MEAM 520

Haptic Rendering and Teleoperation

Katherine J. Kuchenbecker, Ph.D.

General Robotics, Automation, Sensing, and Perception Lab (GRASP)

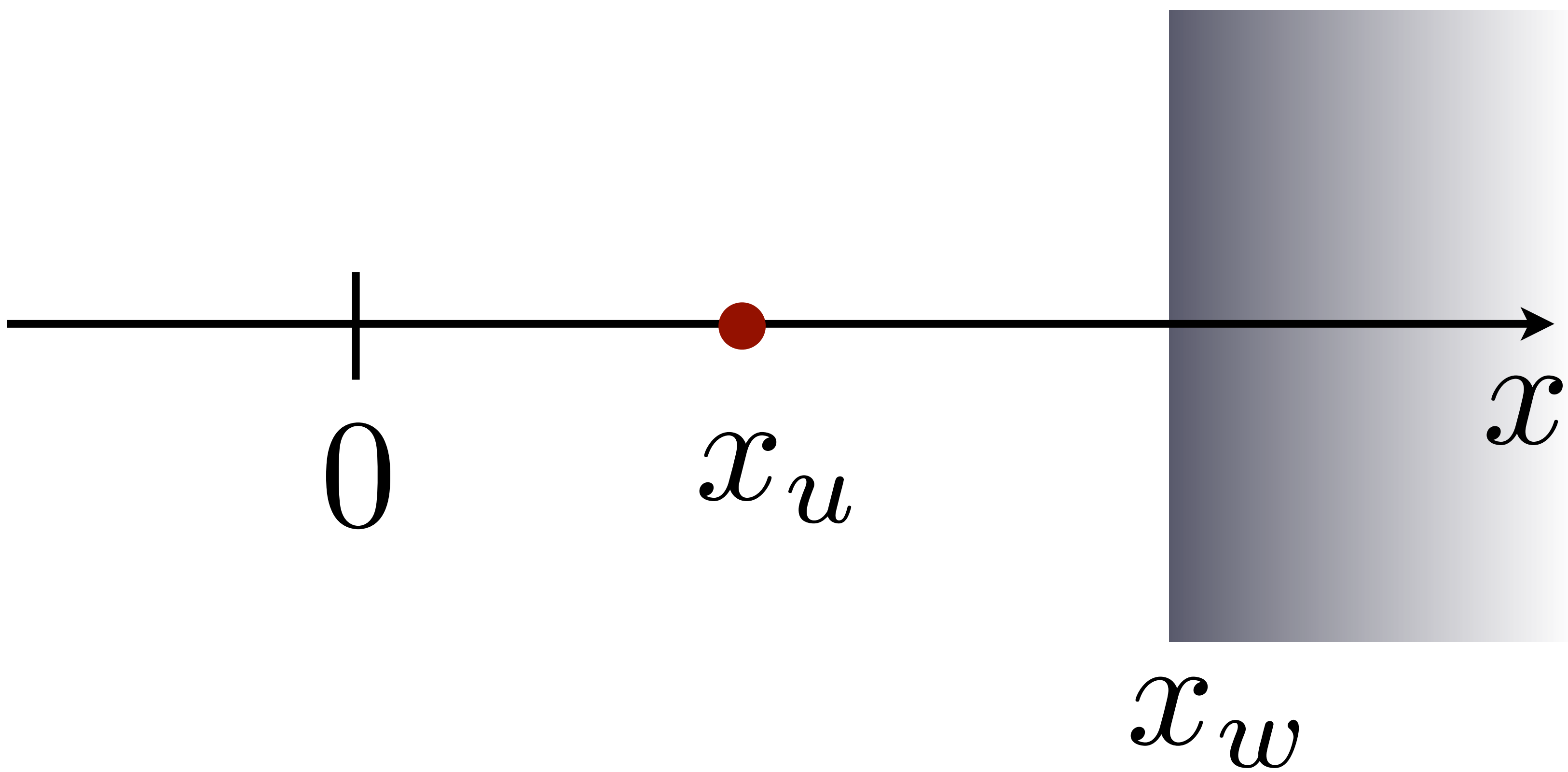
MEAM Department, SEAS, University of Pennsylvania

Lecture 17: November 15, 2012

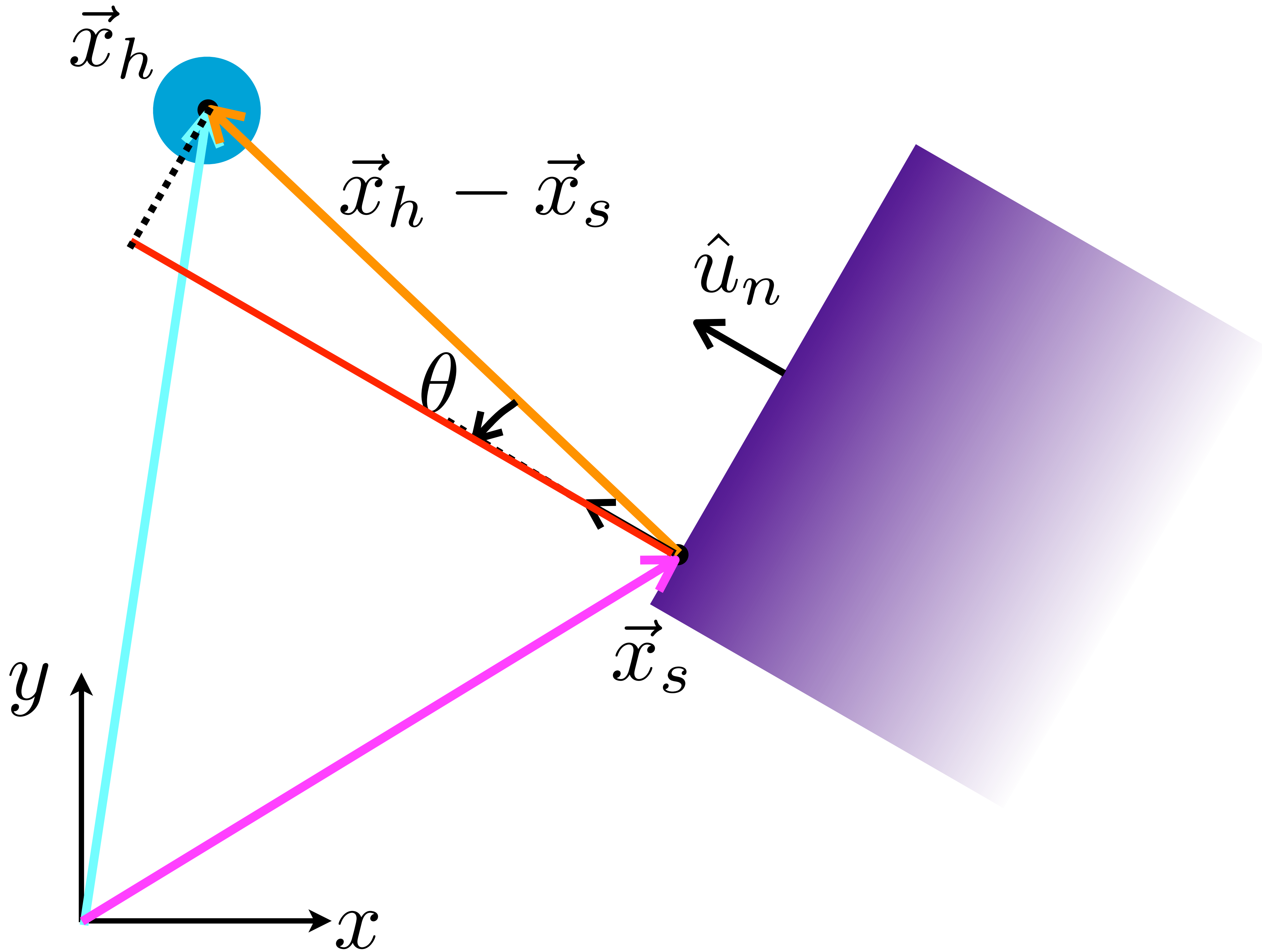
Haptic Rendering

Surface Properties: Hardness

How do you program a one-D virtual wall?

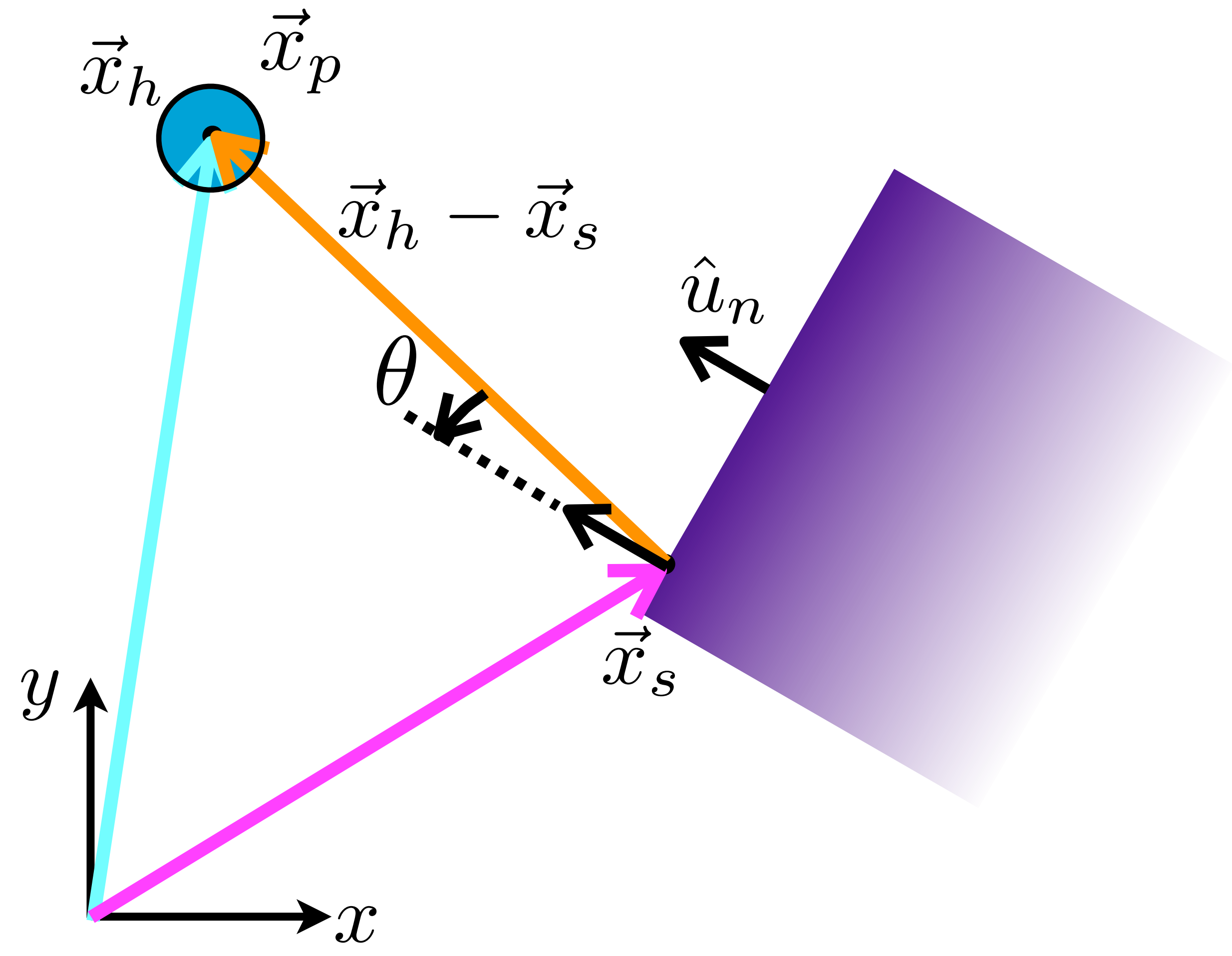


Standard Surface Rendering in 3D



$$\text{test } (\vec{x}_h - \vec{x}_s) \cdot \hat{u}_n = |\vec{x}_h - \vec{x}_s||\hat{u}_n| \cos\theta = d$$

Standard Surface Rendering in 3D



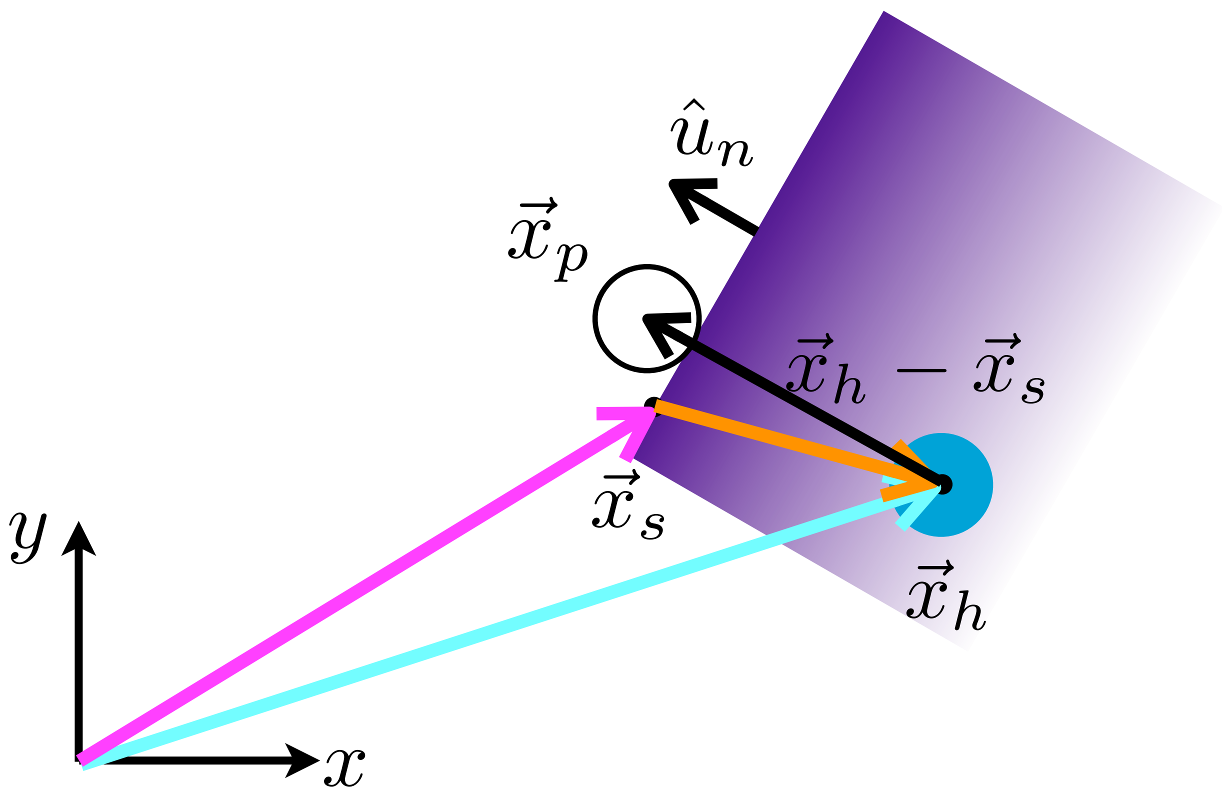
$$d = (\vec{x}_h - \vec{x}_s) \cdot \hat{u}_n$$

