

APPENDIX A

LIE BRACKET PROPERTIES

This appendix contains proofs to verify that a Lie bracket is a vector field and partially constructs the geometric interpretation of a Lie bracket presented in Section 3.2.

A.1 Derivation of the Lie Bracket

A Lie bracket is a vector field if it is linear over the reals and satisfies the derivation property.

A.1.1 Linearity over the Reals

Given $X_p : C_p^\infty \rightarrow \mathbb{R}$, $f, g \in C_p^\infty$ and $\alpha, \beta \in \mathbb{R}$ the Lie bracket is linear if

$$[X, Y]_p(\alpha f + \beta g) = \alpha[X, Y]_p(f) + \beta[X, Y]_p(g).$$

$$\begin{aligned} [X, Y]_p(\alpha f + \beta g) &= X_p(Y(\alpha f + \beta g)) - Y_p(X(\alpha f + \beta g)) \\ &= X_p(\alpha Y(f) + \beta Y(g)) - Y_p(\alpha X(f) + \beta X(g)) \\ &= \alpha X_p(Y(f)) - \alpha Y_p(X(f)) + \beta X_p(Y(g)) - \beta Y_p(X(g)) \\ &= \alpha [X_p(Y(f)) - Y_p(X(f))] + \beta [X_p(Y(g)) - Y_p(X(g))] \\ &= \alpha[X, Y]_p(f) + \beta[X, Y]_p(g). \quad \blacksquare \end{aligned}$$

A.1.2 Derivation Property

$$\begin{aligned}
[X, Y]_p(fg) &= X_p(Y(fg)) - Y_p(X(fg)) \\
&= X_p(Y(f)g + fY(g)) - Y_p(X(f)g + fX(g)) \\
&= X_p(Y(f)g) + X_p(fY(g)) - Y_p(X(f)g) - Y_p(fX(g)) \\
&= X_p(Y(f))g_p + Y_p(f)X_p(g) + X_p(f)Y_p(g) + X_p(Y(g))f_p \\
&\quad - Y_p(X(f))g_p - X_p(f)Y_p(g) - Y_p(f)X_p(g) - Y_p(X(g))f_p \\
&= [X_p(Y(f)) - Y_p(X(f))]g_p + [X_p(Y(g)) - Y_p(X(g))]f_p \\
&\quad + X_p(f)Y_p(g) - X_p(f)Y_p(g) + X_p(g)Y_p(f) - X_p(g)Y_p(f) \\
&= [X, Y]_p fg_p + [X, Y]_p gf_p. \quad \blacksquare
\end{aligned}$$

A.1.3 Geometric Interpretation

Based on the nomenclature shown in Figure 3.1 it is desired to estimate a solution to $\dot{q} = ag_1(q) + bg_2(q)$. The solution is estimated at small time ϵ , *i.e.*, the state of the system is given at $t = \epsilon$ using a Taylor series expansion about time 0 with $a = 1$, $b = 0$ and $q(0) = q_o$. The initial expansion is

$$\begin{aligned}
q(\epsilon) &= q(0) + \dot{q}(0)(\epsilon - 0) + \frac{1}{2}\ddot{q}(0)(\epsilon - 0)^2 + \mathcal{O}(\epsilon^3) \\
&= q_o + \epsilon g_1(q_o) + \frac{1}{2}\epsilon^2 \ddot{q}(0) + \mathcal{O}(\epsilon^3).
\end{aligned}$$

Since

$$\ddot{q}(0) = \frac{d}{dt}\dot{q}(0) = \frac{d}{dt}g_1(q_o) = \frac{\partial g_1(q_o)}{\partial q} \frac{dq(0)}{dt} = \frac{\partial g_1(q_o)}{\partial q} g_1(q_o),$$

the state at time ϵ is approximately

$$q_\epsilon := q(\epsilon) = q_o + \epsilon g_1(q_o) + \frac{1}{2}\epsilon^2 \frac{\partial g_1(q_o)}{\partial q} g_1(q_o) + \mathcal{O}(\epsilon^3).$$

Next, the solution is expanded about $t = \epsilon$ for the solution at 2ϵ . This expansion is

$$\begin{aligned} q(2\epsilon) &= q(\epsilon) + \dot{q}(\epsilon)(2\epsilon - \epsilon) + \frac{1}{2}\ddot{q}(\epsilon)(2\epsilon - \epsilon)^2 + \mathcal{O}(\epsilon^3) \\ &= q_\epsilon + \epsilon g_2(q_\epsilon) + \frac{1}{2}\epsilon^2 \frac{\partial}{\partial q} g_2(q_\epsilon) g_2(q_\epsilon) + \mathcal{O}(\epsilon^3). \end{aligned}$$

Substituting for q_ϵ gives

$$\begin{aligned} q(2\epsilon) &= q_o + \epsilon g_1(q_o) + \frac{1}{2}\epsilon^2 \frac{\partial g_1(q_o)}{\partial q} g_1(q_o) + \epsilon g_2(q_o + \epsilon g_1(q_o)) \\ &\quad + \frac{1}{2}\epsilon^2 \frac{\partial}{\partial q} g_2(q_o) g_2(q_o) + \mathcal{O}(\epsilon^3). \end{aligned}$$

The last two terms reduce since the approximation is second order, and any substitutions beyond ϵ^2 in the ϵg_2 term and beyond ϵ in the last term yield a third-order term in ϵ . Hence, up to second order, $g_2(q_\epsilon) \approx q_o + \epsilon g_1(q_o)$ and $g_2(q_\epsilon) \approx g_2(q_o)$ with the remaining terms going to $\mathcal{O}(\epsilon^3)$.

At this point it is necessary to additionally expand $g_2(q_\epsilon)$ about q_o giving

$$\begin{aligned} g_2(q_\epsilon) &\approx g_2(q_o) + \frac{\partial}{\partial q} g_2(q_o)(q_\epsilon - q_o) \\ &\approx g_2(q_o) + \frac{\partial}{\partial q} g_2(q_o)(q_o + \epsilon g_1(q_o) - q_o) \\ &\approx g_2(q_o) + \epsilon \frac{\partial}{\partial q} g_2(q_o) g_1(q_o). \end{aligned}$$

The state at time 2ϵ is estimated as

$$\begin{aligned} q_{2\epsilon} := q(\epsilon) &= q_o + \epsilon (g_1(q_o) + g_2(q_o)) + \frac{1}{2}\epsilon^2 \left(\frac{\partial}{\partial q} g_1(q_o) g_1(q_o) \right. \\ &\quad \left. + \frac{\partial}{\partial q} g_2(q_o) g_2(q_o) + 2 \frac{\partial}{\partial q} g_2(q_o) g_1(q_o) \right) + \mathcal{O}(\epsilon^3). \end{aligned}$$

It remains to expand this solution for an approximation at 3ϵ and then that solution to obtain the final estimate at 4ϵ .

A.2 Lie Bracket Properties

The two most important properties of Lie brackets in regards to this work are *skew symmetry* and the *Jacobi identity* because vector fields that are dependent through skew symmetry or the Jacobi identity must be eliminated from the Phillip Hall basis. The Phillip Hall basis construction presented in Section 3.2 does this “automatically”. Skew symmetry is easy to show using local coordinates; however, the Jacobi identity is more amenable to a global proof. Only a proof of skew symmetry is given. The proof is constructive.

skew symmetry: $[f, g] = -[g, f]$

From the definition of a Lie bracket

$$\begin{aligned} [f, g] &= \frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g \\ &= - \left(\frac{\partial f}{\partial x} g - \frac{\partial g}{\partial x} f \right) \\ &= -[g, f] \quad \blacksquare \end{aligned}$$

APPENDIX B

THE UNABRIDGED KINEMATIC CAR

This appendix provides additional information associated with the car parking example presented in Section 4.1.1.

B.1 Annihilating Constraint Equations

For each case, it is necessary to check $\omega \cdot f \stackrel{?}{=} 0$. For the first case,

$$\begin{aligned}
 \omega_1 \cdot f &= [\sin(\theta + \phi) \quad -\cos(\theta + \phi) \quad -l \cos \phi \quad 0] \begin{bmatrix} \cos \theta \\ \sin \theta \\ \tan \phi/l \\ 0 \end{bmatrix} \\
 &= (\sin \theta \cos \phi + \cos \theta \sin \phi) \cos \theta - (\cos \theta \cos \phi - \sin \theta \sin \phi) \sin \theta - \frac{l \cos \phi \sin \phi}{\cos \phi l} \\
 &= \cos^2 \theta \sin \phi + \sin^2 \theta \sin \phi - \sin \phi \\
 &= (\cos^2 \theta + \sin^2 \theta) \sin \phi - \sin \phi \\
 &= 0.
 \end{aligned}$$

For the second case,

$$\begin{aligned}
 \omega_2 \cdot f &= (\sin \theta \quad -\cos \theta \quad 0 \quad 0) \begin{pmatrix} \cos \theta \\ \sin \theta \\ \tan \phi/l \\ 0 \end{pmatrix} \\
 &= \sin \theta \cos \theta - \cos \theta \sin \theta \\
 &= 0.
 \end{aligned}$$

For the third case,

$$\begin{aligned} \omega_1 \cdot g &= [\sin(\theta + \phi) \quad -\cos(\theta + \phi) \quad -l \cos \phi \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ &= 0. \end{aligned}$$

Finally,

$$\begin{aligned} \omega_2 \cdot f &= (\sin \theta \quad -\cos \theta \quad 0 \quad 0) \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ &= 0. \end{aligned}$$

B.2 Rank of the Distribution

Checking the dependency of the vector fields shows

$$\begin{bmatrix} \cos \theta & \sin \theta & \tan \phi / l & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-1}{l \cos^2 \phi} & 0 \\ \frac{-\sin \theta}{l \cos^2 \phi} & \frac{\cos \theta}{l \cos^2 \phi} & 0 & 0 \\ 0 & 0 & \frac{-2 \tan \phi}{\cos^2 \phi} & 0 \end{bmatrix} \xrightarrow{R5: -2 \tan \phi l R3 + R5} \begin{bmatrix} \cos \theta & \sin \theta & \tan \phi / l & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-1}{l \cos^2 \phi} & 0 \\ \frac{-\sin \theta}{l \cos^2 \phi} & \frac{\cos \theta}{l \cos^2 \phi} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, $g_5 = f(g_3)$ and either one can be eliminated.

APPENDIX C

ROBOT CODE

This Appendix contains the C-code used to perform the robot motion and manipulation tasks based on the logic presented in Chapter 5. A description of the eleven programs is given below.

- `main.c` – This is the main program to perform robot functions. The options are move a robot in a circle trajectory, move a robot point-to-point, and perform manipulation. It queries the user for appropriate information based on each selection, and calls the necessary functions to perform the desired task.
- `move_options.c` – This program is called from `main.c`, and contains functions to generate a set of desired trajectories based on the user option for the robot tasks. The configuration list is stored in a file for use in determining the inverse kinematics.
- `move_pt2pt.c` – This program is called from `move_options` to generate the desired trajectory to move a robot from point to point.
- `move_circle.c` – This program is called from `move_options` to generate the desired trajectory to move a robot in a circle.
- `inverse_kinematics.c` – This program reads a file containing desired robot configurations and determines the joint angles required to achieve them. It writes two files, `final_angles.dat` which contains the set of calculated joint angles, and `prcounts.dat` which contains a set of relative encoder counts to achieve the calculated angles. If a manipulation is being performed, it writes four such files, one for each robot. The latter is used by `move_robot` to write position commands to the boards. It also returns a pointer to the last set of calculated joint angles.
- `matrix.c` – This program contains several functions to perform vector and matrix operations mostly needed to determine the inverse kinematics.
- `move_robot.c` – This program reads the datafiles containing encoder counts, parses them for each robot and sends the commands to the appropriate motion control boards.

- `acquire_object.c` – This program is called by `main.c` to acquire an object. It generates an identical, initial point-to-point movement for each robot and then uses information from the forces sensors to determine the next position movement for each robot.
- `talk2matlab.c` – This program stores information in `matlab_info.dat` for use with Matlab[®] to determine the joint trajectories for closed loop, reconfigurable manipulation.
- `slip.c` – This program is called from `talk2matlab.c` to check the slip condition after a finger reconfiguration.
- `fuzzy_ctrl.c` – This program is called from `slip.c` and from `acquire_object.c`. It contains the code to implement the fuzzy controller which outputs position commands based on the finger's current position and on the value of a finger's sensor readings.

In addition, there is code defining a set of functions to access the motion control boards. These can be found in [76]. The complete code is listed below. At the cost of readability, the size of the text has been decreased to reduce the number of pages.

C.1 File `main.c`

```
/* *****
*****
* Written by: Neil Petroff *
* Program: main.c *
* Date Written: 22 OCT 2005 *
* Last Date Modified: 14 JUL 2006 *
* Written for: research *
*****
```

```
This is the main program to perform robot functions. The options are
move a robot in a circle trajectory, move a robot point-to-point, and
perform manipulation. It queries the user for appropriate information
based on each selection, and calls the necessary functions to perform
the desired task.
```

```
***** REVISION LOG *****
```

```
10/22/05: The flow of my multi-file program didn't seem very smooth.
So, this is my first attempt at reorganizing it.
11/07/05: Incorporated code to move all the robots.
11/10/05: Added code to have 3 robots touch object based on sensor
readings. Currently, Robot #2 has a joint problem, so I haven't
been using it.
11/11/05: Added code to remove appended datafiles with each new run.
Added code to remove temporary data files. Added code to store the
last x-position for each robot when it's acquiring the object so, if
there's a sensor glitch, and a robot thinks it's touching the object
when it isn't, and starts moving again, it will continue from it's
last position rather than a "global" last position. Added function
acquire_object which is called after manipulation choice to "find"
the object and lift it prior to the motion planning.
04/15/06: Added task type 3 to move a robot using an existing
datafile containing relative counts. This is done to implement robot
simulations in which the joint angles are determined from the contact
constraint equation. Eventually should be capable of reading a file
for each robot.
04/17/06: Task type 3 currently assumes a datafile exists for each
robot.
07/10/06: Added fuzzy controller for finger position control to
```

```

adjust grasp strength.
07/13/06: Began adding code to perform fixed point rotation.

***** */

/* Include Files */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <time.h>
#include "puma.h"
#include "move_options.h"
#include "inverse_kinematics.h"
#include "move_robot.h"
#include "dmclinux.h"
#include "acquire_object.h"
#include "fuzzy_ctrlr.h"
#include "talk2matlab.h"

/* Main Function */

int main()
{
    int i, type, robot_num, object;
    int board_num;
    int cube_side;
    float xo[] = {0,0,0}, xf[] = {0,0,0};
    float x_ctr, y_ctr, z_ctr;
    float radius, ro = 0;
    double rotation_axis[3] = {0,0,0};
    float rotation_amt = 0;
    long int final_counts[6] = {0,0,0,0,0,0};
    long int jtcts[6] = {0,0,0,0,0,0};
    char file_name[20];

    /* delete appended datafiles for a new run */
    for (i = 1; i <= 4; i++)
    {
        sprintf(file_name,"Tdes%d.dat", i);
        remove(file_name);
        sprintf(file_name,"final_angles%d.dat", i);
        remove(file_name);
        sprintf(file_name,"robot%d_pos.dat", i);
        remove(file_name);
    }

    iopl(3);
    Clear(1);
    Clear(2);
    Clear(3);

    for (board_num = 1; board_num <= 3; board_num++)
    Write("DR -2;", board_num);

    /* Get task */
    printf("\nNeil's Robot Controller\n");
    printf("Compiled on %s at %s\n",__DATE__, __TIME__);
    printf("\nWhat type of movement would you like to perform?\n");
    printf("0: circle, 1: point-to-point, 2: manipulation, \
3: from file ... ");
    scanf("%d",&type);

    if (type == 0 || type == 1 || type == 3)
    {
        printf("\nEnter robot number (1-4) you would like to \
use ... ");
        scanf("%d",&robot_num);
        if (robot_num != 1 && robot_num != 2 && robot_num != 3 \
&& robot_num != 4)
        {
            printf("\nNot a valid selection.\n");
            exit(EXIT_FAILURE);
        }
    }

    if (type == 0)
    {
        printf("\nAt the zero configuration, the finger\n");
        printf("is located at %.2f, %.2f, %.2f",12+15, 13-11, \
10-14);
        printf("\nEnter the location of the circle's center \
(x, y, z) ... ");
        scanf("%f %f %f",&x_ctr, &y_ctr, &z_ctr);
        printf("\nEnter the radius of the circle (in.) ... ");
        scanf("%f", &radius);
        move_circle(x_ctr, y_ctr, z_ctr, radius);

        inverse_kinematics(robot_num, type, jtcts);
    }
}

