2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Motion Generation Through Biologically–Inspired Torque Pulses

J. Neubert and N. J. Ferrier

Abstract— **Traditional robot controllers are not designed to produce human–like reactive motion—movements lasting tens of sample periods and requiring large accelerations. One of the major obstacles to producing reactive motions with contemporary controllers is that they rely on kinematic commands. The performance of short duration motions requiring large accelerations is dominated by the motion's dynamics; kinematic commands without an accurate dynamic model of the robot and task will lead to poor performance. Conversely, this paper presents a biologically inspired "torque command" that allows the dynamics of the motion to be communicated to the controller. The commands can be produced with only minor modifications to any existing control scheme that will not impact traditional operation. In addition, sensory input can be mapped *directly* to presented commands for latency sensitive tasks. The ability of the new commands to express a broad range of motions with a small number of parameters is shown experimentally. The experimental results also show that the presented torque commands can be used to learn a ball intercept task.**

I. Introduction

Prosthetics controlled by the human brain are currently a reality [1,2], but are limited to slow, controlled motions. One major reason behind this is the treatment of the user as a motion source, discarding important information about the motion's dynamics [3]. The brain is capable of low-level limb control using muscle activations, similar to robot joint torques. Allowing motions to be specified in such a format will provide the user more control over the device and a mechanism for producing reactive motions. In addition, the ability to produce biological-like reactive motion is becoming more important as robots assume a larger domestic role. If the robotic cat for monitoring the health of an elderly person in [4] could skillfully swat at a string it would be more likely to be treated as a natural part of the environment.

Currently, robots are designed primarily to produce slow, controlled motions because the majority of tasks require such motions. Only a small segment of tasks, such as hitting a baseball, require low latency, quick motions—referred to here as reactive motions. These motions typically last 10–200 sample periods and require large accelerations. This makes factors currently neglected in robot control, such as the discrete nature of the robot and the motion's dynamics, the dominate factors in performance. Constructing a controller that takes into account these factors is an involved and complex process. Moreover, a complex, specialized controller is not ideal for general purpose robots where reactive motions comprise only a small percentage of all motions. Even if existing controllers were able to produce the desired motions the command would have to specify positions, velocities, and accelerations. Such commands are computationally expensive to generate.

This work was supported by NSF IRI-9703352

J. Neubert is with the Dept of Mech. Engr., University of North Dakota, Grand Forks, ND 53202, USA jeremiah.neubert@und.edu

N. J. Ferrier is with the Dept of Mech. Engr., University of Wisconsin-Madison, Madison, WI 53704, USA ferrier@engr.wisc.edu

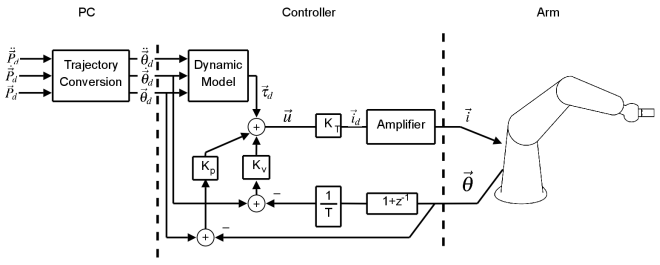


Fig. 1. Schematic of a traditional robot controller consisting of a PD joint controller augmented with a dynamic model for feedforward compensation.

In this paper we present a new method for commanding a motion using joint torques. The proposed method employs a minimal number of free parameters to specify the torques needed to produce a trajectory. The limited dimensionality of the command allows an autonomous learning system to find a mapping from raw measurements to the needed motions. Moreover, the new method for delineating a motion which specifies both the path and destination of the robot. Both simulation and experimentation are used to show the capabilities of the system to produce a wide variety of controlled motion. In addition, the look-and-move system described in [5] and briefly presented here demonstrates that the novel torque commands allow unprocessed sensory input—pixel positions—to be mapped directly to the torque commands needed to intercept a ball.

