15:59:38 OCA PAD INITIATION - PROJECT HEADER INFORMATION 04/12/96

Active

Project #: E-25-L31 Cost share #: Rev #: 0
Center # : 10/24-6-R8879-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: 49X-SU566V Mod #: INITIATION Document : CONT
Prime #: DE-AC05-96OR22464 Contract entity: GTRC

Subprojects ? : N CFDA:
Main project #: PE #:

Project unit: MECH ENGR Unit code: 02.010.126
Project director(s):
BOOK W J MECH ENGR (404)894-3247

Sponsor/division names: OAK RIDGE NAT'L LAB / MARTIN MARIETTA
Sponsor/division codes: 240 / 001

Award period: 960401 to 960930 (performance) 960930 (reports)

Sponsor amount	New this change	Total to date
Contract value	25,909.00	25,909.00
Funded	25,909.00	25,909.00
Cost sharing amount		0.00

Does subcontracting plan apply ?: N

Title: CONTROL METHODS FOR SEAMLESS TELEOPERATION MODE TRANSFER & EMERGENCY...

PROJECT ADMINISTRATION DATA

OCA contact: Anita D. Rowland 894-4820

Sponsor technical contact

REID L. KRESS, RPSD
(423)574-2468

OAK RIDGE NATIONAL LABORATORY
LOCKHEED MARTIN ENERGY SYSTEMS, INC.
P.O. BOX 2002
OAK RIDGE, TN 37831

Sponsor issuing office

JANICE E. JOHNSON
(423)576-4415

OAK RIDGE NATIONAL LABORATORY
LOCKHEED MARTIN ENERGY SYSTEMS, INC.
PROCUREMENT DEPARTMENT
P.O. BOX 2002
OAK RIDGE, TN 37831-6501

Security class (U,C,S,TS) : U ONR resident rep. is ACO (Y/N): N
Defense priority rating : supplemental sheet
Equipment title vests with: Sponsor X GIT
NONE ANTICIPATED
Administrative comments -
INITIATION OF FIVE-MONTH FIXED PRICE AGREEMENT.

4

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 11/08/96

Project No. E-25-L31 Center No. 10/24-6-R8879-0A0

Project Director BOOK W J School/Lab MECH ENGR

Sponsor OAK RIDGE NAT'L LAB/LOCKHEED-MARTIN

Contract/Grant No. 49X-SU566V Contract Entity GTRC

Prime Contract No. DE-AC05-96OR22464

Title CONTROL METHODS FOR SEAMLESS TELEOPERATION MODE TRANSFER & EMERGENCY...

Effective Completion Date 960930 (Performance) 960930 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory & Related Certificate	Y	
Classified Material Certificate	N	
Release and Assignment	N	
Other	N	

Comments

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other	N
	N

NOTE: Final Patent Questionnaire sent to PDPI.

CONTROL METHODS FOR SEAMLESS TELEOPERATION MODE TRANSFER AND EMERGENCY STOPS USING LONG REACH, FLEXIBLE MANIPULATORS

SOFTWARE LISTING

Lonnie J. Love

Submitted to:

Lockheed Martin Energy Systems, Inc.
Oak Ridge National Laboratory
Contract No. SU566-49
Wayne J. Book, Project Director

Period of Performance: 4/1/96 - 9/30/96

Contract Software Deliverable
(E25-L31)

The software developed under this contract has been accessed by electronic means over the Internet and thereby transmitted to the sponsor. Dr. Lonnie Love, who performed the major part of the work while at Georgia Tech, is currently a Post Doctoral Fellow at the sponsor and has thereby assisted in this process. In addition, a listing of the software is attached.

Software

ORNL.CPP - Path planner for triangle trajectory and transfer control

```
// Robtest.cpp
// 9-24-94
//
// Lonnie J. Love
//
// Uses classes Vector, Matrix, Estimator, Filter, and Robot to control
// a 2 dof robot Herb.

////////////////////////////////////////////////////////////////////////
// INCLUDED HEADER FILES
////////////////////////////////////////////////////////////////////////
//#include "envir_t.h"
#include "main.h"
#include "alarm.h"
#include "control2.h"
#include "ds1001_i.h"
#include "timer.h"
#include "display.hpp"
#include "filter.hpp"
#include "estimatr.hpp"
#include "robenvir.hpp"
#include "traj.hpp"
#include <conio.h>
#include <math.h>
////////////////////////////////////////////////////////////////////////
// GLOBAL INITIALIZATION OF OBJECTS
////////////////////////////////////////////////////////////////////////

// Parameters for Robot object Herb
RobotParms HerbParms[2];
void InitParms(void);

// Copy of robot object Herb used in Timer ISR
Robot manipulator(2);

Matrix RawData(4000,6);
Matrix Power(50,3);

Matrix Path(2,3);

hPORT p0;

extern int TRANSMIT_FLAG;
extern int TELEOP_FLAG;
extern int TELE_QUIT_FLAG;
extern int SET_TELE_PD_FLAG;
extern int SET_TELE_IS_FLAG;
extern int SET_TELE_MCF_FLAG;
extern int SAVE_DATA_FLAG;
////////////////////////////////////////////////////////////////////////
// MAIN PROGRAM
////////////////////////////////////////////////////////////////////////
void main(void)
{
	InitParms();
	Robot Herb(HerbParms,2);

	// Copy object Herb to object manipulator used in
	// Timer ISR
	manipulator=Herb;

	Herb.InitRobot();
```

```
	// Initialize Timer ISR
	Herb.InitISR();
	RobotDisplay myDisplay(Herb);
	myDisplay.run();

	// Perform any cleanup before exiting

	// Quit controller on VME
	c_putc(p0, 'Q');
	while(c_txcount(p0) < 1023) {};

	// Stop ISR
	Herb.CleanUpISR();

	clrscr();
	RawData.CPP2MAT("rawdata.mat");
	Power.CPP2MAT("power.mat");
}

////////////////////////////////////////////////////////////////////////////
// INITROBOT FROM ROBOT CLASS
////////////////////////////////////////////////////////////////////////////

void Robot::InitRobot(void)
{
	// Initialize Dual Port Ram
	unsigned board_addr;
	if (get_board_addr("DS1001.SET", &board_addr))
	{
	printf("Error reading DS1001.SET setup file !\n\a");
	exit(0);
	}
	if (DS1001_enable(board_addr))
	{
	printf("Error enabling DS1001 DSP-board !\n\a");
	exit(0);
	}

	// Initialize Serial Port
	p0 = u8250_init(COM2, 38400L, DATABITS8, PARITY_NONE, STOPBITS1);
	install_ipr(p0, RECEIVE, NULL, 1024);
	install_ipr(p0, TRANSMIT, NULL, 1024);
	install_isr(p0, 3, (PIHANDLER) NULL);

	// Flush I/O on Serial ports
	do{}
	while(c_rxflush(p0,0) !=0);

	do{}
	while(c_txflush(p0,0) !=0);

	DS1001_write_dual_port_memory(0x30,False);

}

////////////////////////////////////////////////////////////////////////////
// TIMER ISR FROM ROBOT CLASS
////////////////////////////////////////////////////////////////////////////
// Sample time set in main.h

int DataCount=0;
int DataMax=4000;
float MasterPower=0;
float SlavePower=0;
int TaskIteration=0;

extern hPORT p0;
extern char *SaveStat;
extern char *FrcFlg;
```

```
int data1;
int data2;
unsigned int tempold;
int GRIP_FLAG, PREV_GRIP_FLAG, FORCE_FLAG;
int TRAJ_FLAG=False;
//extern Matrix Path;

void Robot::control ()
{
	static RowVector XYMasterFrc(2);
	static RowVector XYSlaveFrc(2);
	static RowVector XYMasterPos(2);
	static RowVector XYSlavePos(2);
	static RowVector XYMasterVel(2);

	static char chr[]="C";
	static char ch[]=" ";

	#define lobyte 0x003F
	static double tempg;
	static unsigned char string1[]="  ", string2[]="  ";
	static unsigned char force[]="      ";

	static unsigned long	RALF_Q1_ADR = 0X11,
						RALF_Q2_ADR = 0X12,
						RALF_GRIPPER =0x06,
						RALF_S1 = 0X16,
						RALF_S2 = 0X17,
						RALF_S3 = 0X18,
						TRAJ_X  = 0x1A,
						TRAJ_Y  = 0x1B;

	static unsigned int tempq1,tempq2,tempq3,tempui,biflag;
	static double deg2num1 = 0.304;
	static double deg2num2 = 0.271;

	static RowVector x1(2);		// Initial position
	static RowVector x2(2);		// Target Position
	static RowVector x(2);		// Intermediate position vector

	static Trajectory Line(2);

	static int LineIndex;
	static int PointIndex;
	static int NumPoints;
	static int PATH_FLAG=True;

	static double V=1/7; // target velocity (w.r.t. Herb's coordinates
	static double a=5/7; // target acceleration
	static double Ts=0.01;

// Begin Timer ISR

	// This flag dictates what type of target impedance to use
	DS1001_write_dual_port_memory(0x31,TRAJ_FLAG);

	if(TELE_QUIT_FLAG == True)
	{
		biflag=False;
		DS1001_write_dual_port_memory(0x30,biflag);

		c_putc(p0, 'Q');
		while(c_txcount(p0) < 1023) {};
		TELE_QUIT_FLAG = False;
		TRANSMIT_FLAG = False;
		TELEOP_FLAG=False;
	}
```

```
if(TELEOP_FLAG == True)
{
    if(TRAJ_FLAG == True)
    {

        if(PATH_FLAG == True)
        {
            for(int j=0;j<2;j++)
            {
                x1[j]=Path[LineIndex][j];          // Pull initial point from path
                x2[j]=Path[LineIndex+1][j];        // Pull final point from path
            }

            Line.Reset(x1, x2, V, a, Ts);     // Object creation
            NumPoints=Line.GetT();
            PointIndex=0;
            PATH_FLAG=False;
        }

        x=Line.Update(PointIndex);
        PointIndex++;              // Current position

        if(PointIndex==NumPoints-1)
        {
            LineIndex++;
            PATH_FLAG=True;
        }

        if(LineIndex>Path.GetColLength())
            TRAJ_FLAG=False;
    }

    tempui=int(x[0]*32757);
    DS1001_write_dual_port_memory (TRAJ_X, tempui);

    tempui=int(x[1]*32757);
    DS1001_write_dual_port_memory (TRAJ_Y, tempui);

    PREV_GRIP_FLAG=GRIP_FLAG;

    biflag=True;
    DS1001_write_dual_port_memory(0x30,biflag);
    // While waiting for force data to come, get command joint angles
    DS1001_read_dual_port_memory (RALF_Q1_ADR, &tempq1);
    DS1001_read_dual_port_memory (RALF_Q2_ADR, &tempq2);
    DS1001_read_dual_port_memory (RALF_GRIPPER, &tempq3);

    if(tempq1 < 6553) //35.7
        tempq1=6553;
    if(tempq1 > 20024) //110.7
        tempq1=20024;
    data1 = int( deg2num1*(tempq1-6553) );

    if(tempq2 < 4369) //23.4
        tempq2=4369;
    if(tempq2 > 19478) //107.7
        tempq2=19478;
    data2 = int( deg2num2*(tempq2-4369) );

    if(tempq3 > 32768)
        tempg = -((float)(~(tempq3-1)) );
    else
        tempg = (float)(tempq3);

    tempg=tempg/32768;

        if(tempg > .25)
            GRIP_FLAG=True;
        else
```

```
        GRIP_FLAG=False;

    if(GRIP_FLAG != PREV_GRIP_FLAG)
        TaskIteration++;


    // Convert to string of chars
string1[0]=(unsigned char)( (data1>>6) & lobyte);
string2[0]=(unsigned char)( (data2>>6) & lobyte);
string1[1]=(unsigned char)(data1&lobyte);
string2[1]=(unsigned char)(data2&lobyte);
string1[2]=string1[0]+string1[1];
string2[2]=string2[0]+string2[1];

// Blast position data up to VME
chr[0] = 'D';
c_putc(p0, chr[0]);
c_putc(p0, string1[0]);
c_putc(p0, string1[1]);
c_putc(p0, string1[2]);

c_putc(p0, string2[0]);
c_putc(p0, string2[1]);
c_putc(p0, string2[2]);


// Blast 'F' to VME to request latest force data
chr[0] = 'F';
c_putc(p0, chr[0]);

// Get Force Data
chr[0]=c_getc(p0);
if(chr[0]=='F')
{
    FORCE_FLAG=True;
    FrcFlg="True";
    force[1]=c_getc(p0);
    force[2]=c_getc(p0);
    force[3]=c_getc(p0);
    force[4]=c_getc(p0);
    force[5]=c_getc(p0);
    force[6]=c_getc(p0);
    force[7]=c_getc(p0);
    force[8]=c_getc(p0);
    force[9]=c_getc(p0);

    // Error Checking.  If good, update current force data,
    // If there is an error, don't update force value.
    // Shift an extra 4 bits to convert to 16 bit number for DSP
    if((force[1]+force[2])==force[3])
    {
        tempui=(force[1]<<10)+(force[2]<<4);
        DS1001_write_dual_port_memory (RALF_S1, tempui);
    }
    else
        FORCE_FLAG=False;

    if((force[4]+force[5])==force[6])
    {
        tempui=(force[4]<<10)+(force[5]<<4);
        DS1001_write_dual_port_memory (RALF_S2, tempui);
    }
    else
        FORCE_FLAG=False;

    if((force[7]+force[8])==force[9])
    {
        tempui=(force[7]<<10)+(force[8]<<4);
        DS1001_write_dual_port_memory (RALF_S3, tempui);
```

```
            }
            else
            {
                FORCE_FLAG=False;
                FrcFlg="False";
            }
            if(FORCE_FLAG == False)
                c_rxflush(p0, 0);

        }

        XYMasterFrc=GetTipForce();
        XYSlaveFrc=GetSlaveTipForce();
        XYMasterPos=GetTipPosition();
        XYSlavePos=7.0*XYMasterPos;
        XYMasterVel=GetTipVelocity();

        if(SAVE_DATA_FLAG == True)
        {
            RawData[DataCount][0]=XYSlavePos[0];
            RawData[DataCount][1]=XYSlavePos[1];
            RawData[DataCount][2]=XYSlaveFrc[0];
            RawData[DataCount][3]=XYSlaveFrc[1];
            RawData[DataCount][4]=XYMasterFrc[0];
            RawData[DataCount][5]=XYMasterFrc[1];
            DataCount++;
        }

        if(DataCount >= DataMax)
        {
            SAVE_DATA_FLAG = False;
            SaveStat="Off ";
        }

    }

    // Set controller, begins running next sample period
    if(SET_TELE_PD_FLAG == True)
    {
        ch[0]='S';
        ch[1]='1';
        c_putc(p0, ch[0]);
        c_putc(p0, ch[1]);
        while(c_txcount(p0) < 1023) {};
        SET_TELE_PD_FLAG = False;
        TRANSMIT_FLAG = True;
        TELEOP_FLAG=True;
    }

    if(SET_TELE_IS_FLAG == True)
    {
        ch[0]='S';
        ch[1]='2';
        c_putc(p0, ch[0]);
        c_putc(p0, ch[1]);
        while(c_txcount(p0) < 1023) {};
        SET_TELE_IS_FLAG = False;
        TRANSMIT_FLAG = True;
        TELEOP_FLAG=True;
    }

    if(SET_TELE_MCF_FLAG == True)
    {
        ch[0]='S';
        ch[1]='3';
        c_putc(p0, ch[0]);
        c_putc(p0, ch[1]);
        while(c_txcount(p0) < 1023) {};
        SET_TELE_MCF_FLAG = False;
```

```
            TRANSMIT_FLAG = True;
            TELEOP_FLAG=True;
        }

        return;
}


//////////////////////////////////////////////////////////////////////////
// INITIALIZE ROBOTPARAMS STRUCTURE FOR HERB OBJECT
//////////////////////////////////////////////////////////////////////////
void InitParms(void)
{
        float n1=60,n2=20;

        Path[0][0]=0.25;
        Path[0][1]=0.55;

        Path[1][0]=0.25;
        Path[1][1]=0.75;

        Path[2][0]=0.25;
        Path[2][1]=0.55;

        HerbParms[0].JtFrcAddr=0x07;
        HerbParms[1].JtFrcAddr=0x08;
        HerbParms[0].JtPosAddr=0x02;
        HerbParms[1].JtPosAddr=0x03;
        HerbParms[0].JtVelAddr=0x04;
        HerbParms[1].JtVelAddr=0x05;
        HerbParms[0].TipPosAddr=0x09;
        HerbParms[1].TipPosAddr=0x0A;
        HerbParms[0].TipFrcAddr=0x0B;
        HerbParms[1].TipFrcAddr=0x0C;
        HerbParms[0].TipVelAddr=0x0D;
        HerbParms[1].TipVelAddr=0x0E;

        HerbParms[0].JtFrcGain=11.179*n1;
        HerbParms[1].JtFrcGain=4.815*n2;
        HerbParms[0].JtPosGain=82.355/n1;
        HerbParms[1].JtPosGain=82.355/n2;
        HerbParms[0].JtVelGain=349.08/n1;
        HerbParms[1].JtVelGain=349.08/n2;
        HerbParms[0].TipPosGain=1;
        HerbParms[1].TipPosGain=1;
        HerbParms[0].DesTipPosGain=1;
        HerbParms[1].DesTipPosGain=1;
        HerbParms[0].TipFrcGain=466.55;
        HerbParms[1].TipFrcGain=466.55;
        HerbParms[0].TipVelGain=349.08/n2;
        HerbParms[1].TipVelGain=349.08/n2;
}
```