```

```

    move_robot(robot_num, final_counts);
}
else if (type == 1)
{
    printf("\nAt the zero configuration, the finger\n");
    printf("is located at %.2f, %.2f, %.2f", l2+l5, l3-l1, \
    l0-l4);
    printf("\nEnter the coordinates of the starting point \
(x, y, z) ... ");
    scanf("%f %f %f", &xo[0], &xo[1], &xo[2]);
    printf("\nEnter the coordinates of the end point \
(x, y, z) ... ");
    scanf("%f %f %f", &xf[0], &xf[1], &xf[2]);
    move_pt2pt(xo,xf);

    inverse_kinematics(robot_num, type, jtcts);
    move_robot(robot_num, final_counts);
}
else if (type == 3)
{
    move_robot(robot_num, final_counts);
}
else if (type == 2)
{
    printf("\nPlease select object type.\n");
    printf("0: ball, 1: egg, 2: cube ... ");
    scanf("%d",&object);
    if (object == 1)
    {
        printf("\nNot yet available.\n");
        exit(EXIT_FAILURE);
    }
    else if (object == 0 || object == 2)
    {
        if (object == 0)
        ro = 4.25;
        else
        {
            printf("\nPlease select an orientation\n");
            printf("1: plane, 2: edge ... ");
            scanf("%d",&cube_side);
            if (cube_side == 1)
            ro = 2.875;
            else if (cube_side == 2)
            ro = 4.066;
            else
            {
                printf("\nInvalid Selection.\n");
                exit(EXIT_FAILURE);
            }
        }

        printf("\nPlease select rotation axis.\n");
        printf("(A rotation about the z-axis is \
(0, 0, 1) ... ");
        scanf("%lf %lf %lf",&rotation_axis[0],\
&rotation_axis[1],&rotation_axis[2]);
        printf("\nPlease select rotation amount \n");
        printf("in degrees ... ");
        scanf("%f", &rotation_amt);
    }
    else
    {
        printf("\nInvalid Selection.\n");
        exit(EXIT_FAILURE);
    }
    acquire_object(object, type);
    talk2matlab(ro, rotation_axis, rotation_amt, type, object);
}
else
{
    printf("\nNot a valid selection.\n");
    exit(EXIT_FAILURE);
}

if (type == 2)
remove("Tdes.dat");

if (type == 0 || type == 1)
{
    for (i = 1; i <= 4; i++)
    {
        sprintf(file_name, "robot%d_cts.dat", i);
        remove(file_name);
    }
}

sleep(10);

printf("Returning to zero position ...");

```

```

    Write("PRA=-1000;",3);
    Write("PRC=-1000;",3);
    Write("PRE=-1000;",3);
    Write("PRG=-1000;",3);
    Write("BG;",3);

    printf("...");
    sleep(1);
    Write("PA 0,0,0,0,0,0,0,0;BG;", 1);
    Write("PA 0,0,0,0,0,0,0,0;BG;", 2);
    Write("PA 0,0,0,0,0,0,0,0;BG;", 3);

    printf("...");
    sleep(1);

    for (board_num = 1; board_num <= 3; board_num++)
Write("DR 0;", board_num);
    printf("...");

    printf(" done.\n\n");

    return 0;
}

```

C.2 File move_options.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: move_options.c *
* Date Written: 22 OCT 2005 *
* Last Date Modified: 11 NOV 2005 *
* Written for: research *
*****
*/

```

This sub-program is called from main, and contains functions to generate a set of desired trajectories based on the user option for the robot tasks. The configuration list is stored in a file for use in determining the inverse kinematics.

```

***** REVISION LOG *****
10/22/05: I've combined the 3 movement options into this 1 file.
11/11/05: Added code in manipulate() to write both permanent and
temporary datafiles depending on the motion requested. Tdes1-4.dat
stores the complete set of desired configurations for each robot
during a manipulation task. Tdes.dat is the temporary file read by
inverse_kinematics() to calculate the required joint angles. Tdes.dat
is later deleted by the main program.
*/

```

```

/* Include Files */

#include <stdio.h>
#include <math.h>
#include "move_options.h"

/* move a robot in a straight line between 2 points */

void move_pt2pt(float *xo,float *xf)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i, j, k;

    FILE *ofp;

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])* \
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2])) + 1;

    ofp = fopen("Tdes.dat","w");

    if (num_pts == 0)
    {
        for (j = 0; j < 3; j++)
pos_vector[j] = xo[j];
        for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[k][0],\
rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);
    }
    else
    {
        for (i = 1; i <= num_pts; i++)
    {
        for (j = 0; j < 3; j++)

```

```

{
    pos_vector[j] = xo[j] + (i/((float) \
num_pts))*(xf[j] - xo[j]);
}
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);
}
}
    fclose(ofp);
}

void move_circle(float xctr, float yctr, float zctr, float radius)
{
    float step = 0.1;
    float theta;
    float z = 9.4;
    float pos_vector[4] = {0,0,z,1};
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i;

    FILE *ofp;

    ofp = fopen("Tdes.dat","w");

    for (theta = 0; theta <= 2*M_PI; theta += step)
    {
        pos_vector[0] = radius * cos(theta) + xctr;
        pos_vector[1] = radius * sin(theta) + yctr;

        for (i = 0; i < 4; i++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[i][0],\
rotation_matrix[i][1],rotation_matrix[i][2], pos_vector[i]);
    }
    fclose(ofp);
}

void manipulate(float *xo,float *xf, double **rotation_matrix, int \
robot_num)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    //float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i, j, k;

    FILE *ofp, *ofptemp;

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])*\
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2])) + 1;

    ofptemp = fopen("Tdes.dat","w");

    if (robot_num == 1)
ofp = fopen("Tdes1.dat","a");
    else if (robot_num == 2)
ofp = fopen("Tdes2.dat","a");
    else if (robot_num == 3)
ofp = fopen("Tdes3.dat","a");
    else
ofp = fopen("Tdes4.dat","a");

    if (num_pts == 0)
    {
        for (j = 0; j < 3; j++)
pos_vector[j] = xo[j];
        for (k = 0; k < 4; k++)
        {
            fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
            fprintf(ofptemp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
        }
    }
    else
    {
        for (i = 1; i <= num_pts; i++)
        {
            for (j = 0; j < 3; j++)
pos_vector[j] = xo[j] + (i/((float) num_pts))*\
(xf[j] - xo[j]);

            for (k = 0; k < 3; k++)
            {
                fprintf(ofp,"%lf\t%lf\t%lf\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2],\

```

```

pos_vector[k]);
    fprintf(ofptemp,"%lf\t%lf\t%lf\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
}
    fprintf(ofp,"%lf\t%lf\t%lf\t%f\n",0.0,0.0,0.0,\
pos_vector[k]);
    fprintf(ofptemp,"%lf\t%lf\t%lf\t%f\n",\
0.0,0.0,0.0, pos_vector[k]);
}
}

    fclose(ofp);
    fclose(ofptemp);

}

```

C.3 File move_pt2pt.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: move_pt2pt.c *
* Date Written: 22 JUL 2005 *
* Last Date Modified: 22 JUL 2005 *
* Written for: research *
*****

```

This program generates a set of robot configurations to move the end-effector in a straight line.

```
***** REVISION LOG *****
```

```

**** Variable Definitions **** */

/* Include Files */

#include <stdio.h>
#include <math.h>
#include "move_pt2pt.h"
// #include "inverse_kinematics.h"
// #include "matrix.h"

void move_pt2pt(float *xo,float *xf)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i, j, k;

    FILE *ofp;

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])*
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2]));

    ofp = fopen("Tdes.dat","w");

    if (num_pts == 0)
    {
        for (j = 0; j < 3; j++)
            pos_vector[j] = xo[j];
        for (k = 0; k < 4; k++)
            fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[k][0],\
rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);
    }
    else
    {
        for (i = 1; i <= num_pts; i++)
        {
            for (j = 0; j < 3; j++)
            {
                pos_vector[j] = xo[j] + (i/((float) \
num_pts))*(xf[j] - xo[j]);
            }
            for (k = 0; k < 4; k++)
                fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
        }
    }
    fclose(ofp);
}

```

```

}

void get_object(float *xo, float *xf)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    double final_angles[6] = {0,0,0,0,0,0};
    int i, j, k;

    FILE *ofp;

    //gst = (double **) malloc((unsigned) 4*sizeof(double*));

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])*\
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2]));

    ofp = fopen("Tdes.dat", "w");

    if (num_pts == 0)
    {
        for (j = 0; j < 3; j++)
            pos_vector[j] = xo[j];
        for (k = 0; k < 4; k++)
            fprintf(ofp, "%f\t%f\t%f\t%f\n", rotation_matrix[k][0], \
rotation_matrix[k][1], rotation_matrix[k][2], pos_vector[k]);
    }
    else
    {
        for (i = 1; i <= num_pts; i++)
        {
            for (j = 0; j < 3; j++)
            {
                pos_vector[j] = xo[j] + (i/((float) \
num_pts))*(xf[j] - xo[j]);
            }
            for (k = 0; k < 4; k++)
                fprintf(ofp, "%f\t%f\t%f\t%f\n", \
rotation_matrix[k][0], rotation_matrix[k][1], rotation_matrix[k][2], \
pos_vector[k]);
        }
    }

    fclose(ofp);

    inverse_kinematics(final_angles);

    printf("\n");
    for (i = 0; i < 6; i++)
        printf(".2%lf\t", final_angles[i]);
    printf("\n");
    //gst = forward_kinematics(final_angles, gst);
}

```

C.4 File move_circle.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: circle_traj.c *
* Date Written: 30 MAR 2004 *
* Last Date Modified: 02 FEB 2005 *
* Written for: research *
*****
*/

/* Include Files */

#include <math.h>
#include <stdio.h>
#include "move_circle.h"

void move_circle(float xctr, float yctr, float zctr, float radius)
{
    float theta, beta;
    float z = 9.4;
    //float y_center = -5.5, z_center = 9.4,
    float pos_vector[4] = {0,0,z,1};
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i;

    FILE *ofp;

```

```

    ofp = fopen("Tdes.dat","w");

    //beta = M_PI/2; //atan(17.05/17);

    for (theta = 0; theta <= 2*M_PI; theta += STEP)
    {
        //pos_vector[1] = radius * cos(theta-beta) + y_center;
        //pos_vector[2] = radius * sin(theta-beta) + z_center;

        pos_vector[0] = radius * cos(theta) + xctr;
        pos_vector[1] = radius * sin(theta) + yctr;

        for (i = 0; i < 4; i++)
        fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[i][0],\
rotation_matrix[i][1],rotation_matrix[i][2], pos_vector[i]);
    }
    fclose(ofp);
}

```

C.5 File inverse_kinematics.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: inverse_kinematics.c *
* Date Written: 09 JUN 2004 *
* Last Date Modified: 01 AUG 2006 *
* Written for: research *
*****

```

This program reads a file containing desired robot configurations and determines the joint angles required to achieve them. It writes two files, final_angles.dat which contains the set of calculated joint angles, and prcounts.dat which contains a set of relative encoder counts to achieve the calculated angles. The latter is used by move_robot to write pr commands to the boards. It also returns a pointer to the last set of calculated joint angles.

```

***** REVISION LOG *****
02/02/05: The wrist orientation (theta4 and 6) can't be determined if
theta5 is zero. These were determined by equating matrix elements
instead of using the subproblem method because the original zero
configuration chosen wasn't conducive to subproblems,i.e., axes 4 and
6 can't line up.
02/28/05: Changed the zero configuration so axes 4 and 6 are no longer
in line. Redid calculations for theta4, 5, and 6. Also placed tool
frame at wrist, which eliminates l5 (for now).
03/08/05: Followed MLS's method using subproblems, resulting in new
solutions for theta2 and theta1. Used subproblem 1 to determine theta1
but not how they describe in the book, i.e. exp^-z1(theta1)q=c. Instead,
used exp^z1(theta1)p=q, where p now includes twist about theta2.
03/09/05: Everything looks good except for theta6. Rechecking wrist
solutions. Test at zero configuration gives -0.05, 0.17, 0, 0.05, -0.17,
and 57.3 degrees for theta1-theta6, respectively. Obviously, theta6
isn't right! Forgot to take acos of stuff for theta6. Looks good now.
Didn't work on the actual robot. I thought it was because gst(0) was
off a sign for y. Should be l3-l1 not l1-l3 on the actual robot. But
this really messed things up! Theta3 is 0, but everything else seems
off by 90 degrees.
03/10/05: Fixed error in theta3 equation. Theta3 is the only one that
is correct!
07/22/05: backup latest stand-alone version of inverse_kinematics.c.
Now editing this one to work in move_robot.c.
09/28/05: Redid entire inverse kinematics using suproblems. Feel
good about theta1, theta2, and theta3. Has been implemented in Matlab.
10/04/05: Replacing inverse kinematics with suproblem approach.
10/05/05: Did subproblem approach first in Matlab, then algebraically
in Mathematica. I'm using the Mathematica solutions in here so I don't
have to write a bunch of functions to do matrix manipulation. But I
guess I will eventually anyway to compute the forward kinematics. The
subproblem approach gave better position results than what I first had;
the orientation results are the same.
10/19/05: The explicit solutions for the wrist angles were very long,
so I'm writing functions to perform the calculations, basically
converting my Matlab code to C.
10/24/05: Added code to read the current robot position prior to
calculating the invere kinematics. Otherwise, while acquiring the
object, the relative counts will be wrong since the program will think
the robot started from its zero configuration. This is only true
before the first move.
11/11/05: Added code to store both temporary and permanent datafiles.
The temporary ones are used in the motion planning as the robots are
required to do additional movements. These are later deleted by the

```

main program, but the permanent files store the calculated joint angles for each set of movements for each robot.

11/29/05: Changed code so that theta5 = 0 if we are doing robot number 2. This is because joint 5 is not working so well on robot 2. Also, set theta6 = 0 for all robots since rotation in this direction breaks the nonholonomic constraint requirement. Also, that shear can damage the sensors.

