Recursive Algorithm for Motion Primitive Estimation

Aaron R. Enes and Wayne J. Book

***Abstract*— The need for knowing future manipulator motion arises in several robotics applications, including notification or avoidance of imminent collisions and real-time optimization of velocity commands. This paper presents a real-time, low overhead algorithm for identification of future manipulator motions, based on measurements of prior motions and the instantaneous sensed actuator velocity commanded by an operator. Experimental results with a human-controlled, two degree-of-freedom manipulator demonstrate the ability to quickly learn and accurately estimate future manipulator motions.**

I. Introduction and Background

This paper presents a low overhead algorithm for identification of future manipulator motions, based on the instantaneous sensed actuator velocity commanded by the operator. The method is suited for real-time operation. Knowledge of the anticipated manipulator path can provide several powerful benefits to robotics and manually controlled manipulators, and is especially attractive if such capabilities come with little sensing or processing overhead. For example, motion forecasts derived from the current user commands may be used to provide early notification of important or unexpected events, such as imminent collisions. Conventional collision detection schemes require pre-programmed knowledge of the manipulator path [1] which may be unavailable or not known ahead of time. Standard collision detection algorithms can make immediate use of the estimates of future motion described in this paper. Another use for estimates of future actuator motions is to enable online optimization of the velocity input. Such a capability could be readily applied to previous works that study manipulator optimizations but assume prior knowledge of the manipulator path [2,3].

The problem of deducing future motions from sensed operator inputs may be considered a sub-problem of general pattern recognition. There are several classical approaches to solve these problems, including neural networks, hidden Markov models, and linear and nonlinear variations within the vein of principal component analysis.

Principal component analysis (PCA) is a classical technique that is simple to implement and guaranteed to represent the true structure of data near a linear subspace of the high-dimensional input space [4]. PCA is used in many domains to transform a set of correlated variables from some large-dimension space (say, actuator velocity commands and position) into a smaller set of uncorrelated principal components, such as expected actuator displacement before changing direction. Jenkins [5] used a set of measured (human) joint angles to classify observed movements and predict future motions.

Neural networks (NNs) have been used for many applications including pattern recognition and next-in-sequence prediction [6] but are inherently poor at extrapolation which may be necessary for motions that change over time [7], unless a complicated adaption rule is introduced. The basic formulations of NN and PCA task identification approaches work best when the motions have been previously seen and the corresponding classifiers established offline.

In contrast, this paper presents a simple and effective technique to map current actuator position and operator inputs to expected manipulator paths. The method requires no prior knowledge of the task or operator style, and can adapt to variations in task parameters over time.

II. The Manipulator Task

Let $q(t) = [q_1(t), \cdots, q_n(t)]^T$ be generalized actuator coordinates (in actuator or joint space) of a serial manipulator. Here, the actuator trajectory is decomposed into a sequence of piecewise monotonic segments termed *motion primitives*. Many manipulator paths are described independent of time, and involve a sequence of shorter motions during which the direction each actuator moves is constant. Ignoring the possibility of constraints or keep-out regions, the motion between the endpoints of these motion primitives is inconsequential, so only the relative actuator displacement is considered.

A sequence of piecewise monotonic motion primitives describe the actuator motion through the workspace. The motion primitives have a *direction*, Ω, and a *length*, x. Ω is a discrete variable that indicates the direction each actuator moves (positive, negative, or static); so for an n DOF manipulator there are n^3 motion categories. The value of x provides the remaining details of the motion by specifying the actuator displacement relative to the absolute actuator position q at the beginning of the motion primitive. Thus, the operator's task is specified by the category of motion (Ω) and a parameterization giving the "amount" of motion in the direction Ω. This approach is similar to other script-based methods of describing manipulator motions [8]. Unlike other methods, the learning-based approach described here does not require the motion parameters x to be specified ahead of time; rather, they are learned online.

This work was supported by the National Science Foundation Center for Compact and Efficient Fluid Power, contract EEC-0540834 and performed at the Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, Georgia 30318.