Calculate proxy position

$$\text{if } d \geq r_p$$

$$\vec{x}_p = \vec{x}_h, \vec{F} = \vec{0}$$

Standard Surface Rendering in 3D



$$d = (\vec{x}_h - \vec{x}_s) \cdot \hat{u}_n$$

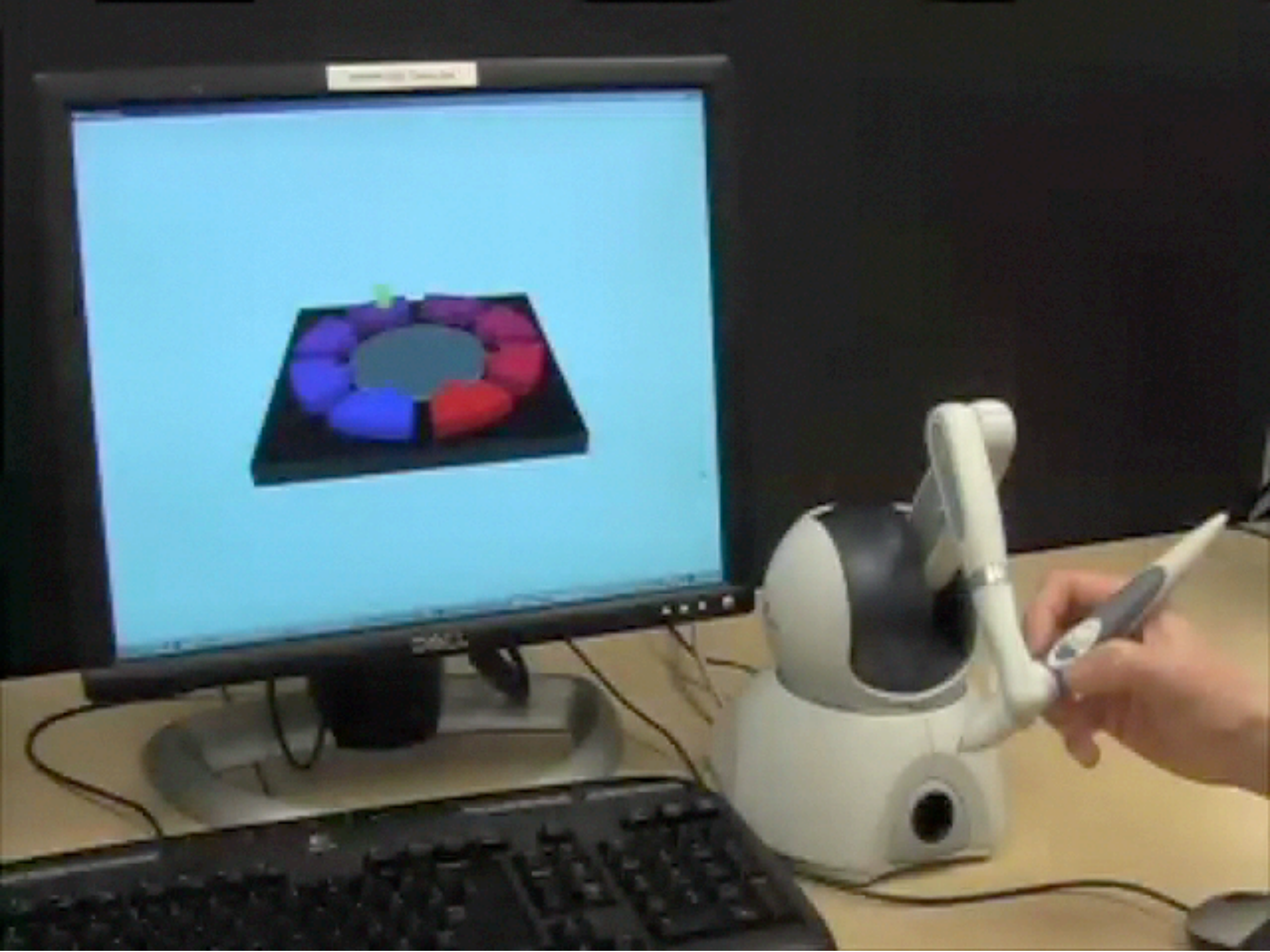
Calculate proxy position

$$\text{if } d < r_p$$

$$\vec{x}_p = \vec{x}_h - d\hat{u}_n + r_p\hat{u}_n$$

$$\vec{F} = -k_s(d - r_p)\hat{u}_n$$

Limited to about 2 N/mm



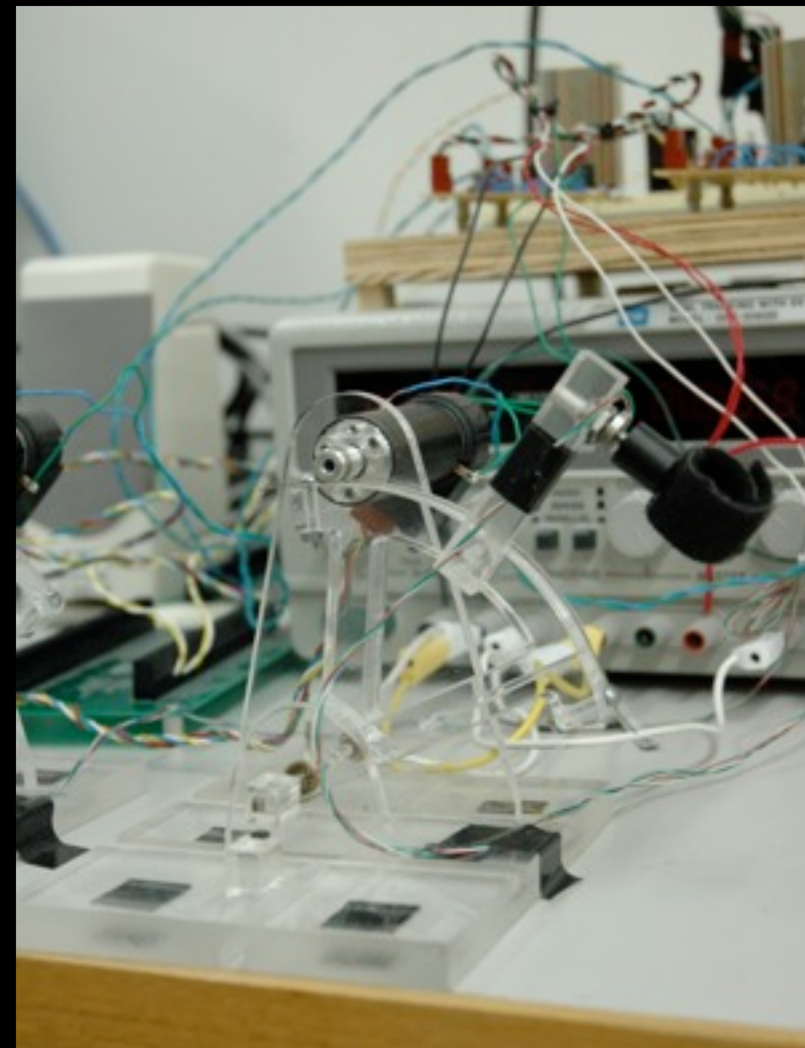
Surface Properties: Hardness

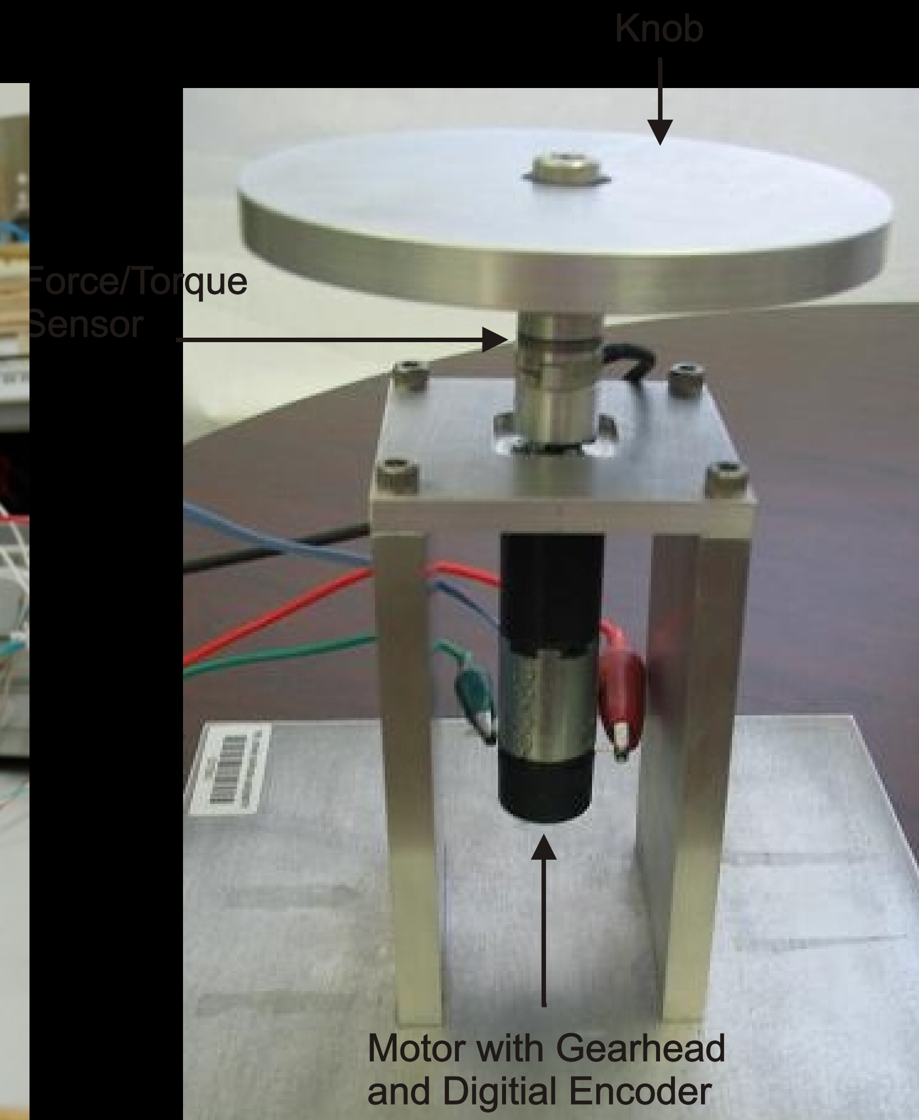
Why would you want to make a wall feel harder?

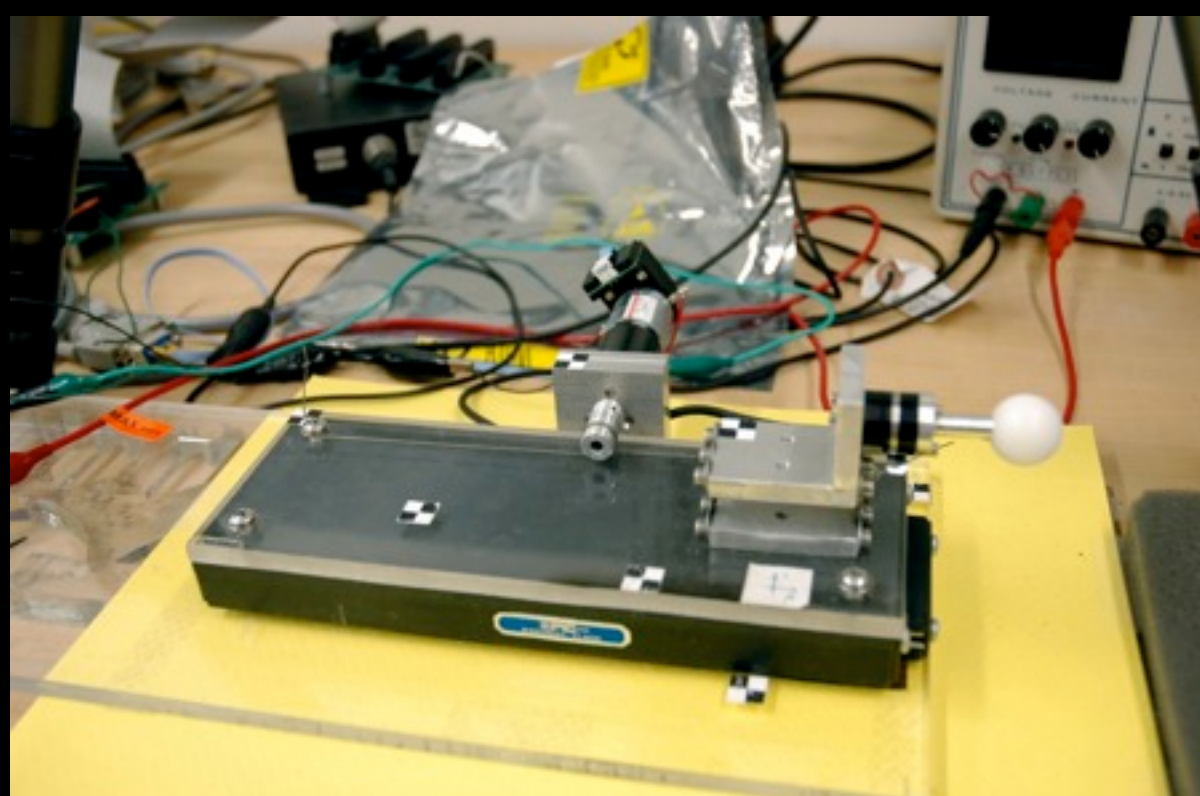
How could you make a wall feel harder?

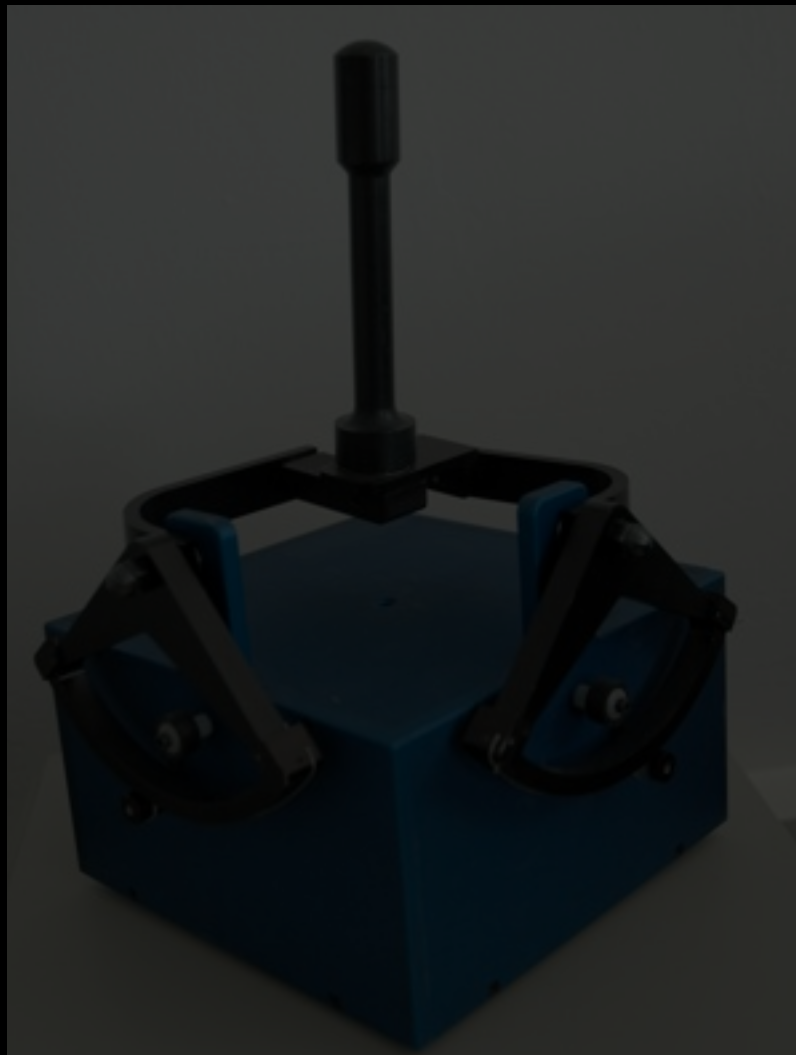
- Buy a better haptic interface.
- Perhaps try nonlinear stiffness.
- Add damping perpendicular to the plane, but only on the way in.
- Add an event-based force transient perpendicular to the plane for a short time after contact. The magnitude of the transient should scale with the magnitude of the perpendicular velocity.