04/26/06: Added finger length variable l5 so the location of the tool frame can be moved. It appears in the calculation for igsto.

07/11/06: Added l5 in puma.h. It is the distance from the wrist, the intersection of joints 4, 5, and 6, to the end of the finger. Turned theta6 "on"

07/14/06: moved definitions for qs and omegas out of puma.h and into here since I couldn't figure a way to #define an array. This fixed the problem with things being previously defined. And I can now use the l's for the finger position instead of hard coding it.

07/22/06: Added corrections to theta5 and theta6 due to mechanical coupling of the wrist.

07/23/06: The wrist angles were determined incorrectly. It only cropped up when I tried to achieve a rotation matrix other than identity. Made corrections.

07/25/06: changed order of wrist joint corrections so that theta6 is corrected prior to theta5. This changes theta6 based on the calculated theta5 instead of theta5's corrected value.

07/26/06: Still making corrections to the wrist angle calculations. I think it's finally correct now. Corrections complete for the inverse kinematics. I'm going to archive this, and start making changes to bring the desired fingertip configurations, which are stored in Tdes, back to the wrist. It appears the inverse kinematics only works with the desired tool frame at the wrist. Once the orientation is far enough from identity, the ball is dropped because the error at the fingertip. There was an error in the calculation for theta3. Making corrections to this, and it now appears the inverse kinematics WILL work with an arbitrary end-effector frame!

07/27/06: Took correction out of theta6 so it is always zero when performing inverse kinematics. This prevents the finger from twisting on the ball while checking the slip condition. Hopefully, this saves some wear on the sensors and prevents the finger from twisting on its frame. The latter could affect the newly added corrections to the sensor locations which are affected by the finger orientation.

07/31/06: Fixed another error in the calculation for theta4, had the indices switched in calculation for u = c - Pw.

08/01/06: Changed theta 6 so that its value equals its previous value. This is to prevent the finger from twisting on the ball while regripping. Originally I had set theta 6 = 0, but this only worked during the object acquisition phase since theta 6 started at zero. After finger reconfiguration, however, this is no longer the case.

```

**** Variable Definitions **** */

/* Include Files */

#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "inverse_kinematics.h"
#include "matrix.h"
#include "puma.h"
#include "dmclinux.h"
#include "fuzzy_ctlr.h"

#define NAN 2147483648UL
#define DOF 6

void inverse_kinematics(int robot_num, int type, long int *jtcts)
{
    char file_name[20];
    long int encoder[256];
    long int robot_enc[6]={0,0,0,0,0,0};
    int board_num, start = 0, which;
    double theta3 = 0.0, th4[2] = {0,0}, th5[2] = {0,0}, cosphi;
    double theta1 = 0.0, theta2 = 0.0, theta4 = 0.0, theta5 = 0.0, \
theta6 = 0.0;
    double x, y, num, num2, den;
    long int counts[6], prev_counts[6] = {0,0,0,0,0,0}, new_cts[6] \
= {0,0,0,0,0,0};
    double gstd[4][4] = {{0,0,0,0}}, **g2;
    double uproj[3] = {0,0,0}, vproj[3] = {0,0,0};
    double mag_uproj = 0, mag_vproj = 0;
    double cross[3] = {0,0,0}, delta = 0;
    double igsto[4][4] = {{1,0,0,-(12+15)},{0,1,0,11-13},\
{0,0,1,14-10},{0,0,0,1}};
    double z[3] = {0,0,0}, alpha = 0, beta = 0, gamma[2] = {0,0};
    double u[4] = {0,0,0,0}, c[3][2] = {{0,0}};
    double v[4] = {0,0,0,0};
    double **dummy1;
    double **dummy2;
    double **dummy3, **dummy4, **dummy5, **prod;
    double mexp1[4][4] = {{0,0,0,0}};

```

```

    double mexp2[4][4] = {{0,0,0,0}}, g2c[4][4] = {{0,0,0,0}};
    double mexp3[4][4] = {{0,0,0,0}}, mexp4[4][4] = {{0,0,0,0}}, \
mexp5[4][4] = {{0,0,0,0}};
    double omegal[3] = {0,0,1}, omega2[3] = {0,1,0}, omega3[3] = \
{0,-1,0};
    double omega4[3] = {0,0,1}, omega5[3] = {0,1,0}, omega6[3] = \
{-1,0,0};
    double w4xw5[3] = {-1,0,0};
    double q1[3] = {0,0,0}, q2[3] = {0,0,1}, q3[3] = {12, 0, 1}, \
Pw[3] = {12, 13-11, 10-14};
    double q4[3] = {12, 13-11, 0}, q5[3] = {12, 0, 10-14}, zero[3] = \
{0,0,0};
    double q6[3] = {0, 13-11, 10-14};
    int i, j = 0, k = 0, joint = 0, end_of_file, f = 1, n;
    FILE *ifp, *ofp, *ofp1[4];

    g2 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy1 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy2 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy3 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy4 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy5 = (double **) malloc((unsigned) 4*sizeof(double*));
    prod = (double **) malloc((unsigned) 4*sizeof(double*));

    for(i = 0; i < 4; i++)
    {
        g2[i]=(double *) malloc((unsigned) 4*sizeof(double));
        dummy1[i]=(double *) malloc((unsigned) 4*sizeof(double));
        dummy2[i]=(double *) malloc((unsigned) 4*sizeof(double));
        dummy3[i]=(double *) malloc((unsigned) 4*sizeof(double));
        dummy4[i]=(double *) malloc((unsigned) 4*sizeof(double));
        dummy5[i]=(double *) malloc((unsigned) 4*sizeof(double));
        prod[i]=(double *) malloc((unsigned) 4*sizeof(double));
    }

    // num_elements = num_cols*(sizeof(mat1)/sizeof(mat1[0]));

    //sleep(1);

    if (robot_num == 1)
start = 12;
    else if (robot_num == 2)
start = 26;
    else if (robot_num == 3)
start = 40;
    else
start = 54;

    for (board_num = 1; board_num <= 3; board_num++)
    {
        if (board_num == 1)
joint = 0;
        else
if (board_num == 2)
joint = 1;
    else
joint = 2;

        ReturnEncoder(encoder,board_num);
        for (k = 0; k < 2; k++)
        {
            robot_enc[joint] = encoder[start+7*k];
            joint += 3;
        }
    }

    for (i = 0; i < 6; i++)
prev_counts[i] = robot_enc[i];

    sprintf(file_name,"final_angles%d.dat",robot_num);
    ofp = fopen(file_name,"a");
    sprintf(file_name,"robot%d_cts.dat",robot_num);
    ofp1[0] = fopen(file_name,"w");
    ifp = fopen(INPUT_FILE,"r");

    if (type == 0 || type == 1)
    {
        for (i = 0; i < 4; i++)
        {
            if (i+1 != robot_num)
            {
                sprintf(file_name,\
"robot%d_cts.dat",i+1);
                ofp1[f] = fopen(file_name,"w");
                f += 1;
            }
        }
    }

    if (ifp == NULL)

```

```

{
    printf("Can't open %s\n",INPUT_FILE);
    exit(EXIT_FAILURE);
}

while ((end_of_file = getc(ifp)) != EOF)
{
    ungetc(end_of_file,ifp);
    delta = 0;
    mag_uproj = 0;
    mag_vproj = 0;
    /* Read the desired configuration. */
    for(i = 0; i < 4; i++)
fscanf(ifp,"%lf%lf%lf%lf",&gstd[i][0],&gstd[i][1],\
&gstd[i][2],&gstd[i][3]);

    /* Solve for theta3 */

    num2 = -2*gstd[0][3]*15*gstd[0][0] + 15*15*gstd[0][0]*\
gstd[0][0] - 2*gstd[1][3]*15*gstd[1][0] + 15*15*gstd[1][0]*\
gstd[1][0] - 2*gstd[2][3]*15*gstd[2][0] + 2*15*10*gstd[2][0] + 15* \
15*gstd[2][0]*gstd[2][0];
    num = gstd[0][3]*gstd[0][3] + gstd[1][3]*gstd[1][3] + \
gstd[2][3]*gstd[2][3] + 10*10 - 11*11 - 12*12 - 13*13 - 14*14 + \
2*11*13 - 2*gstd[2][3]*10 + num2;
    den = 2.0*12*14;

    theta3 = asin(num/den);

    /* Solve for theta2 */
    //num = 10*10 + 11*11 + 12*12 + 13*13 + 14*14 - \
2*11*13 + 2*12*14*sin(theta3) - gstd[0][3]*gstd[0][3] - \
gstd[1][3]*gstd[1][3] - gstd[2][3]*gstd[2][3];
    num = 10*10 + 11*11 + 12*12 + 13*13 + 14*14 - 2*11*13 + \
2*12*14*sin(theta3) - pow(gstd[0][3] - gstd[0][0]*15,2) - \
pow(gstd[1][3] - gstd[1][0]*15,2) - pow(gstd[2][3] - gstd[2][0]*15,2);
    den = 2 * 10 * sqrt(14*cos(theta3)*14*cos(theta3) + \
(12 + 14*sin(theta3))*(12 + 14*sin(theta3)));
    cosphi = num/den;

    x = 10 * 14 * cos(theta3);
    y = 10 * (12 + 14*sin(theta3));
    theta2 = atan2(y,x) - acos(cosphi);

    /* Solve for theta1 */
    //x = gstd[1][3]*(13 - 11) + gstd[0][3]*(12*cos(theta2) \
- 14*sin(theta2-theta3));
    //y = gstd[0][3]*(11 - 13) + gstd[1][3]*(12*cos(theta2) \
- 14*sin(theta2-theta3));
    x = (gstd[1][3] - gstd[1][0]*15)*(13 - 11) + (gstd[0][3] \
- gstd[0][0]*15)*(12*cos(theta2) - 14*sin(theta2-theta3));
    y = (gstd[0][3] - gstd[0][0]*15)*(11 - 13) + \
12*cos(theta2)*(gstd[1][3] - gstd[1][0]*15) - 14*sin(theta2- \
theta3)*(gstd[1][3] - gstd[1][0]*15);
    theta1 = atan2(y,x);

    /* Solve for the wrist angles, theta 5 first */
dummy1 = matrix_exponential(omega1, q1, -theta1, dummy1);
dummy2 = matrix_exponential(omega2, q2, -theta2, dummy2);
dummy3 = matrix_exponential(omega3, q3, -theta3, dummy3);

    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            mexp1[i][j] = dummy1[i][j];
            mexp2[i][j] = dummy2[i][j];
            mexp3[i][j] = dummy3[i][j];
        }
    }

    g2 = matrix_product(5, 4, g2, mexp3, mexp2, mexp1, \
gstd, igsto);

    for (i = 0; i < 3; i++)
u[i] = q6[i] - Pw[i];

    get_delta(g2, q6, Pw, v);

    alpha = dotprod(omega4, v, 3);
    beta = dotprod(omega5, u, 3);
    gamma[0] = sqrt(dotprod(u, u, 3) - alpha*alpha - \
beta*beta);
    gamma[1] = -gamma[0];

    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
        {
            z[j] = alpha*omega4[j] + beta*omega5[j] \
+ gamma[i]*w4w5[j];

```

```

    c[j][i] = z[j] + Pw[j];
}
}

projection(u, omega5, uproj);

for (i = 0; i < 2; i++)
{
    for (j = 0; j < 3; j++)
v[j] = c[j][i] - Pw[j];
    projection(v, omega5, vproj);
    y = 0;
    x = dotprod(uproj, vproj, 3);
    cross_product(uproj, vproj, cross);
    y = dotprod(omega5, cross, 3);

    th5[i] = atan2(y,x);
}

if (fabs(th5[0]) < fabs(th5[1]))
which = 0;
else
which = 1;

    theta5 = th5[which];

    /* Solve for theta4 */
    get_delta(g2, q6, Pw, v);
    projection(v, omega4, vproj);

    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            u[j] = c[j][i] - Pw[j];
        projection(u, omega4, uproj);
        y = 0;
        x = dotprod(uproj, vproj, 3);
        cross_product(uproj, vproj, cross);
        y = dotprod(omega4, cross, 3);

        th4[i] = atan2(y, x);
    }
    if (fabs(th4[0]) < fabs(th4[1]))
        which = 0;
    else
        which = 1;

    theta4 = th4[which];

    /* Solve for theta6 */

    theta6 = prev_counts[5]/RAD2COUNT6;

    /* apply correction to theta6 */
    theta6 = theta6 - c46*theta4 - c56*theta5;
    /* apply correction to theta5 */
    theta5 = theta5 - c45*theta4;

    if ((long int) theta1 == NAN || (long int) theta2 == \
NAN || (long int) theta3 == NAN || (long int) theta4 == NAN || \
(long int) theta5 == NAN || (long int) theta6 == NAN)
    {
        printf("\nFailure during inverse kinematics. \
Configuration can not be acheived.\n\n");
        exit(EXIT_FAILURE);
    }

    counts[0] = theta1*RAD2COUNT1;
    counts[1] = theta2*RAD2COUNT2;
    counts[2] = theta3*RAD2COUNT3;
    counts[3] = theta4*RAD2COUNT4;
    counts[4] = theta5*RAD2COUNT5;
    counts[5] = prev_counts[5]; //theta6*RAD2COUNT6;

    for (i = 0; i < 6; i++)
new_cts[i] = counts[i] - prev_counts[i];

    //printf("%lf %lf %lf %lf %lf %lf\n",theta1,theta2,\
theta3,theta4,theta5,theta6);
    //printf("%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\n",\
theta1*R2D,theta2*R2D,theta3*R2D,theta4*R2D,theta5*R2D,theta6*R2D);
    fprintf(ofp, "%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\n",\
theta1*R2D,theta2*R2D,theta3*R2D,theta4*R2D,theta5*R2D,theta6*R2D);

    fprintf(ofp1[0], "%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n", \
new_cts[0],new_cts[1],new_cts[2],new_cts[3],new_cts[4],new_cts[5]);
    if (type == 0 || type == 1)
{

```

```

    for (i = 1; i < 4; i++)
fprintf(ofp1[i], "0\t0\t0\t0\t0\t0\n");
}

    for (i = 0; i < 6; i++)
prev_counts[i] = counts[i];
}

    /*
final_angles[0] = theta1;
final_angles[1] = theta2;
final_angles[2] = theta3;
final_angles[3] = theta4;
final_angles[4] = theta5;
final_angles[5] = theta6;
*/
fclose(ifp);
fclose(ofp);
if (type == 2)
{
    n = fclose(ofp1[0]);
    if (n == EOF)
    {
        sprintf(file_name, "robot%d_cts.dat", robot_num);
        printf("\nProblem closing file %s\n", file_name);
        exit(EXIT_FAILURE);
    }
}
else
for (i = 0; i < f; i++)
{
    n = fclose(ofp1[i]);
    if (n == EOF)
    {
        sprintf(file_name, "robot%d_cts.dat", i+1);
        printf("\nProblem closing file %s\n", \
file_name);
    }
}
}
}