II. Background

The motion of a robotic manipulator is typically governed by a digital controller such as a digital signal processing (DSP) unit. The DSP contains the control law responsible for determining the control signal, $\vec{u}$, needed to track a position command, $\vec{\theta}$, and reject disturbances. Creating the needed control law is particularly difficult because robotic manipulators are multi-input multi-output (MIMO), highly coupled, nonlinear dynamic systems. For this reason advanced techniques such as sliding mode control [6] and adaptive control [7] have been employed to enhance stability and reduce the time required to eliminate disturbances. Precise tracking of the command is achieved through the use of some type of command feedforward compensation, as shown in Fig. 1. This utilizes prior knowledge of the desired trajectory and the robot's dynamics to predict the joint torques needed to track $\vec{\theta}$. An example of command feedforward robot control can be found in [8]. The main obstacle to produce

such motions creating a full trajectory, including positions, velocities, and accelerations. Construction of a fully specified trajectory can be computationally expensive, requiring operations such as predicting ball speed using the pitcher's arm motion, optimization non-linear constraints and negotiating constraints intrinsic to the robot [9]. This generates an undesirable amount of latency. Moreover, these systems assume the robot is, at least approximately, an analog system and do not allow for task dependant external forces to be specified. A specialized controller with large gains, complex discrete control law, and involved adaptive techniques may provide reasonable performance, but employing a controller optimized to a small segment of motion is undesirable for general purpose robots.

The alternative proposed here is to allow motions to be specified using joint torques. Motions parameterized with torques have been demonstrated to be smooth and human-like [10]. One such system presented by Atkeson and McIntyre [11] uses practice to determine the torques needed at each sample instance *a priori*. These torques are simply played back to perform the motion. The commands proposed in this work move beyond replaying torques, and instead use a limited number of parameters in the encoding. This makes it possible to map sensory input *directly* to the joint torques. This is similar to the mapping of visual input to joint positions outlined in [12].

III. Reactive Motion

One taxonomy of human motion categorizes it as reflexive, reactive, or conscious. Reflexive motions do not involve conscious choice and are performed with minimal latency. Conscious motions, conversely, are closed loop motions requiring constant mental involvement. Reactive motions are a hybrid of the two, consisting of low latency, open-loop motion based on sensory input. The objective of this work is to present a motion command that allows robots to mimic this reactive motor behavior. The resulting motion command is not intended for general use, but rather the small segment of tasks that require quick, low-latency motion. A "reactive" motion

- originates from a fixed starting point,
- is short in duration,
- contains relatively limited variation,
- is repetitive, and
- tolerant to small errors in position.

These criteria are reasonable when compared to human reactive motion. A baseball batter, for example, begins their swing from the same point each time and only attempts to hit the ball over a small region of space. The motion also requires a great deal of practice to achieve optimal performance. Additionally, "reactive" motions—open-loop motions—are not used to complete tasks, such as threading a needle, that require a high degree of precision.

A. Human reactive Motion

The exact format of human reactive motion has been the subject of debate for several decades. Some believe that humans map sensory input to a motion expressed in visual (hand) coordinates [13]. Conversely, others believe that the human brain maps sensory input directly to muscle activations, referred to as motor coordinates [14,15]. Still others believe that it is a combination of the two methods [16]. Visual and motor coordinates each have their strengths and weaknesses. The visual encoding is tightly linked to task space which simplifies learning, but incurs the added latency associated with converting the command to muscle activations. Conversely, motor coordinates require no such conversion and can be used to convey knowledge about the motion dynamics. Unfortunately, such commands are difficult to learn for a given task.

B. Robotic Reactive Motion

The hybrid representation presented in [16] is used in this investigation because it allows the system to take advantage of the visual representation's tight link to task space for efficient learning and the motor coordinate's ability to produce low latency motion and express knowledge of the motion's dynamics.

Visual Coordinates: Human visual or hand coordinates, for all intents and purposes, are equivalent to Cartesian coordinates. A robotic trajectory can be expressed in Cartesian space as a set of poses paired with time values, $\mathscr{P} = \{(\vec{P}_1, t_1), ..., (\vec{P}_m, t_m)\}$, and an objective function or parameter. The poses are represented by a plücker vector,

$$\vec{P}_i = [x \ y \ z \ \theta_x \ \theta_y \ \theta_z]^T, \tag{1}$$

and t_i is a scalar representing the desired time at which the manipulator is to pass through $\vec{P}_i$. This pair is referred to as a via point. The motion of the end effector between via points is specified by the given objective function or parameter that must be optimized. The most common objectives include minimizing motion jerk, motion time, and actuator effort. A description and comparison of various objectives can be found in [17]. Optimization results in a continuous function, $P_d(t)$, that gives the desired end effector position as a function of time, t.