Aaron Enes, Ph.D. is a member of Technical Staff, MIT Lincoln Laboratory, and Professor Wayne Book, Ph.D. holds the HUSCO/Ramirez Distinguished Chair of Fluid Power and Motion Control at Georgia Tech (e-mails: aaron.enes@ll.mit.edu, wayne.book@me.gatech.edu)

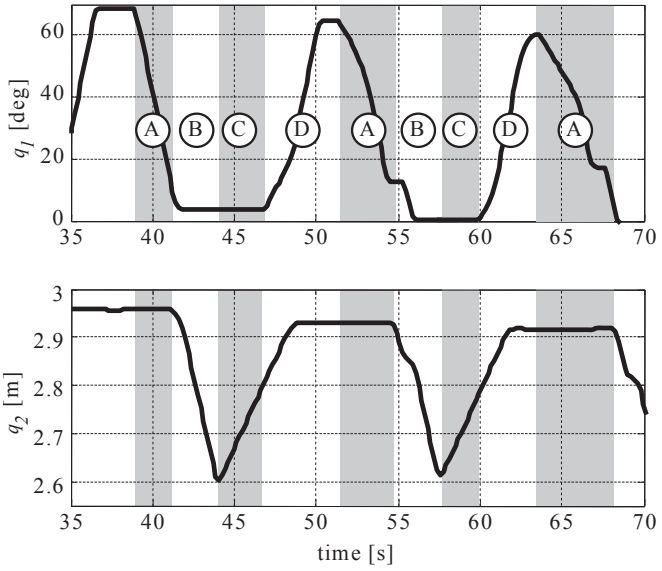


Fig. 1: Actuator displacement for coded by motion primitives.

Consider the 2 DOF trajectory in Fig. 1, which is decomposed into an approximate repeating sequence of motion primitives with categories

$$\Omega : \left\{ \Omega^{(A)}, \Omega^{(B)}, \Omega^{(C)}, \Omega^{(D)} \right\}$$

and corresponding actuator displacements

$$x : \left\{ \begin{bmatrix} 60 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0.32 \end{bmatrix}, \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}, \begin{bmatrix} 60 \\ 1.2 \end{bmatrix} \right\}$$

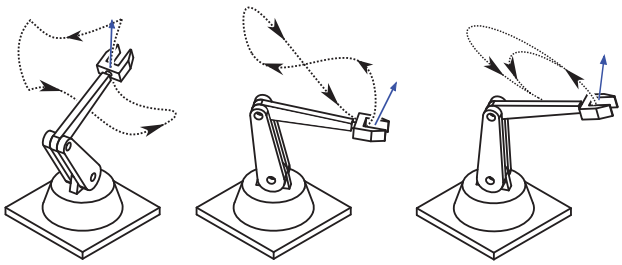
where the meaning of symbols $\Omega^{(A)}$, $\Omega^{(B)}$, $\Omega^{(C)}$, $\Omega^{(D)}$ are "*Retract* q_1", "*Retract* q_2", "*Extend* q_2", and "*Extend* q_1 *and* q_2". Note that only the relative actuator displacement—and not the temporal dependence—is specified by the sequence of motion primitives. The alternating shared regions in Fig. 1 denote transitions to new motion categories Ω, which correspond to the circled values.

III. Limitations on Trajectory Types

The *motion primitive* formulation discussed here implicitly assumes the manipulator path is uniquely parameterized by the generalized actuator coordinates q and the motion category Ω (i.e., the *direction* each actuator is moving). Suitable paths include non-intersecting paths, such as Fig. 2a, or any path that self-intersects at an oblique angle, as in Fig. 2b. Paths that intersect and are tangent, i.e., the paths' velocities are parallel at the intersection point, are *not* parametrized by q and Ω alone. Fig. 2c shows an invalid point along a path at which it is impossible to distinguish the "loop" to be traced by the manipulator.

IV. Algorithm for Motion Primitive Estimation

For many conventional manipulators, one or more joysticks are displaced by the operator; the angular displacement in a given direction maps to the commanded velocity of the corresponding actuator. Ω is directly determined by



(a) No intersections (b) Oblique intersection (c) Tangent intersection

Fig. 2: Relative displacements along paths (a) and (b) are uniquely determined by q and Ω, whereas (c) is not.

