Parallel Processing Architecture for Computing Inverse Differential Kinematic Equations of the PUMA Arm

T.C. Hsia
University of California, Davis
Davis, CA 95616

G.Z. Lu and W.H. Han
Nankai University
Tiangjin, China

Abstract

In advanced robot control problems, on-line computation of inverse Jacobian solution is frequently required. Parallel processing architecture is an effective way to reduce computation time. In this paper, a parallel processing architecture is developed for the inverse Jacobian (inverse differential kinematic equation) of PUMA arm. The proposed pipeline/parallel algorithm can be implemented on IC chip using systolic linear arrays. This implementation requires 27 processing cells and 25 time units. Computation time is thus significantly reduced.

1. Introduction

In many advanced robot control problems, such as with sensor guided manipulations, it is essential that the end effector be appropriately controlled in Cartesian coordinates so that the robot can adapt to a changing environment. This means that we need to compute the inverse Jacobian in real time to provide the required differential change in joint variables for a desired differential change in position and orientation. The speed of this computation directly affects the speed of robot operation. Thus efficient algorithms for computing the inverse Jacobian are needed.

There have been efforts made recently in developing computationally efficient algorithms to solve the Jacobian problem suitable for serial computer implementation [1,2]. In addition, some work has been reported in algorithm development for implementation on pipelined or parallel computer [3]. These results show that such parallel algorithms can reduce computation time significantly.

A more important requirement in robot manipulation is the computing of the inverse Jacobian solution. This is generally a troublesome problem when we try to invert the Jacobian numerically. A more direct approach is to derive an explicit solution of the inverse Jacobian for a given robot. Paul, Shimano, and Mayer [2] have shown that such solutions can be obtained by differentiating the kinematic equations. This approach has shown to result simpler inverse Jacobian solutions with regard to manipulator degeneracies and joint constraints. The inverse Jacobian of the PUMA arm has been solved specifically in [2].

In this paper, we present a pipeline/parallel algorithm and architecture for computing the PUMA arm inverse Jacobian derived in [2]. With rapid advances in VLSI technology, this type of algorithm can be readily implemented on IC chips. These special purpose chips can be connected to a host computer system to achieve real-time Cartesian space control at sufficiently high sample rate. It is noted that a study has been made recently to implement direct kinematic solution on VLSI chips to speed up computation time [4]. The goal here is to further exploit the advantages of VLSI technology for the design of customized chips dedicated to the computing of the inverse Jacobian of PUMA arm.

3. Differential Kinematic Solution of PUMA Arm

Differential changes in joint variables dq can be related to the different changes in translation and rotation dx, dy, dz, dθx, dθy, and dθz of the end effector by the relationship

\[ [dx, dy, dz, dθx, dθy, dθz]^T = J [dq_1, dq_2, ..., dq_n]^T \]  \hspace{1cm} (1)

In which n is the number of joints, and J is the Jacobian matrix. But in advanced robot control problems, we need the solution of dq given the desired differential change dx, dy, dz, dθx, dθy, dθz. That is, we need to compute the inverse problem

\[ [dq_1, dq_2, ..., dq_n]^T = J^{-1} [dx, dy, dz, dθx, dθy, dθz]^T \]  \hspace{1cm} (2)

This represents the inverse differential kinematic solution (inverse Jacobian) of the robot arm.

Instead of relying on the direct computing of the inverse Jacobian matrix J^{-1}, an analytical solution of the inverse Jacobian problem can be frequently obtained, and such a solution for the PUMA arm is given in [2]. For the PUMA arm, the joint variables are the six rotational joint angles q_1, q_2, ..., q_6. Furthermore, the
differential changes in translation and rotation can be related to the differential change of the end effector homogenous matrix $T$ [2]:

$$\Delta T = T - I = \Delta T,$$

where

$$T = \begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Delta T = \begin{bmatrix} \Delta n_x & \Delta d_x & \Delta a_x & \Delta p_x \\ \Delta d_y & \Delta d_y & \Delta a_y & \Delta p_y \\ \Delta d_z & \Delta d_z & \Delta a_z & \Delta p_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore differential changes in translation and rotation can also be specified in terms of the $x, y, z$ elements of $\Delta p, \Delta d_y, \Delta d_z$ (the $d_n$ vector is redundant). The desired solutions of $\Delta T$ in terms of $\Delta p, \Delta d_y, \Delta d_z$ for the PUMA arm obtained in [2] are given in the appendix. A pipeline/parallel processing architecture for computing these equations is now developed below.

4. Systolic Array Processing

VLSI technology has created a new architecture horizon in implementing parallel algorithms directly on hardware. Central to this architecture is the use of systolic linear arrays which consist of interconnected simple and mostly identical processing cells. Algorithms that can be executed using identical operations simultaneously can take advantage of the systolic array architecture to reduce computation time.