Traj.cpp Trajectory class

```
// Traj.cpp
// --------
//
//=========================================================================
//
// Description:
// ------------
//      A 5th order straight line trajectory generator for multiple degree
//      of freedom robots.  Provides ramp increase on acceleration profile.
//      Path provides zero initial and final velocity.  Variables passed to
```

```
//      the path include desired velocity and maximum accleration, as well as
//      sampling rate.  Constructor        computes the total path time as well as
//      the time required for acceleration and deceleration.  In addition,
//      the constructor provides the parameters required for the geteration
//      of the path.  The actual computation of the path is performed during
//      run time.  This increases, slightly, the computational burden of the
//      computer, but adds flexibility to the path if an obstacle or external
//      stimuli is provided to robot.  The "update" method is passed an index
//      which corresponds to the time along the path.  It returns the resulting
//      desired position in the robot's cartesian workspace.
//=====================================================================
//
// Copyright (c) 1995 by Lonnie J. Love
// All rights reserved.
//=====================================================================
//
// modification history
// --------------------
// 01a, 19aug95, ljl     First release
// 02a, 06sep95, ljl  Moved basic trajectory concept to a base class
//                                                    Now have MinTime and MinJerk trajectory classes
//                                                    that inherit basic trajectory methods.
//=====================================================================
//
// Includes
#include "math.h"
#include "matrix.hpp"
#include "traj.hpp"

//=====================================================================
// Base Class - Trajectory
//
// Purpose : Pass current position, desired position, velocity, and
//         max acceleration along with sample rate.  This constructor
//                         computes the parameters that go with a basic three phase
//         trajectory.  Each phase of the trajectory is computed using
//         the model:
//                                                d(t)=a+bt+ct^2+dt^3+et^4
//
// Methods:                Reset                - permits modification of parameters of path in run
//                         time.  This is beneficial to emergency stop, hybrid
//                         teleoperation/automonous commands...
//          Update                - Returns a position vector (in terms of the cartesian
//                         coordinate system) with respect to a time index.
//          GetVelocity - Similar to update, but returns scalar velocity
//
//=========================================================================
Trajectory::Trajectory()
{
     DOF=1;
     Amax=1;
     Vd=1;
     SampleTime=1;
};

Trajectory::Trajectory(RowVector X1, RowVector X2, double V, double A,
     double Ts) : Xtraj(X1.GetVecSize()), m(X1.GetVecSize()), Xhome(X1.GetVecSize())
{

     // Make sure dof is compatable
     if(X1.GetVecSize() != X2.GetVecSize())
          printf("Incompatable Desired Points");

     Xhome=X1;                    // Call first vector "home"

     DOF=X1.GetVecSize();      // Get Degrees of Freedom
     Amax=A;              // Maximum Acceleration
     SampleTime=Ts;         // Sample rate
     D=0;               // Compute distance of path
```

```
    for(int i=0;i<DOF;i++)
        D=D+(X2[i]-X1[i])*(X2[i]-X1[i]);
    D=sqrt(D);

    for(i=0;i<DOF;i++)     // Compute slopes to project path to cart. coord.
        m[i]=(X2[i]-X1[i])/D;

    if(D > V*V/Amax)
    {
        Vd=V;               // Desired Velocity
        T1=Vd/Amax;     // Compute time required for accel. and decell.
        T1i=int(T1/Ts);
        T=T1+D/Vd;            // Compute total time to finish path
        Ti=int(T/Ts);
    }
    else
    {
        Vd=sqrt(D*Amax);
        T1=Vd/Amax;     // Compute time required for accel. and decell.
        T1i=int(T1/Ts);
        T=2*T1;
        Ti=int(T/Ts);
    }

    tau=T-T1;
    a1=0;
    b1=0;
    c1=Amax/2;
    d1=0;
    a2=V*V/(2*Amax);
    b2=Vd;
    c2=0;
    d2=0;
    a3=Amax*(T-T1);
    b3=Vd;
    c3=-Amax/2;
    d3=0;

    va1=0;
    vb1=Amax;
    vc1=0;
    vd1=0;
    va2=Vd;
    vb2=0;
    vc2=0;
    vd2=0;
    va3=Vd+Amax*(T-T1);
    vb3=-Vd;
    vc3=0;
    vd3=0;
};
// Copy constructor
Trajectory::Trajectory(const Trajectory &tmpTraj)
{
    DOF=tmpTraj.DOF;
    Amax=tmpTraj.Amax;
    Vd=tmpTraj.Vd;
    D=tmpTraj.D;
    T=tmpTraj.T;
    T1=tmpTraj.T1;
    Ti=tmpTraj.T;
    T1i=tmpTraj.T1;
    SampleTime=tmpTraj.SampleTime;
//  Xtraj=tmpTraj.Xtraj;
};

// Constructor
Trajectory::Trajectory(int dof) : Xtraj(dof), m(dof), Xhome(dof)
{
```

```
};


// Update method
RowVector Trajectory::Update(int index)
{
	static double t;   // time
	static double t3;   // time cubed

	t=index*SampleTime;

	// Computation of path, d(t)
	if( (index <= T1i) & (index >=0) )		// Check to see if accelerating
	{
		t3=t*t*t;
		d=a1+b1*t+c1*t*t+d1*t3+e1*t3*t;
	}

	else if( (index > T1i) & (index < Ti-T1i) )	// Check to see if holding velocity
		d=a2+b2*t+c2*t*t+d2*t*t*t+e2*t*t*t*t;

	else if( (index >= Ti-T1i) & (index <= Ti) )	// Check to see if decelerating
	{
		t3=t*t*t;
		d=a3+b3*t+c3*t*t+d3*t3+e3*t3*t;
	}

	else			// Check to see if an error has occured
		printf("Index is out of Range");

	// Project to cartesian coordinate system
	// The vector of slopes, m, is computed in the constructor
	for(int i=0;i<DOF;i++)
		Xtraj[i]=Xhome[i]+d*m[i];

	// Return the current desired tip position
	return(Xtraj);
};

double Trajectory::GetVelocity(int index)
{
	static double t;   // time
	static double t3;   // time cubed
	static double vel;

	t=index*SampleTime;

	// Computation of path, d(t)
	if( (index <= T1i) & (index >=0) )		// Check to see if accelerating
	{
		t3=t*t*t;
		vel=va1+vb1*t+vc1*t*t+vd1*t3;
	}

	else if( (index > T1i) & (index < Ti-T1i) )	// Check to see if holding velocity
		vel=va2+vb2*t+vc2*t*t+vd2*t*t*t;

	else if( (index >= Ti-T1i) & (index <= Ti) )	// Check to see if decelerating
	{
		t3=t*t*t;
		vel=va3+vb3*t+vc3*t*t+vd3*t3;
	}

	else			// Check to see if an error has occured
		printf("Index is out of Range");

	return(vel);
};
```

```
double Trajectory::GetAcceleration(int index)
{
	static double t;	// time
	static double acc;

	t=index*SampleTime;

	// Computation of path, d(t)
	if( (index <= T1i) & (index >=0) )		// Check to see if accelerating
	{
		acc=aa1+ab1*t+ac1*t*t;
	}

	else if( (index > T1i) & (index < Ti-T1i) )	// Check to see if holding velocity
		acc=aa2+ab2*t+ac2*t*t;

	else if( (index >= Ti-T1i) & (index <= Ti) )	// Check to see if decelerating
	{
		acc=aa3+ab3*t+ac3*t*t;
	}

	else			// Check to see if an error has occured
		printf("Index is out of Range");

	return(acc);
};

//=====================================================================
// Class - MinTimeTrajectory
//
// Purpose : Inherits basic components of Trajectory class.  Constructor
//	bases parameters of model on a minimum time trajectory.  e.g
//	this approach uses max acceleration until desired acceleration
//	is established and then uses maximum acceleration to stop motion.
//
//=======================================================================
MinTimeTrajectory::MinTimeTrajectory(RowVector X1, RowVector X2, double V, double A,
	double Ts) : Trajectory(X1, X2, V, A, Ts)
{

	// Make sure dof is compatable
	if(X1.GetVecSize() != X2.GetVecSize())
		printf("Incompatable Desired Points");

	Xhome=X1;			// Call first vector "home"

	DOF=X1.GetVecSize();	// Get Degrees of Freedom
	Amax=A;		// Maximum Acceleration
	SampleTime=Ts;		// Sample rate
	D=0;		// Compute distance of path
	for(int i=0;i<DOF;i++)
		D=D+(X2[i]-X1[i])*(X2[i]-X1[i]);
	D=sqrt(D);

	for(i=0;i<DOF;i++)	// Compute slopes to project path to cart. coord.
		m[i]=(X2[i]-X1[i])/D;

	if(D > V*V/Amax)
	{
		Vd=V;		// Desired Velocity
		T1=Vd/Amax;	// Compute time required for accel. and decell.
		T1i=int(T1/Ts);
		T=T1+D/Vd;		// Compute total time to finish path
		Ti=int(T/Ts);
	}
	else
	{
		Vd=sqrt(D*Amax);
		T1=D/Vd;	// Compute time required for accel. and decell.
```

```
        T1i=int(T1/Ts);
        T=2*T1;
        Ti=int(T/Ts);
    }

    tau=T-T1;
    a1=0;
    b1=0;
    c1=Amax/2;
    d1=0;
    e1=0;
    a2=Amax*T1*T1/2-Vd*T1;
    b2=Vd;
    c2=0;
    d2=0;
    e2=0;
    a3=D-0.5*A*T*T;
    b3=Amax*T;
    c3=-Amax/2;
    d3=0;
    e3=0;

    va1=0;
    vb1=Amax;
    vc1=0;
    vd1=0;
    va2=Vd;
    vb2=0;
    vc2=0;
    vd2=0;
    va3=Vd+Amax*(T-T1);
    vb3=-Amax;