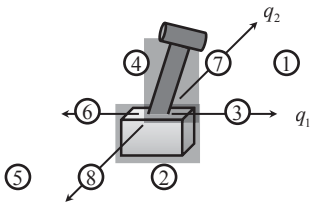


Fig. 3: A mapping between joystick angular displacement, direction of actuator motion, and motion category Ω.

sensing this command. Fig. 3 illustrates how certain joystick displacements map to actuator motions and Ω (the value of Ω is circled). For example, moving the joystick "up and right" commands the displacement of both actuators (q_1 and q_2) to increase; this category of motion is assigned the symbol $\Omega = 1$. Similarly, joystick motion "directly left" commands one actuator (q_1) to retract, classified as motion within category $\Omega = 7$.

While the *direction* of motion is easily determined, the length x of the primitive remains an unknown. This section describes the Recursive Algorithm for Motion Primitive Estimation—named RAMPE—for estimating the displacement of the motion primitives.

> ***Problem Statement:*** At the beginning of the kth detected occurrence of a motion primitive in category Ω, find the expected duration, $x[k] \in R^n$, given the observation $y[k], y[k-1], \cdots, y[1]$ with $y[k] \in R^{n+m}$.

Argument k indicates there have been k previous occurrences of the motion category Ω. For convenience in notating the following sections, the symbols x, Θ, P, etc. are used for all categories of motion Ω even though each Ω has its own set of variables.

The displacement $x[k]$ is assumed to be well approximated by some unknown function

$$x[k] = f(\Omega, y[k]) \tag{1}$$

where Ω is the primitive category and $y[k] = [q[k], \xi[k]] \in R^{m+n}$ is an observation vector taken at the start of the motion primitive. f is assumed to be independent of k, so the path is unchanging with time. In reality, the path *does* change (e.g. in an excavation task, the depth of a trench will

gradually increase thereby requiring longer 'reaches' by the manipulator); the recursions in RAMPE can be augmented with a forgetting factor to account for the dynamics of slowly varying paths. The observation y always consists of the generalized coordinates $q \in R^n$ at the start of Ω, and optional additional measurements $\xi \in R^m$ which may include measures such as the sum of commanded velocities or time.

The trajectory function 1 is approximated as a linear function of the observation $y[k]$ and unknown parameters Θ, so

$$x[k] = H(y[k])\Theta$$

where H is a block diagonal matrix defined as

$$H(y[k]) = \begin{bmatrix} h_1(y[k]) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & h_n(y[k]) \end{bmatrix}$$

The n regression elements $h_j(y[k])$ are row vectors to estimate the jth component of $x[k]$. Each h_j has a unique structure for each component of $x[k]$. Linear supports may be adequate to separate the motion primitives for some applications, while more complex trajectories may require a set higher-order basis functions.

Θ is formed by concatenation of vectors θ_j:

$$\Theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

where each θ_j is a vector of unknown parameters θ_{ij} for estimating the jth component of x, as $x_j = h_j\theta_j$.

Based on all the *previous* observations, the estimated displacement for the current kth occurrence of Ω is then

$$\hat{x}[k] = H(y[k])\hat{\Theta}[k-1] \tag{2}$$

where $\hat{x}$ is the estimated displacement. The model parameters in $\hat{\Theta}[k-1]$ are recursively updated so (2) optimally (in the least squares sense) describes the *observed* relative actuator displacement of the previous $k-1$ occurrences. With this goal, $\Theta[k-1]$ is updated recursively, using the update law [9]

$$\hat{\Theta}[k-1] = \hat{\Theta}[k-2] + K[k-1]\Big(x[k-1] - H(y[k-1])\hat{\Theta}[k-2]\Big)$$

with

$$K[k-1] = (P[k-1])^{-1}H(y[k-1])^T G \tag{3}$$

The matrix $P[k-1]$ is also updated recursively as

$$P[k-1] = P[k-2] + H(y[k-1])^T G H(y[k-1]) \tag{4}$$

G is the inverse of the noise covariance matrix R

$$G^{-1} = R(\Omega) = \begin{bmatrix} \sigma_1 & \cdots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \cdots & \sigma_n \end{bmatrix} \tag{5}$$

where the constants σ_{ij} in $R(\Omega)$ represents the observed standard deviation of the actuators for a given motion category Ω. [1].