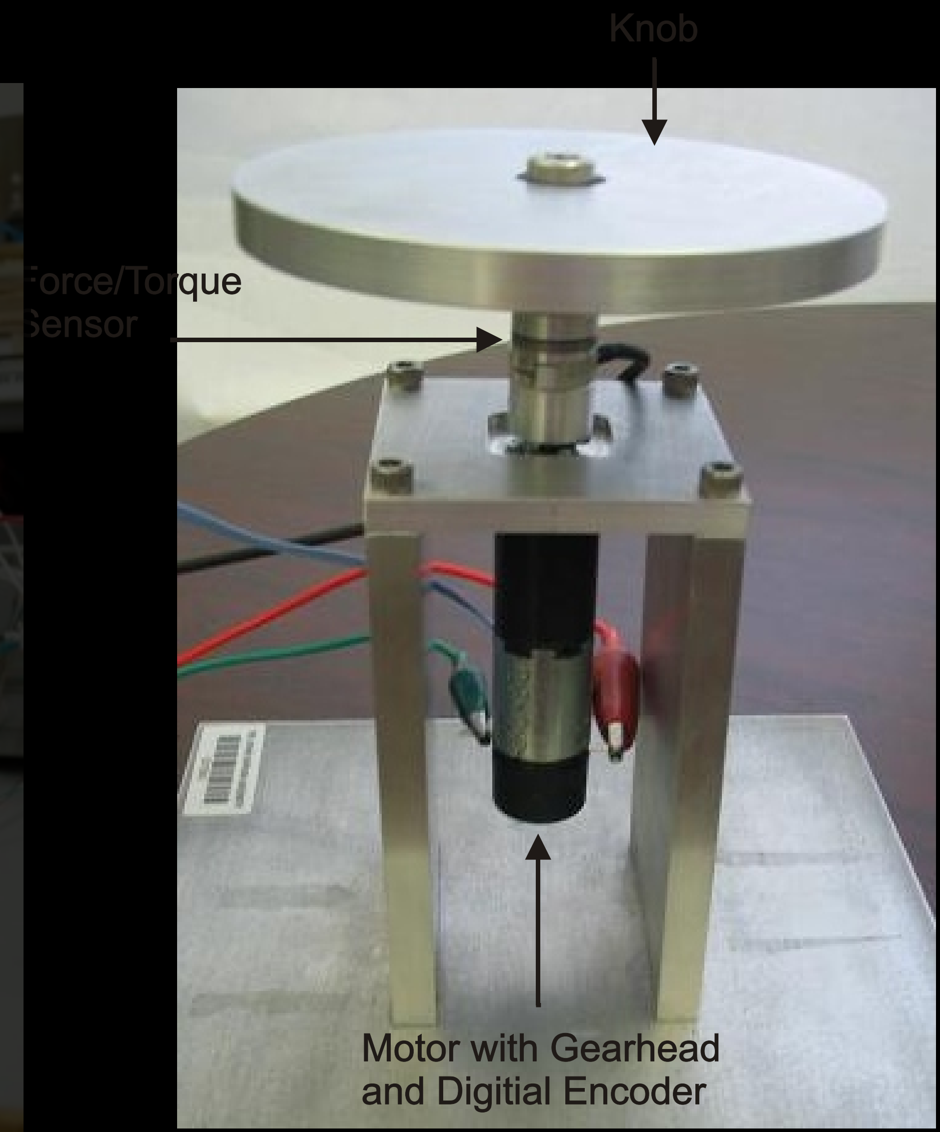
A sample custom haptic device

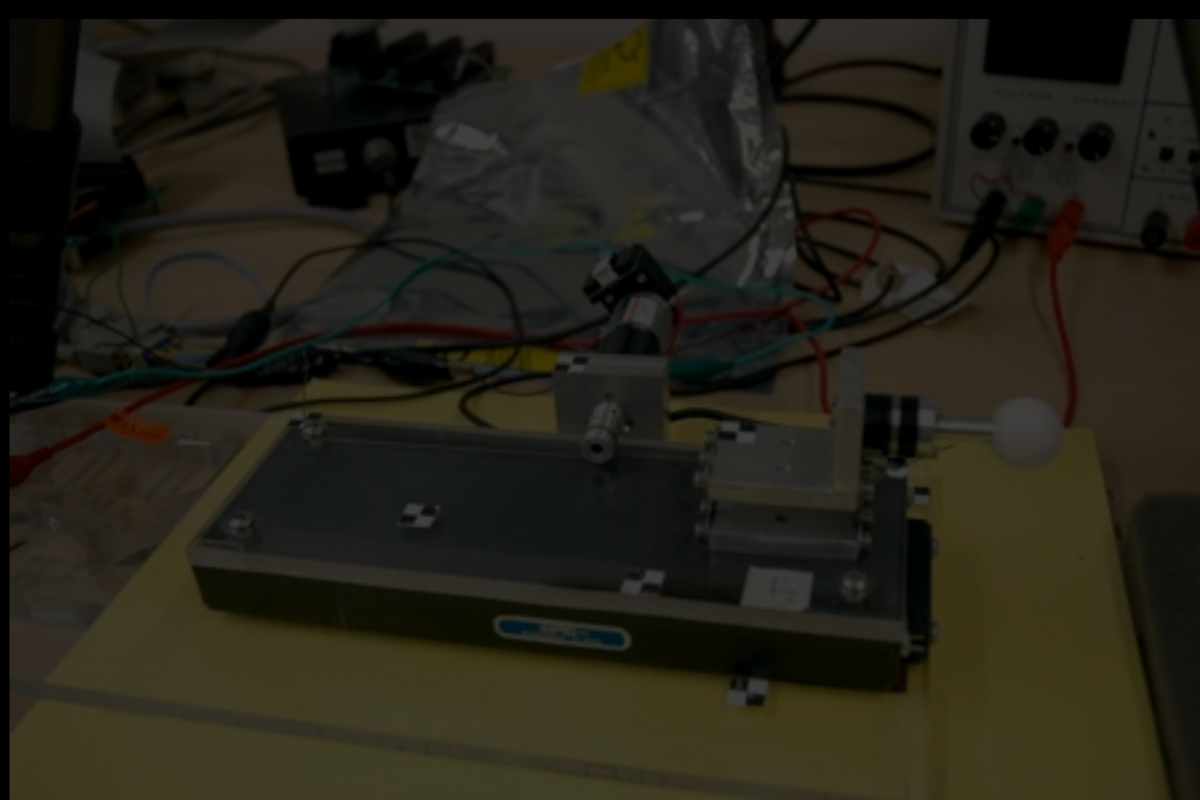


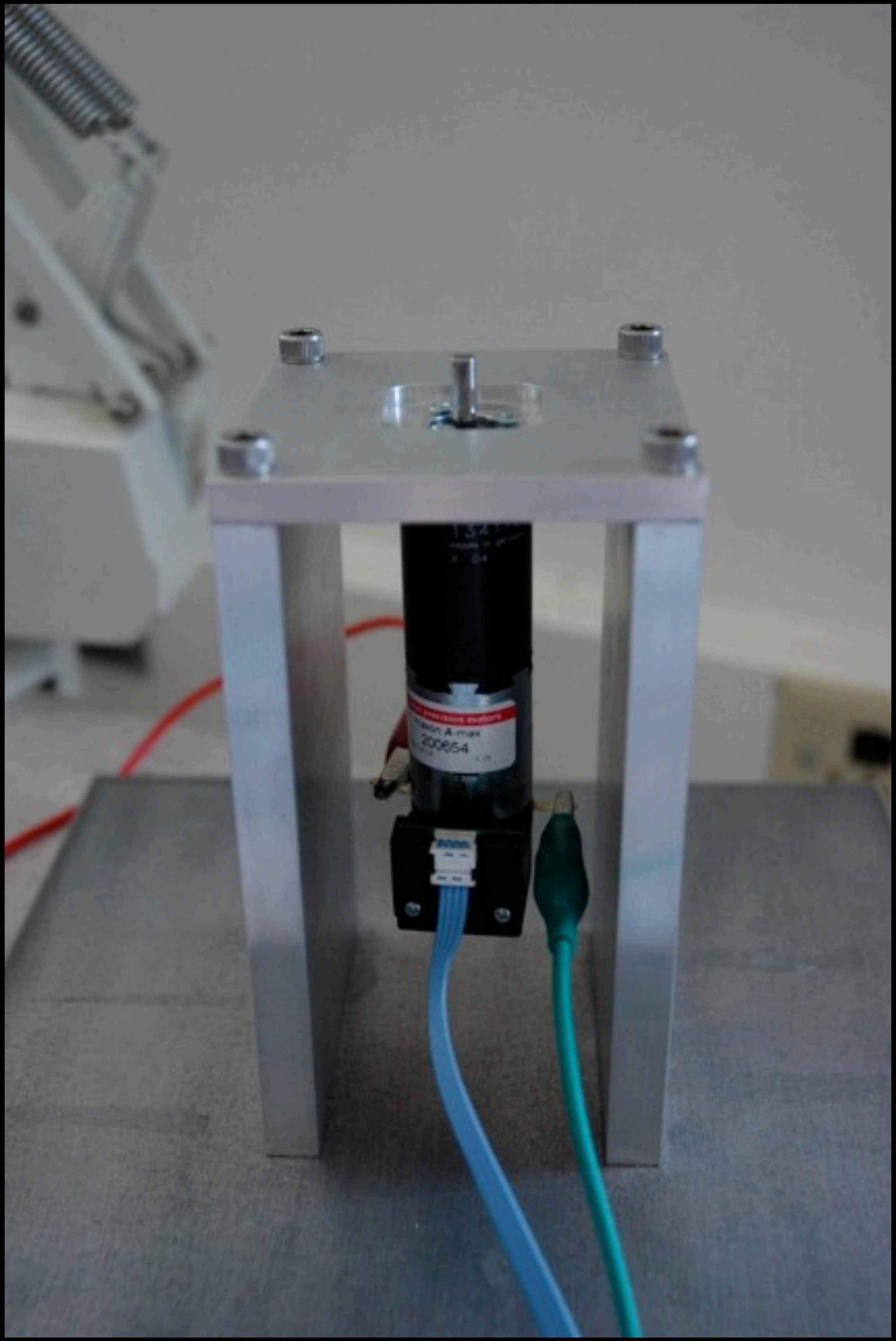

Knob
Force/Torque
Sensor
Motor with Gearhead
and Digitial Encoder

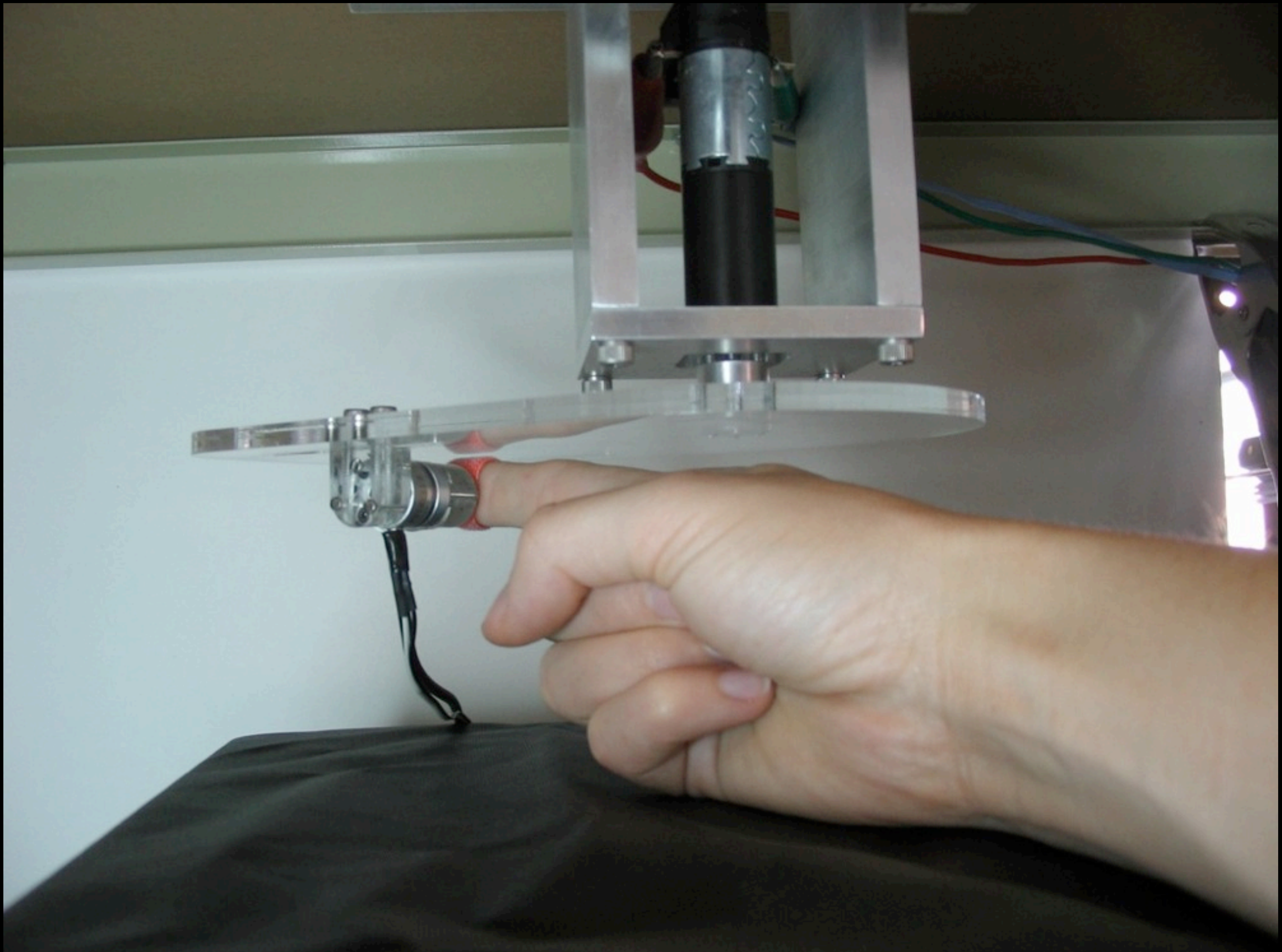




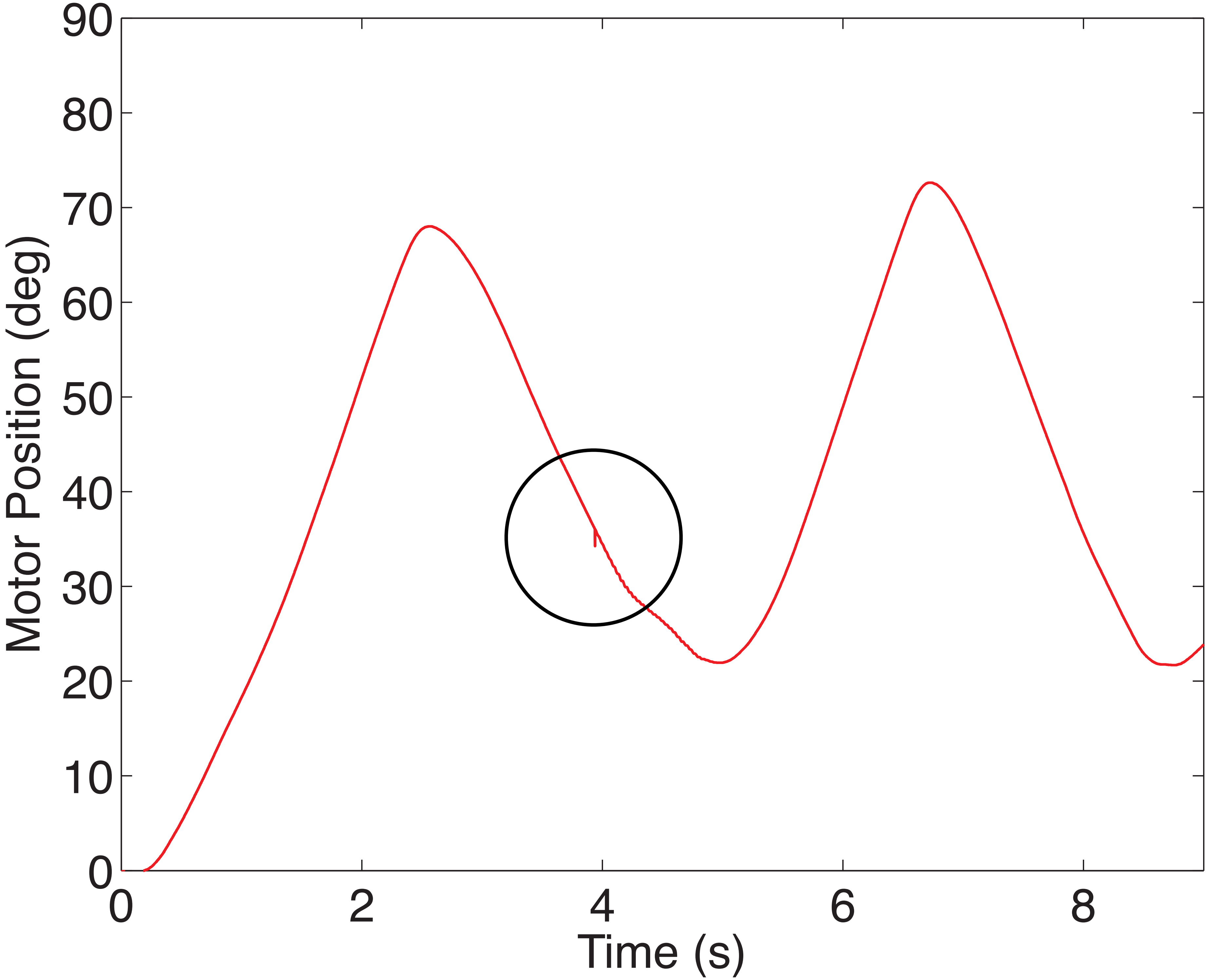

Knob
Force/Torque Sensor