    vc3=0;
    vd3=0;

    aa1=Amax;
    ab1=0;
    ac1=0;
    aa2=0;
    ab2=0;
    ac2=0;
    aa3=-Amax;
    ab3=0;
    ac3=0;
};
MinTimeTrajectory::MinTimeTrajectory(int dof) : Trajectory(dof)
{
};

void MinTimeTrajectory::Reset(RowVector X1, RowVector X2, double V, double A,
double Ts)
{
    // Make sure dof is compatable
    if(X1.GetVecSize() != X2.GetVecSize())
        printf("Incompatable Desired Points");

    Xhome=X1;                   // Call first vector "home"

    DOF=X1.GetVecSize();    // Get Degrees of Freedom
    Amax=A;             // Maximum Acceleration
    SampleTime=Ts;      // Sample rate
    D=0;                // Compute distance of path
    for(int i=0;i<DOF;i++)
        D=D+(X2[i]-X1[i])*(X2[i]-X1[i]);
    D=sqrt(D);

    for(i=0;i<DOF;i++)    // Compute slopes to project path to cart. coord.
        m[i]=(X2[i]-X1[i])/D;
```

```
    if(D > V*V/Amax)
    {
        Vd=V;               // Desired Velocity
        T1=Vd/Amax;     // Compute time required for accel. and decell.
        T1i=int(T1/Ts);
        T=T1+D/Vd;          // Compute total time to finish path
        Ti=int(T/Ts);
    }
    else
    {
        Vd=sqrt(D*Amax);
        T1=D/Vd;    // Compute time required for accel. and decell.
        T1i=int(T1/Ts);
        T=2*T1;
        Ti=int(T/Ts);
    }

    tau=T-T1;
    a1=0;
    b1=0;
    c1=Amax/2;
    d1=0;
    e1=0;
    a2=Amax*T1*T1/2-Vd*T1;
    b2=Vd;
    c2=0;
    d2=0;
    e2=0;
    a3=D-0.5*A*T*T;
    b3=Amax*T;
    c3=-Amax/2;
    d3=0;
    e3=0;

    va1=0;
    vb1=Amax;
    vc1=0;
    vd1=0;
    va2=Vd;
    vb2=0;
    vc2=0;
    vd2=0;
    va3=Vd+Amax*(T-T1);
    vb3=-Amax;
    vc3=0;
    vd3=0;

    aa1=Amax;
    ab1=0;
    ac1=0;
    aa2=0;
    ab2=0;
    ac2=0;
    aa3=-Amax;
    ab3=0;
    ac3=0;
};

// Emergency Stop.  Pass index so it can look up current velocity
// This method modifies trajectory so that only last portion of
// trajtectory is executed.
void MinTimeTrajectory::Stop(int index)
{
    Xhome=Update(index);
    Vd=GetVelocity(index);  // Base trajtectory on current velocity
    T1=0;    // Compute time required for accel. and decell.
    T1i=int(T1/SampleTime)-1;
    T=Vd/Amax;    // Compute time required for accel. and decell.
    Ti=int(T/SampleTime);
```

```
	tau=T-T1;
	a2=0;
	b2=Vd;
	c2=-Amax/2;
	d2=0;
	e2=0;
	a3=0;
	b3=0;
	c3=0;
	d3=0;
	e3=0;
	a1=0;
	b1=0;
	c1=0;
	d1=0;
	e1=0;
	va2=Vd;
	vb2=-Amax;
	vc2=0;
	vd2=0;
};

//=========================================================================
// Class - MinTimeTrajectory
//
// Purpose : Inherits basic components of Trajectory class.  Constructor
//	bases parameters of model on smooth jerk trajectory.  e.g
//	this approach uses departs from the bang bang approach used
//	in the minimum time approach to ensure finite jerk during motion.
//
//=========================================================================
MinJerkTrajectory::MinJerkTrajectory(RowVector X1, RowVector X2, double V, double A,
	double Ts) : Trajectory(X1, X2, V, A, Ts)
{
	// Make sure dof is compatable
	if(X1.GetVecSize() != X2.GetVecSize())
		printf("Incompatable Desired Points");

	Xhome=X1;			// Call first vector "home"

	DOF=X1.GetVecSize();	// Get Degrees of Freedom
	Amax=A;		// Maximum Acceleration
	SampleTime=Ts;		// Sample rate
	D=0;		// Compute distance of path
	for(int i=0;i<DOF;i++)
		D=D+(X2[i]-X1[i])*(X2[i]-X1[i]);
	D=sqrt(D);

	for(i=0;i<DOF;i++)	// Compute slopes to project path to cart. coord.
		m[i]=(X2[i]-X1[i])/D;

	if(D > 3*V/(2*Amax))
	{
		Vd=V;		// Desired Velocity
		T1=(3*Vd)/(2*Amax);	// Compute time required for accel. and decell.
		T1i=int(T1/Ts);
		T=T1+D/Vd;		// Compute total time to finish path
		Ti=int(T/Ts);
	}
	else
	{
		Vd=sqrt(2*D*Amax/3);
		T1=sqrt(3*D/(2*Amax));
		T1i=int(T1/Ts);
		T=2*T1;
		Ti=int(T/Ts);
	}

	tau=T-T1;
```

```
	double tau2=tau*tau;
	double tau3=tau2*tau;
	double tau4=tau3*tau;
	double T12=T1*T1;
	double T13=T12*T1;
	a1=0;
	b1=0;
	c1=0;
	d1=Vd/T12;
	e1=-Vd/(2*T13);
	a2=-Vd*T1/2;
	b2=Vd;
	c2=0;
	d2=0;
	e2=0;
	a3=Vd*(tau4/(2*T13)+tau3/T12-T1/2);
	b3=Vd*(1-3*tau2/T12-2*tau3/T13);
	c3=Vd*(3*tau/T12+3*tau2/T13);
	d3=-Vd*(1/T12+2*tau/T13);
	e3=Vd/(2*T13);

	va1=0;
	vb1=0;
	vc1=3*Vd/(T1*T1);
	vd1=-2*Vd/(T1*T1*T1);
	va2=Vd;
	vb2=0;
	vc2=0;
	vd2=0;
	va3=T*T*Vd*(3-2*T/T1)/(T1*T1);
	vb3=6*T*Vd*(T/T1 - 1)/(T1*T1);
	vc3=3*Vd*(1-2*T/T1)/(T1*T1);
	vd3=2*Vd/(T1*T1*T1);

	aa1=0;
	ab1=6*Vd/(T1*T1);
	ac1=-6*Vd/(T1*T1*T1);
	aa2=0;
	ab2=0;
	ac2=0;
	aa3=6*T*Vd*(T/T1 - 1)/(T1*T1);
	ab3=6*Vd*(1-2*T/T1)/(T1*T1);
	ac3=6*Vd/(T1*T1*T1);
};

MinJerkTrajectory::MinJerkTrajectory(int dof) : Trajectory(dof)
{
};

void MinJerkTrajectory::Reset(RowVector X1, RowVector X2, double V, double A,
double Ts)
{
	// Make sure dof is compatable
	if(X1.GetVecSize() != X2.GetVecSize())
		printf("Incompatable Desired Points");

	Xhome=X1;			// Call first vector "home"

	DOF=X1.GetVecSize();	// Get Degrees of Freedom
	Amax=A;		// Maximum Acceleration
	SampleTime=Ts;		// Sample rate
	D=0;		// Compute distance of path
	for(int i=0;i<DOF;i++)
		D=D+(X2[i]-X1[i])*(X2[i]-X1[i]);
	D=sqrt(D);

	for(i=0;i<DOF;i++)	// Compute slopes to project path to cart. coord.
		m[i]=(X2[i]-X1[i])/D;
```

```
    if(D > 3*V/(2*Amax))
    {
        Vd=V;               // Desired Velocity
        T1=(3*Vd)/(2*Amax);     // Compute time required for accel. and decell.
        T1i=int(T1/Ts);
        T=T1+D/Vd;              // Compute total time to finish path
        Ti=int(T/Ts);
    }
    else
    {
        Vd=sqrt(2*D*Amax/3);
        T1=sqrt(3*D/(2*Amax));
        T1i=int(T1/Ts);
        T=2*T1;
        Ti=int(T/Ts);
    }
    tau=T-T1;

    double tau2=tau*tau;
    double tau3=tau2*tau;
    double tau4=tau3*tau;
    double T12=T1*T1;
    double T13=T12*T1;
    a1=0;
    b1=0;
    c1=0;
    d1=Vd/T12;
    e1=-Vd/(2*T13);
    a2=-Vd*T1/2;
    b2=Vd;
    c2=0;
    d2=0;
    e2=0;
    a3=Vd*(tau4/(2*T13)+tau3/T12-T1/2);
    b3=Vd*(1-3*tau2/T12-2*tau3/T13);
    c3=Vd*(3*tau/T12+3*tau2/T13);
    d3=-Vd*(1/T12+2*tau/T13);
    e3=Vd/(2*T13);

    va1=0;
    vb1=0;
    vc1=3*Vd/(T1*T1);
    vd1=-2*Vd/(T1*T1*T1);
    va2=Vd;
    vb2=0;
    vc2=0;
    vd2=0;
    va3=T*T*Vd*(3-2*T/T1)/(T1*T1);
    vb3=6*T*Vd*(T/T1 - 1)/(T1*T1);
    vc3=3*Vd*(1-2*T/T1)/(T1*T1);
    vd3=2*Vd/(T1*T1*T1);

    aa1=0;
    ab1=6*Vd/(T1*T1);
    ac1=-6*Vd/(T1*T1*T1);
    aa2=0;
    ab2=0;
    ac2=0;
    aa3=6*T*Vd*(T/T1 - 1)/(T1*T1);
    ab3=6*Vd*(1-2*T/T1)/(T1*T1);
    ac3=6*Vd/(T1*T1*T1);
};

// Emergency Stop.  Pass index so it can look up current velocity
// This method modifies trajectory so that only last portion of
// trajtectory is executed.
void MinJerkTrajectory::Stop(int index)
{
    Xhome=Update(index);
```

```
	Vd=GetVelocity(index);  // Base trajtectory on current velocity
	T1=0;	// Compute time required for accel. and decell.
	T1i=int(T1/SampleTime);
	T=3*Vd/(2*Amax);	// Compute time required for accel. and decell.
	Ti=int(T/SampleTime);
//	cout << "T1i=" << T1i << " and Ti=" << Ti;
	double temp=3*Vd/(2*Amax);
	tau=T-T1;
	a2=0;
	b2=Vd;
	c2=0;
	d2=-2*Amax/(3*temp);
	e2=Amax/(3*temp*temp);
	a3=0;
	b3=0;
	c3=0;
	d3=0;
	e3=0;
	a1=0;
	b1=0;
	c1=0;
	d1=0;
	e1=0;
	va2=Vd;
	vb2=0;
	vc2=-2*Amax/temp;
	vd2=4*Amax/(3*temp*temp);
};

#if !defined(__TRAJECTORY_HPP)
#define __TRAJECTORY_HPP 1

#include "matrix.hpp"

class Trajectory
{
	public:
		Trajectory();
		~Trajectory(void){}
		Trajectory(RowVector X1, RowVector X2, double v, double a, double ts);
		Trajectory(const Trajectory &tmpTraj);
		Trajectory(int dof);
		RowVector Update(int index);
		int GetT(void){ return Ti; }
		int GetStopIndex(void){ return (Ti-T1i);}
		double GetVelocity(int index);
		double GetAcceleration(int index);

	protected:
		double Vd;
		double Amax;
		double ctheta;
		double stheta;
		double D;
		double d;
		double SampleTime;
		double T;
		double T1;
		double a1,b1,c1,d1,e1,
				a2,b2,c2,d2,e2,
				a3,b3,c3,d3,e3,
				tau;
		double va1,vb1,vc1,vd1,
			va2,vb2,vc2,vd2,
			va3,vb3,vc3,vd3;

		double aa1,ab1,ac1,
			aa2,ab2,ac2,
			aa3,ab3,ac3;
```

```
        int Ti;
        int T1i;
        int DOF;
        RowVector Xtraj;
        RowVector Xhome;
        RowVector m;    // slopes
};

class MinTimeTrajectory : public Trajectory
{
    public:
        MinTimeTrajectory(RowVector x1, RowVector x2, double V, double A, double Ts);
        MinTimeTrajectory(int dof);
        void Reset(RowVector X1, RowVector X2, double V, double A, double Ts);
        void Stop(int index);
};

class MinJerkTrajectory : public Trajectory
{
    public:
        MinJerkTrajectory(RowVector x1, RowVector x2, double V, double A, double Ts);
        MinJerkTrajectory(int dof);
        void Reset(RowVector X1, RowVector X2, double V, double A, double Ts);
        void Stop(int index);
};

#endif;
```