```

C.6 File matrix.c

```

#include "matrix.h"
#include <stdarg.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

/* take the vector omega and return the skew symmetric matrix \
smmat */

void skewsm(const double *omega, double *smmat)
{
    *smmat = 0.0;
    *(smmat + 1) = -omega[2];
    *(smmat + 2) = omega[1];
    *(smmat + 3) = omega[2];
    *(smmat + 4) = 0.0;
    *(smmat + 5) = -omega[0];
    *(smmat + 6) = -omega[1];
    *(smmat + 7) = omega[0];
    *(smmat + 8) = 0.0;
}

/* return the scalar product of two vectors */
double dotprod(const double *vector1, const double *vector2, const \
int size)
{
    int i;
    double result = 0.0;

    for (i = 0; i < size; i++)
result += vector1[i]*vector2[i];
    return result;
}

/* return the cross product of two vectors */
void cross_product(const double *vector1, const double *vector2, \
double *result)
{
    result[0] = vector1[1]*vector2[2] - vector1[2]*vector2[1];
    result[1] = vector1[2]*vector2[0] - vector1[0]*vector2[2];
}

```

```

    result[2] = vector1[0]*vector2[1] - vector1[1]*vector2[0];
}

double** matrix_product(int n, int r, double **c, ...)
{
    va_list ap;
    int i, j, m, s, a;
    double sum = 0.0;
    double *mat1, *mat2;
    mat1 = malloc(r*r * sizeof(double));
    mat2 = malloc(r*r * sizeof(double));

    for (i = 0; i < r; i++)
    for (j = 0; j < r; j++)
        c[i][j] = 0.0;

    va_start(ap, c);
    mat1 = va_arg(ap, double*);

    for (m = 1; m < n; m++)
    {
        s = 0;
        mat2 = va_arg(ap, double*);

        for (a = 0; a < r; a++)
        {
            for (i = 0; i < r; i++)
            {
                sum = 0.0;
                for (j = 0; j < r; j++)
                sum += *(mat1 + r*a + j) * *(mat2 + \
r*j + i);
                //sum += mat1[r*a+j] * mat2[r*j+i];
                c[a][i] = sum;
            }
        }

        for (i = 0; i < r; i++)
        {
            for (j = 0; j < r; j++)
            {
                *(mat1 + s) = c[i][j];
                s += 1;
            }
        }

        va_end(ap);

        return c;
    }
}

double** matrix_exponential(double *w, double *q, double th, double \
**result)
{
    int i, j;
    int eye3[3][3] = {{1,0,0},{0,1,0},{0,0,1}};
    double smmat[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
    double temp[3][3] = {{0,0,0}};
    double **prod;
    double v[3] = {0,0,0};
    double v1[3] = {0,0,0};
    double wt[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
    double omg_exp[3][3] = {{0,0,0},{0,0,0},{0,0,0}};

    prod = (double **) malloc((unsigned) 3*sizeof(double*));
    for(i = 0; i < 3; i++) {
    prod[i] = (double *) malloc((unsigned) 3*sizeof(double));
    }

    for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
        result[i][j] = 0.0;

    for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        wt[i][j] = w[i]*w[j];

    cross_product(q, w, v);
    cross_product(w, v, v1);

    skewsm(w, smmat[0]);

    for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        temp[i][j] = smmat[i][j];

    prod = matrix_product(2, 3, prod, temp);
}

```

```

    for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
    {
        omg_exp[i][j] = eye3[i][j] + smmat[i][j] * sin(th) \
+ (prod[i][j] * (1 - cos(th)));
        result[i][j] = omg_exp[i][j];
    }

    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            result[i][3] += (eye3[i][j] - omg_exp[i][j]) * \
v1[j] + wt[i][j]*v[j]*th;
        }
    }

    /* for a revolute joint, e(zhat*th) = [ e^(what*th) \
[0; 0; 0; 0 1] */

    for (i = 0; i < 3; i++)
    result[3][i] = 0.0;
    result[3][3] = 1.0;

    return result;
}

void get_delta(double **g2, double *qa, double *qb, double *p)
{
    int i, j;
    double sum;
    double qc[4] = {qa[0],qa[1],qa[2],1};
    double qd[4] = {qb[0], qb[1], qb[2], 1};

    for (i = 0; i < 4; i++)
    {
        sum = 0;
        for (j = 0; j < 4; j++)
        sum += g2[i][j] *qc[j];
        p[i] = sum - qd[i];
    }
}

void projection(double *v, double *w, double *p)
{
    int i, j;
    double sum;
    double wt[3][3] = {{0,0,0},{0,0,0},{0,0,0}};

    for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        wt[i][j] = w[i]*w[j];

    for (i = 0; i < 3; i++)
    {
        sum = 0;
        for (j = 0; j < 3; j++)
        sum += wt[i][j] *v[j];
        p[i] = v[i] - sum;
    }
}

double** forward_kinematics(double *qs, double *omegas, double \
*ths, double *gst, double **gst)
{
    int i, j;
    double **mexp1, **mexp2, **mexp3, **mexp4, **mexp5, **mexp6;
    double mexp11[4][4] = {{0,0,0,0}}, mexp22[4][4] = {{0,0,0,0}}, \
mexp33[4][4] = {{0,0,0,0}};
    double mexp44[4][4] = {{0,0,0,0}}, mexp55[4][4] = {{0,0,0,0}}, \
mexp66[4][4] = {{0,0,0,0}};

    mexp1 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp2 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp3 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp4 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp5 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp6 = (double **) malloc((unsigned) 4*sizeof(double*));

    for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
        gst[i][j] = 0.0;

    for(i = 0; i < 4; i++)
    {

```

```

    mexp1[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp2[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp3[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp4[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp5[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp6[i]=(double *) malloc((unsigned) 4*sizeof(double));
}

    mexp1 = matrix_exponential(&omegas[0], &qs[0], ths[0], mexp1);
    mexp2 = matrix_exponential(&omegas[3], &qs[3], ths[1], mexp2);
    mexp3 = matrix_exponential(&omegas[6], &qs[6], ths[2], mexp3);
    mexp4 = matrix_exponential(&omegas[9], &qs[9], ths[3], mexp4);
    mexp5 = matrix_exponential(&omegas[12], &qs[12], ths[4], mexp5);
    mexp6 = matrix_exponential(&omegas[15], &qs[15], ths[5], mexp6);

    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            mexp11[i][j] = mexp1[i][j];
            mexp22[i][j] = mexp2[i][j];
            mexp33[i][j] = mexp3[i][j];
            mexp44[i][j] = mexp4[i][j];
            mexp55[i][j] = mexp5[i][j];
            mexp66[i][j] = mexp6[i][j];
        }
    }

    gst = matrix_product(7, 4, gst, mexp1, mexp2, mexp3, mexp4, \
mexp55, mexp66, &gst0[0]);

    return gst;
}

```

C.7 File move_robot.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: move_robot.c *
* Date Written: 18 JUL 2005 *
* Last Date Modified: 30 JUL 2006 *
* Written for: research *
*****
*****

```

This program queries the user for the type of robot motion to perform. It calls the appropriate functions to determine the subsequent encoder counts to perform the motion. It reads this file and writes it to the boards.

```
***** REVISION LOG *****
```

```

7/18/05: This is a copy of circle2.c. Which would write the board
information after reading and parsing prcounts.dat. To generate a
robot motion currently takes 3 runs - 1 to determine the trajectory
which generates the desired robot configuration at each point
(Tdes.dat), 1 to perform the inversed kinematics which generates the
desired encoder counts (prcounts.dat), then finally circle2. I want
this all to be done in one run. Which would ultimately include
manipulation choices.
7/26/05: Cleaned up code to read each line of prcounts.dat.
10/23/05: This used to be the main program to query the user for a
robot motion, determine the inverse kinematics, and send commands to
the control boards. I've redone the program flow so that, now, this
is a sub-program called by main to only send commands to the boards.
11/04/05: This program now needs to read four input files, one for
each robot, and combine each line. Right now, however, the lines
aren't being read correctly.
11/07/05: This is working now. It reads 4 separate encoder files,
parses them, and writes them to the boards to move each robot.
11/11/05: Added code to write all files for the encoder counts for
manipulation or only 1 for specific robot if simple move is selected.
Added timestamp to the datafiles.
11/29/05: Added robot 2 to the mix.
04/15/06: Added code to remove temporary robot_cts.dat datafiles if
one of the datafiles cannot be read. Otherwise, we are left with
empty files.
07/30/06: Changed the way the encoder counts are stored and printed
to a file. Just put them all in a matrix instead of having 4 arrays.
They didn't appear to be printing properly the first way. And I
know the new way works because I wrote and tested it just recently
in slip.c.

```

```
**** Variable Definitions ****
```

```

NUMBRDS - number of motion control boards (3). Each board is capable
of controlling 8 axes. Board 1 controls joints 1 and 4 on each of
the 4 robots, board 2 controls joints 2 and 5, and board 3
controls joints 3 and 6.
NUMJOINTS - number of joints controlled by each board (8)
CYCLES - number of times to move each joint
response - response buffer from the boards after interrogation, a
string
joints - string containing the axes definitions for each joint. A
and B refer to robot 1, C and D to robot 2, etc.

commands - string containing the a 2-letter, executable instruction for
a board.
move_amt - a string that sets the encoder counts for a position command.
brd_cmd - a string containing a board command, e.g. PRA=2000 means set
the position relative to 2000 encoder counts on joint A. Depending on
the board, this would be joint 1, 2, or 3 on robot 1. The command is
built through combinations of commands, joints, and move_amt.
Currently, only position commands are done
i,j - loop counters
board_num - loop counter to loop through each board, set by NUMBRDS
which_joint - loop counter to loop through each joint, set by
NUMJOINTS

***** */

/* Include Files */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include "dmclinux.h"
#include "move_robot.h"
#include "puma.h"

#define NUMBRDS 3
#define NUMJOINTS 8

/* Function Prototypes */
void TellPosition(const char *, const char *);
void PositionAbsolute(const long int *, const long int *, const long \
int *, const long int *, const int, char *);

/* Main Function */

void move_robot(int robot_num, long int *final_counts)
{
    char brd_cmd[20], file_name[20];
    int brd_cmds = 0;
    long int move_amt;
    long int robot1[] = {0,0,0,0,0,0};
    long int robot2[] = {0,0,0,0,0,0};
    long int robot3[] = {0,0,0,0,0,0};
    long int robot4[] = {0,0,0,0,0,0};
    long int robot_enc_cts[NUM_ROBOTS][6] = {{0,0,0,0,0,0}};
    long int encoder[256];
    int i = 0, counter, j = 0, k, m, board_num, joint = 0, ch;
    int ready;
    long int sum[] = {0,0,0,0,0,0};
    FILE *ifp[4], *ofpp, *ofp[4];

    time_t start_time = time(NULL);

    for (i = 0; i < 4; i++)
    {
        sprintf(file_name,"robot%d_cts.dat",i+1);
        ifp[i] = fopen(file_name,"r");
        if (ifp[i] == NULL)
        {
            printf("Can't open %s\n",file_name);
            for (i = 1; i <= 4; i++)
            {
                sprintf(file_name, "robot%d_cts.dat", i);
                remove(file_name);
            }
            exit(EXIT_FAILURE);
        }
    }

    /*#if 0
    for (j = 0; j <= sizeof(brd_cmd); j++) /* initialize brd_cmd \
array */
    brd_cmd[j] = '\0';

    //for (board_num = 1; board_num <= NUMBRDS; board_num++)
    //Write("SH;",board_num);

```

```

        if (robot_num == 5) /* there are only 4 robots, a 5 indicates
                           manipulation in which all robots are used */
    {
        for (i = 0; i < 4; i++)
        {
            sprintf(file_name,"robot%d_pos.dat", i+1);
            ofp[i] = fopen(file_name,"a");
        }
    }
    else
    {
        sprintf(file_name,"robot%d_pos.dat", robot_num);
        ofpp = fopen(file_name,"w");
    }

    i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
    {
        ReturnEncoder(encoder,board_num);
        for (m = 0; m < 4; m++)
        {
            if (board_num == 1)
            joint = 0;
            else
            if (board_num == 2)
            joint = 1;
            else
            joint = 2;
            for (k = 0; k < 2; k++)
            {
                robot_enc_cts[m][joint] = encoder[12+7*i];
                i += 1;
                joint += 3;
            }
        }
        i = 0;
    }

    if (robot_num == 5)
    {
        fprintf(ofp[0],"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
        fprintf(ofp[1],"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
        fprintf(ofp[2],"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
        fprintf(ofp[3],"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);
    }
    else if (robot_num == 1)
    fprintf(ofpp,"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    else if (robot_num == 2)
    {
        fprintf(ofpp,"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    }
    else if (robot_num == 3)
    fprintf(ofpp,"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    else if (robot_num == 4)
    fprintf(ofpp,"%g\t%d\t%d\t%d\t%d\t%d\t%d\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);

    /* enter contour mode */
    for (j = 1; j <= NUMBRDS; j++)
    {
        Write("CM ABCDEFGH;",j);
        Write("DT 8;",j);
    }

    /* Input file contains rows of, ultimately, 24 encoder counts
    for a desired trajectory. Currently, there are only six,

```

```

corresponding to the first robot. Board 1 controls joints 1,4; board
2 controls joints 2,5; board 3 controls joints 3,6. */

    while ((ch = getc(ifp[0])) != EOF)
    {
        ungetc(ch, ifp[0]);
        for (i = 0; i < 4; i++)
        {
            /* read until a line of data. The first 6
            entries belong to robot 1, the next 6 to robot 2, and so on. */

            counter = 0;
            while ((ch = getc(ifp[i])) != '\n')
            {
                ungetc(ch, ifp[i]);
                fscanf(ifp[i], "%ld", &move_amt);
                //printf("\n%ld\t%d\t%d\t%d\t%d", move_amt, j, \
                counter, ch);
                if (i == 0)
                {
                    robot1[counter] = move_amt;
                    //printf("robot1[%d] = %ld\t", \
                    counter, move_amt);
                }
                else if (i == 1)
                {
                    robot2[counter] = move_amt;
                    //printf("robot2[%d] = %ld\t", \
                    counter, move_amt);
                }
                else if (i == 2)
                {
                    robot3[counter] = move_amt;
                }
                else
                {
                    robot4[counter] = move_amt;
                    sum[i] += move_amt;
                    counter += 1;
                }
            }
        }

        /*#if 0

        /* write the Command Data to each board. Board 1 gets
        robot[0], robot[3] (joints 1 and 4), board 2 gets robot[1], robot[4]
        (joints 2 and 5), and Board 3 gets robot[2], robot[5] (joints 3 and 6)
        for each robot. */

        for (board_num = 1; board_num <= NUMBRDS; board_num++)
        {
            /* the function name is a remnant from when I
            was trying to use PA commands instead of CD. */
            PositionAbsolute(robot1, robot2, robot3, robot4, \
            board_num, brd_cmd);
            //printf("\n%s", brd_cmd);