$P[k-1]$ may be singular, or nearly singular, if the regressors in $H(y[k-1])$ are linearly dependent. Thus, the inverse $(P[k-1])^{-1}$ required in (3) may be ill-conditioned or not unique.

As a remedy, $(P[k-1])^{-1}$ is computed using the singular value decomposition (SVD) of $P[k-1]$, giving

$$P[k-1] = U\Sigma V^T$$

where U and V are sets of orthonormal basis vectors and Σ is the diagonal matrix of singular values. To guarantee convergence in a fixed number of computation steps, the diagonal pseudoinverse of Σ is calculated by transposing the matrix obtained after inverting each element along the diagonal of Σ. If the matrix $P[k-1]$ is nearly singular, some of the elements in Σ are near zero, so only those diagonal elements larger than a given tolerance *tol* $<< 1$ are inverted; the other elements are set to zero. Finally, the pseudoinverse of $P[k-1]$ is computed as

$$P[k-1]^{-1} = V\Sigma^{-1}U^T$$

A. Algorithm Initialization

An estimate $\hat{x}$ is computed only after N_{init} prior observations. To prime the RLS algorithm, an initial set of parameters $\hat{\Theta}$ is computed based on the first N_{init} observations. An initial regressor matrix H_0 is formed by placing the submatrices $h_{j,\text{init}}$ along its main diagonal, where

$$h_{j,\text{init}} = \left[h_j(y(1))^T, h_j(y(2))^T, \cdots, h_j(y(N_{\text{init}}))^T\right]^T \tag{6}$$

Similarly, the vector x_{init} is formed by "unwrapping" the previous measurements to produce

$$x_{\text{init}} = [x_{1,1}, \cdots, x_{1,N_{\text{init}}}, x_{2,1}, \cdots, x_{2,N_{\text{init}}}, \cdots]^T$$

where $x_{j,k}$ is component j of the kth observation $x[k]$. The initial estimate of the parameter $\hat{\Theta}[N_{\text{init}}-1]$ is obtained by solving the system of equations in (2).

SVD is used to solve the system by direct computation using

$$\hat{\Theta}[N_{\text{init}}-1] = Vr \tag{7}$$

where r is the solution to the diagonal system

$$\Sigma r = U^T G_0 H_0 x_{\text{init}}$$

and U, Σ, V compromise the singular value decomposition of matrix $[G_0 H_0]$. G_0 is the inverse of the initial covariance matrix R_0, where R_0 is a zero-padded version of (5) and H_0 is the initial set of regressor matrices (6). As before, inversion of r is carried out by setting to zero the components of r corresponding to the diagonal elements of Σ below a specified threshold. The matrix $P[N_{\text{init}}-1]$ for use in (4) is

$$P[N_{\text{init}}-1] = H_0^T G_0 H_0$$

[1]Another key role of R scaling the elements of x, because each element may have units of very different magnitudes (i.e., *degrees* versus *meters*)