The processing cell structure we will employ is the "inner product step processor" which performs matrix-vector multiplication using one-way pipeline algorithms. For example, computing

$$A b = p$$

where $A$ is $m \times n$ and $b$ is $n \times 1$, can be carried out in the following recurrence manner:

$$p_i^{(0)} = 0$$

$$p_i^{(k+1)} = p_i^{(k)} + a_{ik} b_k$$

$$k = 1, m, i = 1, n$$

$$p_i = p_i^{(m)}$$

This operation can be implemented by a linear array of $m$ inner product step processors shown in Figure 1.

In the following section, we will reformulate the inverse differential kinematic equation given in the appendix in terms of a set of matrix-vector multiplications which can be computed in parallel and pipelining fashion.

5. Algorithm Development

In this section, we present the matrix-vector multiplication processing schemes for computing the differentials $\Delta d_i$, $i = 1, 2, ..., 6$. Here we assume that the trigonometric functions required are available. Typically these functions can be generated by employing ROM look-up techniques [5,6]. The algorithm is broken down into 15 steps as described below. The notation $S_i, S_i, C_i, C_i, C_i$ are used.

(1) $A_1 b_1 = p_1$

$$A_1^T = \begin{bmatrix} dp_y & dp_x & px & py & ay & ax & dx & -ax & dy & oy & ox & dx & doy \\ -dp_x & dp_y & px & py & ay & ax & dx & -ax & dy & oy & ox & doy & -dox \end{bmatrix}$$
\[ b_1^T = [c_1 \, s_1] \quad p_1^T = [p_{11} \ldots p_{19} \, p_{110} \ldots p_{113}] \]

output: \( d_0 = p_{11} / p_{13} \)

(2) \[ f_1 \quad m_1 + g_1 = h_1 \]
\[ f_1^T = [p_{14} \, p_{15} \, p_{18} \, p_{110} \, p_{111}] \quad m_1 = d_0 \]
\[ g_1^T = [p_{12} \, p_{17} \, p_{14} \, p_{112} \, p_{113}] \]
\[ h_1^T = [h_{11} \ldots h_{15}] \]

(3) \[ A_2 b_2 = p_2 \]
\[ A_2 = \begin{bmatrix} -a_3 \, d_4 \, p_z \, -p_{13} \\ d_4 \, a_3 \, p_{13} \, p_z \end{bmatrix} \]
\[ b_2 = \begin{bmatrix} s_3 \\ c_3 \end{bmatrix} \]
\[ p_2^T = [p_{21} \ldots p_{24}] \]

(4) \[ f_2 \quad m_2 + g_2 = h_2 \]
\[ f_2^T = [c_3 \, s_3 \, p_{21} \, p_{23} \, p_{24}] \]
\[ g_2^T = [a_3 \, d_4 \, 0 \, 0 \, 0] \]
\[ m_2 = e_2, \quad h_2^T = [h_{21} \ldots h_{25}] \]

(5) \[ A_3 b_3 = p_3 \]
\[ A_3 = [-p_z \, p_{13}] \quad b_3^T = [dp_z \, h_{11}] \quad p_3 = p_{31} \]

output: \( d_0 = p_{31} / h_{23} \)

(6) \[ A_4 b_4 = p_4 \]
\[ A_4 = \begin{bmatrix} p_z \, -p_{13} \\ p_{13} \, p_z \end{bmatrix} \]
\[ b_4 = \begin{bmatrix} h_{21} \\ h_{22} \end{bmatrix} \]
\[ p_4 = \begin{bmatrix} p_{41} \\ p_{42} \end{bmatrix} \]

(7) \[ A_5 b_5 = p_5 \]
\[ A_5 = \begin{bmatrix} -p_{21} \, h_{22} \, -p_{24} \\ h_{22} \, h_{21} \, h_{25} \end{bmatrix} \]
\[ b_5^T = [dp_z \, h_{11} \, d_0_3] \]
\[ p_5^T = [p_{51} \, p_{52}] \]

(8) \[ A_6 b_6 = p_6 \]
\[ A_6 = \begin{bmatrix} p_{42} \, -a_z \, p_{16} \, -p_{51} \, h_{12} \, d_{a_2} \, p_{111} \, -d_{a_2} \, p_{114} \, -d_{a_2} \\ -p_{41} \, -p_{16} \, -a_z \, -p_{52} \, -d_{a_2} \, h_{12} \, -d_{a_2} \, h_{14} \, -d_{a_2} \, h_{14} \end{bmatrix} \]
\[ b_6^T = [c_{23} \, s_{23}] \]
\[ p_6^T = [p_{61} \ldots p_{610}] \]

outputs: \( d_0_{23} = p_{64} / p_{61} \quad d_0 = d_0_{23} - d_0_3 \)

319
The data flow timing table for these computations are given in Tables 1 and 2. It is shown that the solution requires 25 time units and 27 processing cells.