The function $P_d(t)$ is important because it is needed to construct the task function [18],

$$e\left(P_d(t) - P\left(\vec{q}_{est}(t)\right)\right), \tag{2}$$

for use in identifying the optimal pulse parameters. $P\left(\vec{q}_{est}(t)\right)$ represents the realized end effector pose calculated by using the measured joint displacements at time t, $\vec{q}_{est}(t) \in \mathbb{R}^n$, where n is the number of robot joints. The visual coordinate representation is obtained from a standard trajectory planning algorithm. Because it is used primarily in training, the visual representation can be generated after the motion is complete using all available sensor data to ensure it represents the "ideal" trajectory.

Motor Coordinates: The second representation of the motion is expressed in motor coordinates, otherwise referred to as muscle activations. A variety of theories have been forwarded on how the activations are encoded. The most

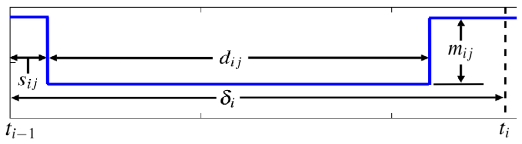


Fig. 2. An example torque pulse for joint j during the i^{th} interval. The distance the joint moves is dictated by the pulse magnitude m_{ij}, while path of the end effector is dictated by s_{ij} and d_{ij}.

intriguing theory put forth is presented by [19]. They theorize that the activations are stored as a series of pulses. The timing and magnitude of the pulses are adjusted to control total displacement and trajectory of the limb. This method for representing the motor commands is desirable because it contains a small number of degrees of freedom. By limiting the dimensionality of the solution space the search for the optimal command becomes a tractable task.

In a robotic system, the control signal, $\vec{u}$, would be comparable to muscle activations. u is a voltage that dictates the desired armature current as shown in Fig. 3. The armature current is directly proportional to joint torque. Thus, the pulses are referred to as torque pulses. More complex motions can be produced by combining multiple pulses to get $\mathscr{C} = \{c_1, ..., c_m\}$. The elements of $\mathscr{C}$ are referred to as torque commands. The torque command c_i produces the i^{th} segment of a motion which must be completed in the pulse interval δ_i. The torque command contains one pulse for each joint, $c_i = \{\vec{c}_{i1}, ..., \vec{c}_{in}\}$. The torque pulse for the j^{th} joint in the i^{th} segment is delineated by $\vec{c}_{ij} \in \mathbb{R}^3$ consisting of a magnitude, m_{ij}, a pulse initiation time, s_{ij}, and a pulse duration, d_{ij}. An example pulse is shown in Fig. 2.

C. Robot Controller for Producing Reactive Motions

The custom controller shown in Fig. 3 is similar to that in Fig. 1 with one important modification—the ability to accept a torque command, $\vec{m}_d$, as well as the standard joint space command, $\vec{\theta}_d$. In addition, the dynamic model has been removed, reducing the computational complexity of the control law. During a reactive motion the torque command is used to move the robot from one attractor point to another. The location of the attractor points is specified with the position command, $\vec{\theta}_d$. The value of $\vec{\theta}_d$ is fixed as the robot moves, during which time a new value of $\vec{\theta}_d$ is calculated. The value of $\vec{\theta}_d$ is only updated after the motion is complete. Note that the controller does utilize a velocity command because this simplifies the control law. Furthermore, one major reason for using torque pulses is to avoid the generation of a velocity command. To maintain stability the differential portion of the control law, $-K_v * \frac{d\vec{\theta}}{dt}$, assumes zero desired velocity; thus a large differential gain, K_v, will prevent the robot from achieving high speeds during a reactive motion. While increasing the complexity of the controller is undesirable, the problem can be addressed by lowering or eliminating the contribution from the differential part of the controller when $\|\vec{m}_d\| \neq 0$.