Command Filtering and Path Planning for Remote Manipulation of a Long Reach Flexible Robot

Lonnie J. Love and David P. Magee
The George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0405
http://davinci.marc.gatech.edu/

Abstract

Long reach manipulators are characterized by their light weight and large workspace. In order to fully utilize this workspace under teleoperated commands, motion amplification must exist between the master robot motion and the commanded motion of the slave robot. Unfortunately, this limits the accuracy with which an operator is capable of remotely executing a task. To extend the capabilities of our long reach testbed, our work focuses on the fusion of autonomous and teleoperated commands. This combination provides full use of the robot's workspace without requiring large motion amplification between a master and slave robot.

Combining autonomous and teleoperated commands provides the potential for large variations in the commanded momentum of the flexible robot. These variations excite the lightly damped, low resonant frequencies associated with these manipulators. This phenomenon provides the motivation for further investigation on the effect of joint control and path planning techniques on the tracking performance of flexible robots. Two techniques are proposed to reduce the vibration during sudden stops in the commanded motion of a flexible manipulator. First, a new command filtering approach that permits shorter delay times than standard input shaping methods is presented. Next, we propose dynamic alteration of the desired trajectory. Our investigation shows that filtering techniques exhibit an oscillatory response, more so than standard PD control algorithms, during hard stopping conditions. However, the shorter time delay filtering algorithm has less vibration than standard input shaping techniques. Furthermore, any vibration may be eliminated by commanding the robot to decelerate instead of immediately stopping the motion. Analysis and experimental results are provided.

1. Introduction

The advantage of long reach manipulators has been well documented over the past twenty years [3], [15]. These robots are characterized by their large workspace and light structural weight. However, this reduction in structural mass results in lower natural frequency values. Therefore, these robots have a tendency to vibrate during the execution of tasks. This vibratory effect has led to a flurry of mechanical and control design concepts. Bayo [2] and Kwon [8] showed fast response with little vibration by using inverse dynamic techniques. Alberts [1] showed that these techniques, in combination with passive damping on the elastic links, can reduce the magnitude of a broad band of frequencies during slewing motions. Unfortunately, the inverse dynamics technique requires an accurate definition of the robot's dynamic equations of motion which may prove difficult for multi-link robot systems. More recently, input shaping [13] and command filtering [11] techniques demonstrate reduced oscillatory effects without the sensitivity to modeling errors experienced with inverse dynamic techniques. Furthermore, Magee has shown that filtering techniques applied to a rigid manipulator attached to the end of a long reach flexible robot reduces the residual vibration of this combined system while still performing meaningful tasks [12].

Recently, attention is shifting to the utility of such systems for complex problems associated with handling hazardous materials [7]. Preliminary experiments using force reflecting teleoperation of flexible robots illustrate a few problems that exist during simple contact tasks [10]. The large workspace associated with these robots requires motion amplification between the master and slave robots. This scaling reduces the positioning accuracy that is necessary during contact tasks. Alternative techniques are sought that provide the advantage of teleoperation without requiring the operator's constant interaction through teleoperation or large motion amplification. First, a new approach is described to seamlessly combine both autonomous and teleoperated commands. Autonomous commands provide course positioning of the robot in its workspace. Teleoperated commands are superimposed on these commands to provide a perturbation from the desired path of the robot. In essence, the teleoperated commands permit the operator to execute articulated maneuvers while the autonomous component takes care of globally positioning the end of the robot.