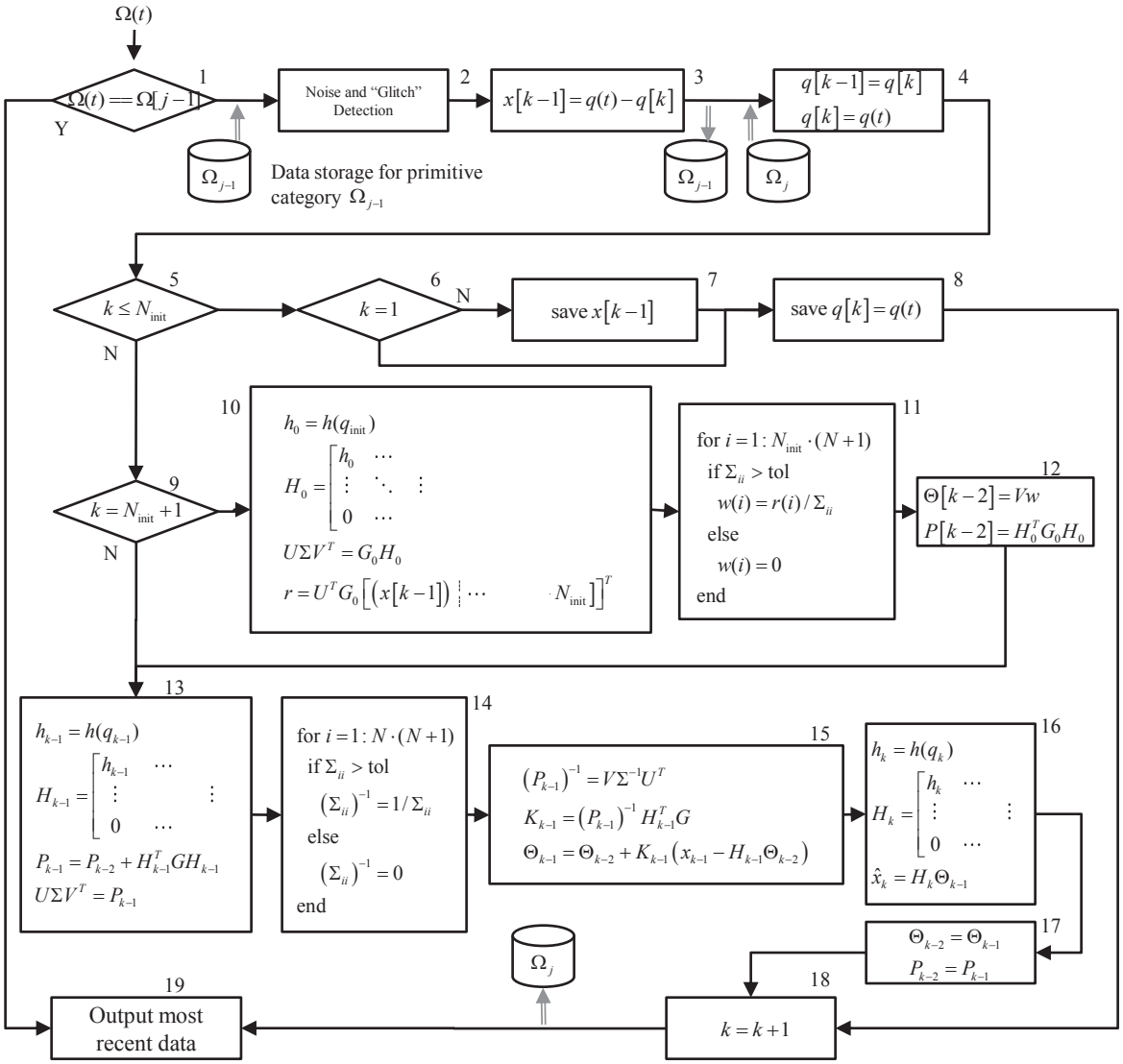


Fig. 4: Elements of the RAMPE algorithm.

To compute the initial estimate, first compute $\hat{\Theta}[N_{\text{init}} - 1]$ using (7), then compute $\hat{x}[k]$ using (2).

Fig. 4 shows a flow chart for calculating the estimate $\hat{x}[k]$. The double lines indicate transfers into or out of memory for all data ($x[k], q[k], \Theta$, etc.) associated with a particular Ω. A description of each numbered block is provided in Table I.

B. Engineering Tests with the RAMPE *Algorithm*

An experiment was performed to demonstrate the RAMPE algorithm. The operator used a gaming joystick to move the end effector of a 2 DOF simulated hydraulic manipulator displayed a standard LCD monitor. The operator's goal was to repeatedly maneuver the end effector through a sequence of targets shown in Fig. 5:

$$A \to B \to C \to D \to C \to B \to A \to \cdots$$

The control inputs were the swing rate of the arm ($\dot{q}_1$) and the velocity of the boom hydraulic cylinder ($\dot{q}_2$). Table II shows the location of each target in actuator space q, and the minimum displacement x between targets.

The generalized coordinates q for the swing and boom during a portion of the cycles are shown in Fig. 6. A total of 23 cycles were completed.

With two active actuators, there are 3^2 categories of motion. The regression model chosen for this example assumes all components of x are linearly dependent on q—plus a possible constant offset—so

$$h_j(q[k]) = \left[q[k]^T, 1\right] \tag{8}$$

Each vector of unknown parameters, θ_j, has 3 components in this case.