The traditional PD controller is left intact for three reasons. First, it is needed to complete the vast majority of tasks that do not require reactive motions. Recall that only a small

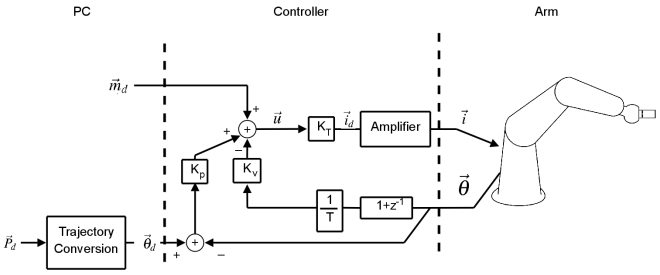


Fig. 3. The robot controller from Fig. 1 modified to allow a torque command $\vec{m}_d$ to be used to move the robot from one attractor point to another. The location of the attractor points is specified by $\vec{\theta}_d$.

segment of tasks meet the required criteria for a reactive motion. Second, the controller provides the attraction points that punctuate the beginning and end of a motion, guaranteeing stability. Third, it resists the motion generated by the pulses. This is important because a human limb has protagonist and antagonist muscles. As the protagonist actuates the joint, the antagonist is stretched generating increasing resistance to the motion. The proportional portion of the control signal, $K_v * \vec{e}$, acts in a similar manner. As the pulse moves the joint the proportional part of the controller provides spring–like resistance reducing the magnitude of the pulse as the motion concludes. The result is more human–like motion and a reduction in motion jerk at the conclusion of a pulse.

IV. Determining the Desired Torque Command

One objective of this work is demonstrating that torque commands can produce a variety of motions with a specific trajectory. To accomplish this an efficient method is needed for locating the optimal pulse parameters. The optimal set of pulses must pass through all the via points so that

$$e\left(\vec{P}_i - P\left(\vec{q}_{est}\left(t_i\right)\right)\right) < \lambda_{via} \;\forall\; \left(P_i,\; t_i\right) \;\in \mathscr{P}, \tag{3}$$

and minimize

$$\sigma_{dev} = \int_0^{t_f} \left\| e\left(P_d(t) - P\left(\vec{q}_{est}(t)\right)\right)\right\| dt. \tag{4}$$

The threshold λ_{via} in (3) is task dependant. In (4) the task function is integrated with respect to time because it allows larger deviations to be tolerated in portions of the motion where the robot is moving with greater speed. This is optimal because as speed increases larger errors should be expected. The search begins by locating the timing parameters needed to minimize (4). Once the optimal timing is found, the final estimate of the magnitudes are determined to ensure that the criterion presented in (3) is met.

Determining the optimal pulse timing is difficult because there is no clear mechanism to guide the search. Fortunately, robot controllers are typically discrete, which limits the search to a finite set of values. While the solution space is finite, attempting each combination would be extremely time consuming. Through experimentation it has been found that the value of (4) varies smoothly with the timing parameters. Based on this, the number of timing parameter combinations evaluated is reduced by using a coarse to fine search. In this type of search an initial coarse sampling of s and d space is used to identify a small set of regions likely to contain the

global minima of (4). These areas then become the focus of the search, allowing the majority of the solution space to be ignored.

To be able to determine the quality of a particular set of timing parameters the pulse magnitudes must be found so the motion can be performed. The magnitudes are found by repeating a motion several times and adjusting them based on the joint position errors. The desired joint positions, $\vec{q}_i$, at the end of pulse interval i are determined by using the inverse kinematics of the robot and the desired robot pose, $\vec{P}_i$. The difference between the desired joint displacement, $\vec{q}_i$, and the measured joint displacements at the end of the interval, $\vec{q}_{est}(t_i)$, is the joint error, $\vec{e}_i = \vec{q}_i - \vec{q}_{est}(t_i)$. The estimate of the optimal pulse magnitude for a joint j is updated using the equation

$$m'_{ij} = m_{ij} + k_j * e_{ij}, \tag{5}$$

where k_j is a user selected gain. In the authors' experience the system converges with 10 practice motions for $\lambda_{via} \approx 0.001m$. This is significantly reduced when a similar motion's pulse magnitudes are used as a starting point for the search.

The time required to find the optimal pulse timing can be further reduced by replacing λ_{via} in (3) with λ_{search}, where $\lambda_{search} \cong 10\lambda_{via}$. Increasing the tolerance for errors at the via points requires a modification to the evaluation of the motion's shape so only the errors related to pulse timing are reflected in σ_{dev}. This is accomplished by combining the realized poses, $P(\vec{q}_{est}(t_{i-1}))$ and $P(\vec{q}_{est}(t_i))$, with the objective function to generate $\hat{P}_d(t)$. $\hat{P}_d(t)$ is then used in (4) to get a reasonable estimate of σ_{dev} with the effects of the via point errors removed. This allows the pulse magnitudes to be found in about one fifth the time. Experiments have demonstrated that relaxing the first criteria has minimal effect on the search results. In the rare case where the solution was altered the difference was minor, *e.g.* a timing parameter differed by one sample period with little effect on the final σ_{dev}. Once the optimal pulse timing has been found the original value of λ_{via} is restored so that the final pulse magnitudes can be found which satisfy (3).