            /*#if 0

            /* keep writing to the board until it's ready to receive
            another command. I don't remember exactly why I do this. I think the
            computer would write too fast to the board and some of the data would
            be missed, I guess if both the holding and processing buffers were full.
            Therefore, the desired trajectory would not be followed and the
            C-program would end well before the robot motion was finished. */
            ready = Write(brd_cmd, board_num);
            while (ready)
            ready = Write(brd_cmd, board_num);
        }

        brd_cmds += 1;
        printf("%d\n", brd_cmds);

        i = 0;
        for (board_num = 1; board_num <= 3; board_num++)
        {
            ReturnEncoder(encoder, board_num);
            for (m = 0; m < 4; m++)
            {
                if (board_num == 1)
                joint = 0;
                else
                if (board_num == 2)
                joint = 1;
                else
                joint = 2;
                for (k = 0; k < 2; k++)
                {
                    robot_enc_cts[m][joint] = encoder[12+7*i];
                    i += 1;
                    joint += 3;
                }
            }
        }

        i = 0;

```

```

}

    if (robot_num == 5) /* write data for all the robots */
    {
        fprintf(ofp[0], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[0][0], \
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3], \
robot_enc_cts[0][4],robot_enc_cts[0][5]);
        fprintf(ofp[1], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[1][0], \
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3], \
robot_enc_cts[1][4],robot_enc_cts[1][5]);
        fprintf(ofp[2], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[2][0], \
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3], \
robot_enc_cts[2][4],robot_enc_cts[2][5]);
        fprintf(ofp[3], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[3][0], \
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3], \
robot_enc_cts[3][4],robot_enc_cts[3][5]);
    }

    else if (robot_num == 1)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[0][0], \
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3], \
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    else if (robot_num == 2)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[1][0], \
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3], \
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    else if (robot_num == 3)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[2][0], \
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3], \
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    else if (robot_num == 4)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[3][0], \
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3], \
robot_enc_cts[3][4],robot_enc_cts[3][5]);
}

/* wait to make sure the boards are ready to receive more
commands and then turn the contour mode off. */

    sleep(2);
    for (j = 1; j <= 3; j++)
    {
        Write("DT0;",j);
        Write("CD0;",j);
    }

    i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
    {
        ReturnEncoder(encoder,board_num);
        for (m = 0; m < 4; m++)
        {
            if (board_num == 1)
            joint = 0;
            else
            if (board_num == 2)
            joint = 1;
            else
            joint = 2;
            for (k = 0; k < 2; k++)
            {
                robot_enc_cts[m][joint] = \
encoder[12+7*i];
                i += 1;
                joint += 3;
            }
        }

        i = 0;
    }

    if (robot_num == 5)
    {
        fprintf(ofp[0], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[0][0], \
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3], \
robot_enc_cts[0][4],robot_enc_cts[0][5]);
        fprintf(ofp[1], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[1][0], \
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3], \
robot_enc_cts[1][4],robot_enc_cts[1][5]);
        fprintf(ofp[2], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL),start_time), robot_enc_cts[2][0], \

```

```

robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    fprintf(ofp[3], "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL), start_time), robot_enc_cts[3][0], \
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3], \
robot_enc_cts[3][4],robot_enc_cts[3][5]);
}
    else if (robot_num == 1)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL), start_time), robot_enc_cts[0][0], \
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3], \
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    else if (robot_num == 2)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL), start_time), robot_enc_cts[1][0], \
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3], \
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    else if (robot_num == 3)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL), start_time), robot_enc_cts[2][0], \
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3], \
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    else if (robot_num == 4)
fprintf(ofpp, "%g\t%d\t%d\t%d\t%d\t%d\t%d\n", \
difftime(time(NULL), start_time), robot_enc_cts[3][0], \
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3], \
robot_enc_cts[3][4],robot_enc_cts[3][5]);

    for (i = 0; i < 4; i++)
fclose(ifp[i]);

    if (robot_num == 5)
{
    for (i = 0; i < 4; i++)
fclose(ofp[i]);
}
    else
fclose(ofpp);
}

/* This function changes the command that is sent to the board in
the main program. */

void PositionAbsolute(const long int *robot1, const long int \
*robot2, const long int *robot3, const long int *robot4, const int \
board_num, char *full_cmd)

{
    switch(board_num)
    {
default:
    printf("\nError occurred in file move_robot --- No valid
controller specified\n");
case 1:
    sprintf(full_cmd, "CD %ld,%ld,%ld,%ld,%ld,%ld,%ld; \
WC;", robot1[0],robot1[3],robot2[0],robot2[3],robot3[0],robot3[3], \
robot4[0],robot4[3]);
    break;
case 2:
    sprintf(full_cmd, "CD %ld,%ld,%ld,%ld,%ld,%ld,%ld; \
WC;", robot1[1],robot1[4],robot2[1],robot2[4],robot3[1],robot3[4], \
robot4[1],robot4[4]);
    break;
case 3:
    sprintf(full_cmd, "CD %ld,%ld,%ld,%ld,%ld,%ld,%ld; \
WC;", robot1[2],robot1[5],robot2[2],robot2[5],robot3[2],robot3[5], \
robot4[2],robot4[5]);
    break;
}
}
}

```

C.8 File acquire_object.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: read_sensors.c *
* Date Written: 11 NOV 2005 *
* Last Date Modified: 11 AUG 2006 *
* Written for: robots *
*****

```

This is a sub-program called by main to check whether the end-effector has sufficiently contacted the object at the beginning of the manipulation routine.

```

***** REVISION LOG *****
11/11/05: Moved code to touch the object based on sensor values from
main to here. Going to add code to lift the object, and then return
to call the motion planning.
11/29/05: Changed code to incorporate robot 2.
12/19/05: Added position code to set position vector to acquire cube,
and added code to store the last force sensor data. This is used to
determine object compliance.
04/17/06: Added code to remove the temporary counts datafiles, then
rename openloop motion plan files, and call move_robot again. The
openloop datafiles are currently generated in using the grasp
constraint equation.
04/20/06: Changed code so that robot 2 isn't used. Joint 5 is dead.
07/10/06: Added call to fuzzy controller for change in x position
of finger. Turned Robot 2 back on since joint 5's motor was replaced.
07/12/06: Replaced code that checked all 8 analog inputs to determine
if a robot was sufficiently contacting the object. The new code first
determines the maxium force associated with each robot by comparing
the 6 sensor readings. Then, only the max is checked.
07/30/06: Changed code so average force values for each finger are
sent to the fuzzy controller instead of just the max for each finger.
08/11/06: added object to call of fuzzy controller.

```

```

***** */
#include <time.h>
#include <math.h>
#include "acquire_object.h"
#include "move_robot.h"
#include "dmclinux.h"
#include "move_options.h"
#include "puma.h"
#include "inverse_kinematics.h"
#include "fuzzy_ctlr.h"

int acquire_object(int object, int type)
{
    int i, j;
    int robot_num;
    int board_num;
    int robot_flag[4] = {1,1,1,0}; /* flag to stop robot. 1 if true,
0 if false */
    short int anin[256];
    double **Rd;
    double robot_anin[4][6]={{0,0,0,0,0,0}};
    long int jctcts[6] = {0,0,0,0,0,0};
    long int final_counts[6] = {0,0,0,0,0,0};
    float xo[] = {0,0,0}, xf[] = {0,0,0}, xnew[3] = {0,0,0}, \
xnext[4] = {0,0,0,0};
    float xlift[3] = {0,0,0};
    float dx = 0;
    short int new_max;
    short int max_force[4] = {0,0,0,0};
    double sum, avg_force[4] = {0,0,0};
    char file_name[20];
    FILE *ofp;
    //time_t start_time = time(NULL);

    Rd = (double **) malloc((unsigned) 3*sizeof(double*));

    for (i = 0; i < 3; i++)
Rd[i] = (double *) malloc((unsigned) 4*sizeof(double));

    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            if (i == j)
Rd[i][j] = 1;
            else
Rd[i][j] = 0;
        }
    }

    printf("\n acquiring object ... \n");

    xo[0] = 12 + 15;
    xo[1] = 13 - 11;
    xo[2] = 10 - 14;

    if (object == 0)
    {
        xf[0] = 28;
        xf[1] = 0;
        xf[2] = 8.5;
    }
    else if (object == 2)
    {
        xf[0] = 29;
    }
}

```

```

        xf[1] = 0;
        xf[2] = 8.5;
    }
    for (i = 0; i < 4; i++)
xnext[i] = xf[0];

    for (robot_num = 1; robot_num <= 4; robot_num++)
    {
        manipulate(xo, xf, Rd, robot_num);
        inverse_kinematics(robot_num, type, jtcts);
    }
    move_robot(5, final_counts);

    for (board_num = 1; board_num <= 3; board_num++)
    {
        printf("\n");
        ReturnAnlg(anin, board_num);

        /* if force readings < 0, set them to 0 */
        for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

        /* for (j = 0; j < 8; j++)
        {
            if (j == 0)
robot_anin[k][i] = anin[33+14*j]*ANALOG_RES;
            else if (j == 1)
robot_anin[k][i+1] = anin[33+14*j]*ANALOG_RES;
            else if (j == 2)
robot_anin[k+1][i] = anin[33+14*j]*ANALOG_RES;
            else if (j == 3)
robot_anin[k+1][i+1] = anin[33+14*j]*ANALOG_RES;
            else if (j == 4)
robot_anin[k+2][i] = anin[33+14*j]*ANALOG_RES;
            else if (j == 5)
robot_anin[k+2][i+1] = anin[33+14*j]*ANALOG_RES;
            else if (j == 6)
robot_anin[k+3][i] = anin[33+14*j]*ANALOG_RES;
            else
robot_anin[k+3][i+1] = anin[33+14*j]*ANALOG_RES;
        }
        i += 2; */

        for (robot_num = 1; robot_num <= 4; robot_num++)
        {
            if (robot_num == 1)
            {
                new_max = MAX(anin[117], anin[131]);
                if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
            }
            else if (robot_num == 2)
            {
                new_max = MAX(anin[61], anin[75]);
                if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
            }
            else if (robot_num == 3)
            {
                new_max = MAX(anin[89], anin[103]);
                if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
            }
            else if (robot_num == 4)
            {
                new_max = MAX(anin[33], anin[47]);
                if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
            }
        }
    }

    for (i = 0; i < 4; i++)
    {
        sum = 0;
        for (j = 0; j < 6; j++)
sum += robot_anin[i][j];
        avg_force[i] = sum/6.0;
    }

    while (robot_flag[0] || robot_flag[1] || robot_flag[2] || \
robot_flag[3])
    {
        for (i = 0; i < 4; i++)
        {
            if (robot_flag[i])
            {
                xf[0] = xnext[i];
                dx = fuzzy_controller(max_force[i]* \
ANALOG_RES, xf[0], object);
            }
        }
    }

```

```

    xnext[i] = xf[0] + dx; //0.25;
    xnew[0] = xnext[i];
    xnew[1] = xf[1];
    xnew[2] = xf[2];

    manipulate(xf, xnew, Rd, i+1);
    inverse_kinematics(i+1, type, jtcts);
    printf("New trajectory for Robot #%d \
calculated.\n\n",i+1);
}
else
{
    sprintf(file_name,"robot%d_cts.dat", i+1);
    ofp = fopen(file_name, "w");
    fprintf(ofp,"0\t0\t0\t0\t0\n");
    fprintf(ofp,"0\t0\t0\t0\t0\n");
    fclose(ofp);
}
if (fabs(dx) <= TOL)
robot_flag[i] = 0;
}
move_robot(5, final_counts);

for (board_num = 1; board_num <= 3; board_num++)
{
    printf("\n");
    ReturnAnlg(anin, board_num);

    /* if force readings < 0, set them to 0 */
    for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

    /* for (j = 0; j < 8; j++)
    {
        if (j == 0)
robot_anin[k][i] = anin[33+14*j]* \
ANALOG_RES;
        else if (j == 1)
robot_anin[k][i+1] = anin[33+14*j]* \
ANALOG_RES;
        else if (j == 2)
robot_anin[k+1][i] = anin[33+14*j]* \
ANALOG_RES;
        else if (j == 3)
robot_anin[k+1][i+1] = anin[33+14*j]* \
ANALOG_RES;
        else if (j == 4)
robot_anin[k+2][i] = anin[33+14*j]* \
ANALOG_RES;
        else if (j == 5)
robot_anin[k+2][i+1] = anin[33+14*j]* \
ANALOG_RES;
        else if (j == 6)
robot_anin[k+3][i] = anin[33+14*j]* \
ANALOG_RES;
        else
robot_anin[k+3][i+1] = anin[33+14*j]* \
ANALOG_RES;
    }
i += 2; */

    for (robot_num = 1; robot_num <= 4; robot_num++)
    {
        if (robot_num == 1)
        {
            new_max = MAX(anin[117],anin[131]);
            if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
        }
        else if (robot_num == 2)
        {
            new_max = MAX(anin[61],anin[75]);
            if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
        }
        else if (robot_num == 3)
        {
            new_max = MAX(anin[89],anin[103]);
            if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
        }
        else if (robot_num == 4)
        {
            new_max = MAX(anin[33],anin[47]);

```

```

    if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
}
}
for (robot_num = 1; robot_num <= 4; robot_num++)
printf("%.2f\t", max_force[robot_num - 1]*ANALOG_RES);

printf("\n");

for (i = 0; i < 4; i++)
{
sum = 0;
for (j = 0; j < 6; j++)
sum += robot_anin[i][j];
avg_force[i] = sum/6.0;
}

/* lift object */
printf("\nlifting object\n\n");

for (i = 0; i < 4; i++)
{
xnew[0] = xnext[i];
xlift[0] = xnew[0];
xlift[1] = xnew[1];
xlift[2] = xnew[2] + 5.0;
if (xlift[2] < 6.0)
{
printf("\nWARNING: motion may damage robot, \
aborting\n");
exit(EXIT_FAILURE);
}
manipulate(xnew, xlift, Rd, i+1);
inverse_kinematics(i+1, type, jctcs);
}

move_robot(5, final_counts);

return 0;
}

```

C.9 File talk2matlab.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: talk2matlab.c *
* Date Written: 20 JUL 2006 *
* Last Date Modified: 05 AUG 2006 *
* Written for: research *
*****

This sub-program is called from main, and contains code to perform a
fixed-point rotation of an object once it has been acquired by the
robots.