The resulting trajectory is plotted in the $q_1 - q_2$ plane shown in Fig. 7. Points along the trajectory are coded with a symbol shape corresponding to the motion primitive category Ω executed at that instant; this coding follows the convention in Fig. 3. Fig. 7 illustrates the typical properties of manually-controlled motion: the human operator is not

TABLE I: Description of the flow chart blocks in Fig. 4

	Description of flow chart elements in Fig. 4
1.	Check if the primitive category has changed.
2.	Measured velocity commands are filtered to avoid mis-classifying unintentional motion (noise) as actual manipulator motion.
3.	The length of previous primitive $\Omega(j-1)$ is calculated as $x[k-1] = q(t) - q[k]$, where $q(t)$ is the value of the current generalized coordinate, and $q[k]$ is the (recorded) starting value of primitive $\Omega(j-1)$. This updated information for $\Omega(j-1)$ is saved.
4.	Update the starting point of the previous iteration ($q[k-1] = q[k]$) and the current iteration ($q[k] = q(t)$).
5.	Check if this primitive category is still in the initialization phase.
6.	Check if this is the first time primitive $\Omega(j)$ is encountered.
7.	Append the initialization data for the primitive length x.
8.	Append the starting point q with the previous cycles.
9.	Check if this is the first estimate of $\hat{x}$
10.	Prepare variables to use Least Squares via SVD to compute the initial estimate.
11.	Compute the projection vector w for the Least Squares estimate by computing the (reduced) pseudo-inverse the singular values matrix.
12.	Compute the least squares estimate of the parameter vector Θ and the matrix P to be used in computing the initial estimate of $\hat{x}[k]$.
13.	Compute the necessary terms to update the recursion relationship for RLS estimation. This includes calculating the SVD of $P[k-1]$.
14.	To invert $P[k-1]$, the reduced pseudo-inverse of the matrix of singular values Σ is used.
15.	Update the variables for the RLS estimation of $x[k]$.
16.	Compute the estimate of the primitive length, $\hat{x}[k]$ using the present generalized coordinates $q[k]$ as a basis, and the most up-to-date parameters $\hat{\Theta}[k-1]$.
17.	Store the variables for use during the next iteration.
18.	Update the number of times primitive category Ω_j has been encountered. Push all data for Ω_j back to memory.
19.	Output the most recent estimates, including $\hat{x}[k]$.

TABLE II: Location of weighpoints to guide manipulator cycle

	Generalized coordinate, q		Displacement x to next target	
Target	q_1 [deg]	q_2 [m]	x_1 [deg]	x_2 [m]
A	64.4	2.79	-41.7	0.16
B	22.7	2.95	-22.7	-0.16
C	0	2.79	0	-0.19
D	0	2.60	0	0.19

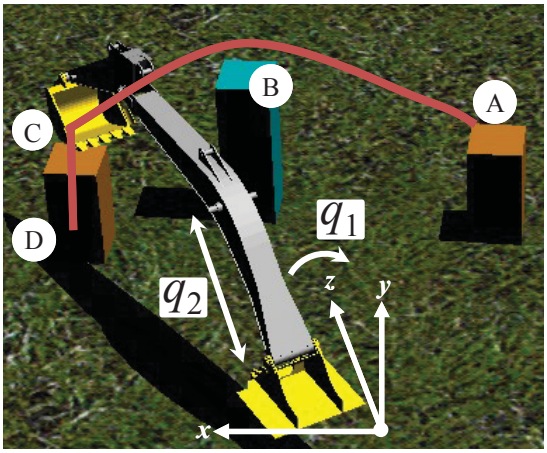


Fig. 5: Operator display with path weighpoints overlaid.

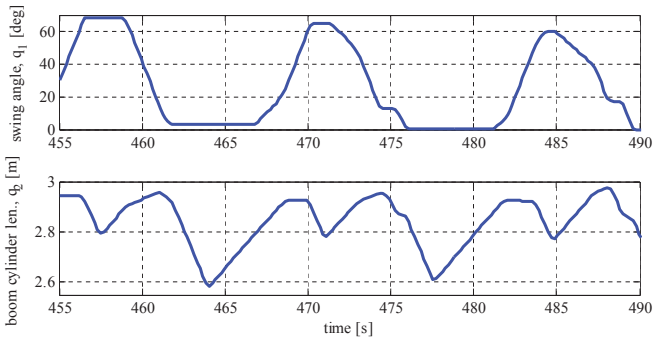


Fig. 6: Actuator coordinates during test cycle.