Motor with Gearhead and Digitial Encoder



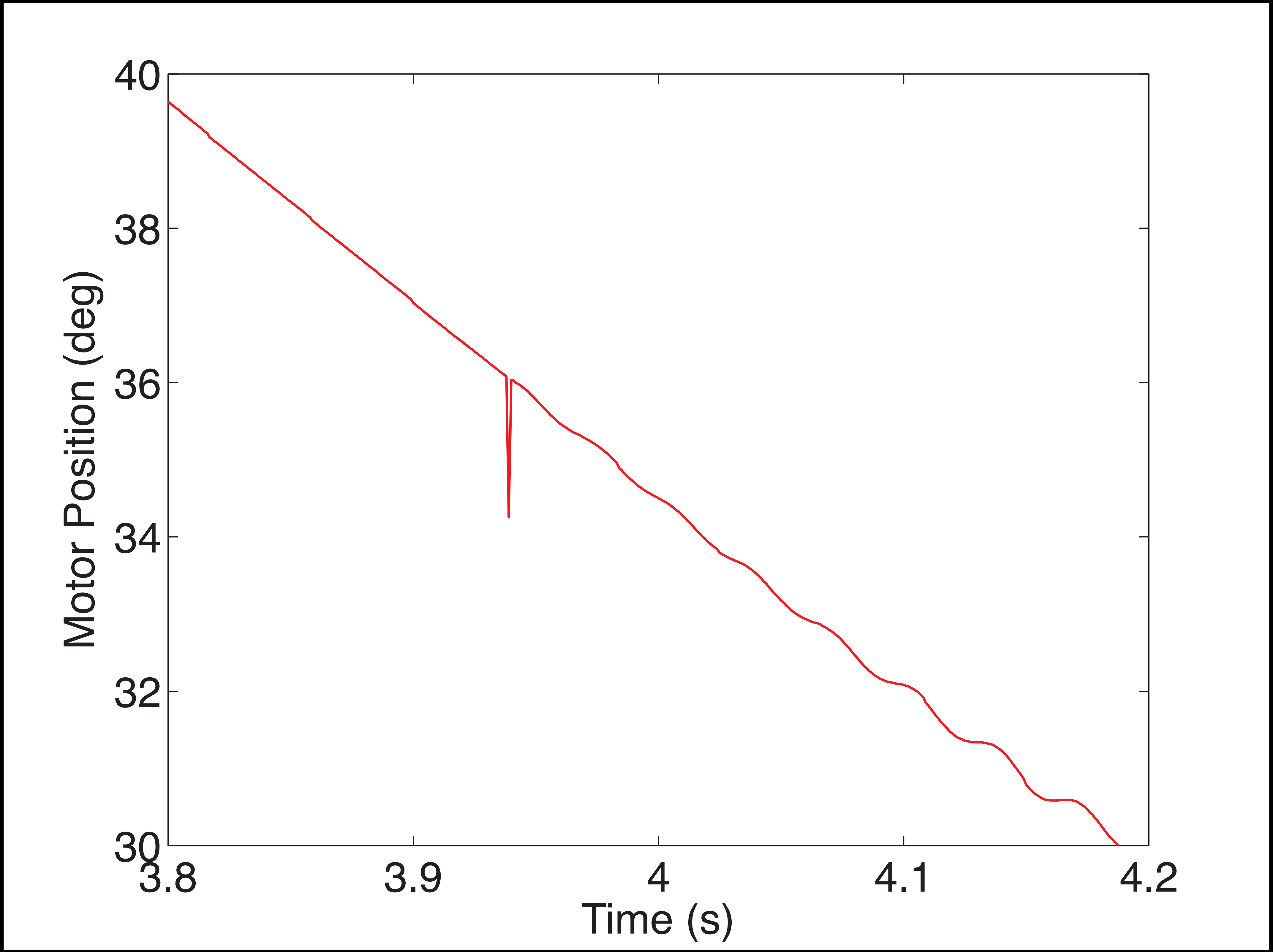

precision motors
A-max
200654



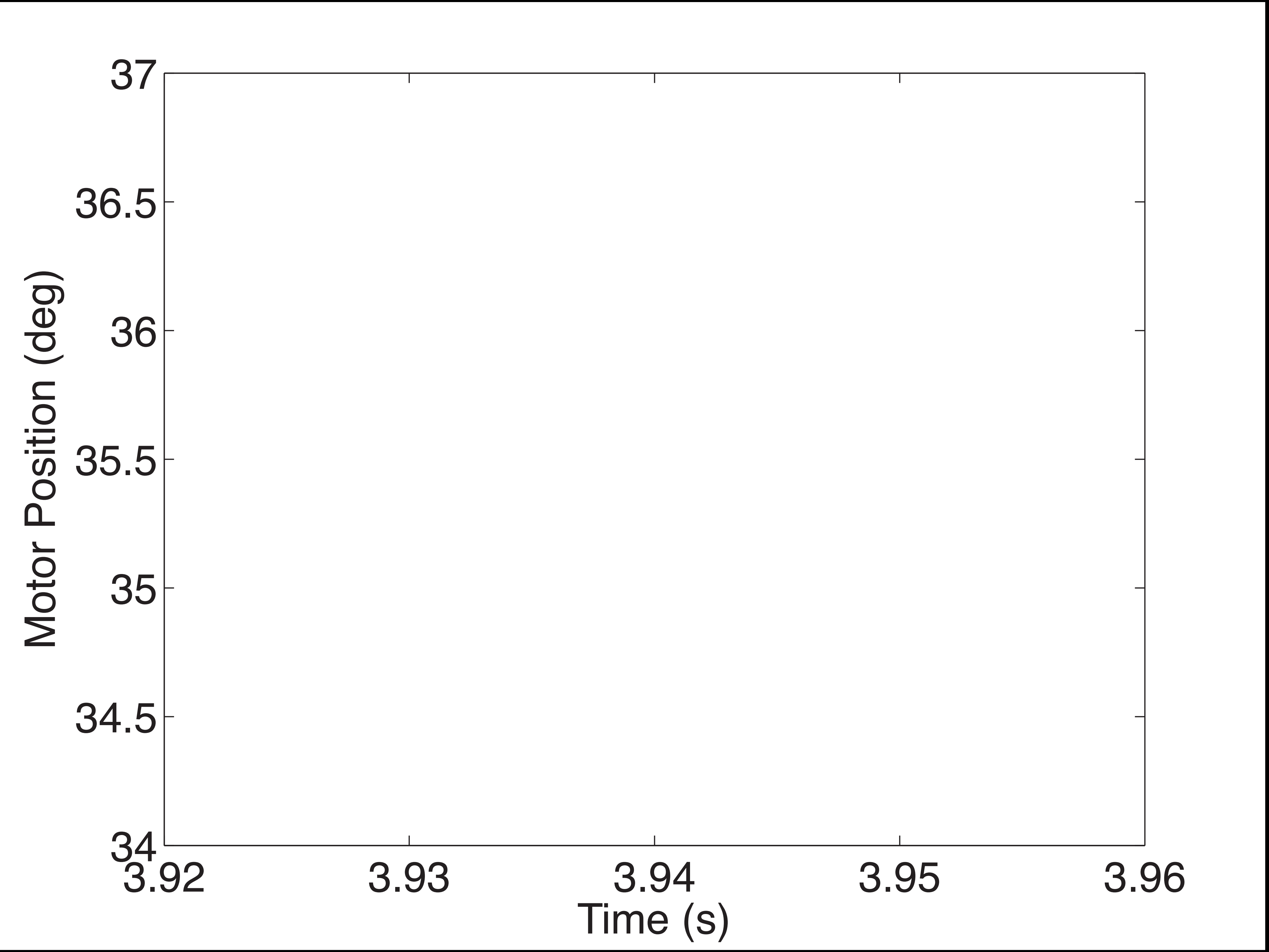
$$\tau_m = k_p(\theta_d - \underline{\theta_m}) + k_d(\omega_d - \underline{\omega_m})$$

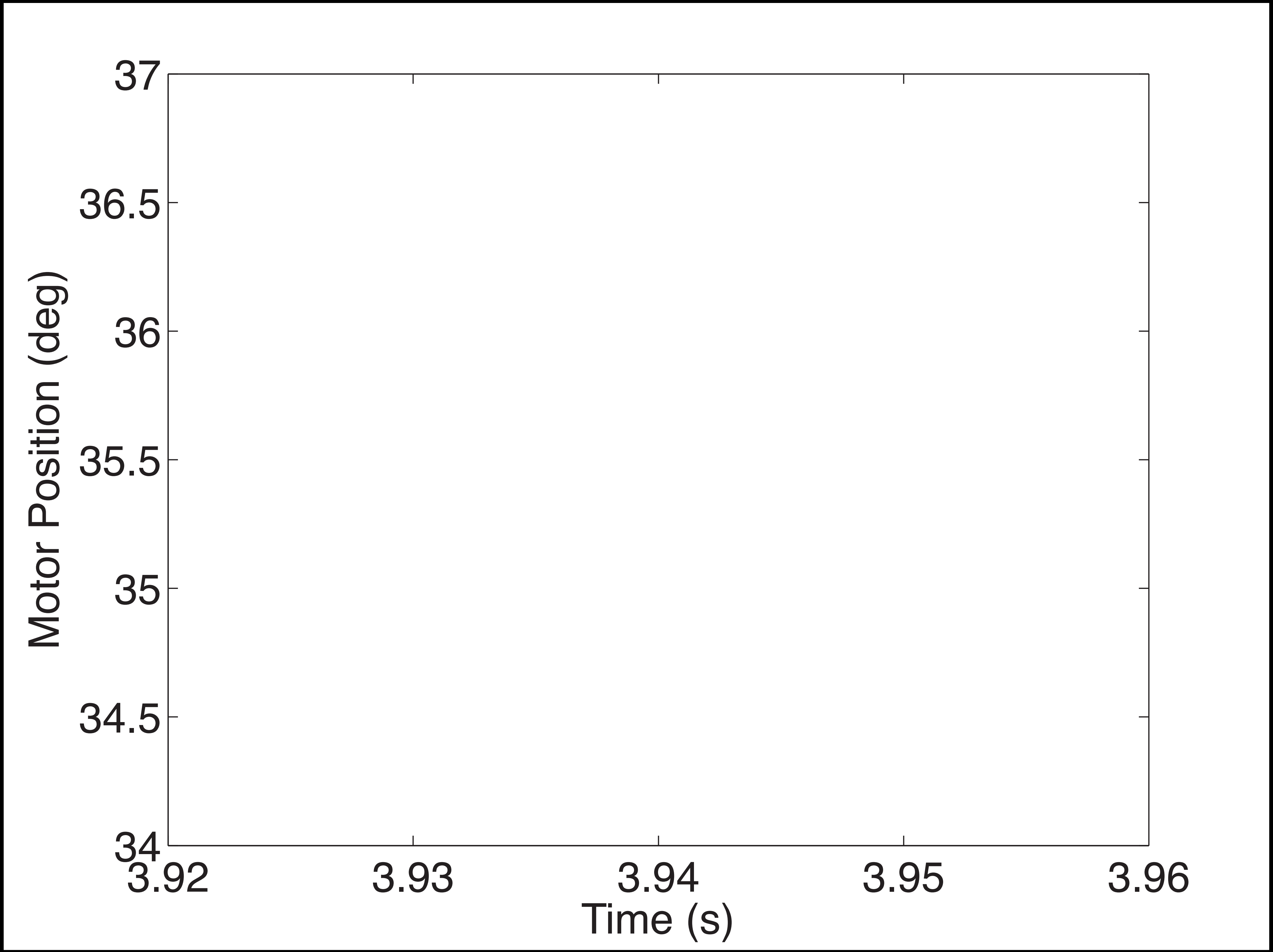


Motor Position (deg)
90
80
70
60
50
40
30
20
10
0
0
2
4
6
8
Time (s)

