joints to control the torso pitch and yaw. Both the wrist joint and torso yaw remain constant. Dynamixel AX-12+ actuators are used for the joint servos.
BrainBot runs a Windows XP computer with an Intel Core 2 Quad CPU at 2.83 GHz and with 2.96 GB of RAM. We interact with the hardware through a server developed in Squeak that runs on the BrainBot computer.

4.2. Data Structures

The model is implemented in Matlab, with some functions written in C++ and used in Matlab. The MBPP cell structure is implemented as a cell array, so each cell has a unique integer cell index. We set the cell dimensions to be 2x2x2 in., $\sigma$ as 0.2 in.$^3$, and $\lambda$ as 0.5.

We implement the ethological memory layer as a set of dynamically-sized hash tables. That is, each hash table corresponds to an ethological function and all the movements for a single ethological function are stored in the same hash table.

Each movement uses an ordered pair consisting of the indices of the cells that contain its starting and final positions as its key. These two integers are converted to a single string (the first integer, a space, and then the second integer), which is then used in the djb2 algorithm developed by Dan Bernstein. The hash function therefore computes a unique value for each key. Each entry in the hash table points to a doubly linked list – where each node in the list stores a movement – in order to handle collisions. The size of the hash table is doubled when its load factor reaches 0.5. Stored movements are never deleted from the table (only replaced by other movements), so we do not consider the case of decreasing the table size.

4.3. Source Code Credit

The forward kinematics function used in this implementation was developed by Richard Lange and implemented in Matlab by Daniel Muldrew. The min-priority queue used in the D* Lite algorithm was written by Andrea Tagliasacchi [19] and modified by the author of this paper.
Code used to create TCP/IP sockets to connect to the BrainBot’s server from within Matlab was created by Peter Rydesäter et al. [15].

5. Testing and Results

In this section, we explain the results of preliminary testing of the model and then relate the results to specific components of the model.

5.1. Preliminary Testing

We demonstrated the use of the model on the robot’s right arm in environments with and without obstacles for 2 different cases, where each case has different initial and final positions.

Figure 5. The initial and final postures assumed during preliminary testing. The pictures on top show the initial posture (top left) and final posture (top right) assumed for the reach target in Case 1. The pictures on the bottom are the initial posture (bottom left) and final posture (bottom right) assumed for the reach target given for Case 2.
The starting posture for Case 1 was the arm straight out to the robot’s side and the reach target was in front of and above the robot, while the starting posture for Case 2 was the arm down in front of the robot and the reach target was to the front and right of the robot (see Figure 5).

<table>
<thead>
<tr>
<th></th>
<th>Distance (in.) from initial to target position</th>
<th>Number of intermediate postures</th>
<th>Time (sec) with no stored movements or postures</th>
<th>Time (sec) with usable stored movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1 – no obstacles</td>
<td>19.5154</td>
<td>11</td>
<td>3.6792</td>
<td>1.3515</td>
</tr>
<tr>
<td>Case 1 – obstacles</td>
<td>19.5154</td>
<td>9</td>
<td>225.5019</td>
<td>0.9334</td>
</tr>
<tr>
<td>Case 2 – no obstacles</td>
<td>17.1319</td>
<td>10</td>
<td>3.1961</td>
<td>1.1481</td>
</tr>
<tr>
<td>Case 2 – obstacles</td>
<td>17.4057</td>
<td>8</td>
<td>34.3187</td>
<td>0.8560</td>
</tr>
</tbody>
</table>

Table 1. This table shows the amount of time the model took to successfully generate movements to different reach targets. Timings do not include initializing data structures, determining the robot’s starting posture, executing the movement, or storing the movement since these actions (with the exception of movement execution) are done before the robot receives a task or after it has achieved the task.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Distance (in.)</th>
<th>Number of Points on Path</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight Line Plan</td>
<td>20</td>
<td>20</td>
<td>.0030</td>
</tr>
<tr>
<td>D* Lite</td>
<td>20</td>
<td>34</td>
<td>43.6916</td>
</tr>
</tbody>
</table>

Table 2. The results and amount of time taken for the Level 3 and Level 4 path planning algorithms to compute a path between two opposite ends of the MBPP cell structure when no obstacles are present.