***** REVISION LOG *****
07/20/06: This is a streamlined version of rotate_object.c. I'm
removing everything that isn't needed when combining with matlab.
07/25/06: changed matlab_info.dat to save current counts instead of
coverting to angles. The extra step isn't really necessary, and
should make it easier to perform wrist joint corrections in matlab.
07/30/06: Changed finding the finger's contact coordinate from the
maximum sensor value to the centroid of all sensors.
08/05/06: Added current rotation and object's twist axis to matlab's
datafile

**** Variable Definitions **** */

/* Include Files */
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "acquire_object.h"
#include "move_robot.h"
#include "dmclinux.h"
#include "move_options.h"
#include "puma.h"
#include "inverse_kinematics.h"
#include "fuzzy_ctrlr.h"
#include "matrix.h"
#include "slip.h"

```

```

#include "talk2matlab.h"

int talk2matlab(const float ro, const double *rot_axis, const float \
rot_amt, int type, int object)
{
    int i, j, k, kb = 1, joint, board_num, robot_num, times;
    long int final_counts[6] = {0,0,0,0,0,0};
    long int robot_enc_cts[NUM_ROBOTS][6] = {{0,0,0,0,0,0}};
    long int encoder[256];
    float current_rot = 0;
    double sensor_radius = 3.0/16.0;
    double robot_anin[4][6]={{0,0,0,0,0,0}};
    double uf[NUM_ROBOTS] = {0,0,0,0};
    double vf[NUM_ROBOTS] = {0,0,0,0};
    double sum_force[NUM_ROBOTS] = {0,0,0,0};
    double numu[NUM_ROBOTS] = {0,0,0,0};
    double numv[4] = {0,0,0,0};
    double du[6] = {sensor_radius, sensor_radius, -sensor_radius, \
-sensor_radius, -sensor_radius, sensor_radius};
    double dv[6] = {0, -2*sensor_radius, -2*sensor_radius, 0, \
2*sensor_radius, 2*sensor_radius};
    short int anin[256];
    double po[3] = {47, 47, 9+ro};
    double ks[3] = {0,0,0};
    double Rgen[3][3] = {{0,0,0}}, Rdot[3][3] = {{0,0,0}}, \
Vpos[6] = {0,0,0,0,0,0};

    double vth = 1 - cos(THF);
    double thdot = THF/ROT_T;
    FILE *ofp;

    /* determine general object twist */

    for (i = 0; i < 3; i++)
ks[i] = rot_axis[i]/sqrt(dotprod(rot_axis,rot_axis,3));

    /* calculate general rotation matrix */

    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
        if (i == j)
Rgen[i][j] = ks[i]*ks[j]*vth + cos(THF);
        else
Rgen[i][j] = ks[i]*ks[j]*vth;
    }
}

    Rgen[0][1] += -ks[2]*sin(THF);
    Rgen[0][2] += ks[1]*sin(THF);
    Rgen[1][0] += ks[2]*sin(THF);
    Rgen[1][2] += -ks[1]*sin(THF);
    Rgen[2][0] = ks[0]*ks[2]*vth - ks[1]*sin(THF);
    Rgen[2][1] = ks[1]*ks[2]*vth + ks[0]*sin(THF);

    skewsm(ks,*Rdot);

    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
Rdot[i][j] = thdot * Rdot[i][j];
}

    /* spatial velocity of object wrt palm */
    for (i = 0; i < 3; i++)
Vpos[0] += -Rdot[0][i] * po[i];
    for (i = 0; i < 3; i++)
Vpos[1] += -Rdot[1][i] * po[i];
    for (i = 0; i < 3; i++)
Vpos[2] += -Rdot[2][i] * po[i];

    Vpos[3] = Rdot[2][1];
    Vpos[4] = Rdot[0][2];
    Vpos[5] = Rdot[1][0];

    while (rot_amt - 1 > current_rot)
{
    i = 0;
    k = 0;
    for (board_num = 1; board_num <= 3; board_num++)
{
        printf("\n");
        ReturnAnlg(anin, board_num);

        /* if force readings < 0, set them to 0 */
        for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

        for (j = 0; j < 8; j++)
{

```

```

    if (j == 0)
robot_anin[k][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 1)
robot_anin[k][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 2)
robot_anin[k+1][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 3)
robot_anin[k+1][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 4)
robot_anin[k+2][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 5)
robot_anin[k+2][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 6)
robot_anin[k+3][i] = anin[33+14*j]* \
ANALOG_RES;
    else
robot_anin[k+3][i+1] = anin[33+14*j]* \
ANALOG_RES;
}
    i += 2;
}

/* calculate the centroid of the force readings and make
this the contact coordinate */

for (robot_num = 1; robot_num <= 4; robot_num++)
{
    numu[robot_num-1] = 0;
    numv[robot_num-1] = 0;
    sum_force[robot_num-1] = 0;

    for (i = 0; i < 6; i++)
    {
        numu[robot_num-1] += du[i]* \
robot_anin[robot_num-1][i];
        numv[robot_num-1] += dv[i]* \
robot_anin[robot_num-1][i];
        sum_force[robot_num-1] += \
robot_anin[robot_num-1][i];
    }

    uf[robot_num-1] = numu[robot_num-1] / \
sum_force[robot_num-1];
    vf[robot_num-1] = numv[robot_num-1] / \
sum_force[robot_num-1];
}

    i = 0;
    sleep(1);

    for (board_num = 1; board_num <= 3; board_num++)
    {
        ReturnEncoder(encoder,board_num);
        for (robot_num = 0; robot_num < 4; robot_num++)
        {
            if (board_num == 1)
joint = 0;
            else
if (board_num == 2)
joint = 1;
            else
joint = 2;
            for (k = 0; k < 2; k++)
            {
                robot_enc_cts[robot_num][joint] \
= encoder[12+7*i];
                i += 1;
                joint += 3;
            }
        }

        i = 0;
    }

    ofp = fopen("matlab_info.dat","w");
    fprintf(ofp,"% .4f\t%.4f\t%.4f\t%.4f\t%.4f\t% \
.4f\n",Vpos[0],Vpos[1],Vpos[2],Vpos[3],Vpos[4],Vpos[5]);
    fprintf(ofp,"% .4f\t%.4f\t%.4f\t%.4f\t%.4f\t% \
%.4f\n",uf[0],uf[1],uf[2],uf[3],0.0,0.0);
    fprintf(ofp,"% .4f\t%.4f\t%.4f\t%.4f\t%.4f\t% \
%.4f\n",vf[0],vf[1],vf[2],vf[3],0.0,0.0);

    for (robot_num = 0; robot_num < 4; robot_num++)
fprintf(ofp,"%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n", \
robot_enc_cts[robot_num][0],robot_enc_cts[robot_num][1], \
robot_enc_cts[robot_num][2],robot_enc_cts[robot_num][3], \

```

```

robot_enc_cts[robot_num][4],robot_enc_cts[robot_num][5]);

fclose(ofp);
printf("\nContact and joint information saved.\n");
printf("\nReady to rotate object ... \n");

printf("\nWaiting for new encoder count files,\n");
printf("enter 0 to continue ... ");

while (kb)
scanf("%d",&kb);
kb = 1;

/* rotate the object */

move_robot(5, final_counts);
sleep(1);
// slipp(type);
//printf("\n ... done.\n");

current_rot += THF*R2D;
printf("\n rotation = %f deg.\n",current_rot);

if (rot_amt - 1 > current_rot)
{
for (times = 0; times < 1; times++)
{
/* reconfigure fingers */

for (board_num = 1; board_num <= 3; \
board_num++)
{
printf("\n");
ReturnAnlg(anin, board_num);

/* if force readings < 0, set
them to 0 */
for (j = 0; j < 8; j++)
anin[33+14*j] = \
MAX(anin[33+14*j], 0);

for (j = 0; j < 8; j++)
{
if (j == 0)
robot_anin[k][i] = \
anin[33+14*j]*ANALOG_RES;
else if (j == 1)
robot_anin[k][i+1] \
= anin[33+14*j]*ANALOG_RES;
else if (j == 2)
robot_anin[k+1][i] \
= anin[33+14*j]*ANALOG_RES;
else if (j == 3)
robot_anin[k+1][i+1] \
= anin[33+14*j]*ANALOG_RES;
else if (j == 4)
robot_anin[k+2][i] \
= anin[33+14*j]*ANALOG_RES;
else if (j == 5)
robot_anin[k+2][i+1] \
= anin[33+14*j]*ANALOG_RES;
else if (j == 6)
robot_anin[k+3][i] \
= anin[33+14*j]*ANALOG_RES;
else
robot_anin[k+3][i+1] \
= anin[33+14*j]*ANALOG_RES;
}
i += 2;
}

/* calculate the centroid of the force
readings and make this the contact coordinate */

for (robot_num = 1; robot_num <= 4; \
robot_num++)
{
numu[robot_num-1] = 0;
numv[robot_num-1] = 0;
sum_force[robot_num-1] = 0;

for (i = 0; i < 6; i++)
{
numu[robot_num-1] += \
du[i]*robot_anin[robot_num-1][i];
numv[robot_num-1] += \
dv[i]*robot_anin[robot_num-1][i];
sum_force[robot_num-1] \
+= robot_anin[robot_num-1][i];
}
}
}

```

```

}
uf[robot_num-1] = \
numu[robot_num-1] / sum_force[robot_num-1];
vf[robot_num-1] = \
numv[robot_num-1] / sum_force[robot_num-1];
}

i = 0;
sleep(1);

for (board_num = 1; board_num <= 3; \
board_num++)
{
ReturnEncoder(encoder,board_num);
for (robot_num = 0; robot_num < \
4; robot_num++)
{
if (board_num == 1)
joint = 0;
else
if (board_num == 2)
joint = 1;
else
joint = 2;
for (k = 0; k < 2; k++)
{
\
robot_enc_cts[robot_num][joint] = encoder[12+7*i];
i += 1;
joint += 3;
}
}
i = 0;
}

k = 0;

/* during reconfiguration Vpos = 0 */

ofp = fopen("matlab_info.dat","w");
fprintf(ofp,"% .4lf\t%.4lf\t%.4lf\t \
%.4lf\t%.4lf\t%.4lf\n",0.0,0.0,0.0,0.0,0.0,0.0);
fprintf(ofp,"% .4lf\t%.4lf\t%.4lf\t \
%.4lf\t%.4lf\t%.4lf\n",uf[0],uf[1],uf[2],uf[3],0.0,0.0);
fprintf(ofp,"% .4lf\t%.4lf\t%.4lf\t \
%.4lf\t%.4lf\t%.4lf\n",vf[0],vf[1],vf[2],vf[3],0.0,0.0);
for (robot_num = 0; robot_num < 4; \
robot_num++)
fprintf(ofp,"%ld\t%ld\t%ld\t%ld\t \
%ld\t%ld\n",robot_enc_cts[robot_num][0],robot_enc_cts[robot_num][1], \
robot_enc_cts[robot_num][2],robot_enc_cts[robot_num][3], \
robot_enc_cts[robot_num][4],robot_enc_cts[robot_num][5]);
fprintf(ofp,"%f\t%.4lf\t%.4lf\t%.4lf \
\t%d\t%d\n", current_rot, ks[0], ks[1], ks[2], 0, 0);
fclose(ofp);

printf("\nContact and joint information \
saved.\n");
printf("\nReady to reposition fingers \
... \n");

printf("\nWaiting for new encoder count \
files,\n");
printf("enter 0 to continue \
... ");

while (kb)
scanf("%d",&kb);
kb = 1;

move_robot(5, final_counts);
sleep(1);
if (times == 1)
{
slip(type, object);
printf("\n ... done.\n");
}
}
}
}

return 0;
}

```

C.10 File slip.c

```
/* *****
*****
* Written by: Neil Petroff *
* Program: slip.c *
* Date Written: 23 JUL 2006 *
* Last Date Modified: 30 JUL 2006 *
* Written for: research *
*****
*****

This function checks the slip condition between an object and the
fingers, and makes adjustments based on a fuzzy controller output
for the finger's x-position. It is called from talk2matlab.c after
each object rotation and finger repositioning.

***** REVISION LOG *****
07/30/06: Changed code so average force values for each finger
are sent to the fuzzy controller instead of just the max for each
finger.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "acquire_object.h"
#include "move_robot.h"
#include "matrix.h"
#include "inverse_kinematics.h"
#include "puma.h"
#include "dmclinux.h"
#include "talk2matlab.h"
#include "fuzzy_ctrlr.h"
#include "move_options.h"
#include "slip.h"

#define NAN 2147483648UL
#define DOF 6

void slip(int type, int object)
{
    int robot_flag[4] = {1,1,1,0}; /* flag to stop robot 1 if
true, 0 if false */
    int i, j, k = 0, joint, robot_num, board_num;
    float dx = 0, xf[3] = {0,0,0}, xnew[3] = {0,0,0};
    long int robot_enc_cts[NUM_ROBOTS][6] = {{0,0,0,0,0,0}};
    double robot_anin[4][6]={{0,0,0,0,0,0}};
    long int encoder[256], final_counts[6] = {0,0,0,0,0,0};
    short int max_force[NUM_ROBOTS] = {0,0,0,0}, anin[256], new_max \
= 0;
    double sum, avg_force[4] = {0,0,0};
    double **gst, **Rd;
    double xpos[4] = {0,0,0,0}, ypos[4] = {0,0,0,0}, zpos[4] = \
{0,0,0,0};
    double current_jt_angles[6] = {0,0,0,0,0,0};
    double omegas[6][3] = {{0,0,1},{0,1,0},{0,-1,0},{0,0,1},\
{0,1,0},{-1,0,0}};
    double qs[6][3] = {{0,0,0},{0,0,10},{12, 0, 10},{12, 13-11, 0},\
{12, 0, 10-14},{0, 13-11, 10-14}};
    double gsto[4][4] = {{1,0,0,12+15},{0,1,0,13-11},{0,0,1,10-14},\
{0,0,0,1}};
    FILE *ofp;
    char file_name[20];

    gst = (double **) malloc((unsigned) 4*sizeof(double*));
    Rd = (double **) malloc((unsigned) 3*sizeof(double*));
    for (i = 0; i < 4; i++)
gst[i] = (double *) malloc((unsigned) 4*sizeof(double));
    for (i = 0; i < 3; i++)
Rd[i] = (double *) malloc((unsigned) 4*sizeof(double));

    printf("\nchecking slip condition ..\n\n");

    while (robot_flag[0] || robot_flag[1] || robot_flag[2] || \
robot_flag[3])
    {
        for (board_num = 1; board_num <= 3; board_num++)
        {
            ReturnAnlg(anin, board_num);

            /* if force readings < 0, set them to 0 */
            for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

            /* for (j = 0; j < 8; j++)
            {
                if (j == 0)

```

```

        robot_anin[k][i] = anin[33+14*j]*ANALOG_RES;
        else if (j == 1)
            robot_anin[k][i+1] = anin[33+14*j]*ANALOG_RES;
        else if (j == 2)
            robot_anin[k+1][i] = anin[33+14*j]*ANALOG_RES;
        else if (j == 3)
            robot_anin[k+1][i+1] = anin[33+14*j]*ANALOG_RES;
        else if (j == 4)
            robot_anin[k+2][i] = anin[33+14*j]*ANALOG_RES;
        else if (j == 5)
            robot_anin[k+2][i+1] = anin[33+14*j]*ANALOG_RES;
        else if (j == 6)
            robot_anin[k+3][i] = anin[33+14*j]*ANALOG_RES;
        else
            robot_anin[k+3][i+1] = anin[33+14*j]*ANALOG_RES;
        }
        i += 2; */

    for (robot_num = 1; robot_num <= 4; robot_num++)
    {
        if (robot_num == 1)
        {
            new_max = MAX(anin[117],anin[131]);
            if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
        }
        else if (robot_num == 2)
        {
            new_max = MAX(anin[61],anin[75]);
            if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
        }
        else if (robot_num == 3)
        {
            new_max = MAX(anin[89],anin[103]);
            if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
        }
        else if (robot_num == 4)
        {
            new_max = MAX(anin[33],anin[47]);
            if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
        }
    }

    sleep(1);
    i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
    {
        ReturnEncoder(encoder,board_num);
        for (robot_num = 0; robot_num < 4; robot_num++)
        {
            if (board_num == 1)
                joint = 0;
            else
                joint = 1;
            if (board_num == 2)
                joint = 1;
            else
                joint = 2;
            for (k = 0; k < 2; k++)
            {
                robot_enc_cts[robot_num][joint] \
= encoder[12+7*i];
                i += 1;
                joint += 3;
            }
        }
        i = 0;
    }

    for (i = 0; i < 4; i++)
    {
        if (robot_flag[i])
        {
            current_jt_angles[0] = \
robot_enc_cts[i][0]*COUNT2RAD1;
            current_jt_angles[1] = \
robot_enc_cts[i][1]*COUNT2RAD2;
            current_jt_angles[2] = \
robot_enc_cts[i][2]*COUNT2RAD3;

```