V. Experiments and Results

There are four parts to the investigation. In the first part a simulated robot, modeled after the CRS 465, is moved to 300 points uniformly distributed over a volume of space to demonstrate the variety of motions that can be produced with the presented command. In the second part of the investigation it is shown that the timing of the pulses is important for shaping the trajectory of the robot. The third part of the investigation verifies the results of the simulation by repeating the previous experiment with the CRS 465. The investigation concluded with a summary of the results presented in [5], demonstrating that sensory input can be mapped *directly* to torque commands to accomplish a ball intercept task. A video of the results of the final experiment is supplied. During this investigation the pulse interval used was $\delta = 0.13$ *sec*. For a motion to be successful, the end effector had to be within $\lambda_{via} = 0.001m$ of the via points. The algorithm presented in Section IV was used to find the optimal $\mathscr{C}$. The short duration and simplicity of the motions dictated that only one pulse was necessary—thus $|\mathscr{C}| = 1$.

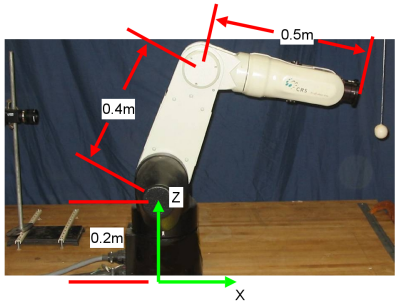


Fig. 4. Robot modeled for simulation and later used in the investigation.

A. Experimental Setup

The experiments were conducted on the open architecture 6 dof (degree of freedom) articulated robot shown in Fig. 4. In these experiments the wrist (last 3 dof) was neglected. The robot controller was a desktop computer running WinCon real time kernel and outfitted with a Quanser MultiQ card that allowed full control of the robot's amplifiers and access to its joint encoders. The control law, shown in Fig. 3, was implemented on the computer in Simulink. The computer could reliably execute the control law at 100 Hz. It should be noted that dedicated controllers can easily exceed 500 Hz. Despite the reduced bandwidth of the controller it was able to show the expressiveness of the pulses and reliably intercept a ball.

The controller's sample period of 0.01 *sec* and the use of $\delta = 0.13$ *sec* dictated that s and d were limited to 13 possible values. To maximize the amount of motion that the robot could produce, the largest possible pulse duration that did not limit the system's ability to shape the robot's trajectory was used. The desired pulse duration was identified by conducting an investigation with the following values for d: 0.08, 0.09, 0.10, 0.11, and 0.12 *sec*. It was found that there was little difference in the variety of motion that could be produced using a duration of 0.08, 0.09, or 0.10 *sec*, but values larger than 0.10 *sec* had a significant impact on trajectory control. For this reason the pulse duration was fixed at 0.10 *sec*. Using the stated pulse duration allowed the end effector to be moved up to $0.2m$ in $\delta = 0.13$ *sec* time.

B. Simulation

The dynamic model of the robot used in the simulation is described by the differential equation

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \mathbf{M} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} + \mathbf{C}(\theta_1, \theta_2, \theta_3, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3) \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} + \mathbf{F}(\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3) + \mathbf{G}(\theta_1, \theta_2, \theta_3), \tag{6}$$

where θ_j, $\dot{\theta}_j$, and $\ddot{\theta}_j$ are joint j's position, velocity, and acceleration respectively. The physical dimensions of the robot used in the motion can be found in Fig. 4. The joint friction, $\mathbf{F}(\cdot)$, is based on the model presented in [20]. The term $\mathbf{G}(\cdot)$ denotes the forces exerted on the robot due to gravity. The shorter link is assumed to have mass of $2.3kg$

while the assumed mass of the other is $2.5kg$. Based on the link masses and the assumption of slender cylindrical links, the computation of inertia, $\mathbf{M}(\cdot)$, and coriolis, $\mathbf{C}(\cdot)$, matrices is relatively straightforward. A detailed description of manipulator dynamics along with the terms $\mathbf{M}(\cdot)$ and $\mathbf{C}(\cdot)$ can be found in [21]. The differential equations were solved using the Runge–Kutta method.