In the obstacle-free environments for Cases 1 and 2, the model first entered Level 1 to determine if any stored movements are usable. If there were no stored movements, the model entered Level 3 and planned a straight-line path to the target. Level 2 then successfully generated a movement for the given path. The process took a little over 3 seconds in both cases. If in Level 1 a usable movement was found, the model did not need to enter any higher level and the process was completed in less than 2 seconds.
For the second type of environment, obstacles were placed in cells that the straight-line path traveled through in order to force the model to enter Level 4. For Case 2, the model followed the same procedure described as above with the exception that the straight-line path in Level 3 failed. Therefore, the model then entered Level 4 and determined an optimal path for the end effector using D* Lite. Level 2 successfully generated a movement for the path returned by Level 4. The multilevel process took approximately 34 seconds to generate a movement for an environment with obstacles and no available stored movements.

While time was still on the order of seconds for Case 2, it was on the order of minutes for Case 1 when obstacles were present. The model followed the same procedure outlined above for Case 2, but could not develop a collision-free posture for one of the path points while in Level 2. As a result, the obstacle flag for the cell containing the relevant path point was set and the model returned to Level 4 to re-plan a new path. Level 2 was then able to generate a collision-free movement once Level 4 returned a new path. Thus, the use of D* Lite twice for Case 1, but only once for Case 2 accounts for the significant time discrepancy between the two cases when obstacles are present.

In either case with obstacles present, the process finished in less than 1 second if a usable movement was found in Level 1.

Qualitatively, the movements generated look fairly smooth and natural when executed by the robot and no unnatural poses are adopted, which is an important standard for our model. When we look at the angle displacements across the movements for each joint, we see that they indeed follow smooth curves for most of the movements. Unfortunately, this is not true for Case 1 when obstacles are present. Here, the curves appear jagged around the fifth and sixth intermediate postures in the movement, which is when the arm is circumventing the obstacle.
Figure 6. The top left graph shows the path planned by Level 3 for the end effector of the robot in the absence of obstacles for Case 1. The shaded green data point represents the reach target. The remaining graphs show the angle displacement of torso pitch 1 (top right), torso pitch 2 (middle left), shoulder rotation (middle right), shoulder pitch (bottom left), and the elbow (bottom right) across the movement generated for the path shown in the top left graph. We can see that the angle displacements follow smooth curves. This is generally true of the movements that were generated during testing.
Figure 7. The top left graph shows the path planned by Level 4 for the end effector of the robot in the presence of obstacles for Case 1. The red region represents an obstacle that the robot must avoid. The remaining graphs show the angle displacement of torso pitch 1 (top right), torso pitch 2 (middle left), shoulder rotation (middle right), shoulder pitch (bottom left), and the elbow (bottom right) across the movement generated for the path shown in the top left graph. We see that the angle displacements do not follow smooth curves like those in Figure 6. However, the movement did appear smooth and natural when executed by the robot.
5.2. Computational Advantages of the Model

One of the biggest advantages to our framework is the discretization of the workspace. The MBPP cell structure divides the workspace into a finite number of cells, which then allows for the use of discrete path planning algorithms in levels 3 and 4. This also reduces the complexity of planning from a dimensionality of 7 in the C-space, where the angle for each joint is dependent on the one before it, to a dimensionality of 3 in the workspace. Collision detection is also simplified as we saw in section 3.3.7 because we only have to check a relatively small number of cells for the presence of obstructions.

Another advantage to our framework is that storing movements and postures enables us to avoid re-planning and recalculating movements for a task that has been previously accomplished. Moreover, with a mapping of postures and movements to workspace positions, we need only perform local searches when looking up movements and generating new movements, which provide faster performance than global searches. Our use of hash tables enables us to perform local searches in constant time. Usable movements were returned in approximately 1 second during preliminary testing.

The movements generated from our model are also more tailored to the requirements of the function they are intended to serve due to the rating system. The rating system allows for the ability to filter out less efficient or useful movements over time as more and potentially better movements and postures are learned.

Through testing, however, we discovered the importance of setting an appropriate threshold $\lambda$. If $\lambda$ is set too high, then the model will try to generate a new movement when a stored movement for the given task already exists and has the highest rating any movement for
that task will be able to achieve. If $\lambda$ is too low, then new movements will never be generated when a stored one exists even if better movements are possible.