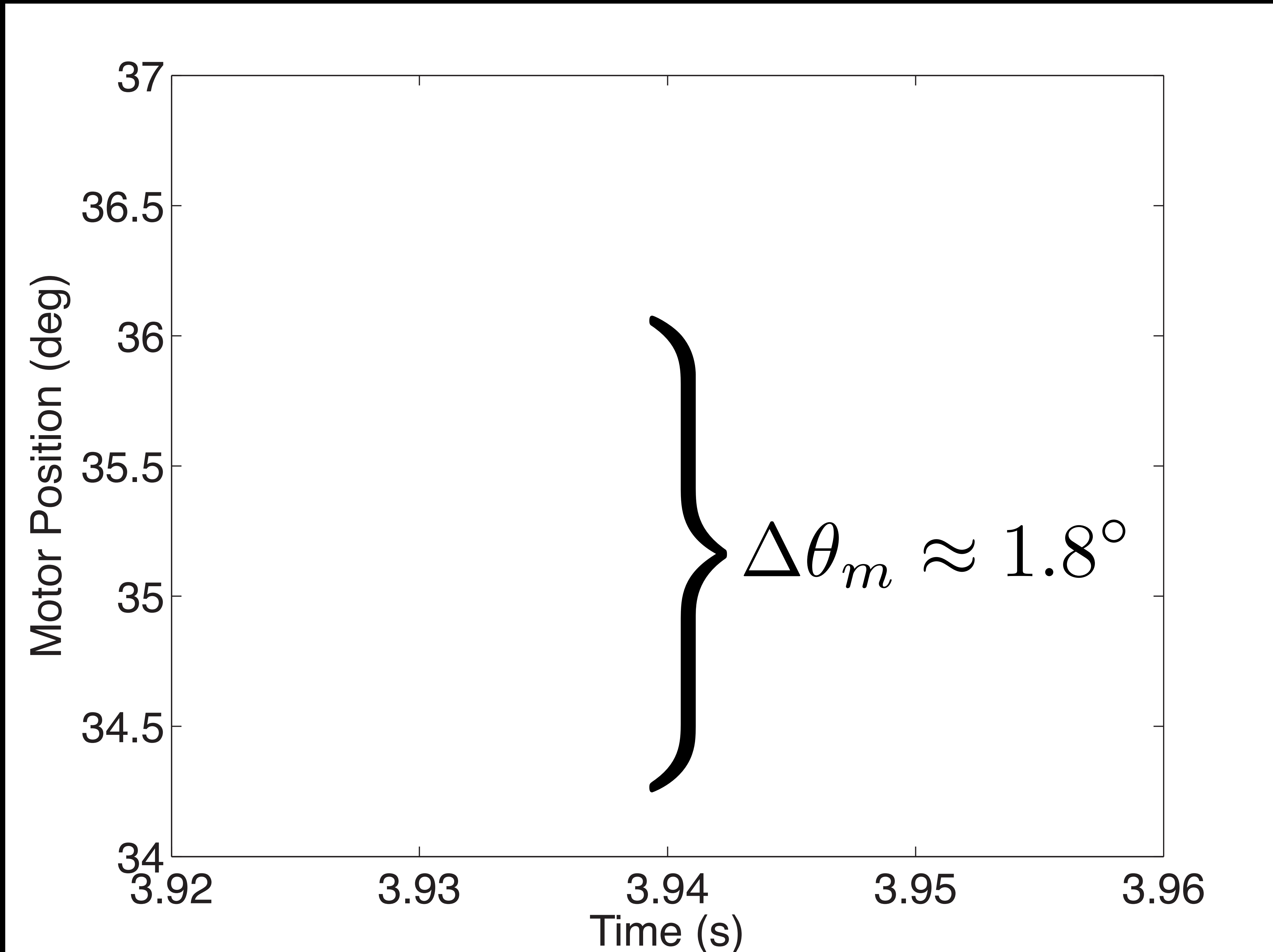


40
38
36
34
32
30
Motor Position (deg)
3.8
3.9
4
4.1
4.2
Time (s)


37
36.5
36
35.5
35
34.5
34
Motor Position (deg)
3.92
3.93
3.94
3.95
3.96
Time (s)

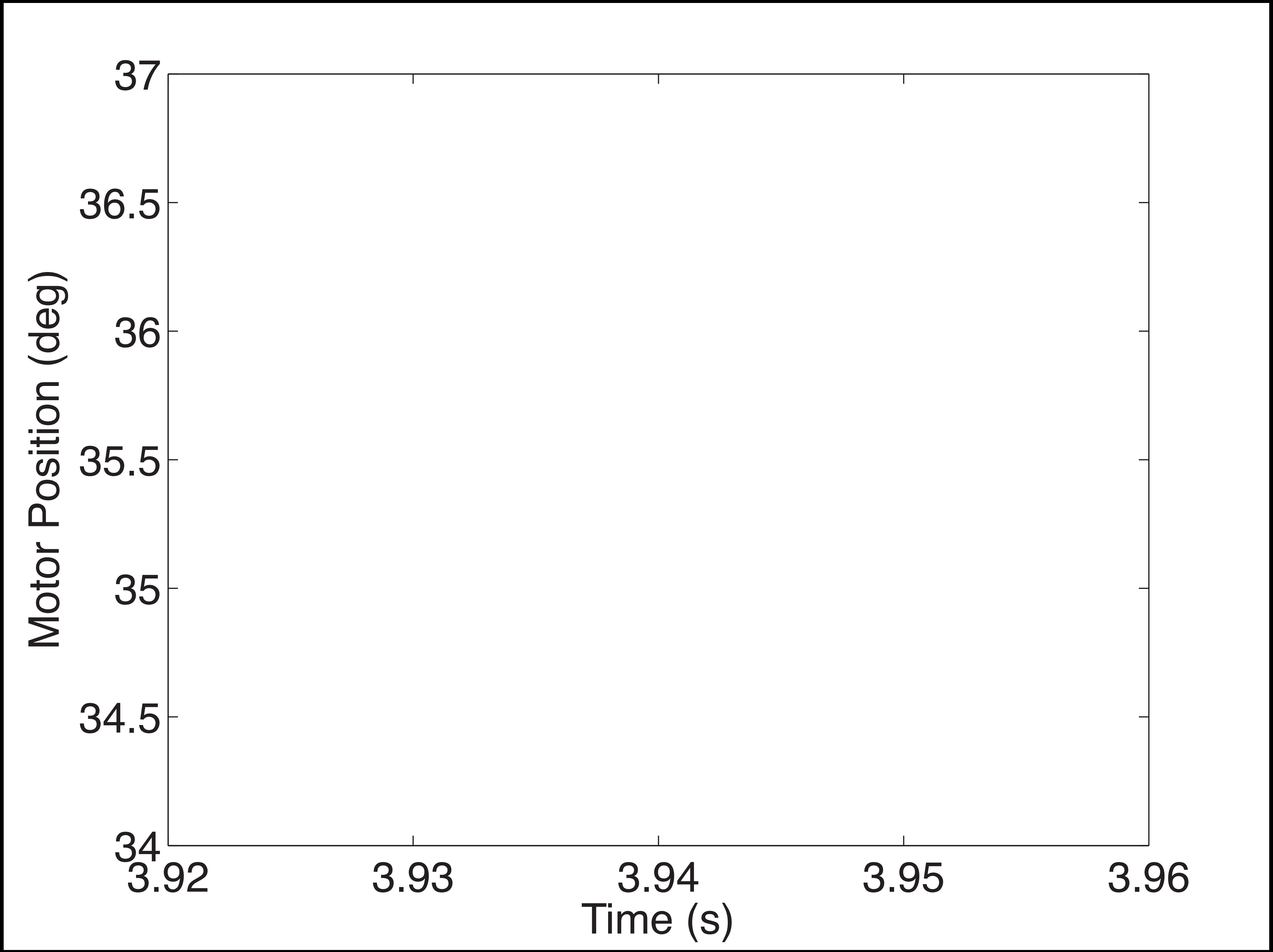


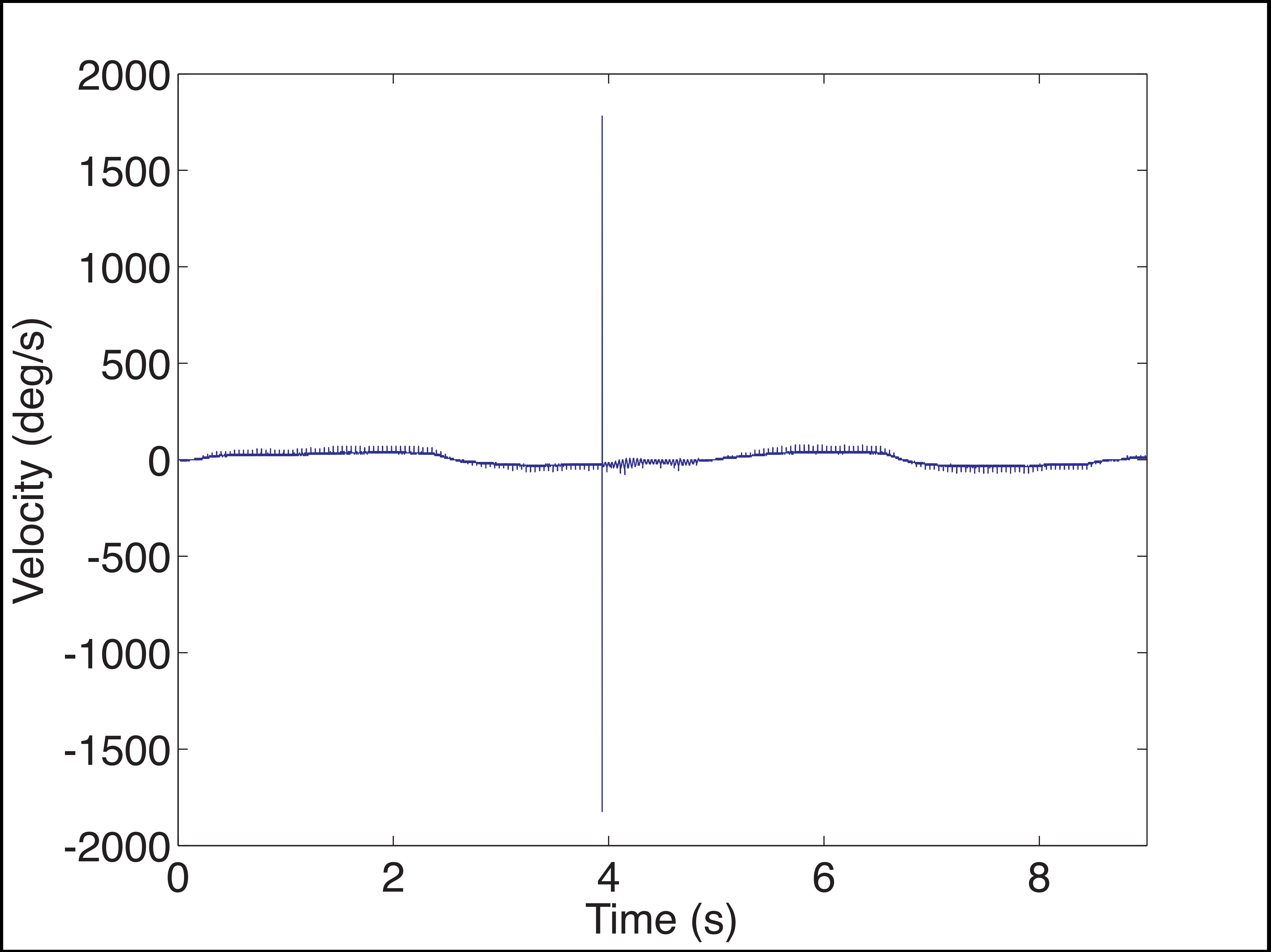
37
36.5
36
35.5
35
34.5
34
Motor Position (deg)
3.92
3.93
3.94
3.95
3.96
Time (s)



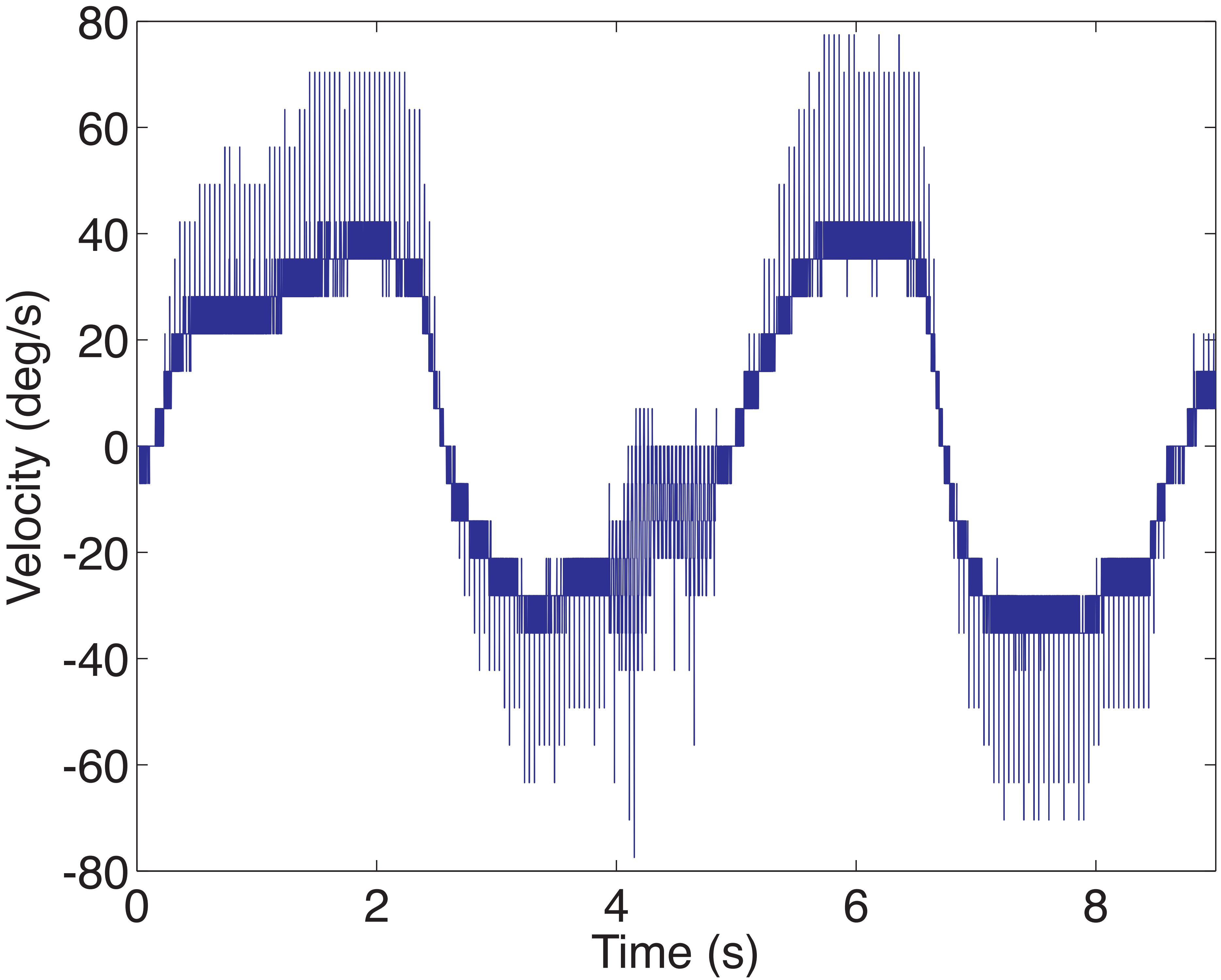
37
36.5
36
35.5
35
34.5
34
Motor Position (deg)
3.92
3.93
3.94
3.95
3.96
Time (s)
$\Delta\theta_m \approx 1.8^\circ$

$$\Delta\theta_m = 1.8^\circ \cdot \frac{51200 \text{ counts}}{360^\circ} = 256 \text{ counts}$$