Experiments show that the flexibility of these robots permits vibration during the transition between these two modes of operation. This investigation examines control techniques and trajectory generation methods in an attempt to isolate this difficult problem. Our investigation compares the performance of two control techniques, PD and command filtering, as well as an adaptive bang-bang path generator.

2. Seamless Transfer Between Autonomous and Teleoperated Commands

Methods are sought that permit the combination of both autonomous and teleoperated commands for

manipulation. Our approach consists of treating teleoperated commands as perturbations from the command trajectory provided by an autonomous path planner. This approach differs slightly from the techniques described by Guo [5]. They proposed event-based planning and control as a means of fusing autonomous and teleoperated commands. Their system contains four basic functions: Stop, SlowDown, SpeedUp, and Orthogonal. The commands from the master robot, a spaceball in their system, provide velocity modifications to the command trajectory of the slave robot. Our approach is slightly different in that commands from the master robot consist of position modifications to the command trajectory of the slave robot. This approach permits easy implementation of force reflection but requires careful consideration during the transition between velocity and position commands.

Our testbed, described by Book [4], requires a position amplification of 7:1 between master and slave robots. Scaling provides a comfortable match between the slave robot's workspace and the human operator. For pure teleoperated commands, end point accuracy is limited when using position based control schemes. This position amplification may be reduced by using autonomous commands for large motions and teleoperated commands for fine articulated manipulation.

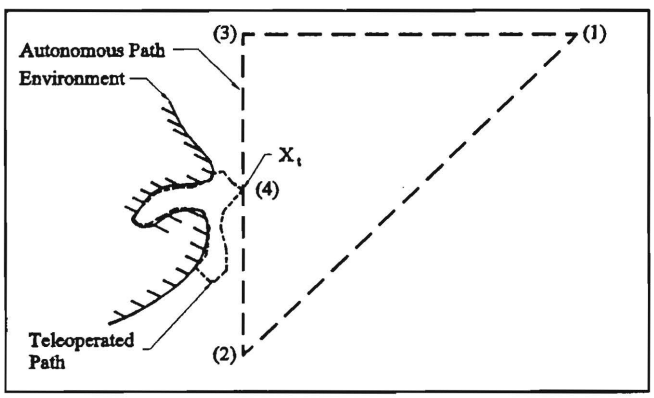

Figure 1. Autonomous and Teleoperated Task Execution

Consider the task illustrated in Figure 1. The robot uses autonomous commands for global positioning. However, when interaction between the robot and the environment is required, the system switches to a teleoperated mode. After completing the task, the control system transfers back to an autonomous mode and continues along its path. The following section describes how the switching between autonomous and teleoperated modes is accomplished and the problems that exist.

2.1 Impedance Controlled Master Robot

Our testbed consists of two kinematically dissimilar manipulators. The slave robot, RALF, is a two link, long reach manipulator. Each link is ten feet in length. Furthermore, the structural weight of the robot does not exceed 100 pounds while its payload capacity is approximately 60 pounds. The first natural frequency of this robot is about 4.5 Hz with a damping ratio of 0.01.

The master robot, HURBIRT, is a two degree of freedom impedance controlled robot designed for studies in the interaction of humans and robots [9]. The target impedance for the robot is defined as

$$M_t \ddot{x}_m + B_t \dot{x}_m + F_v = F_h + \frac{1}{A} F_e \qquad (1)$$

where x_m is the position of the master robot, F_h is the force applied by the human operator and F_e is the force applied by the environment. The target mass and damping matrices, M_t and B_t respectively, control the ease with which the operator moves the master robot. The virtual force, F_v, represents the repulsive force produced by deforming virtual fixtures in the robot's workspace. One example using the target impedance on a master robot is illustrated in Figure 2.

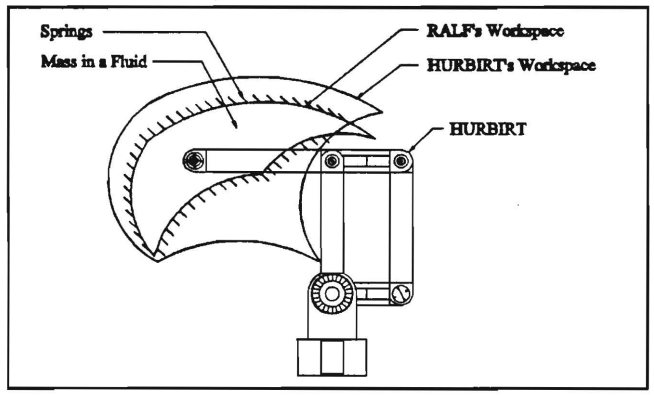

Figure 2. Virtual Walls for Bilateral Teleoperation

Since the workspaces of the master and slave manipulators are dissimilar, simple tasks such as moving the slave robot to its home position prove to be difficult by visual cues alone. Virtual walls are used to constrain the motion of the master robot to the scaled workspace of the slave robot. The target impedance of the master robot, using the same philosophy of superimposing impedances described by Hogan [6], is augmented with virtual walls that constrain the operator from commanding the slave robot outside it's workspace. Four compliant circles replicate the limits of the slave robot's workspace mapped inside the master robot's workspace. If the operator manipulates inside the scaled slave robot's workspace, the robot effectively "feels" like a mass moving through a viscous fluid. However, if the human attempts to command the robot outside it's workspace, the virtual walls attempt to push the operator back into the workspace. Position commands from the master robot to the slave robot are scaled by the amplification, A (for our testbed, $A = 7.0$),

$$x_s = A x_m \qquad (2)$$

where x_s is the position of the slave robot. The operator maneuvers the robot about its workspace tracing the trajectory to follow during autonomous manipulation.

During autonomous motion, a slightly different target impedance for the master robot is selected. The virtual force

$$F_v = K e^{-a|x_m - x_0|} \left(x_m - x_0\right) \qquad (3)$$

is now a decaying potential well. This force can provide a localized equilibrium position on the master robot. The stiffness, K, controls the attractive potential while α controls the rate of decay of the force as the operator moves away from the equilibrium position. After the tip of the robot moves sufficiently far away from equilibrium, controlled by α, the robot behaves like a mass moving through a viscous fluid. Without any external forces applied to the master robot, the tip position of the master robot stays at the equilibrium position, x_0. If the tip position of the slave robot is within a defined radius of the equilibrium position, δ, the slave robot is under autonomous commands alone. A vector x_t denotes the current commanded position along a desired trajectory and k specifies the discrete time index. In purely autonomous mode, the current position of the trajectory is passed along to the slave robot as the new desired slave position x_s.

$$x_s = x_t[k] \tag{4}$$

If the human grabs the master robot and moves it away from the equilibrium position, the control system transforms from autonomous to teleoperated mode. First, the time index, k, associated with the command trajectory, $x_t[k]$, is suspended. The command to the slave robot now consists of two components, the last position on the trajectory and the perturbation provided by the human through the master robot.

$$x_s = x_t[k] + x_m - x_0 \tag{5}$$

Commands from the master robot provide a deviation from the commanded path. This approach provides a natural method of switching between autonomous and teleoperated commands. The operator needs only to grab the master robot and move it to switch between modes. Furthermore, after completing the teleoperated task, the operator needs only to move the master robot into the vicinity of the equilibrium position and release the master robot. The attractive potential field will draw the robot to the equilibrium position and the robot will then switch back to autonomous mode.

Of central concern now is the transfer between autonomous and teleoperated modes. The transition may require a dramatic shift in the commanded momentum of the flexible slave robot. Furthermore, when the operator completes the teleoperated task and the system switches back to autonomous mode, it must again accelerate to the command velocity. These issues are aggravated by the compliance associated with the long reach slave manipulator. A shift in momentum may excite vibration in the link structure of the robot. The following sections compare the performance of path planners and joint motion controllers and their influence upon the vibration of the slave robot during abrupt changes in momentum.

3. Command Filtering

The command filtering approach used here is based on pole-zero cancellation of the second-order equations of motion describing the flexible behavior. The three term filter takes the form

$$F(s) = \frac{1 - 2\cos(\omega_1 T)\, e^{\sigma_1 T} e^{-sT} + e^{2\sigma_1 T} e^{-s2T}}{1 - 2\cos(\omega_1 T)\, e^{\sigma_1 T} + e^{2\sigma_1 T}} \tag{6}$$

to cancel poles located at $s = \sigma_1 \pm j\omega_1$. This s-domain filter can be transformed to the digital domain with the transformation $z = e^{sT_s}$ where T_s is the inverse of the sampling rate of the discrete-time system. See [12] for a more detailed discussion of the command filtering method.

After identifying the poles of the system to be canceled, the delay time value, T, must be chosen. Previous filter design work has shown that an effective gain can be generated if the delay time is shorter than one-half the damped period of the second-order system. In standard shaping methods, the maximum gain is unity because the method is restricted to one value of delay time. In this work, we compare two different delay times. First, we use the delay time associated with Singer's input shaping technique (IS) [14]. For our system, the delay time is 0.091 seconds. Next, we use a general command filter (CF) which has a shorter delay time of 0.045 seconds. These filters are applied to the feedback error signal in a PD control scheme on the slave manipulator (i.e. RALF) in a similar manner as given in [11].

4. Trajectory Generators

When the robot switches between these autonomous and teleoperated modes, dramatic shifts in the commanded momentum of the robot exist. To reduce this effect, smooth blending between constant velocity trajectories and teleoperated commands are proposed. As an example, consider the case where the robot is commanded to depart from an existing constant velocity trajectory to teleoperated commands in an orthogonal direction. Simply switching from velocity to position commands excites lower modes of vibration in the compliant slave robot. An alternative approach is to smoothly blend these commands so the robot reduces its momentum before switching completely over to teleoperated commands. The problem also exists when transferring back fromteleoperated to autonomous commands.

4.1 BangBang Acceleration Profile

The first trajectory considered in this investigation is the Bang Bang acceleration profile. This profile accelerates at a maximum rate until the desired velocity is reached. When close to the destination, the robot decelerates at its peak rate.

$$v(t) = \begin{cases} A_{\max} t & t \le T_1 \\ V_d & T_1 < t < T - T_1 \\ V_d - A_{\max} t + A_{\max}(T - T_1) & T - T_1 \le t \end{cases} \tag{7}$$

$$T_1 = \frac{V_d}{A_{\max}} \qquad T = \frac{D}{V_d} + \frac{V_d}{A_{\max}} \tag{8}$$

Equation (7) provides the velocity profile with the time constants defined in Equation (8) where D is the distance of the path, V_d is the desired velocity and $A_{\max}$ is the maximum acceleration.

4.2 Seamless Transfer

To permit smooth transitions between autonomous and teleoperated states, dynamic alteration of the commanded path is required. To smoothly decrease the momentum of the robot when transforming from autonomous mode to teleoperation, the present velocity of the manipulator is first measured. Next, the parameters of the profile to go from this initial velocity to zero velocity are computed. This computation is easily accomplished within the sampling rate of the robot's controller. Thus, during the first few cycles of the teleoperated commands, the effective trajectory of the autonomous mode decelerates. This deceleration provides a smooth transition between autonomous and teleoperated modes. After completing a teleoperated task, the human moves the master robot to its equilibrium position. When the master robot reaches the equilibrium position, the current position of the slave robot is measured and the parameters of the trajectory are updated. Finally, the slave robot smoothly accelerates along its path towards its next target point.