```

    current_jt_angles[3] = \
robot_enc_cts[i][3]*COUNT2RAD4;
    current_jt_angles[4] = \
robot_enc_cts[i][4]*COUNT2RAD5;
    current_jt_angles[5] = \
robot_enc_cts[i][5]*COUNT2RAD6;

    gst = forward_kinematics(&qs[0][0], \
&omegas[0][0],current_jt_angles,&gsto[0][0],gst);
    xpos[i] = gst[0][3];
    ypos[i] = gst[1][3];
    zpos[i] = gst[2][3];

    xf[0] = gst[0][3];
    xf[1] = gst[1][3];
    xf[2] = gst[2][3];

    for (j = 0; j < 3; j++)
    {
        for (k = 0; k < 3; k++)
        Rd[j][k] = gst[j][k];
    }

    dx = fuzzy_controller(max_force[i]* \
ANALOG_RES, xpos[i], object);
    xpos[i] += dx*gst[0][0];
    ypos[i] += dx*gst[1][0];
    zpos[i] += dx*gst[2][0];
    xnew[0] = xpos[i];
    xnew[1] = ypos[i];
    xnew[2] = zpos[i];

    manipulate(xf, xnew, Rd, i+1);
    inverse_kinematics(i+1, type, \
&robot_enc_cts[i][0]);
    printf("New trajectory for Robot #%d \
calculated.\n",i+1);
}
else
{
    sprintf(file_name,"robot%d_cts.dat", i+1);
    ofp = fopen(file_name, "w");
    fprintf(ofp,"0\t0\t0\t0\t0\t0\n");
    fprintf(ofp,"0\t0\t0\t0\t0\t0\n");
    fclose(ofp);
}
if (fabs(dx) <= TOL)
robot_flag[i] = 0;
}

/* for (robot_num = 1; robot_num <= 4; robot_num++)
printf("%.2f\t", max_force[robot_num - 1]*ANALOG_RES);
printf("\n"); */

move_robot(5, final_counts);

free(gst);
free(Rd);
}

```

C.11 File fuzzy_ctrl.c

```

/* *****
*****
* Written by: Neil Petroff *
* Program: fuzzy_ctrl.c *
* Date Written: 10 JUL 2006 *
* Last Date Modified: 14 AUG 2006 *
* Written for: research *
*****

```

This sub-program is called from main, and contains a fuzzy controller used to determine the local x-position change of the finger to acquire/maintain proper contact with an object. The inputs are object weight, object compliance index, and the current contact force. The output is the change in x-position of the fingertip. It is assumed that the rest of the desired configuration is constant.

```

***** REVISION LOG *****
07/10/06: new program.
07/13/06: changed inputs to function call. Currently, they are
current maximum force, and current x-position.
08/11/06: added object to function call. The range of the x-pos
variable has to change depending on the object type. Currently have
values for ball and cube. Also partially implemented automated

```

```

membership function calculations based on the range for an input
variable. Only did it for x-pos input since I've been tweaking this
a lot.
08/13/06: Changing range on x-position input to controller for the
ball. Seems the robots drop the ball a lot during slip check when
far from the zero position.
08/14/06: Can't strike a good balance with the finger positions
changing so much during manipulation. Either it squeezes the ball
too tight when it first grabs it, or it drops it during the slip
check later. Instead of changing the x-pos range again, I'm going
to try weighting the force input. Start with a weight of 1.5.
*/

/* Include Files */
#include <stdio.h>
#include <stdlib.h>
#include "fuzzy_ctrlr.h"

float fuzzy_controller(float contact_force, float current_x, const \
int object)
{
    int rules[NUM_RULES] [LEN]={0,0,4},{1,0,4},{2,0,4},{3,0,3}, \
{4,0,2},{0,1,4},{1,1,4},{2,1,3},{3,1,2},{4,1,1},{0,2,4},{1,2,3}, \
{2,2,2},{3,2,1},{4,2,0},{0,3,3},{1,3,2},{2,3,1},{3,3,0},{4,3,0}, \
{0,4,2},{1,4,1},{2,4,0},{3,4,0},{4,4,0}};
    int i, j;
    float mu_force = 0.0, mu_xpos = 0.0;
    float dx = 0.0, implication, intersect = 0.0, area = 0.0;
    float numerator = 0.0, denominator = 0.0, watot = 0.0, atot = 0.0;
    float peak_ball[3][5] = {{0,1,2,3,4},{0,0,0,0,0}, \
{-1,-0.5,0,0.5,1}};
    float span_ball[3][5] = {{1,2,2,2,1},{0,0,0,0,0}, \
{0.5,1,1,1,0.5}};
    float peak_cube[3][5] = {{0,1.25,2.5,3.75,5},{0,0,0,0,0}, \
{-1,-0.5,0,0.5,1}};
    float span_cube[3][5] = {{1.25,2.5,2.5,2.5,1.25},{0,0,0,0,0}, \
{0.5,1,1,1,0.5}};
    float range_ball[2] = {28, 31};
    float range_cube[2] = {29, 31};
    float range = 0, wgt = 1.0;
    float peak[3][5] = {{0,0,0,0,0}};
    float span[3][5] = {{0,0,0,0,0}};

    /* the rule vectors can take on values from 0-4 which represent
    linguistic variables LN, N, Z, P, and LP, respectively. The first 2
    are for the input variables, contact force and current x position,
    and the last element is for the output variable dx.*/

    if (object == 0)
    {
        range = range_ball[1] - range_ball[0];
        peak_ball[1][0] = range_ball[0];
        peak_ball[1][4] = range_ball[1];
        span_ball[1][0] = range/4.;
        span_ball[1][4] = range/4.;
        for (i = 1; i < 4; i++)
        {
            peak_ball[1][i] = range_ball[0] + ((i * range)/4.);
            span_ball[1][i] = range / 2.;
        }
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 5; j++)
            {
                peak[i][j] = peak_ball[i][j];
                span[i][j] = span_ball[i][j];
            }
        }
    }
    else if (object == 2)
    {
        range = range_cube[1] - range_cube[0];
        peak_cube[1][0] = range_cube[0];
        peak_cube[1][4] = range_cube[1];
        span_cube[1][0] = range/4.;
        span_cube[1][4] = range/4.;
        for (i = 1; i < 4; i++)
        {
            peak_cube[1][i] = range_cube[0] + ((i * \
range)/4.);
            span_cube[1][i] = range / 2.;
        }
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 5; j++)
            {
                peak[i][j] = peak_cube[i][j];
                span[i][j] = span_cube[i][j];
            }
        }
    }
}

```

```

}
}
else
{
printf("\nNo controller specified for selected object.\n");
exit(EXIT_FAILURE);
}

/* determine the membership value of the contact force in the
linguistic variable maxforce */

printf("\nf\tf\t",contact_force, current_x);

for (i = 0; i < NUM_RULES; i++)
{
if (rules[i][0] == 0) /* if input1 is LN */
{
if (contact_force < peak[0][0])
mu_force = 1.0;
else if (contact_force >= peak[0][0] && \
contact_force <= (peak[0][0] + span[0][0]))
mu_force = (1./span[0][0])*(peak[0][0] - \
contact_force) + 1;
else if (contact_force > (peak[0][0]+span[0][0]))
mu_force = 0.;
}
else if (rules[i][0] == 1) /* if input1 is N */
{
if (contact_force < (peak[0][1]-span[0][1]/2.) \
|| contact_force > (peak[0][1]+span[0][1]/2.))
mu_force = 0.;
else if (contact_force >= peak[0][1]- \
span[0][1]/2. && contact_force <= peak[0][1])
mu_force = (2/span[0][1])*(contact_force - \
peak[0][1]) + 1;
else if (contact_force > peak[0][1] && \
contact_force <= peak[0][1] + span[0][1]/2.)
mu_force = (2/span[0][1])*(-contact_force + \
peak[0][1]) + 1;
}
else if (rules[i][0] == 2) /* if input1 is zero */
{
if (contact_force < (peak[0][2]-span[0][2]/2.) \
|| contact_force > (peak[0][2]+span[0][2]/2.))
mu_force = 0.;
else if (contact_force >= peak[0][2]- \
span[0][2]/2. && contact_force <= peak[0][2])
mu_force = (2/span[0][2])*(contact_force - \
peak[0][2]) + 1;
else if (contact_force > peak[0][2] && \
contact_force <= peak[0][2] + span[0][2]/2.)
mu_force = (2/span[0][2])*(-contact_force + \
peak[0][2]) + 1;
}
else if (rules[i][0] == 3) /* if input1 is P */
{
if (contact_force < (peak[0][3]-span[0][3]/2.) \
|| contact_force > (peak[0][3]+span[0][3]/2.))
mu_force = 0.;
else if (contact_force >= peak[0][3]- \
span[0][3]/2. && contact_force <= peak[0][3])
mu_force = (2/span[0][3])*(contact_force - \
peak[0][3]) + 1;
else if (contact_force > peak[0][3] && \
contact_force <= peak[0][3] + span[0][3]/2.)
mu_force = (2/span[0][3])*(-contact_force + \
peak[0][3]) + 1;
}
else if (rules[i][0] == 4) /* if input1 is LP */
{
if (contact_force < peak[0][4]-span[0][4])
mu_force = 0.;
else if (contact_force >= peak[0][4]-span[0][4] \
&& contact_force <= peak[0][4])
mu_force = (1/span[0][4])*(contact_force - \
peak[0][4]) + 1;
else if (contact_force > peak[0][4])
mu_force = 1.0;
}
/* determine the membership value of x-position in
linguistic variable currentx */
if (rules[i][1] == 0) /* if input2 is LN */
{
if (current_x < peak[1][0])
mu_xpos = 1.0;
else if (current_x >= peak[1][0] && current_x <= \
(peak[1][0] + span[1][0]))
mu_xpos = (1./span[1][0])*(peak[1][0] - \
current_x) + 1;
}
}
}
}

```

```

else if (current_x > (peak[1][0]+span[1][0]))
    mu_xpos = 0.;
}
else if (rules[i][1] == 1) /* if input2 is N */
{
if (current_x < (peak[1][1]-span[1][1]/2.) || \
current_x > (peak[1][1]+span[1][1]/2.))
    mu_xpos = 0.;
else if (current_x >= peak[1][1]-span[1][1]/2. \
&& current_x <= peak[1][1])
    mu_xpos = (2/span[1][1])*(current_x - \
peak[1][1]) + 1;
else if (current_x > peak[1][1] && current_x \
<= peak[1][1] + span[1][1]/2.)
    mu_xpos = (2/span[1][1])*(-current_x + \
peak[1][1]) + 1;
}
else if (rules[i][1] == 2) /* if input2 is zero */
{
if (current_x < (peak[1][2]-span[1][2]/2.) || \
current_x > (peak[1][2]+span[1][2]/2.))
    mu_xpos = 0.;
else if (current_x >= peak[1][2]-span[1][2]/2. \
&& current_x <= peak[1][2])
    mu_xpos = (2/span[1][2])*(current_x - \
peak[1][2]) + 1;
else if (current_x > peak[1][2] && current_x \
<= peak[1][2] + span[1][2]/2.)
    mu_xpos = (2/span[1][2])*(-current_x + \
peak[1][2]) + 1;
}
else if (rules[i][1] == 3) /* if input2 is P */
{
if (current_x < (peak[1][3]-span[1][3]/2.) || \
current_x > (peak[1][3]+span[1][3]/2.))
    mu_xpos = 0.;
else if (current_x >= peak[1][3]-span[1][3]/2. \
&& current_x <= peak[1][3])
    mu_xpos = (2/span[1][3])*(current_x - \
peak[1][3]) + 1;
else if (current_x > peak[1][3] && current_x \
<= peak[1][3] + span[1][3]/2.)
    mu_xpos = (2/span[1][3])*(-current_x + \
peak[1][3]) + 1;
}
else if (rules[i][1] == 4) /* if input2 is LP */
{
if (current_x < peak[1][4]-span[1][4])
    mu_xpos = 0.;
else if (current_x >= peak[1][4]-span[1][4] && \
current_x <= peak[1][4])
    mu_xpos = (1/span[1][4])*(current_x - \
peak[1][4]) + 1;
else if (current_x > peak[1][4])
    mu_xpos = 1.0;
}

/* do the rule evaluation */
/* since each rule is an and, simply take the min
membership value */

implication = MIN(mu_force, mu_xpos);

/* now, apply these to each output to do the implication.
This gives a fuzzy output set for each rule */

/* determine fuzzy output membership function areas for
change in x */

if (implication != 0)
{
if (rules[i][2] == 0) /* then output is LN */
{
intersect = span[2][0]*(1 - implication);
area = (1./2.)*implication*(intersect + \
span[2][0]);
numerator = area * (peak[2][0] + \
(span[2][0]/3.));
/* numerator is area of cropped
fuzzy set */
denominator = area;
/* times center (peak) of original fuzzy
set - symmetry */
if (implication == mu_force)
{
numerator = numerator * wgt;
denominator = denominator * wgt;
}
}
}
}

```

```

else if (rules[i][2] == 1) /* then output is N */
{
intersect = span[2][1]*(1 - implication);
area = (1/2.)*implication*(intersect + \
span[2][1]);
numerator = area * peak[2][1];
denominator = area;
if (implication == mu_force)
{
numerator = numerator * wgt;
denominator = denominator * wgt;
}
}
else if (rules[i][2] == 2) /* then output is
zero */
{
intersect = span[2][2]*(1 - implication);
area = (1/2.)*implication*(intersect + \
span[2][2]);
numerator = area * peak[2][2];
denominator = area;
if (implication == mu_force)
{
numerator = numerator * wgt;
denominator = denominator * wgt;
}
}
else if (rules[i][2] == 3) /* then output is P */
{
intersect = span[2][3]*(1 - implication);
area = (1./2.)*implication*(intersect + \
span[2][3]);
numerator = area * peak[2][3];
denominator = area;
if (implication == mu_force)
{
numerator = numerator * wgt;
denominator = denominator * wgt;
}
}
else if (rules[i][2] == 4) /* then output is LP */
{
intersect = span[2][4]*(1 - implication);
area = (1./2.)*implication*(intersect + \
span[2][4]);
numerator = area * (peak[2][4] - \
(span[2][4]/3.));
denominator = area;
if (implication == mu_force)
{
numerator = numerator * wgt;
denominator = denominator * wgt;
}
}
watot += numerator;
atot += denominator;
}
}

/* calculate crisp output */
if (atot != 0.)
dx = watot / atot;
else
dx = 0.;

printf("%f\n",dx);

return dx;
}

```

APPENDIX D

A MATHEMATICA PROGRAM FOR DETERMINING THE INVOLUTIVE CLOSURE OF AN UNDERACTUATED SYSTEM

This appendix contains the code for a Mathematica[®] notebook to generate the involutive closure of an underactuated system. It uses the function `LieB` to calculate a Lie bracket `vout` given two existing vector fields `v1` and `v2` and a set of local coordinates `list`. It can then be used to verify that the set of vector fields generates the involutive closure for the system. The results shown are for a spherical object rolling on a flat plane given by Equation 5.2.