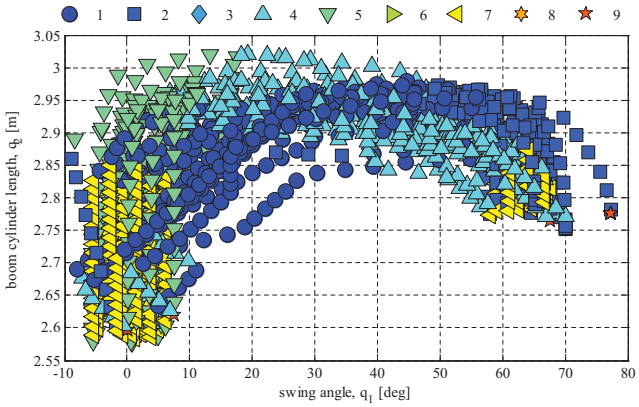


Fig. 7: Resulting manipulator motion plotted in the q-plane. The marker shape denotes the instantaneous motion primitive category, Ω. Symbol shape denotes motion category, Ω

precise in commanding the motion, as evidenced by the general "spread" of the plot. Further, some motion primitives only occur in one region of the q-plane, while others occur at multiple regions. These distinct clusterings of motions are successfully seperated due to the state (q) dependent basis functions $h(q)$.

Fig. 8 shows the start points ($q[k]$), estimated displacements ($\hat{x}[k]$), and actual displacements ($x[k]$). The results for the five most encountered primitives are shown. Estimates during the three initialization periods ($k < N_{\text{init}} = 3$) were set to $\hat{x}[k] = 0$. For $\Omega = 1$, $\Omega = 4$, and $\Omega = 7$ there are two distinct clusters from which the motion primitive begins. This illustrates that the linear supports 8 were adequate to separate the two clusters of displacements that occur for the same category of motion. In general, the estimation error is small relative to the total actuator displacement.

Fig. 9 shows the estimation error $E = x[k] - \hat{x}[k]$ plotted versus occurrence count k. Large errors are present during the initialization phase and the error generally decreases by increasing the number of prior occurrences. The error never settles to zero because of the inherently unpredictable nature of a human-controlled task.

V. Conclusions

The motion primitive formulation describes manipulator paths as a sequence of monotonic actuator displacements

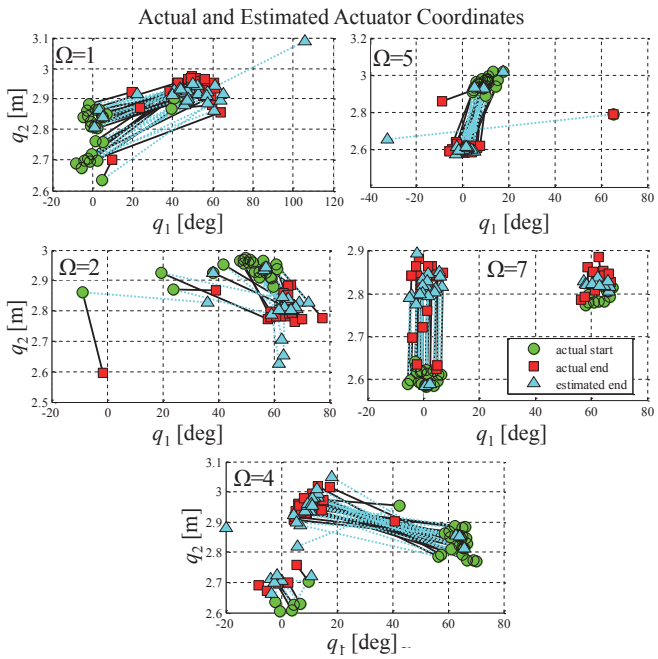


Fig. 8: Actuator and estimated displacements of the five most common motion categories. Green circles represent position $q[k]$ at the beginning of the motion primitive; cyan triangles mark the estimated end point $\hat{x} + q[k]$; red squares mark the actual end point. The solid black and dashed cyan lines link the start and end points for a particular cycle iteration k.