The objective of the first part of the investigation was to show that torque commands could be used to generate a wide range of motions. For each motion the initial position of the end effector was $(0.45m,\ 0.0m,\ 0.40m)$ with the elbow up. It was moved, using a linear motion, from the initial position to three hundred points that were selected at random from a cubic *volume* defined by the corners $(0.47m,\ -0.06m,\ 0.46m)$ and $(0.53m,\ 0.06m,\ 0.56m)$ with a uniform distribution. The motions produced had an average error area, the area between the desired and actual trajectory, of $16.0*10^{-5}m^2$. Error area is used here because it is a more widely understood metric of performance than (4). There were seven failed attempts to find a torque command to move the robot within λ_{via} of the endpoint. The failures occurred when the arm needed to be nearly fully extended, a singular configuration, to reach the endpoint.

The objective of the second part of the investigation was to show that the linear motions were not just a product of the robot configuration. To accomplish this the motion produced by the torque commands was compared to a motion produced using a traditional PD controller. For this experiment the end effector was moved from $(0.57m, 0.03m, 0.51m)$ to $(0.61m, -0.03m, 0.41m)$. The plot in Fig. 5(a) shows the end effector trajectory produced using only PD control as a blue dashed line with circles, the motion generated by the torque command is shown as a red line with triangles, and the ideal path is shown with a black line. Looking at the plot, it becomes evident that the torque command follows the desired path more closely. The error area between the PD trajectory and the ideal motion is $25.0*10^{-5}m^2$, approximately five times the error area of torque pulse motion, $5.3*10^{-5}m^2$, demonstrating that pulse timing is a powerful tool for controlling the trajectory of the robot.

C. Verification of Simulation

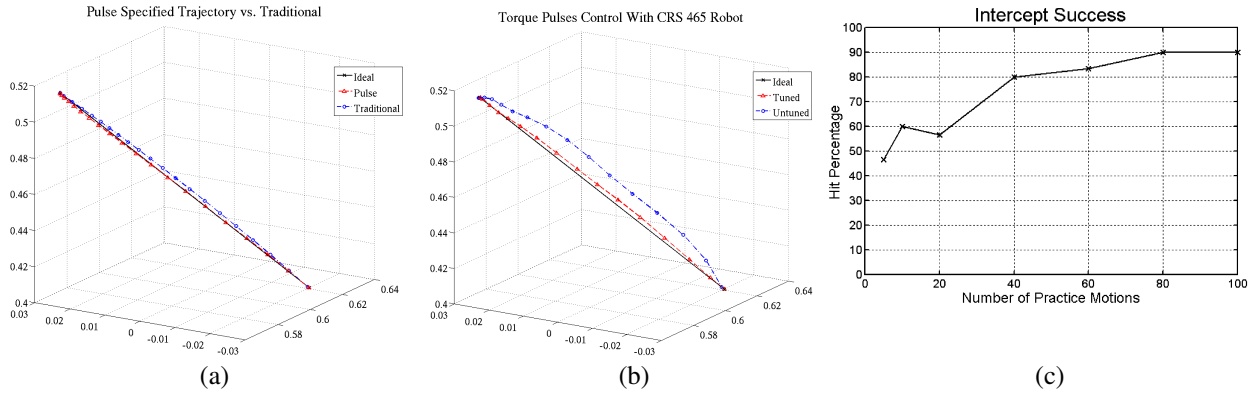
In the third part of the investigation the results from the simulation were verified on the CRS 465 robot using the same motion. Again, a baseline was needed to compare the results. In this case, one motion was produced using a randomly selected set of timing parameters and another with the optimal values. The results can be seen in Fig. 5(b). The motion produced by the optimal torque command is shown as a red line with triangles. The error area of the optimal motion was $74.4*10^{-5}m^2$. This was higher than that of the simulated motion, but differences were expected. The simulation uses a relatively simple model with assumptions such as cylindrical links. While there are moderate differences, the overall results of the simulation match that the CRS 465 robot.