```
(* lie_bracket.nb
   Written by: Neil Petroff *)

g1={0,-Sec[uf],rf*Sin[si],rf*Cos[si],Tan[uf]};
g2={1,0,rf*Cos[si],-rf*Sin[si],0};
xs={uf,vf,uo,vo,si};
LieB[v1_,v2_,list_,vout_]:=
  Module[{jm1,jm2},
    Do[jm1=Table[D[v1[[i]],list[[j]]],{i,Length[v1]},
      {j,Length[list]},{i,Length[v1]},{j,Length[list]}];
    Do[jm2=Table[D[v2[[i]],list[[j]]],{i,Length[v2]},
      {j,Length[list]},{i,Length[v2]},{j,Length[list]}];
    vout=FullSimplify[jm2.v1-jm1.v2]
  LieB[g1,g2,xs,g3];
  LieB[g1,g3,xs,g4];
  LieB[g2,g3,xs,g5];
  closure=FullSimplify[Transpose[{g1,g2,g3,g4,g5}]];
  MatrixRank[closure]
```

5

MatrixForm[closure]

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ -\text{Sec}[uf] & 0 & \text{Sec}[uf] \text{Tan}[uf] & 0 & \text{Sec}[uf] (\text{Sec}[uf]^2 + \text{Tan}[uf]^2) \\ rf \text{Sin}[si] & rf \text{Cos}[si] & -rf \text{Sin}[si] \text{Tan}[uf] & rf \text{Cos}[si] & -2 rf \text{Sec}[uf]^2 \text{Sin}[si] \\ rf \text{Cos}[si] & -rf \text{Sin}[si] & -rf \text{Cos}[si] \text{Tan}[uf] & -rf \text{Sin}[si] & -2 rf \text{Cos}[si] \text{Sec}[uf]^2 \\ \text{Tan}[uf] & 0 & -\text{Sec}[uf]^2 & 0 & -2 \text{Sec}[uf]^2 \text{Tan}[uf] \end{pmatrix}$$

BIBLIOGRAPHY

- [1] P. E. Agre and D. Chapman, What are plans for? In P. Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 17–34, The MIT Press: Cambridge, MA, USA (1990).
- [2] J. S. Albus and A. M. Meystel, *Engineering of Mind: An Introduction to the Science of Intelligent systems*. John Wiley and Sons, Inc. (2001).
- [3] J. R. Anderson, M. V. Albert and J. M. Fincham, Tracing problem solving in real time: fMRI analysis of the subject-paced tower of Hanoi. *Journal of Cognitive Neuroscience*, 17: 1261–1274 (2005).
- [4] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere and Y. L. Qin, An integrated theory of the mind. *Psychological Review*, 111(4): 1036–1060 (2004).
- [5] Automated assembly device with remote center compliance: Compensator from ATI. webpage (January 31, 2006), http://www.atia.com/products/compliance/assembly_compliance_device.aspx.
- [6] E. T. Baumgartner and S. B. Skaar, An autonomous vision-based mobile robot. *IEEE Transactions on Automatic Control*, 39: 493–502 (March 1994).
- [7] R. A. Brooks, *Flesh and Machines : How Robots Will Change Us*. Pantheon Books (2002).
- [8] M. Buss and H. Hashimoto, Dextrous robot hand experiments. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1680–1686 (May 1995).
- [9] C. Cai and B. Roth, On the spatial motion of rigid bodies with point contact. In *Proceedings of the 1987 International IEEE Conference on Robotics and Automation*, pages 686–695, Raleigh, NC (1987).
- [10] W. T. Cerven and F. Bullo, On trajectory optimization for polynomial systems via series expansions. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 1, pages 772–777, Sydney, Australia (December 2000).
- [11] D. C. Chang and M. R. Cutkosky, Rolling with deformable fingertips. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 194–199 (1995).

- [12] W. Z. Chen, U. A. Korde and S. B. Skaar, Position control experiments using vision. *The International Journal of Robotics Research*, 13(3): 199–208 (June 1994).
- [13] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Addison-Wesley, second edition (1989).
- [14] J. DeSchutter and H. V. Brussel, Compliant robot motion i. a formalism or specifying compliant motion tasks. *The International Journal of Robotics Research*, 7(4): 3–17 (1988).
- [15] M. J. Er and Y. L. Sun, Hybrid fuzzy proportional-integral plus conventional derivative control of linear and nonlinear systems. *IEEE Transactions on Industrial Electronics*, 48(6): 1109–1117 (December 2001).
- [16] R. S. Fearing and J. M. Hollerbach, Basic solid mechanics for tactile sensing. *The International Journal of Robotics Research*, 4(3): 40–54 (1985).
- [17] J. M. Fincham, C. S. Carter, V. van Veen, V. A. Stenger and J. R. Anderson, Neural mechanisms of planning: A computational analysis using event-related fMRI. *Proceedings of the National Academy of Sciences*, 99(5): 3346–3351 (2002).
- [18] V. Goel and J. Grafman, Are the frontal lobes implicated in planning functions? interpreting data from the tower of Hanoi. *Neuropsychologia*, 33: 632–642 (1995).
- [19] J. W. Goodwine, *Control of Stratified Systems with Robotic Applications*. Ph.D. thesis, California Institute of Technology (1998).
- [20] I. A. Gravagne and I. D. Walker, Manipulability, force, and compliance analysis for planar continuum manipulators. *IEEE Transactions on Robotics and Automation*, 3(18): 263–273 (June 2002).
- [21] N. Gulley and R. J. S. Jang, *Fuzzy Logic Toolbox for use with Matlab[®]*. The MathWorks, Inc. (1995).
- [22] L. Han, Y. S. Guan, Z. X. Li, Q. Shi and J. C. Trinkle, Dextrous manipulation with rolling contacts. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 992–997 (1997).
- [23] K. Harada, M. Kaneko and T. Tsuji, Active force closure for multiple objects. *Journal of Robotic Systems*, 19(3): 133–141 (2002).
- [24] B. Hayes-Roth, K. Pflieger, P. Morignot, P. Lalanda and M. Balabanovic, 1993. available at <http://citeseer.ist.psu.edu/hayes-roth93plans.html> (1993).
- [25] R. Hermann, *Differential Geometry and the Calculus of Variations*. Academic Press (1968).
- [26] M. Hershkovitz, U. Tasch and M. Teboulle, Toward a formulation of the human grasping quality sense. *Journal of Robotic Systems*, 12(4): 249–256 (1995).

- [27] S. A. Huettel, A. W. Song and G. McCarthy, Decisions under uncertainty: Probabilistic context influences activation of prefrontal and parietal cortices. *Journal of Neuroscience*, 25(13): 3304–3311 (March 2005).
- [28] E. L. Ince, *Ordinary Differential Equations*. Dover Publications, Inc. (1956).
- [29] R. S. Johansson and G. Westling, Roles of glabrous skin receptors and sensorimotor memory in automatic control of precision grip when lifting rougher or more slippery objects. *Experiments in Brain Research*, 56: 550–564 (1984).
- [30] T. J. Koo, Stable model reference adaptive fuzzy control of a class of nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 9(4): 624–636 (August 2001).
- [31] B. Kosko, *Fuzzy Thinking: The New Science of Fuzzy Logic*. Hyperion (1993).
- [32] B. Kosko, *Fuzzy Engineering*. Prentice Hall (1997).
- [33] S. A. Kyle, Non-contact measurement for robot calibration. In R. Bernhardt and S. L. Albright, editors, *Robot Calibration*, pages 79–100, Chapman & Hall (1993).
- [34] G. Lafferriere and H. J. Sussmann, A differential geometric approach to motion planning. In X. Li and J. F. Canny, editors, *Nonholonomic Motion Planning*, pages 235–270, Kluwer (1993).
- [35] S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning. available at <http://msl.cs.uiuc.edu/rrt/papers.html>.
- [36] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at <http://msl.cs.uiuc.edu/planning/>) (2006).
- [37] S. J. Lederman and R. L. Klatzky, The intelligent hand: An experimental approach to human object recognition and implications for robotics and AI. *AI Magazine*, 15(1): 26–38 (1994).
- [38] J.-W. Li, H. Liu and H.-G. Cai, On computing three-finger force-closure grasps of 2-D and 3-D objects. *IEEE Transactions on Robotics and Automation*, 19(1): 155–161 (2003).
- [39] D. Liberzon and A. S. Morse, Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19(5): 59–70 (October 1999).
- [40] H.-O. Lim and K. Tanie, Human safety mechanisms of human-friendly robots: Passive viscoelastic trunk and passively movable base. *The International Journal of Robotics Research*, 19(4): 307–335 (April 2000).
- [41] H. Liu, T. Iberall and G. A. Bekey, The multi-dimensional quality of task requirements for dexterous robot hand control. In *Proceedings of the 1989 IEEE International Conference of Robotics Automation*, pages 452–457 (May 1989).
- [42] S. Liu and H. Asada, Transferring manipulative skills to robots: Representation and acquisition of tool manipulative skills using a process dynamic model. In *Modeling and Control of Compliant Rigid Motion Systems*, volume 31, ASME (1991).

- [43] M. T. Mason and J. J. Kenneth Salisbury, *Robot Hands and the Mechanics of Manipulation*. The MIT Press (1985).
- [44] A. M. Meystel and J. S. Albus, *Intelligent Systems : Architecture, Design, and Control*. John Wiley and Sons, Inc. (2002).
- [45] D. J. Montana, The kinematics of contact and grasp. *The International Journal of Robotics Research*, 7(3): 17–32 (1988).
- [46] D. J. Montana, The kinematics of contact with compliance. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, volume 2, pages 770–774 (1989).
- [47] R. M. Murray, Z. Li and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press (1994).
- [48] R. M. Murray and S. S. Sastry, Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38: 700–716 (May 1993).
- [49] C. Natale and L. Villani, Adaptive control of a robot manipulator in contact with a curved compliant surface. In *Proceedings of the American Control Conference*, pages 288–292, San Diego, California (June 1999).
- [50] S. D. Newman, P. A. Carpenter, S. Varma and M. A. Just, Frontal and parietal participation in problem solving in the tower of london: fMRI and computational modeling of planning and high-level perception. *Neuropsychologia*, 41: 1668–1682 (2003).
- [51] V.-D. Nguyen, Constructing force-closure grasps. *The International Journal of Robotics Research*, 7(3): 3–16 (1988).
- [52] B. E. Paden, *Kinematics and Control of Robot Manipulators*. Ph.D. thesis, University of California at Berkeley (1985).
- [53] K. Passino and S. Yurkovich, *Fuzzy Control*. Addison-Wesley (1998).
- [54] M. A. Peshkin, *Robotic Manipulation Strategies*. Prentice Hall (1990).
- [55] N. Petroff, A neural network for determining forward kinematics of a 6 degree of freedom robot. Technical report, University of Notre Dame (2000), AME 598, Applications of Artificial intelligence in engineering, Spring 2000.
- [56] N. Petroff, Nonorthogonal coordinate maps for robotic manipulation. Project report for AME 598, Solid Modeling.
- [57] H. E. Rauch, Autonomous control reconfiguration. *IEEE Control Systems Magazine*, 15(6): 37–48 (December 1995).
- [58] S. J. Remis and M. M. Stanisic, Design of a singularity-free articulated arm-subassembly. *Journal of Robotics and Automation*, 9(6): 816–824 (1994).
- [59] L. Reznik, *Fuzzy Controllers*. Newnes (1997).

- [60] Z. S. Roth, W. B. Mooring and B. Ravani, An overview of robot calibration. *IEEE Journal of Robotics and Automation*, RA-3(5): 377–384 (1987).
- [61] S. Sastry, *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag (1999).
- [62] J. M. Selig, *Geometrical Methods in Robotics*. Springer (1996).
- [63] K. B. Shimoga, Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3): 230–266 (June 1996).
- [64] M. R. Spiegel, *Mathematical Handbook of Formulas and Tables*. McGraw-Hill Book Company (1968).
- [65] L. A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge University press (1987).
- [66] H. J. Sussmann, A product expansion for the Chen series. In C. I. Byrnes and A. Lindquist, editors, *Theory and Applications of Nonlinear Control Systems*, pages 323–335, Elsevier Science (1986).
- [67] K. Tanaka, M. Iwasaki and H. O. Wang, Switching control of an R/C hovercraft: Stabilization and smooth switching. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, 31(6): 853–863 (2001).
- [68] K. S. Tang, K. F. Man, G. Chen and S. Kwong, An optimal fuzzy PID controller. *IEEE Transactions on Industrial Electronics*, 48(4): 757–765 (August 2001).
- [69] S. Tian-Soon, M. H. A. Jr. and L. Kah-Bin, A compliant end-effector coupling for vertical assembly: Design and evaluation. *Robotics and Computer-Integrated Manufacturing*, 13(1): 21–30 (March 1997).
- [70] J. Trinkle and R. Paul, Planning for dexterous manipulation with sliding contacts. *The International Journal of Robotics Research*, 9(3): 24–48 (1990).
- [71] Unimate PUMA mark II robot: 500 series equipment manual for VAL II and VAL PLUS operating systems.
- [72] V. S. Varadarajan, *The Selected Works of V. S. Varadarajan*. American Mathematical Society (1999).
- [73] A. H. Wallace, *Differential Topology: First Steps*. W. A. Benjamin, Inc. (1968).
- [74] J. Wang, S. J. Dodds and W. N. Bailey, Co-ordinated control of multiple robotic manipulators handling a common object — theory and experiments. *IEE Proceedings — Control Theory and Applications*, 144: 73–84 (January 1997).
- [75] J. R. Weeks, *The Shape of Space*. Marcel Dekker (2002).
- [76] Y. Wei, *Theoretical and Experimental Investigation of Stratified Robotic Finger Gaiting and Manipulation*. Ph.D. thesis, University of Notre Dame (2002).

- [77] J. M. Wiitala and M. M. Stanasic, Design of an overconstrained and dextrous spherical wrist. *Journal of Mechanical Design*, 122(1): 347–353 (2000).
- [78] D. Willis, *The Sand Dollar and the Slide Rule: Drawing Blueprints from Nature*. Addison-Wesley (1995).
- [79] A. M. Wing, P. Harggard and R. J. Flanagan, *Hand and Brain: The Neurophysiology and Psychology of Hand Movements*. Academic Press, Inc. (1996).
- [80] R. R. Yager and D. P. Filev, *Essentials of Fuzzy Modeling and Control*. John Wiley & Sons, Inc. (1994).
- [81] G. Yan, *Decomposable Closed-Form Inverse Kinematics for Reconfigurable Robots using Product-of-Exponentials Formula*. Master’s thesis, Nanyang Technological University (2000).
- [82] B. Yao and M. Tomizuka, Adaptive control of robot manipulators in constrained motion-controller design. *Journal of Dynamic Systems, Measurement, and Control*, 117: 320–328 (September 1995).
- [83] C. A. Yates, *Design of a Three-Fingered Gripper with the Capability of Manipulating an Object*. Master’s thesis, University of Florida (1999).
- [84] K.-Y. Young and C.-C. Fan, Control of voluntary limb movements by using a fuzzy system. In *Proceedings of the 32nd Conference on Decision and Control*, pages 1759–1764, IEEE (1993).
- [85] L. A. Zadeh, Fuzzy sets. *Information and Control*, 8: 338–353 (1965).
- [86] M. H. Zand, P. Torab and A. Bahri, Hybrid position/force control of a dexterous hand based on fuzzy control strategy. In *Proceedings of the 1997 IEEE 8th International Conference on Advanced Robotics*, pages 133–139 (July 7–9 1997).