As we saw in the previous section, significant gains in time were achieved by the inclusion of a simple path planning Level (Level 3) before using D* Lite (Level 4). When in an obstacle-free environment or an environment where obstacles do not obstruct the most direct path to the target, the speed and simplicity of the straight-line path planning algorithm allows the model to achieve a task in seconds. Even if the straight-line path fails and we need to resort to D* Lite, the time spent running the straight-line path planning algorithm can be considered negligible.

![End effector path in the workspace](image)

Figure 8. The path planned by Level 3 in the absence of obstacles for Case 2 is shown in blue, while the path planned by Level 4 in the presence of obstacles for Case 2 is shown in green. Both paths converge at the shaded green data point, which represents the reach target given for Case 2.
5.3. Next Steps in Testing

While we have demonstrated functionality of the model in fairly simple cases, testing needs to be done in more highly constrained environments to prove the model’s robustness. Specifically, the model needs to be tested in environments that require the use of unnatural postures in order for successful completion of a task.

Before this kind of testing can be done, however, the robot’s repertoire of stored movements and postures needs to be built up more through experience and possibly imitation learning. Imitation learning can be as simple as manually moving the robot’s arm through a sequence of postures and recording the joint angles for each one or it can be based on motion capture data from a human subject. The motivation for imitation learning comes from studies in which human infants and certain monkey species are unable to retrieve an object from an opaque box when the opening to the box is on the side rather than directly in front due to lack of inhibitory control [17]. Once they are shown the correct solution to retrieving the object, the infants and monkeys can then complete the task themselves. Our model behaves the same way – it cannot perform unnatural postures or more complicated movements until it has learned or experienced them – but more testing needs to be done in this area.

6. Discussion

The model presented in this paper is a holistic approach to motor control that strives not only to reflect human and primate motor control, but also to increase computational efficiency and reduce computational complexity. The multilevel motor control model comprehensively covers the range of an agent’s motor control response by first trying what it already knows either inherently or by habit (the lookup of stored movements in Level 1), assessing the most direct
route to its goal (straight-line planning in Level 3), resorting to more complex problem solving that may require first moving farther from the goal (D* Lite in Level 4), dictating how its body should move to accomplish its goal (movement generation and execution in levels 2 and 1, respectively), and learning new and better movements with experience (movement rating and storage in Level 1). By assembling these different responses into one model, we are able to achieve certain computational advantages without an increase in complexity and computation.

There are still many ways in which this model can be improved, expanded, and enhanced. The most prominent addition to this model would be the implementation of a visual system. Currently, there is no feedback system. Instead, the robot implicitly “knows” where obstacles are, where segments of its arm lie in the workspace for any posture (rather than using visual servoing [5], and what ethological function it is trying to carry out. The addition of visual feedback to the system would be a significant step forward in making the model even more representative of human and primate motor control.

Similarly, the inclusion of motion dynamics may also enhance the model. The consideration of dynamics was left out in the current version of the model due to the complexity of the area and its lack of necessity in our implementation for the BrainBot. The current interface for the BrainBot’s servos does not grant us fine-tuned control of the servo speeds, so attention was not given to velocity, acceleration, torque, and forces due to gravity and the Coriolis effect. The addition of motion dynamics, however, may make the model more applicable to other robotic systems and simulations in which such aspects play a crucial role in the control and function of the system.

A more sophisticated learning scheme might also improve the model. Currently, movements are classified and stored according to the ethological function they fulfill. This setup
makes the incorporation of learning and movement generation using Principle Component Analysis on stored movements, as used in [8] and [16], much easier in terms of accessing movements that are the same type of movement primitive (which are equivalent to our ethological function categories).

7. Acknowledgements

I would first like to thank my advisors, Jerald Kralik and Devin Balkcom, for their continual guidance and support. I would also like to thank Daniel Muldrew, Richard Lange, the other researchers in our lab, and Laura Ray for their help and feedback. This work would not be possible without Richard Granger and his involvement in the Brain Engineering Laboratory at Dartmouth College and the development of the BrainBot.

8. References