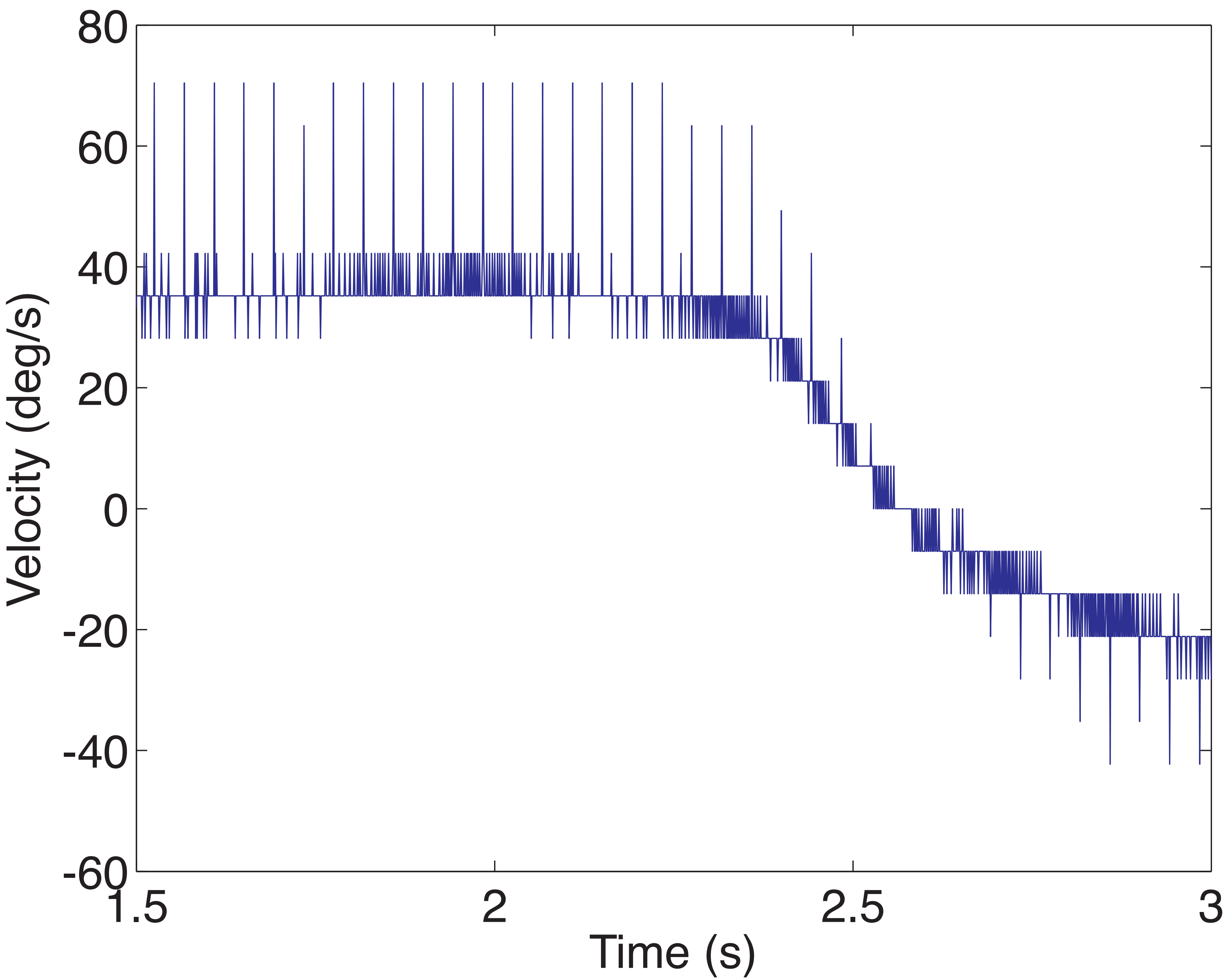

Motor Position (deg)
37
36.5
36
35.5
35
34.5
34
3.92
3.93
3.94
3.95
3.96
Time (s)



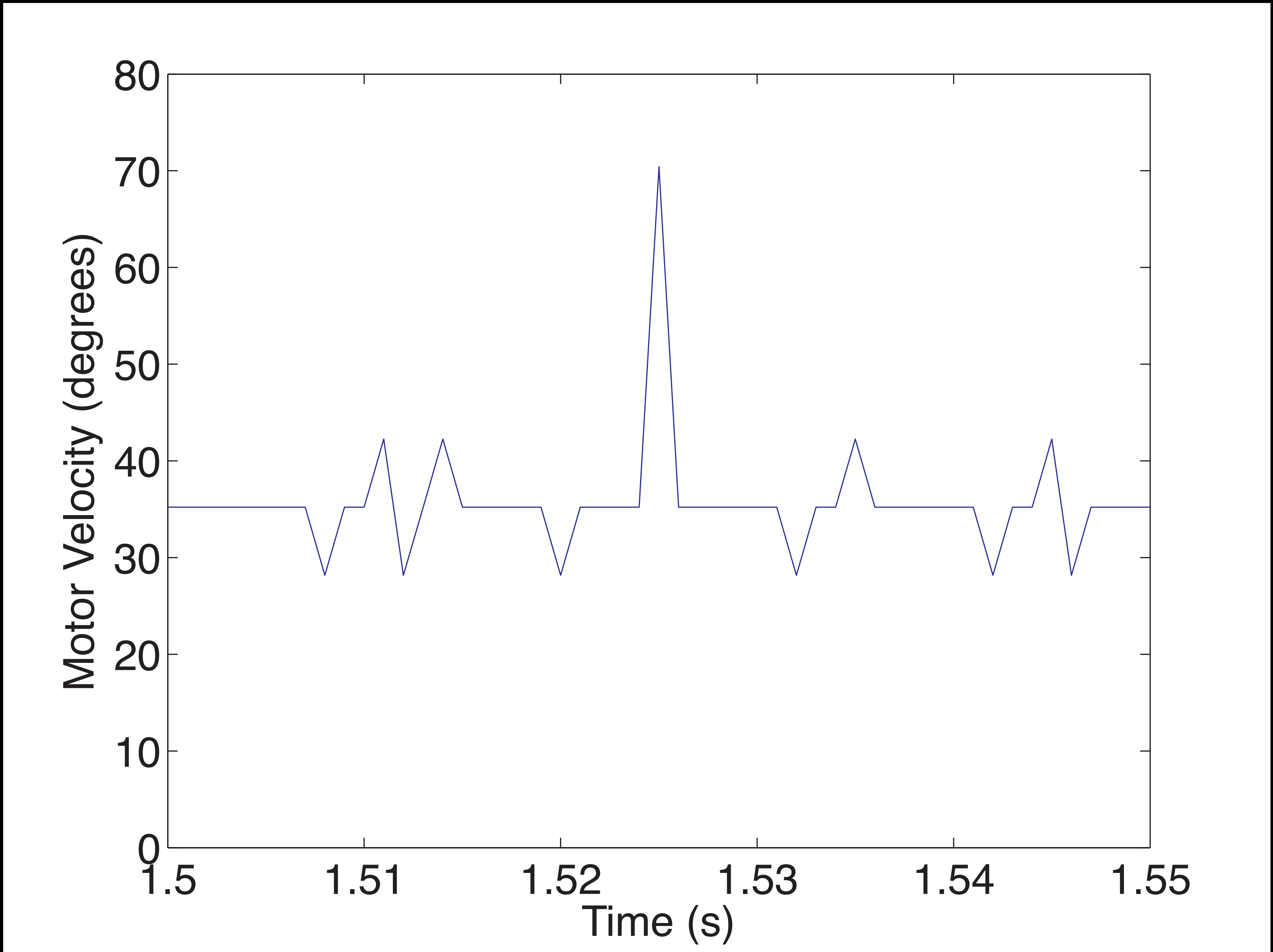
2000
1500
1000
500
0
-500
-1000
-1500
-2000
Velocity (deg/s)
0
2
4
6
8
Time (s)



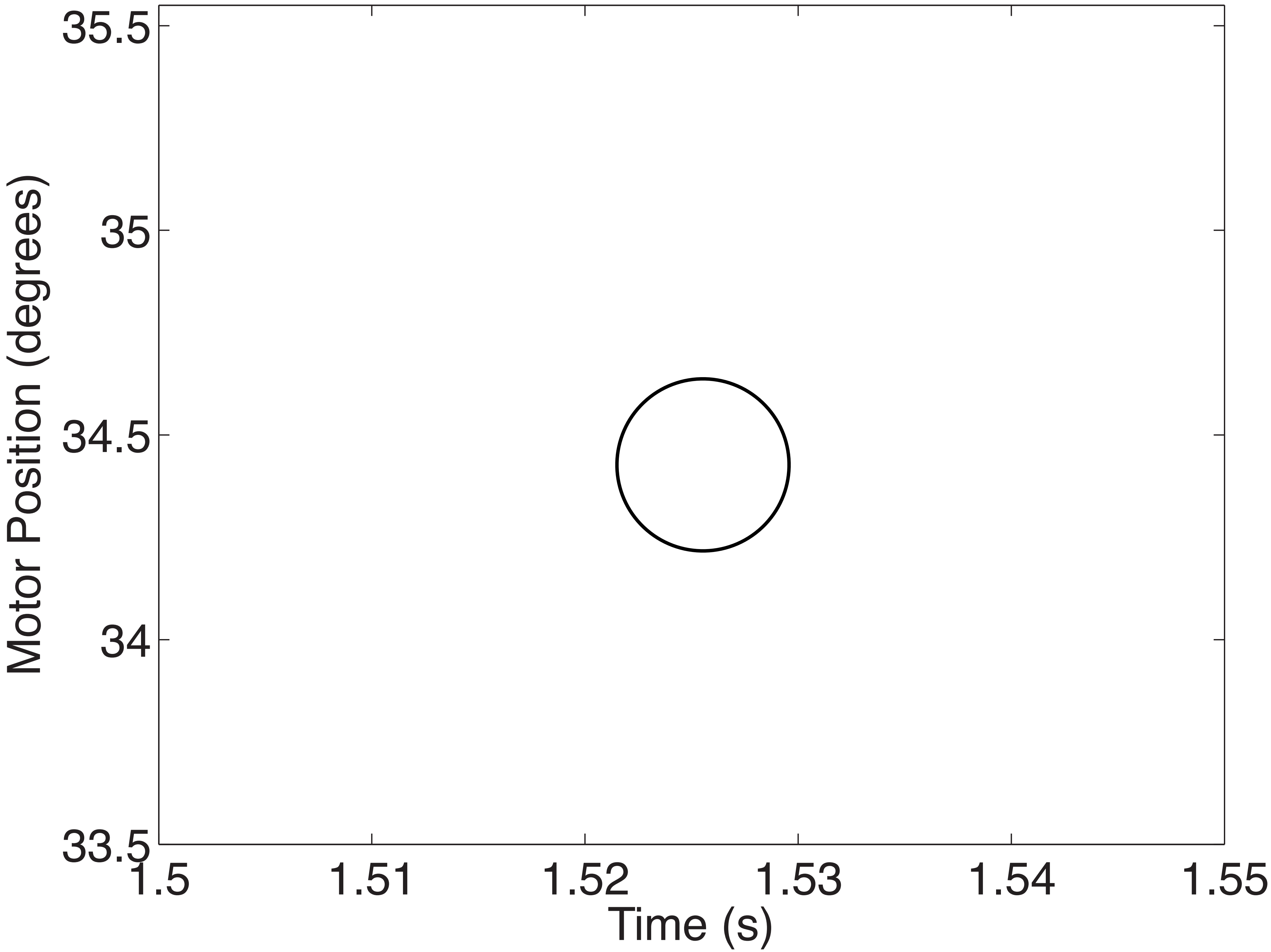
80
60
40
20
0
-20
-40
-60
-80
Velocity (deg/s)
0
2
4
6
8
Time (s)



80
60
40
20
0
-20
-40
-60
Velocity (deg/s)
1.5
2
2.5
3
Time (s)



Motor Velocity (degrees)
Time (s)
0
10
20
30
40
50
60
70
80
1.5
1.51
1.52
1.53
1.54
1.55



Motor Position (degrees)
35.5
35
34.5
34
33.5
1.5
1.51
1.52
1.53
1.54
1.55
Time (s)

Emacs@tinosa.local

```
            PostMessage(win, WM_DESTROY, NULL, NULL);
        }
        force_bias_initialize = true;

        // Configure quadrature board.
        ULStat = cbC7266Config (QUAD_BOARD_NUM, MOTOR_ROT, X4_QUAD, NORMAL_MODE, BINARY_ENCODING,
                INDEX_DISABLED, DISABLED, CARRY_BORROW, DISABLED);


        // Initialize the quadrature board
        LoadValue = 800000;
        ULStat = cbCLoad32 (QUAD_BOARD_NUM, COUNT1, LoadValue);
        ULStat = cbCLoad32 (QUAD_BOARD_NUM, COUNT2, LoadValue);
        ULStat = cbCLoad32 (QUAD_BOARD_NUM, COUNT3, LoadValue);
        ULStat = cbCLoad32 (QUAD_BOARD_NUM, COUNT4, LoadValue);

        ULStat = cbCLoad(QUAD_BOARD_NUM, PRESCALER1, 1);
        ULStat = cbCLoad(QUAD_BOARD_NUM, PRESCALER2, 1);
        ULStat = cbCLoad(QUAD_BOARD_NUM, PRESCALER3, 1);
        ULStat = cbCLoad(QUAD_BOARD_NUM, PRESCALER4, 1);

        // Get the high resolution counter's accuracy.
        QueryPerformanceFrequency(&ticksPerSecond);
        sprintf(clockResult, "There are %I64d ticks per second", ticksPerSecond.QuadPart);

        // Seed the random-number generator with current time.
        srand((unsigned)time(NULL));

        // Start the graphics timer
        SetTimer(win, 0, GRAPHIC_UPDATE_PERIOD, NULL);

        // Start the haptic thread
        g_HapticThread.Start(HAPTICS_UPDATE_PERIOD, Haptic_Function, NULL);

        return 0;


    case WM_MOUSEMOVE:

        SetCursor(LoadCursor(NULL, IDC_ARROW));
        return 0;


    case WM_DESTROY :

        // Stop the Haptic Thread
        g_HapticThread.Stop();
```

--(DOS)-- knob_07_01_05.cpp   24% L333   (C++ Abbrev)---------------------------

Emacs@tinosa.local

```cpp
/*************************************************************
 Haptic_Function
        This is the function that updates the system's forces
*************************************************************/


void __stdcall Haptic_Function(void *pv)

{
        int i;
        static double timer = 0;  // Used as a timer for several different purposes.

        ////////////////////////////////////////
        // *** TIMING ***

        // Cache the time of the previous haptic function call.
        lastTime = thisTime;

        // Find out what time it is now.  This information facilitates accurate velocity calculation.
        QueryPerformanceCounter(&thisTime);

        // Calculate time since last call in clock cycles and then convert to seconds.
        deltaTime.QuadPart = (thisTime.QuadPart - lastTime.QuadPart);
        deltaTimeS = (float) deltaTime.LowPart / (float) ticksPerSecond.QuadPart;

        ////////////////////////////////////////
        // *** FORCE/TORQUE MEASUREMENTS ***

        // Get present voltage values from f/t sensor
        RawVoltage(tempRawVoltage);

        // Filter voltage
        for (i=0 ; i<7 ; i++) {
                filteredRawVoltage[i] = LowPass1((double)1.0/(2.0*PI*50.0), deltaTimeS, (double)tempRawVol
tage[i],  (double)filteredRawVoltage[i]);
        }

        // Handle initialization of force/torque sensor
        if ((force_bias_initialize) && (filter_wait > 50))
        {
                if (Number_of_Samples < MAX_NUMBER_OF_SAMPLES) {
                        for (int CONV_r = 0; CONV_r < 7; CONV_r++) {
                                VoltageBiasTemp[CONV_r][Number_of_Samples] = filteredRawVoltage[CONV_r];
                        }
                        Number_of_Samples++;
```

--(DOS)-- knob_07_01_05.cpp   63% L918   (C++ Abbrev)--------------------------------