The final portion of the investigation was to verify that the pulses facilitated the process of learning to map unprocessed observations, $\mathscr{F}$, to a set of torque commands, $\mathscr{C}$, to complete a simple task, intercepting a ball. The experimental setup shown for this is shown in Fig. 4. It consists of a CRS 465 robot at the origin of the depicted frame and a camera mounted at $(-0.432m,\ 0.3175m,\ 0.432m)$. The camera captures *two* images of the ball $0.03sec$ apart. Given the possible motion of the robot in $0.13sec$ only a ball that passes through a $0.178m$ by $0.127m$ window approximately $0.610m$ from the robot can be intercepted. To ensure the ball passes through this window it was mounted on a string and suspended from the ceiling.

To determine the efficiency of learning to mapping $\mathscr{F}$ to the desired $\mathscr{C}$, the system was tested after 5, 10, 20, 40, 60, 80, and 100 practice motions. Each test consisted of 30 attempts to intercept the ball. The release point of the ball and the length of the string was changed randomly before each drop. From the images two features are collected, the centroid of the ball in the first image and the change in the centroid's position in the second image corresponding to $\mathscr{F} = [u_o\ v_o\ \Delta u\ \Delta v]^T$. This input vector, expressed in pixels, was used as the input to a set of feedforward multilayer perceptions, consisting of 36 hidden neurons total, to determine the $\mathscr{C}$ needed to intercept the ball—a detailed description of the structure and training of the neural network can be found in [5]. The optimal $\mathscr{C}$ intercepts the ball with the front of the end effector $0.13sec$ after the second image at a location in the $0.178m$ by $0.127m$ window using a linear, open-loop motion.

The results shown in Fig. 5(c) demonstrate that the format of the commands is conducive to efficient learning. The majority of learning was complete after 40 practice motions. The system achieved a 90% success rate after just 80 practice motions, but the 10% failure rate could not be eliminated. The reason for the failures was likely variations in network latency. As with any robotic system, the robot commands were created on a secondary computer and transmitted to the computer responsible for executing the control law. The communication between the PCs was done using TCP/IP because it provided the minimum median latency. Unfortunately, at times there was visible lag between the command and the motion. The observed latency may be due to the interrupt routine employed inside the operating system. The communication software did not run inside the real time kernel due to constraints of the WinCon system, so there was no guarantee that the interrupt would be handled in a timely fashion. Serial communication was also tried, and while latency variation was reduced, the median latency was several orders of magnitude greater than that of the TCP/IP commands.

A video of the robot performing the interception task is supplied. The video shows several releases of the ball. At each release the robot collects the needed features $\mathscr{F}$ and maps them *directly* to a command $\mathscr{C}$ using a trained neural network to intercept the ball. During these trails the velocity of the ball is controlled by varying the release point. It is important to note that, while the string length was unchanged in this video, during training and testing the length of the



(a) (b) (c)

Fig. 5. The numbers on the axes of (a) and (b) denote position with respect to the world frame in meters. (a) A plot showing the path of the simulated robot's end effector as it moves between two points using a linear motion. The black line is the ideal motion, while the red is the motion produced with the pulses. (b) is a verification of the results shown in (a). Again the ideal motion is shown in black and the motion produced using the optimal pulses is shown in red. (c) The effect of practice on the ability of the robot to intercept a ball.

string was varied with each trial.

VI. Discussion and Future Work

The objective of this work was to explore the use of torque pulses to specify robotic motion. The simulation showed that as long as the destination point of the robot is not near a singularity the command can be used successfully. In addition, the results showed that the timing parameters can be used to shape motions even with a robot controller having a sample period five times greater than that of contemporary controllers. An increased sample rate may marginally improve the pulse's performance, but any substantial improvement is unlikely given the small error area with the current system. Additionally, the results showed that with less than 100 practice motions it is possible to learn to map sensory input directly to the required pulses to complete a simple task. This result is important as the goal of this work is the creation of robotic reactive motions.

Continued work is needed in several areas. First, an intelligent method of segmenting motions is needed. While anecdotal evidence suggests that a fixed pulse interval will provide reasonable performance, an intelligent motion segmentation algorithm will allow (4) to be further minimized. Additionally, the large jerk associated with square torque pulses are not desirable. While the motor amplifiers act as low pass filters, further reduction of the motion jerk is desirable. This can be accomplished by altering the shape of the pulses or by using a low pass filter with the current square pulses. Lastly, work needs to be conducted to determine if such a system can be used to generate reactive motions with a prosthetic limb controlled using neural output. Mapping the neural output to such commands has the potential to greatly improve the lives of prosthetics users by allowing them to participate in activities such as baseball.