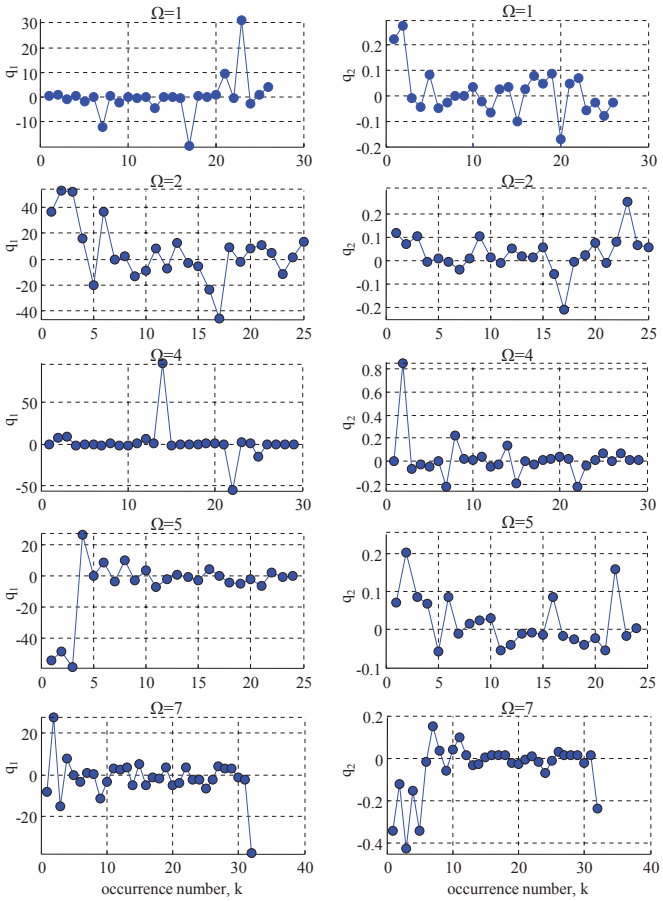


Fig. 9: Residual estimation error $E = \hat{x}[k] - x[k]$. The abscissa marks the number of prior occurrences of the particular Ω. Units for q_1 and q_2 are degrees and meters.

and is well suited for describing motions within an obstacle free environment. Further, the motion is conveniently decoupled into two elements: the motion primitive category, Ω, and the relative actuator displacement, x. The category Ω is determined by sensing and filtering the operator input commands, while the expected displacement, $\hat{x}$, is estimated using previous observations in a recursive algorithm called RAMPE. Only the point-to-point displacement of the actuator across each primitive is saved, thus simplifying the description of actuator motions and the corresponding computation load associated with processing complex trajectories. This simplicity comes at the expense of lower-fidelity path descriptions. Knowledge of future actuator motions—especially when such knowledge can be incorporated into a system with little overhead—has several useful applications to robotics including early collision notification as well as optimization of velocity commands necessary to achieve the desired goal.

The application example demonstrated good estimation performance for a human-controlled point-to-point motion task.

REFERENCES

[1] P. Xavier, "Fast swept-volume distance for robust collision detection," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Albuquerque, NM, 1997.

[2] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[3] A. Enes and W. Book, "Optimizing point to point motion of net velocity constrained manipulators," in *Proc. IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, 2010.

[4] K. V. Mardia, *Multivariate analysis*. London: Academic Press, 1979.

[5] O. C. Jenkins, M. J. Mataric, and S. Weber, "Primitive-based movement classification for humanoid imitation," in *Proc. IEEE International Conference on Humanoid Robots (Humanoids)*, Cambridge, MA, 2000.

[6] C. Bishop, *Neural networks for pattern recognition*. New York: Oxford University Press, 1995.

[7] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the theory of neural computation*. Redwood City, CA: Addison-Wesley, 1991.

[8] P. Rowe and A. Stentz, "Parameterized scripts for motion planning," in *Proc. of International Conference on Intelligent Robots and Systems (IROS)*, Grenoble, France, 1997.

[9] A. H. Jazwinski, *Stochastic processes and filtering theory*. New York: Academic Press, 1970.