Emacs@tinosa.local

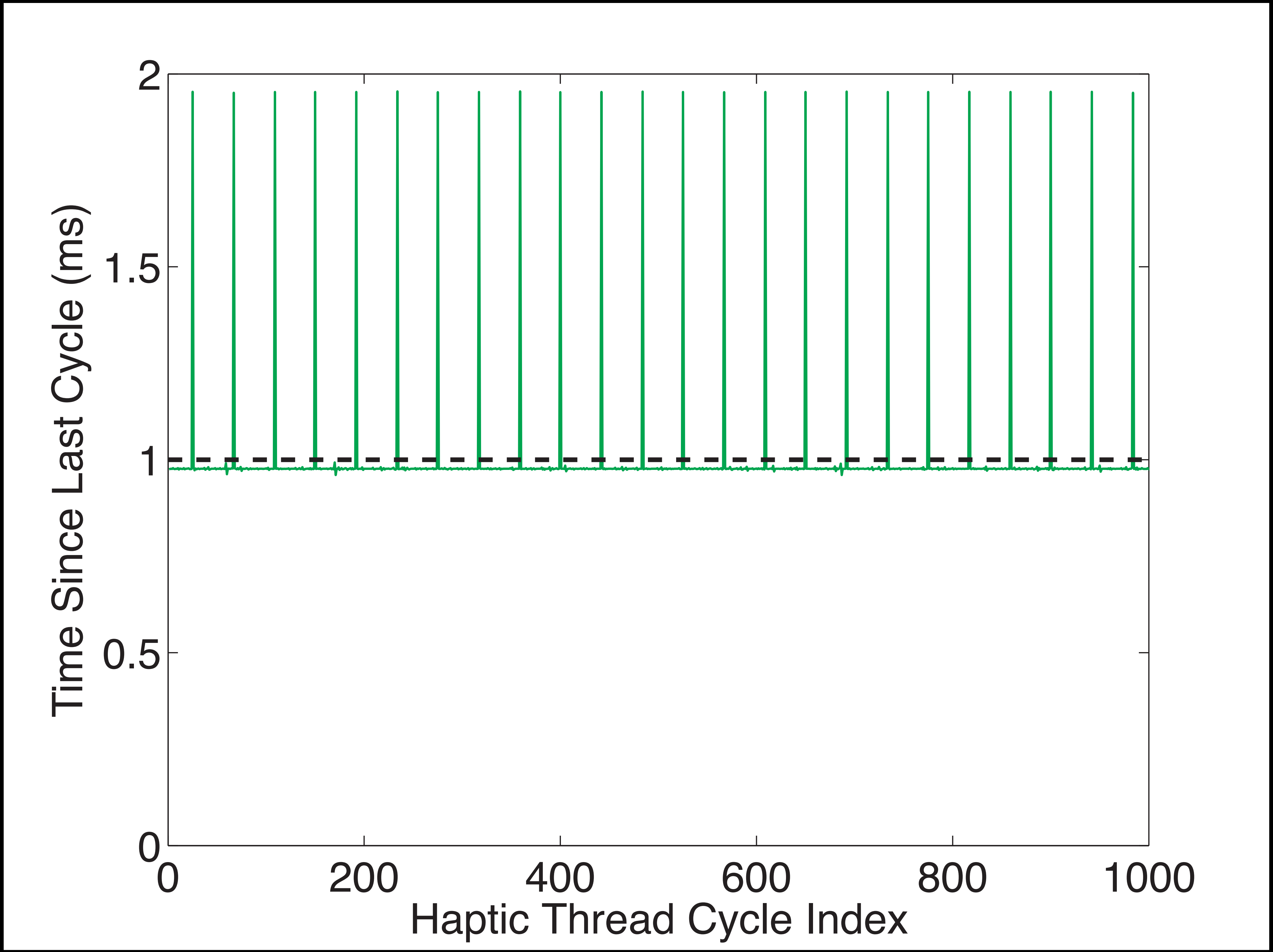
```cpp
        // *** MOTOR CONTROL ***

        // Save last position for velocity computation.
        lastPosDeg = curPosDeg;

        // Read in encoder signals from the QUAD04 board
        ULStat = cbCIn32 (QUAD_BOARD_NUM, MOTOR_ROT, &rot_cts);

        //Convert to signed counts
        rot_cts_signed =  rot_cts;

        // Convert signed counts to degrees
        curPos = rot_cts_signed - LoadValue;
        curPosDeg = curPos / CTS_PER_DEG;                     // Converts position to units of degrees

        // Check for freak position reads - if change is too much, discard this reading, and use the last 
one.
        if (fabs(curPosDeg - lastPosDeg) > 1) {
                curPosDeg = lastPosDeg;
        }

        // Compute velocity and low-pass filter.
        unfiltVelDeg = (curPosDeg - lastPosDeg) / deltaTimeS;
        curVelDeg = LowPass1(1/(2*PI*50), deltaTimeS, unfiltVelDeg, curVelDeg);

        // F/T transducer safety checks.
        if(fabs(FTValues[0])>200 || fabs(FTValues[1])>200 || fabs(FTValues[2])>500 || fabs(FTValues[3])>15
00 || fabs(FTValues[4])>1500 || fabs(FTValues[5])>2000) {
                // If over limits, make desired position present position with no output.
                desPosDeg = curPosDeg;
                desVelDeg = curVelDeg;
                current = 0;
                voltage = 0;
        } else {
                // Calculate the proxy's position and velocity during a trial for all of the different sta
tes.
                switch (state) {
                case waitingForParameters:
                case ready:
                        // Trial set will start soon.  Keep proxy at zero position.
                        proxyPosDeg = 0;
                        proxyVelDeg = 0;
                        break;
                case showingCommand:
                        // Next trial will start soon.  Keep proxy at its current position, sitting still.
                        proxyPosDeg = proxyPosDeg;
                        proxyVelDeg = 0;
```

--(DOS)** knob_07_01_05.cpp   69% L1001   (C++ Abbrev)------------------------------------

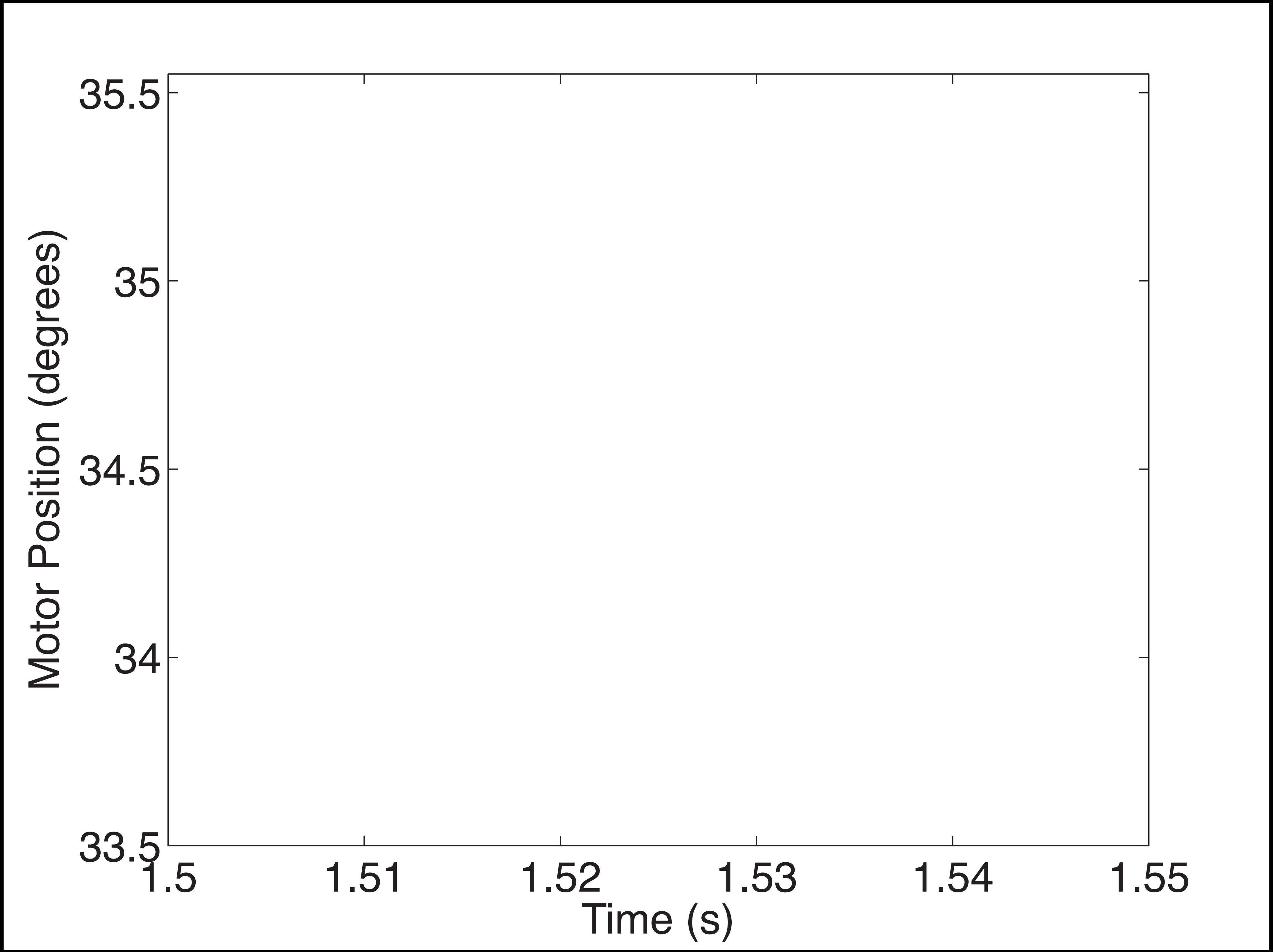
Emacs@tinosa.local

```
otFeedback ? 'D' : 'd', proprioceptiveFeedback ? 'P' : 'p', tactileFeedback ? 'T' : 't', commandPosDeg, co
mmandWidthDeg);
        //      return;
        //}

        // Output the desired values to the file.
        // Write parameters.
        fprintf(output_file, "subjectNumber = %d;\n\n", subjectNumber);
        fprintf(output_file, "setNumber = %d;\n\n", setNumber);
        fprintf(output_file, "trialNumber = %d;\n\n", trialNumber);
        fprintf(output_file, "lineFeedback = %d;\n\n", lineFeedback);
        fprintf(output_file, "dotFeedback = %d;\n\n", dotFeedback);
        fprintf(output_file, "proprioceptiveFeedback = %d;\n\n", proprioceptiveFeedback);
        fprintf(output_file, "tactileFeedback = %d;\n\n", tactileFeedback);
        fprintf(output_file, "commandPosition = %d;\n\n", commandPosDeg);
        fprintf(output_file, "commandWidth = %d;\n\n", commandWidthDeg);
        fprintf(output_file, "proxyAdmittance = %f;\n\n", proxyAdmittance);
        fprintf(output_file, "k = %f;\n\n", k);
        fprintf(output_file, "b = %f;\n\n", b);

        // Write the real time vector.
        fprintf(output_file, "clockTicksPerSecond = %I64d;\n\n", ticksPerSecond);
        fprintf(output_file, "tClock = [");
        for(i=0; i<dataIndex; i++) {
                fprintf(output_file, "%I64d\t", timeArray[i]);
        }
        fprintf(output_file, "]' - %I64d;\n", timeArray[0]);
        fprintf(output_file, "t = tClock / clockTicksPerSecond;\n\n");

        // Write time-varying data.
        fprintf(output_file, "dacVoltage = [");
        for(i=0; i<dataIndex; i++) {
                fprintf(output_file, "%.9f\t", dacVoltageArray[i]);
        }
        fprintf(output_file, "]';\n\n");

        fprintf(output_file, "fingerForce = [");
        for(i=0; i<dataIndex; i++) {
                fprintf(output_file, "%.9f\t", fingerForceArray[i]);
        }
        fprintf(output_file, "]';\n\n");

        fprintf(output_file, "motorPosition = [");
        for(i=0; i<dataIndex; i++) {
                fprintf(output_file, "%.9f\t", motorPositionArray[i]);
        }
        fprintf(output_file, "]';\n\n");
```

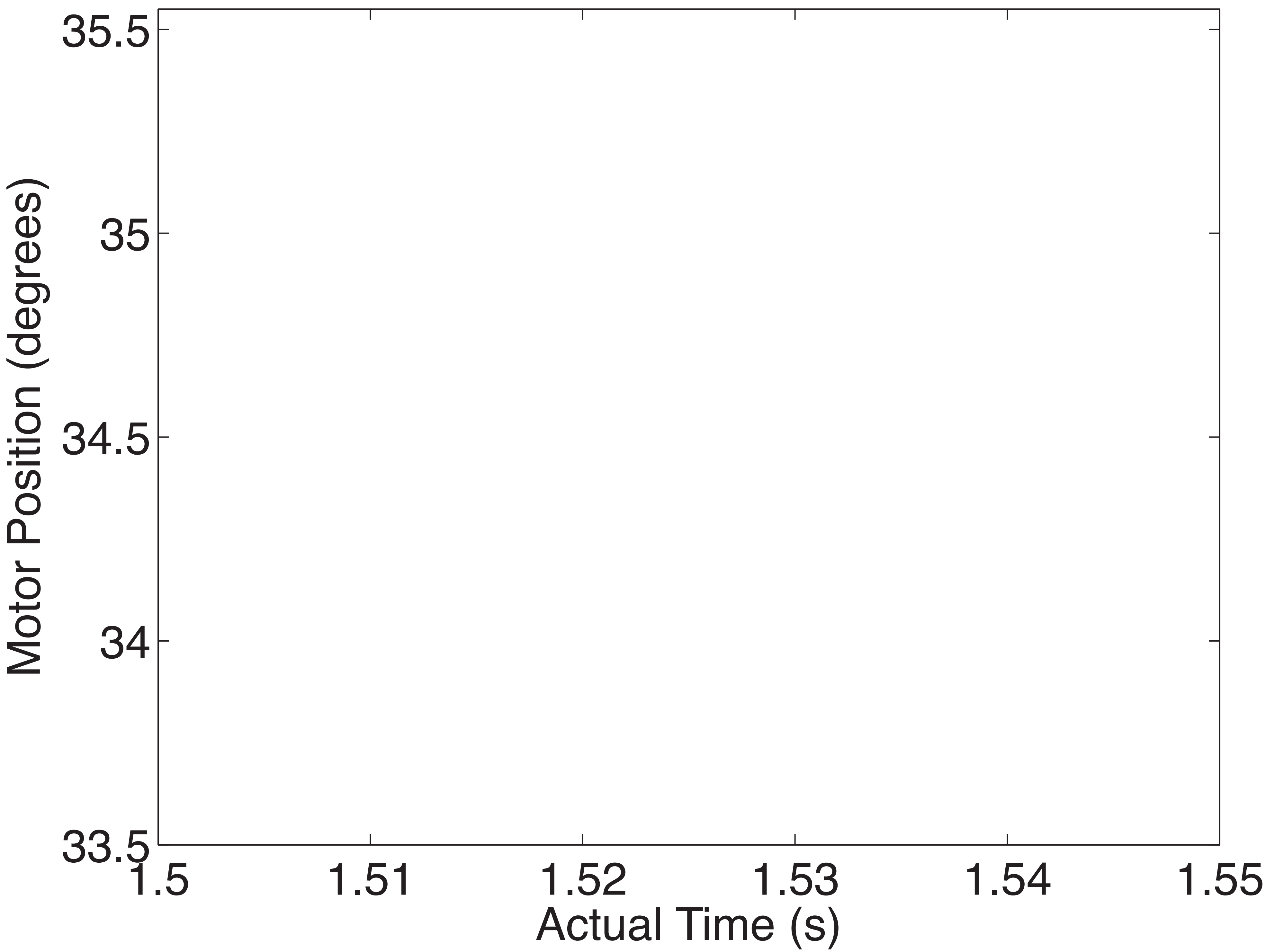
--(DOS)-- knob_07_01_05.cpp 93% L1346 (C++ Abbrev)--------------------------------