5. Experimental Results

The following series of experiments illustrate the effect joint control and path planning have in the vibration response of a flexible robot during abrupt changes in the commanded momentum. Examples where this effect is relevant include the transfer between autonomous and teleoperated commands as well as emergency stop commands. The slave robot is commanded to follow a triangular path, illustrated in Figure 1. The horizontal and vertical portions of the path are one meter in length. The desired velocity along each segment of the triangle is 0.75 m/s with a maximum acceleration of 3.5 m/s^2. During the vertical line segment, at point (4) in Figure 1, the slave robot executes an abrupt stop. This effect can represent a transition produced through human intervention in teleoperation or an emergent stop situation.

Three joint control schemes are considered. First, the robot has a PD algorithm that is tuned to provide excellent joint tracking capabilities as illustrated in Figure 3.

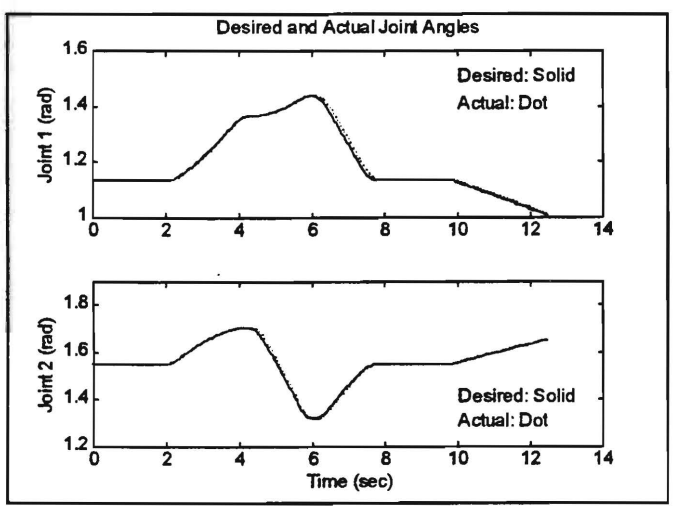

Figure 3. Joint Angle Response for Triangular Trajectory

The commands are then modified using either IS or CF, described in Section 3. Figure 4 illustrates the tracking performance of each of these algorithms. A landmark tracking system provides absolute tip position measurement. Evidently, some tracking error is due to the static deflection of the manipulators links. Joint controllers alone do not compensate for this effect.

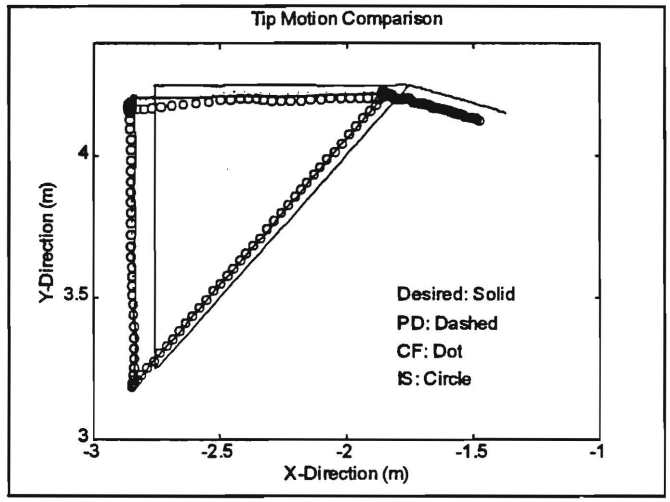

Figure 4. Tip Motion During Triangular Trajectory

Figure 5 illustrates the vibration produced in link 1 of RALF during the process of this task. It is evident that both filtering techniques reduce the level of vibration during motion. This reduction in vibration is also evident in the spectral response in Figure 6. Both filtering techniques reduce the magnitude of vibration of the first mode by approximately 20 dB.

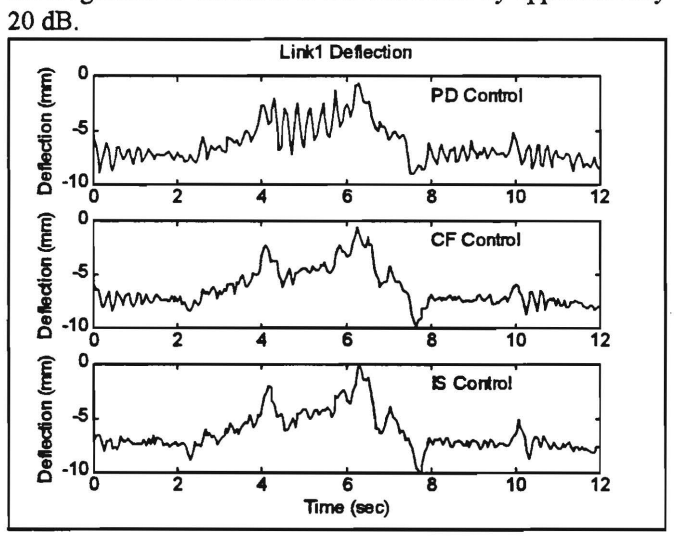

Figure 5. Link 1 Deflection During Triangular Trajectory

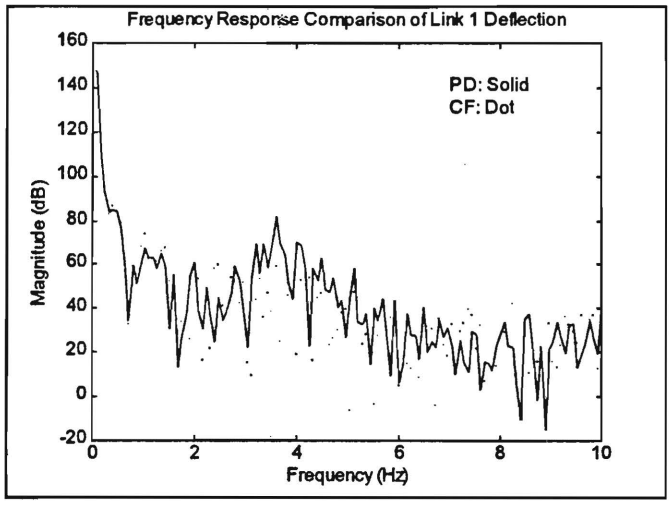

Figure 6. Frequency Response of Link 1 Deflection

Vibration is also evident during the hard stop, illustrated in Figures 7 and 8. This condition actually favors the PD controller over the filtering techniques. The increase in vibration for the shaping methods is due to the delayed response provided by the filtering process. The stop is initiated when the y position is 3.7 m. The PD controller overshoots the stopping point by 19 cm. The CF controller has a maximum overshoot of 31 cm while the IS has an overshoot of 52 cm. The CF controller produces less overshoot because it has an overall shorter delay time than the IS controller.

An alternative approach to stopping the momentum of the robot consists of commanding a smooth stop. This is accomplished by commanding the robot to execute the final stage of the velocity profile when the system is commanded to stop. Figure 9 illustrates the performance of the PD and CF controllers during this soft stop. It is evident that the magnitude of overshoot has decreased dramatically. The CF controller has a maximum overshoot of 1.75 cm while the PD controller has a maximum overshoot of 0.4 cm.

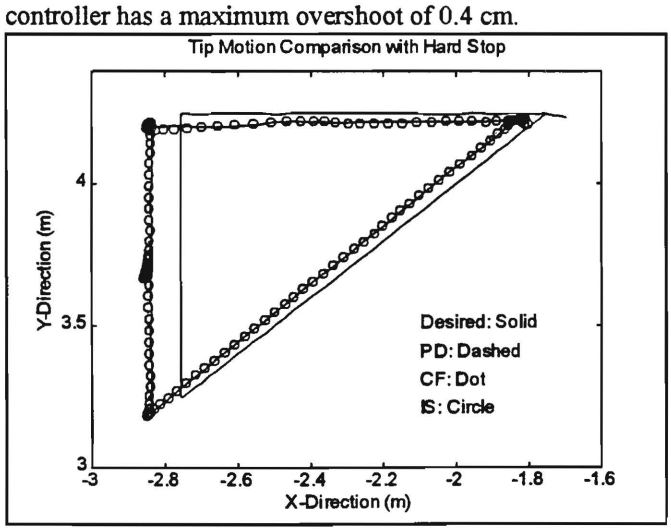

Figure 7. Tip Motion Comparison During Triangular Trajectory with Hard Stop

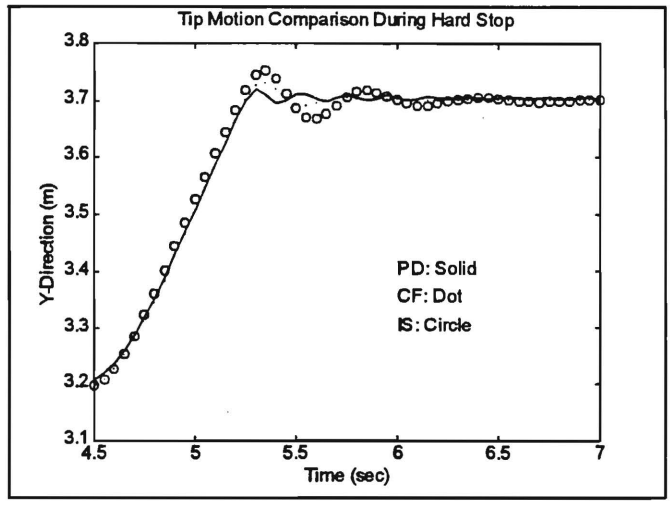

Figure 8. Tip Motion Comparison During Hard Stop

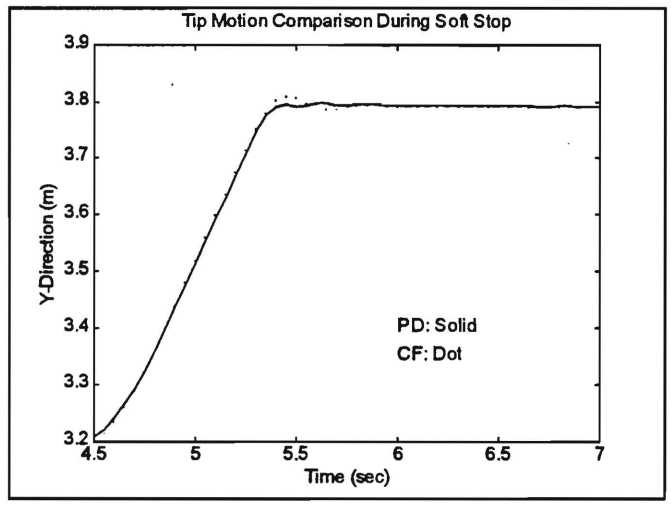

Figure 9. Tip Motion Comparison During Soft Stop

This reduction in vibration is somewhat deceiving because the actual commanded endpoint with the soft stop does not correspond to the point where the stop was initiated. While the approach reduces the magnitude of oscillation during a stop, it increases the error between the desired and actual robot stop positions.