The results of 25 time units is a significant reduction of computation time in comparison with that when a serial computer is used to compute the original solution. The total number of multiplications of that solution is about 150. This is equivalent to 150 time units in the systolic array processing system as opposed to the 25 time units we have achieved by exploiting parallelism.
Table 1. Data flow timing table for steps 1 through 8 which compute $dQ_1$, $dQ_2$, and $dQ_3$. Numbers on top row indicate time units.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d\Phi_y$</td>
<td>- $d\Phi_x$</td>
<td>$d\Phi_x$</td>
<td>$d\Phi_y$</td>
<td>$p_x$</td>
<td>$p_y$</td>
<td>$-p_x$</td>
<td>$e_y$</td>
<td>$-e_x$</td>
<td>$e_x$</td>
<td>$e_y$</td>
<td>$d_x$</td>
<td>$d_y$</td>
<td>$d_x$</td>
<td>$-e_x$</td>
<td>$-e_y$</td>
<td>$d_y$</td>
<td>$d_x$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$-p_1$</td>
<td>$p_2$</td>
<td>$h_2$</td>
<td>$-h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$-h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
<td>$h_2$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$s_3$</td>
<td>$p_{21}$</td>
<td>$p_{23}$</td>
<td>$p_{24}$</td>
<td>$-p_2$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
<td>$p_{13}$</td>
</tr>
<tr>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
</tr>
<tr>
<td>$p_{14}$</td>
<td>$p_{15}$</td>
<td>$p_{18}$</td>
<td>$-p_{110}$</td>
<td>$p_{111}$</td>
<td>$-p_{111}$</td>
<td>$h_{11}$</td>
<td>$d_{11}$</td>
<td>$h_{21}$</td>
<td>$d_{21}$</td>
<td>$h_{31}$</td>
<td>$d_{31}$</td>
<td>$h_{41}$</td>
<td>$d_{41}$</td>
<td>$h_{51}$</td>
<td>$d_{51}$</td>
<td>$h_{61}$</td>
<td>$d_{61}$</td>
</tr>
<tr>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
<td>$d_4$</td>
</tr>
</tbody>
</table>

321
Table 2. Data flow timing table for steps 9 through 15 which compute $d_{04}$, $d_{05}$, $d_{06}$.

<table>
<thead>
<tr>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>p15</td>
<td>p63</td>
<td>h31</td>
<td>-h13</td>
<td>p15</td>
<td>-p63</td>
<td>h31</td>
<td>h13</td>
<td>p67</td>
<td>p110</td>
<td>-p67</td>
<td>h33</td>
<td>h15</td>
<td>-p67</td>
<td>-p110</td>
<td>h15</td>
</tr>
<tr>
<td>p91</td>
<td>p96</td>
<td>p86</td>
<td>p81</td>
<td>p86</td>
<td>p81</td>
<td>p86</td>
<td>p81</td>
<td>h41</td>
<td>-h32</td>
<td>-p6G</td>
<td>p83</td>
<td>-h42</td>
<td>-h34</td>
<td>p92</td>
<td>d04</td>
</tr>
</tbody>
</table>

6. Conclusion

It has been demonstrated in this paper that parallel computing architecture can be developed for the inverse differential kinematic equation of the PUMA arm. By using systolic linear arrays employed in VLSI chip design, the computation can be completed with 27 processing cells in 25 time units.

The differential kinematic equation in its original form requires about 150 multiplications to compute. If one multiplication is counted as one time unit, the parallel architecture definitely provides a substantial reduction in computation time. A customized IC chip dedicated to this algorithm can be fabricated.

7. References


Appendix: Inverse Differential Kinematic Equations for PUMA arm

\[ d_0 = \frac{C_1 d_p_y - S_1 d_p_x}{C_1 x + S_1 y} \]

\[ d_1 = \frac{d_1}{d_1 (d_4 + 1) - d_3} \]

\[ d_1 = f_{11}(p) d_{11}(p) - f_{12}(p) d_{12}(p) \]

\[ f_{11}(p) = C_1 d_p_x = S_1 d_p_y \]

\[ d_{11}(p) = -S_1 d_p_x d_{e1} + C_1 d_p_x + C_1 d_y d_{e1} + S_1 d_p_y \]

\[ f_{12}(p) = -d_p_z \]

\[ d_{23} = \frac{-S_{23} d_{23} d_{u1} - C_{23} d_{u1}}{C_{23} d_{u2} - S_{23} d_{u1}} \]