References

[1] A. Schwartz, "Cortical neural prosthetics." *Annu Rev Neurosci*, vol. 27, pp. 487–507, 2004.

[2] M. Velliste, S. Perel, C. Spalding, A. Whitford, and A. Schwartz, "Cortical control of a prosthetis are for self feeding," *Nature*, vol. 483, pp. 1098–1101, 2008.

[3] H. Kim, S. Park, and M. Srinivasan, "Developments in brain-machine interfaces from the perspective of robotics," *Human Movement Science*, vol. 28, no. 2, pp. 191–203, 2009.

[4] A. Libin and J. Cohen-Mansfield, "Therapeutic robocat for nursing home residents with dementia: Preliminary inquiry," *Am J Alzheimers Dis Other Demen*, vol. 19, no. 2, pp. 111–116, 2004.

[5] J. Neubert and N. Ferrier, "Direct mapping of visual input to motor torques," in *IEEE Intr. Conf. on Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 634–638.

[6] H. Hu and P. Woo, "Fuzzy supervisory sliding-mode and neural-network control for robotic manipulators," *IEEE Trans. on Ind. Electronics*, vol. 53, no. 3, pp. 929–940, June 2006.

[7] N. Hung, H. Tuan, T. Narikiyo, and P. Apkarian, "Adaptive control for nonlinearly parameterized uncertainties in robot manipulators," *IEEE Trans. on Control Systems Tech.*, vol. 16, no. 3, May 2008.

[8] n. V. Santiba and R. Kelly, "PD control with feedforward compensation for robot manipulators: Analysis and experimentation," *Robotica*, vol. 19, no. 1, pp. 11–19, 2001.

[9] A. Gasparetto and V. Zanotto, "A technique for time-jerk optimal planning of robot trajectories," *Robotics and Computer-integrated Manufacturing*, vol. 24, pp. 415–426, 2008.

[10] Y. Wada and M. Kawato, "A theory for cursive handwriting based on the minimization principle," *Bio Cybrn*, vol. 73, no. 1, pp. 3–13, 1995.

[11] C. Atkeson and J. McIntyre, "Robot trajectory learning through practice," in *IEEE Intr. Conf. on Robot. and Autom.*, vol. 3, April 1986, pp. 1737–1742.

[12] F. Nori, L. Natale, G. Sandini, and G. Metta, "Autonomous learning of 3D reaching in a humanoid robot," in *IEEE Intr. Conf. on Robot. and Autom.*, vol. 2. IEEE, Oct 2007, pp. 1142–1147.

[13] D. Moran and A. Schwartz, "Motor cortical representation of speed and direction during reaching." *J Neurophysiol*, vol. 82, no. 5, pp. 2676–2692, November 1999.

[14] A. d'Avella, A. Portone, L. Fernandez, and F. Lacquaniti, "Control of fast-reaching movements by muscle synergy combinations," *J. Neurosci.*, vol. 26, no. 30, pp. 7791–7810, 2006.

[15] J. Krakauer, G. Maria-Felice, and C. Ghez, "Independent learning of internal models for kinematic and dynamic control of reaching," *Nature Neuroscience*, vol. 2, no. 11, pp. 1026–1031, 1999.

[16] H. Nakahara, K. Doya, and O. Hikosaka, "Parallel cortico-basal ganglia mechanisms for acquisition and execution of visuo-motor sequences : A computational approach," *J. of Cognitive Neuroscience*, vol. 13, no. 5, pp. 626–647, 2001.

[17] A. Gasparetto and V. Zanotto, "A new method for smooth trajectory planning of robot manipulators," *Mechanism and Machine Theory*, vol. 42, no. 4, pp. 455 – 471, 2007.

[18] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Trans. Robotics and Automation*, vol. 8, no. 3, pp. 313–326, 1992.

[19] A. Fagg, A. Barto, and J. Houk, "Learning to reach via corrective movements," in *Yale Workshop on Adaptive and Learning Systems*, June 1998, pp. 179–185.

[20] P. Dupont, "Friction modeling in dynamic robot simulation," in *IEEE Intr. Conf. on Robot. and Autom.*, vol. 2, 1990, pp. 1370–1376.

[21] R. Schilling, *Fundamentals of Robotics: Analysis and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1990, pp. 208–212.