6. Conclusions and Recommendations

This investigation presented a new approach for the teleoperation of long reach flexible manipulators. Experiments showed that command filtering techniques provide excellent vibration suppression during normal operations. However, if a dramatic shift in the commanded momentum of the robot occurs, the performance of the filtering techniques decreases. By shortening the filter's delay time, the amplitude of vibration was reduced.

Tapering the command trajectory also reduced the level of vibration during hard stopping conditions. Further investigation is necessary to determine what profile or delay

time provides sufficient vibration absorption during slewing and hard stop trajectories. Also, these experiments suggest that some form of tip position feedback is necessary to compensate for the static deflection in the elastic robot.

Acknowledgments

This research was sponsored in part by Sandia National Laboratories under contract No. 18-4379G with the cooperation of Pacific Northwest Laboratory and Oak Ridge National Laboratory.

References

1. Alberts, T.A., Love, L.J., Bayo, E., and Moulin, H., "Experiments with End-Point Control of a Flexible Link using the Inverse Dynamics Approach and Passive Damping," *Proceedings of the 1990 American Control Conference*, Vol.1, San Diego, CA, May 23-25,1990, pp.350-355.
2. Bayo, E., Papadopoulos, P., Stubbe, J. and Serna, M.A., "Inverse Dynamics and Kinematics ofMulti-Link Elastic Robots: An Iterative Frequency Domain Approach," *The International Journal of Robotics Research*, Vol.8, No.6, December 1989, pp.4962.
3. Book, W.J., "Controlled Motion in an Elastic World," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol. 115, No. 2B, June 1993, pp.252-261.
4. Book, W.J., Lane, H., Love, L.J., Magee, D.P., Obergfell, K., "A Novel Teleoperated Long Reach Manipulator Testbed and its Remote Capabilities via the Internet," *to appear in the Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, MN, April 22-28, 1996.
5. Guo, C., Tarn, T.J., Ning, X., Bejczy, A.K., "Fusion of Human and Machine Intelligence for Telerobotic Systems," *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Vol.3, Nagoya, Aichi, Japan, May 21-27, 1995, pp.3110-3115.
6. Hogan, N., "Impedance Control: An Approach to Manipulation: Part III - Applications," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol.107, No.1, March 1985, pp.17-24.
7. Kwon, D.S., Hwang, D.H., Babcock, S.M., and Burks, B.L., "Input Shaping Filter Methods for the Control of Structurally Flexible, Long-Reach Manipulators," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, Vol.4, San Diego, CA, May 8-13, 1994, pp.3259-3264.
8. Kwon, D.S., Book, W.J., "A Time-Domain Inverse Tracking Control of a Single-Link Flexible Manipulator," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol.116, No.2, June 1994, pp.193-200.
9. Love, L.J., and Book, W.J., "Design and Control of a Multiple Degree of Freedom Haptic Interface," *Dynamic Systems and Control*, DSC-Vol.55-2, ASME Winter Annual Meeting, Chicago, IL, November 6-11, 1994, pp.851-856.
10. Love, L.J., and Book, W.J., "Adaptive Impedance Control for Bilateral Teleoperation of Long Reach, Flexible Manipulators," *to appear in the Proceedings of the World Congress of International Federation of Automatic Control*, San Francisco, CA, June 30-July 5, 1996.
11. Magee, D.P. and Book, W.J., "Experimental Verification of Modified Command Shaping using a Flexible Manipulator," *Proceedings of the First International Conference on Motion and Vibration Control*, Vol.1, Yokohama, Japan, September 7-11, 1992, pp.55358.
12. Magee, D.P. and Book, W.J., "Filtering Micro-Manipulator Wrist Commands to Prevent Flexible Base Motion," *Proceedings of the 1995 American Control Conference*, Vol.1, Seattle, WA, June 21-23, 1995, pp.924-928.
13. Meckl, P.H., and Seering, W.P., "Experimental Evaluation of Shaped Inputs to Reduce Vibration for a Cartesian Robot," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol.112, No.2, June 1990, pp.159-165.
14. Singer, N.C., and Seering, W.P., "Preshaping Command Inputs to Reduce System Vibration," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol.112, No.1, March 1990, pp.7682.
15. Whitney, D.E., Book, W.J., Lynch, P.M., "Design and Control Considerations for Industrial and Space Manipulators," *Proceedings of the 1974 Joint Automatic Control Conference*, Vol.1, Austin, TX, June 18-21, 1974, pp.591-598.

E-25-L31
1

CONTROL METHODS FOR SEAMLESS TELEOPERATION MODE TRANSFER AND EMERGENCY STOPS USING LONG REACH, FLEXIBLE MANIPULATORS

FINAL REPORT

Lonnie J. Love

Submitted to:

Lockheed Martin Energy Systems, Inc.
Oak Ridge National Laboratory
Contract No. SU566-49
Wayne J. Book, Project Director

Period of Performance: 4/1/96 - 9/30/96

Final Report

CONTROL METHODS FOR SEAMLESS TELEOPERATION MODE TRANSFER AND EMERGENCY STOPS USING LONG REACH, FLEXIBLE MANIPULATORS

Summary

The work performed during the term of this contract is broken into two subtasks as described in the statement of work. The first subtask focuses on the transfer between robotic and teleoperated commands. Issues addressed during this phase of the contract included the control of a force reflecting master robot and the adaptation of robotic path planning techniques. The second subtask focuses on an investigation of control techniques during the emergency stop condition of long reach, flexible manipulators. The following provides a brief description of the findings of this investigation. Additional details of each investigation are described in the detailed description of each project subtask.

Seamless Teleoperation Mode Transfer

The first phase of this contract focused on the development of a control method that combines robotic and teleoperation commands. This method treats teleoperated commands from an impedance controlled master robot as position perturbations along the robotic path of the long reach manipulator. Furthermore, a new impedance field was developed for the master robot during the term of this investigation. This field provides a decaying attractive potential. If the master robot is maneuvered close to its equilibrium point, the potential field will provide a stable asymptotic attraction to the equilibrium point. However, if the operator maneuvers the slave robot sufficiently away from the potential field, the resistance on the master robot will smoothly transfer to a mass moving through a viscous fluid. The advantage of this approach is two fold. First, when the master robot is near the equilibrium position, it will maintain this position without human interaction. During standard robotic operation, the operator does not have to maintain contact with the master robot. If the operator observes a condition which requires teleoperation, he or she needs only to grasp the master robot and move it to transfer to teleoperated commands.

The next issue addressed during this phase of the investigation focused on the fusion of robotic and teleoperated commands. Teleoperated commands, in this investigation, are considered desired position perturbations from robotic path. When an operator moves the master robot sufficiently away from its equilibrium position, the robotic path planner suspends its operation. Deviations from the equilibrium position of the master robot are added to the last position computed from the robotic path planner. When the operator completes the teleoperated phase of the task, he or she moves the master robot into the vicinity of the attractive potential field and releases the master robot. When the master robot moves sufficiently close to its equilibrium position, the system transfers back to robotic mode and continues along its original path. An additional issue addressed during this phase of the investigation included the dynamic adaptation of the robotic path during the transition between robotic and teleoperated commands to reduce the excitation of vibrations modes on the flexible slave manipulator.

Control of Long Reach, Flexible Robots During Emergency Stop Commands

The second subtask executed during this contract addressed the control of long reach, flexible robots during emergency stop commands. This investigation compared two approaches to stopping the momentum of the robot and the performance of three joint level controllers. The three methods of stopping the momentum of the robot included:

- Stopping at the position computed during the stop command (Hard Stop).
- Smoothly stopping using the last phase of a minimum time trajectory (Soft Stop).

Furthermore, the three joint level controller compared during this investigation included:

- PD
- PD with Command Filter set at half the damped natural frequency
- PD with Command Filter set at one quarter the damped natural frequency.

The results of this investigation show that shaping techniques increase the maximum overshoot of the tip position of the flexible robot during emergency stop commands. However, filtering techniques with shorter delay times provide less overshoot during emergency stops. Furthermore, vibration can be eliminated by softly stopping the momentum of the robot. Unfortunately, this changes the final command position of the robot. The smooth stop, while eliminating any vibration during the stop command, provides a final tip position that exceeds the maximum overshoot exhibited by all the controllers during hard stop conditions.

Project Subtask: Seamless Teleoperation Mode Transfer

The first phase of this contract focused on control methods that enable the seamless transfer between robotic and teleoperated commands. This investigation is motivated by the size discrepancy that exists between a master robot's workspace and a long reach slave manipulator, illustrated in Figure 1. Pure teleoperation control requires large motion amplification between the master and slave robot. This amplification reduces the accuracy with which an operator can position the tip of the slave robot. A strategy that combines robotic and teleoperated commands permits robotic commands to globally position the slave robot while teleoperation commands facilitate fine, articulated commands.

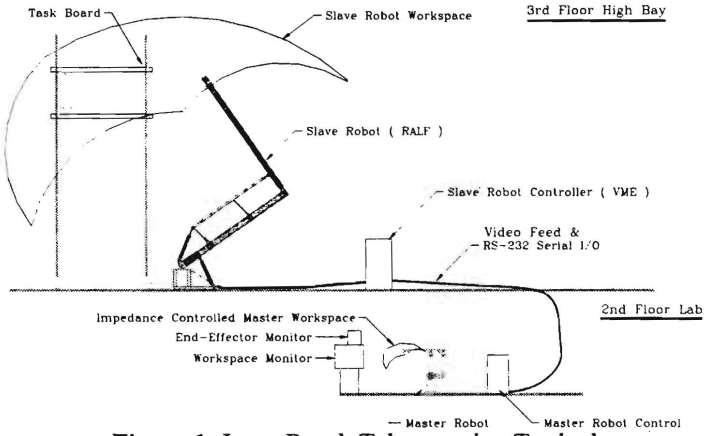

Figure 1: Long Reach Teleoperation Testbed

This subtask is divided into two additional tasks. First, an adaptive impedance control scheme is described for the force reflecting master robot. Two target impedance fields control the tactile response an operator experiences during robotic commands. The master and slave robots used during this investigation are kinematically dissimilar. The first field, developed prior to the execution of this contract, is selected during pure teleoperation commands. Compliant spheres, illustrated in Figure 2, constrain the motion of the master robot to the scaled workspace of the slave robot. If the operator maneuvers the master robot inside the scaled workspace of the slave robot, the master robot behave like a mass moving through a viscous fluid. However, the compliant spheres provide additional resistance if the operator attempts to command the slave robot outside of its workspace. This in effect is most relevant when the operator maneuvers the slave robot to its home position.