\[ v_1 = -u_2 f_{11}(p) + u_1 p_z \]

\[ v_2 = u_1 f_{11}(p) + u_2 p_z \]

\[ u_1 = a_2 c_3 + a_3 \]

\[ u_2 = d_4 + a_2 s_3 \]

\[ d_{u1} = -u_2 d_{11}(p) - a_2 f_{11}(p) c_3 d_{e3} + u_1 d_p_z - a_2 s_3 p_z d_{e3} \]

\[ d_{u2} = u_1 d_{11}(p) - a_2 s_{11}(p) d_{e3} + u_2 d_p_z + a_2 c_3 p_z d_{e3} \]

\[ d_{e2} = d_{e23} - d_{e3} \]

\[ d_{e4} = \frac{N C_{4} d(N S_{4}) - N S_{4} d(N C_{4})}{(N S_{4})^2 + (N C_{4})^2} \]

\[ N C_{4} = C_{23} d_{41} - S_{23} d_{2} \]

\[ D_{41} = C_{1} d_x + S_{1} d_y \]

\[ d(N S_{4}) = -C_{1} d_y d_{e1} - S_{1} d_y d_{e1} + C_{1} d_x z + S_{1} d_x \]

\[ N S_{4} = -S_{1} d_x + C_{1} d_y \]

\[ d(N C_{4}) = -S_{23} D_{41} d_{e23} + C_{23} D_{41} = C_{23} d_{2} d_{e23} - S_{23} d_{e2} \]

\[ d D_{41} = -S_{1} d_x d_{e1} + C_{1} d_x z + C_{1} d_y d_{e1} + S_{1} d_z \]

\[ d d_{e5} = C_{5} s d_{5} - S_{5} d C_{5} \]

\[ d s_{5} = -S_{4} N C_{4} d_{e4} + C_{4} d(N C_{4}) + C_{4} d e_{4} N S_{4} + S_{4} d(N S_{4}) \]

\[ d C_{5} = C_{23} d e_{23} D_{41} + S_{23} d D_{41} = -S_{23} d_{2} d_{e23} + C_{23} d_{e2} \]

\[ S_{5} = C_{4} N C_{4} + S_{4} N S_{4} \]

\[ C_{5} = S_{23} D_{41} + C_{23} d_{e2} \]
\[ d_{6} = C_{6}o_{5}S_{6} - S_{6}o_{5}C_{6} \]

\[ S_{6} = -C_{6}N_{61} - S_{5}N_{612} \]

\[ C_{6} = -S_{4}N_{611} + C_{4}N_{6112} \]

\[ dS_{6} = S_{5}N_{61}d_{e_{5}} - C_{5}d_{N_{61}} - C_{5}N_{612}d_{e_{5}} - S_{5}d_{N_{612}} \]

\[ dC_{6} = -dS_{4}N_{611} - S_{4}d_{N_{611}} + dC_{4}N_{6112} + C_{4}d_{N_{6112}} \]

\[ = -C_{4}N_{611}d_{e_{4}} - S_{4}d_{N_{611}} - S_{4}N_{6112}d_{e_{4}} + C_{4}d_{N_{6112}} \]

\[ N_{61} = C_{4}N_{611} + S_{4}N_{6112} \]

\[ N_{611} = C_{23}N_{61111} - S_{23}O_{23} \]

\[ N_{6112} = -S_{1}O_{x} + C_{1}O_{y} \]

\[ dN_{61} = -S_{4}N_{611}d_{e_{4}} + C_{4}d_{N_{611}} + C_{4}N_{6112}d_{e_{4}} + S_{4}d_{N_{6112}} \]

\[ dN_{611} = -S_{23}N_{61111}d_{e_{23}} + C_{23}d_{N_{61111}} = C_{23}O_{23}d_{e_{23}} - S_{23}O_{23} \]

\[ dN_{6112} = -C_{1}O_{x}d_{e_{1}} - S_{1}O_{x}d_{e_{1}} + C_{1}O_{y}d_{e_{1}} + C_{1}d_{O_{y}} \]

\[ N_{6111} = C_{1}O_{x} + S_{1}O_{y} \]

\[ dN_{6111} = -S_{1}O_{x}d_{e_{1}} + C_{1}d_{O_{x}} + C_{1}O_{y}d_{e_{1}} + S_{1}d_{O_{y}} \]

\[ N_{612} = -S_{23}N_{61111} - C_{23}O_{23} \]

\[ dN_{612} = -C_{23}N_{61111}d_{e_{23}} - S_{23}d_{N_{11111}} + S_{23}O_{23}d_{e_{23}} - C_{23}O_{23} \]

---

Figure 1. Inner product step processor