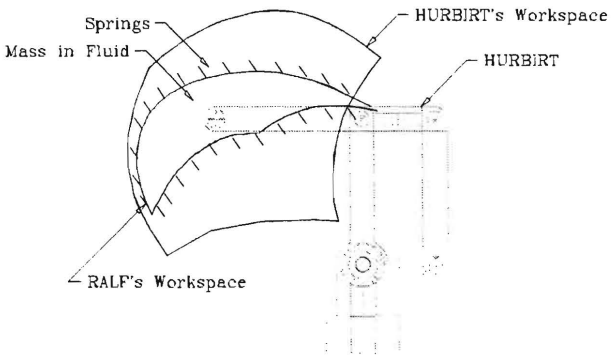

Figure 2: Virtual Walls for Bilateral Teleoperation

The second impedance field, developed during the term of this contract, is a decaying potential well. Equation (1) described the target impedance of the master robot.

$$M_t \ddot{x}_m + B_t \dot{x}_m + F_{vf} = F_h + \frac{1}{A} F_e \tag{1}$$

The mass and damping matrix, M_t and B_t respectively, control the effective resistance of the master robot. The external forces provided to the robot include the human applied force, F_h, and the external force reflected from the slave robot, F_e. The scaling factor, A, is the motion amplification from the master robot to the slave robot. The virtual force, F_{vf}, is a synthetic force. For pure teleoperation, the scale factor A is 7.0 and the virtual force is provided by the compliant spheres described previously. When operating in the hybrid robotic/teleoperated mode, the scale factor changes to unity. This provides a higher range of manipulation accuracy than the standard teleoperation mode. Furthermore, the virtual force switches to the decaying potential well, Eq.(2), when operating in the hybrid mode.

$$F_{vf} = K e^{-\alpha |x_m - x_0|} (x_m - x_0) \tag{2}$$

Without any external forces applied to the master robot, the tip position of the master robot stays at the equilibrium position, x_0. If the tip position of the slave robot is within a defined radius of the equilibrium position, δ, the slave robot is under autonomous commands alone, where x_t is the current command position along a trajectory where k is a time index corresponding to the command trajectory.

$$x_s = x_t(k) \tag{3}$$

However, if the human grabs the master robot and moves it away from the equilibrium position, the system transforms from autonomous to teleoperated modes. First, the time index associated with the command trajectory is frozen. The command to the slave robot now consists of two components, the last position on the trajectory and the perturbation provided by the human through the master robot.

$$x_s = x_t(k) + x_m - x_0 \tag{4}$$

This approach provided a natural method of switching between autonomous and teleoperated commands. The operator needs only to grab the master robot and move it to switch between modes. Furthermore, after completing the teleoperated tasks, the operator needs only to move the master robot into the vicinity of the equilibrium position and release the master robot. The attractive potential field will draw the robot to the equilibrium position and the robot will then switch back to the autonomous mode.

An additional issue is the effect that a drastic shift in commanded momentum may have on the control of a flexible robot. Consider the limiting case where the slave robot is moving at a constant velocity and is commanded by the teleoperator to immediately move in an orthogonal direction. One approach to reduce the effect of this shift in command velocity is to smoothly stop the robotic commands while blending with the teleoperation commands. Likewise, when shifting from teleoperation to robotic mode, it is advisable to smoothly accelerate from zero command velocity to the maximum velocity. These issues are covered in detail in the next section.

Project Subtask: Control of Long Reach, Flexible Robots During Emergency Stop Commands

The next phase of this contract focused on the effect emergency stop commands have on the performance of long reach, flexible manipulators. A large body of research has focused on vibration suppression and control of manipulators with elastic links. An additional problem not yet addressed by the robotics community is the effect this elasticity may have during an emergency stop condition. Consider the case where the robot is moving at a high speed and either end point motion sensors or a human detects that the robot is about to collide with an object. It may be disadvantageous to apply brakes to the joint actuators or suspend the desired joint angles. An immediate command to stop the motion at the robot's joints may result in a large overshoot at the end of the flexible robot. This overshoot may result in hard contact with the robot's environment.

This investigation focused on the performance of three joint level controllers combined with two emergency stop trajectories. Endpoint position sensors were employed to measure the tip deflection and vibration for each case. A landmark tracking system provides a measure of the tip position with respect to the elastic robot's Cartesian coordinate system. Furthermore, lateral effect photo diodes measure the link deformation. The elastic robot's controllers were developed by David P. Magee under the combined support of PNL and Sandia National Laboratories. These algorithms, written in C, run on a Motorola 68040 located on a VME bus system. The trajectory generators, described in this report, were written by Lonnie Love and executed the PC platform that controls the master robot. All of these algorithms, developed for this project, were written in C++.

The experiments conducted for this investigation consist of commanding the elastic robot to execute a triangular trajectory, illustrated in Figure 3. The command velocity along each leg of the trajectory is 0.75m/s. Three controllers were compared during this investigation. These controllers include PD, PD with a command filter with a delay of half the damped natural frequency, and PD with a command filter with a delay of one quarter the damped natural frequency. Figure 4 illustrate the effect the filtering has on the link vibration of the robot during motion. There is approximately 20 dB decrease in the magnitude of vibration when either of the command filtering techniques is used. These results show that command filtering techniques successfully reduce the vibration of a flexible robot during manipulation.

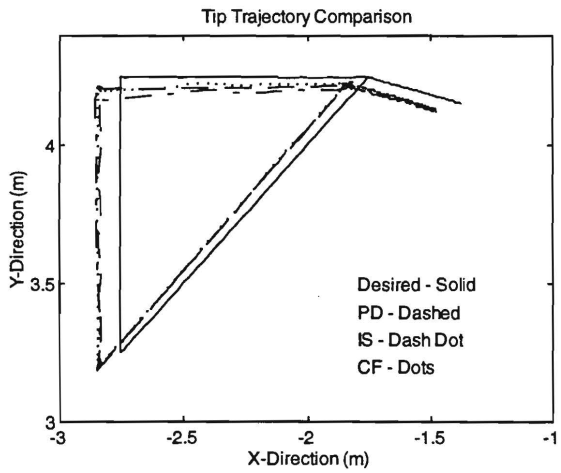

Figure 3: Tip Trajectory

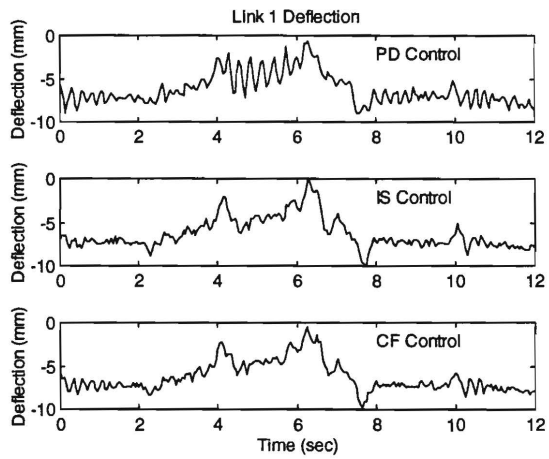

Figure 4: Link Deformation

The following series of experiments simulated an emergency stop condition. During the second leg of the trajectory, a vertical maneuver, the robot is commanded to stop midway through the path. The first series of experiments command the robot to execute a hard stop. This consists of commanding the robot to maintain the last command position at the time when the emergency stop was executed. Figure 5 illustrates the motion of the tip during this experiment. This figure shows that the smallest overshoot occurs when the filtering techniques are disabled. However, the shorter delay filter does have less overshoot than standard filtering techniques. The next series of experiments commanded the robot to decelerate using the last phase of a minimum time trajectory. Figure 6 shows that vibration can be eliminated if the robot is commanded to execute a smooth stop. However, the position of the tip of the robot overshoots beyond the maximum overshoot experienced when using a hard stop command.

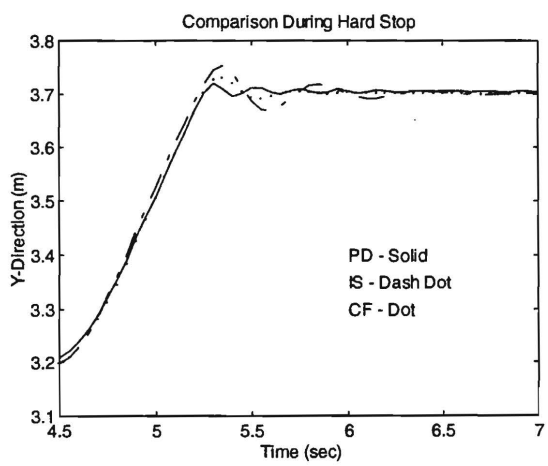

Figure 5: Tip Response During Hard Stops

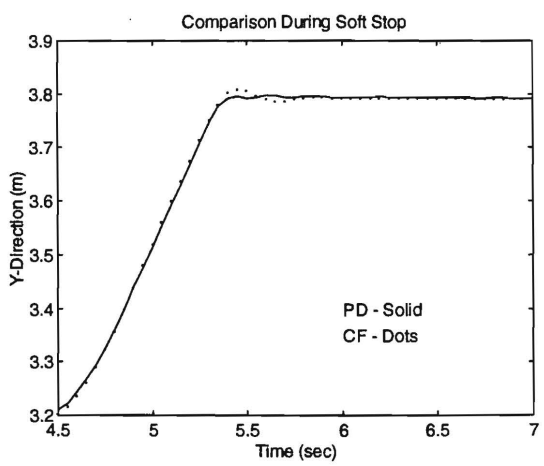

Figure 6: Tip Response During Soft Stops

Conclusions and Recommendations

This investigation provided two significant contributions. First, a new approach to teleoperation was developed during the term of this contract. A new approach to combining robotic and teleoperated commands provides the advantage of long reach manipulators without sacrificing the end point position accuracy of the teleoperation system. Under standard teleoperation commands, motion amplification must exist between the master and slave robots to provide full use of the slave robot's workspace. However, this motion amplification reduces the resolution of commands provided by the master robot. An alternative approach is to use robotic commands for global positioning of the slave robot and teleoperated commands for fine articulation manipulation.

The second contribution of this work was an investigation of the performance of a flexible robot during an emergency stop command. This investigation shows that, while filtering techniques reduce vibration during slewing motion, they increase the maximum overshoot during an emergency stop. This overshot may be reduced by either disabling the filtering or commanding the robot to decelerate instead of executing an immediate stop. Unfortunately, while soft stops reduce the residual vibration after an emergency stop, the final end point position may exceed the maximum overshoot experienced during a hard stop.

Additional work is proposed for the emergency stop condition. Alternative stopping paths may be available. One possibility is the inverse dynamic solution. There is the possibility of deriving the inverse dynamic solution to minimize the stopping distance based upon an initial velocity. Another possibility is through shorter delay filters. This investigation shows reduced overshoot with shorter delay filters. This would be advantageous in maintaining the benefit of vibration suppression during slewing motion.

Papers and Reports Written (Appended):

Love, L., and Magee, D., 1995, "Command Filtering and Path Planning for Autonomous/Teleoperated Control of Long Reach Flexible Manipulators," Submitted to the 1996 IEEE Robotics and Automation Conference.