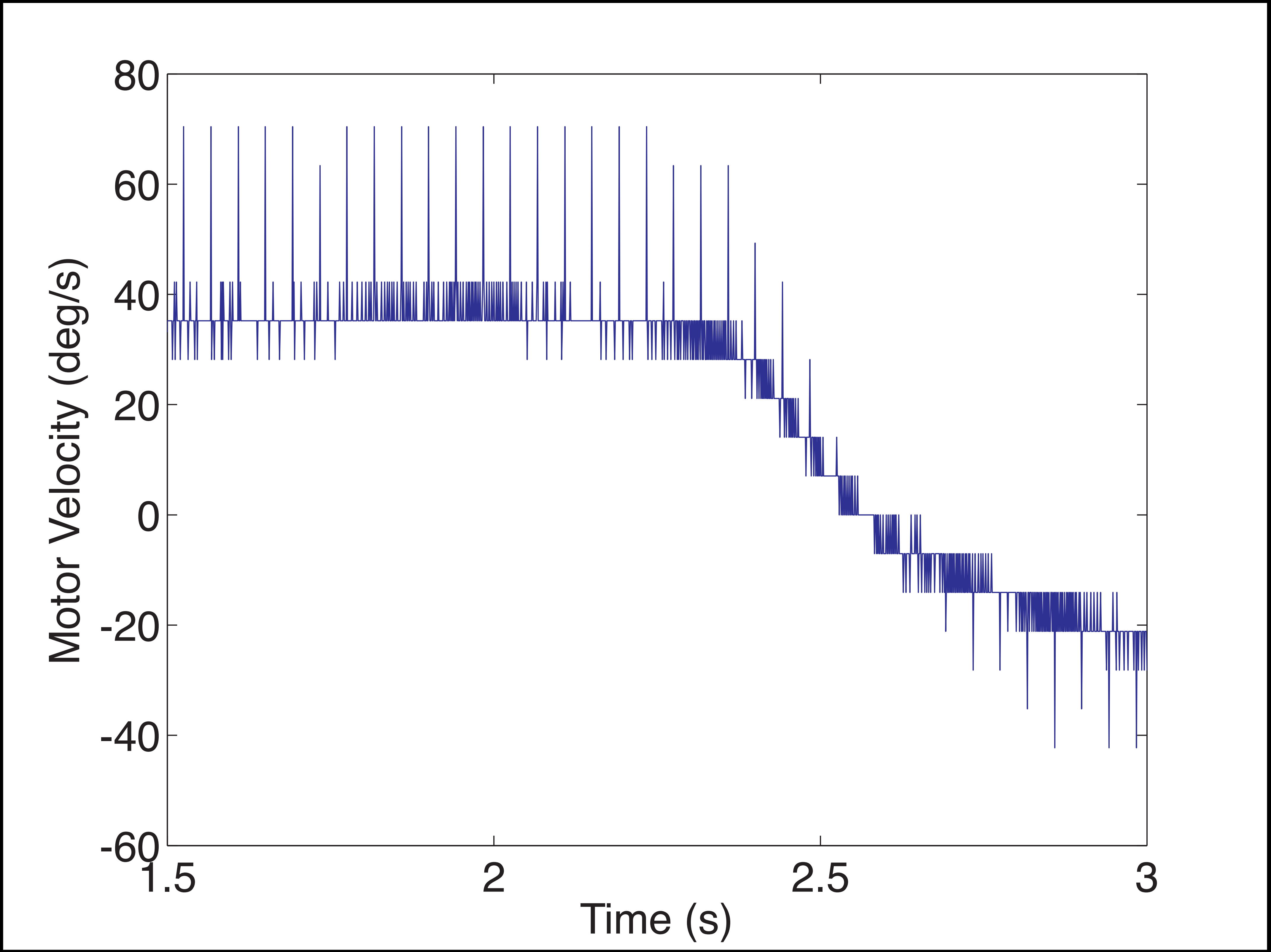


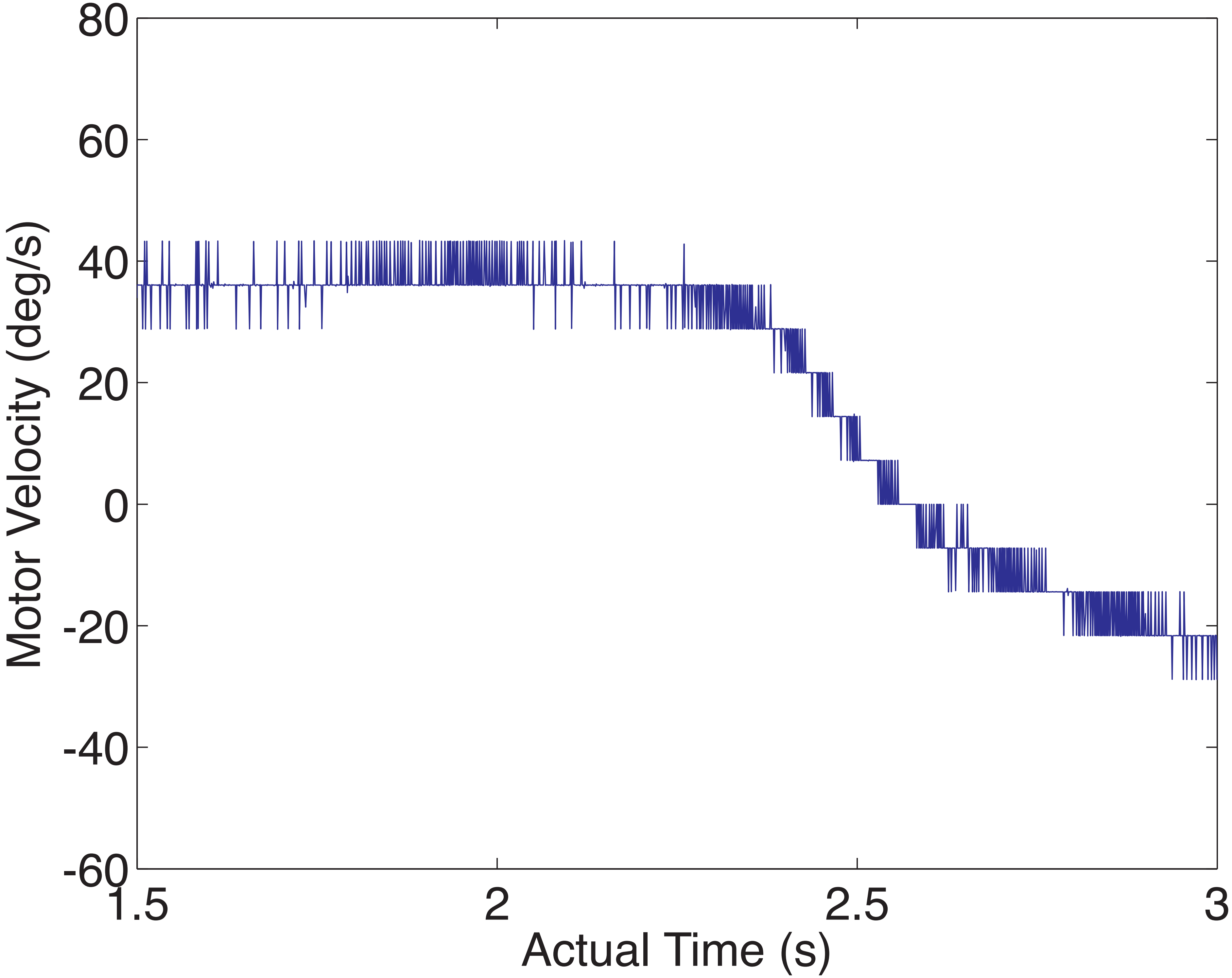
Time Since Last Cycle (ms)
0
0.5
1
1.5
2
0
200
400
600
800
1000
Haptic Thread Cycle Index



Motor Position (degrees)
35.5
35
34.5
34
33.5
1.5
1.51
1.52
1.53
1.54
1.55
Time (s)

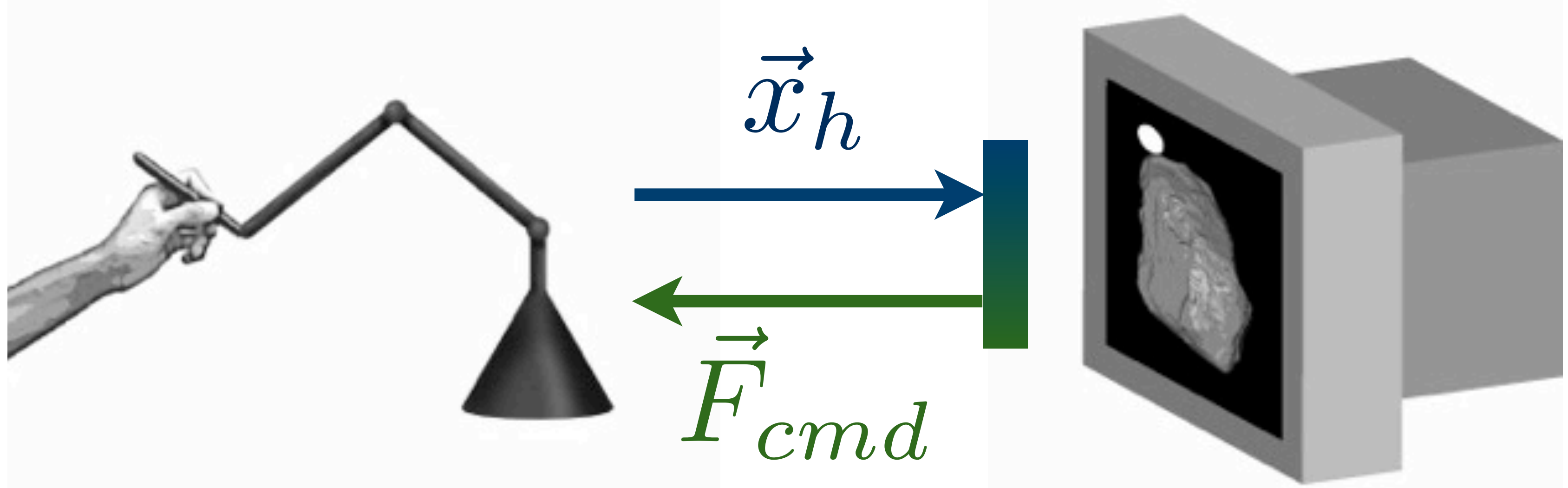

35.5
35
34.5
34
33.5
Motor Position (degrees)
1.5
1.51
1.52
1.53
1.54
1.55
Actual Time (s)


80
60
40
20
0
-20
-40
-60
Motor Velocity (deg/s)
1.5
2
2.5
3
Time (s)

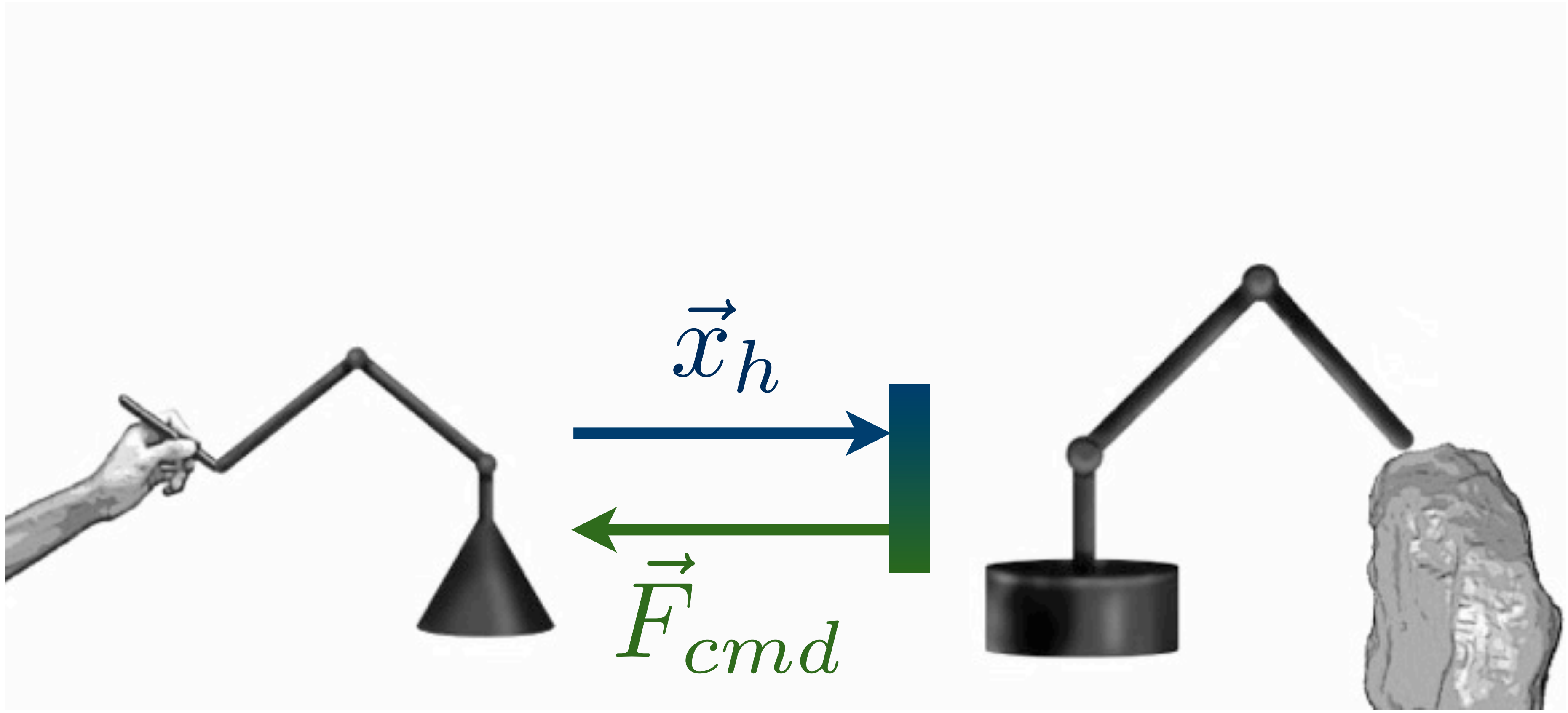


80
60
40
20
0
-20
-40
-60
1.5
2
2.5
3
Motor Velocity (deg/s)
Actual Time (s)

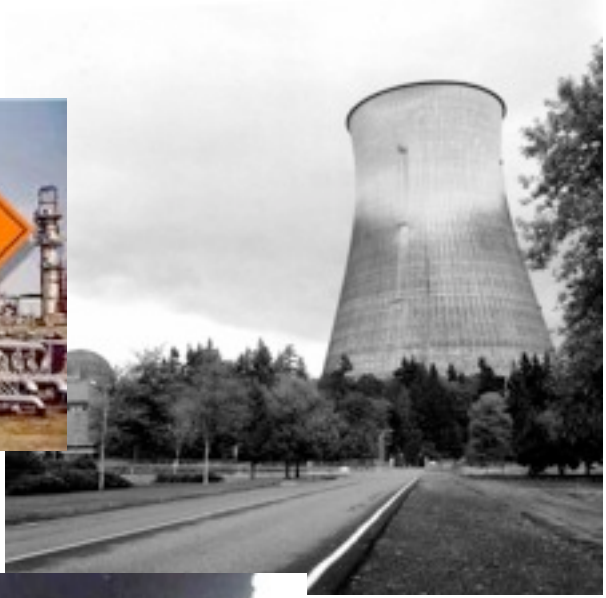
Haptic Virtual Environment



Haptic Remote Environment

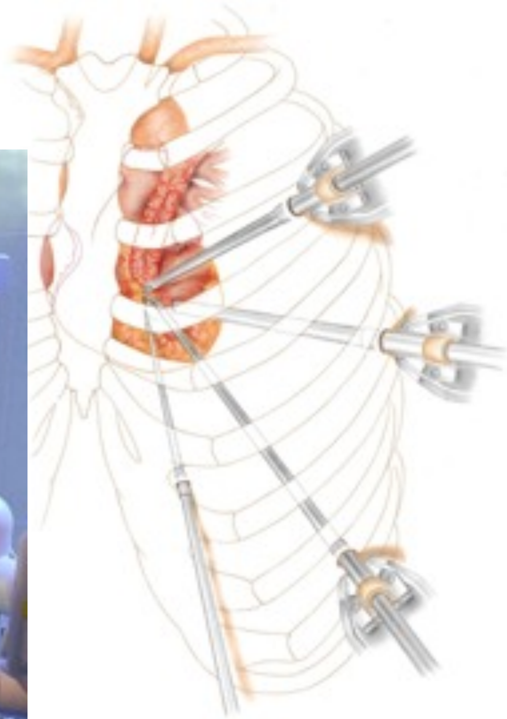


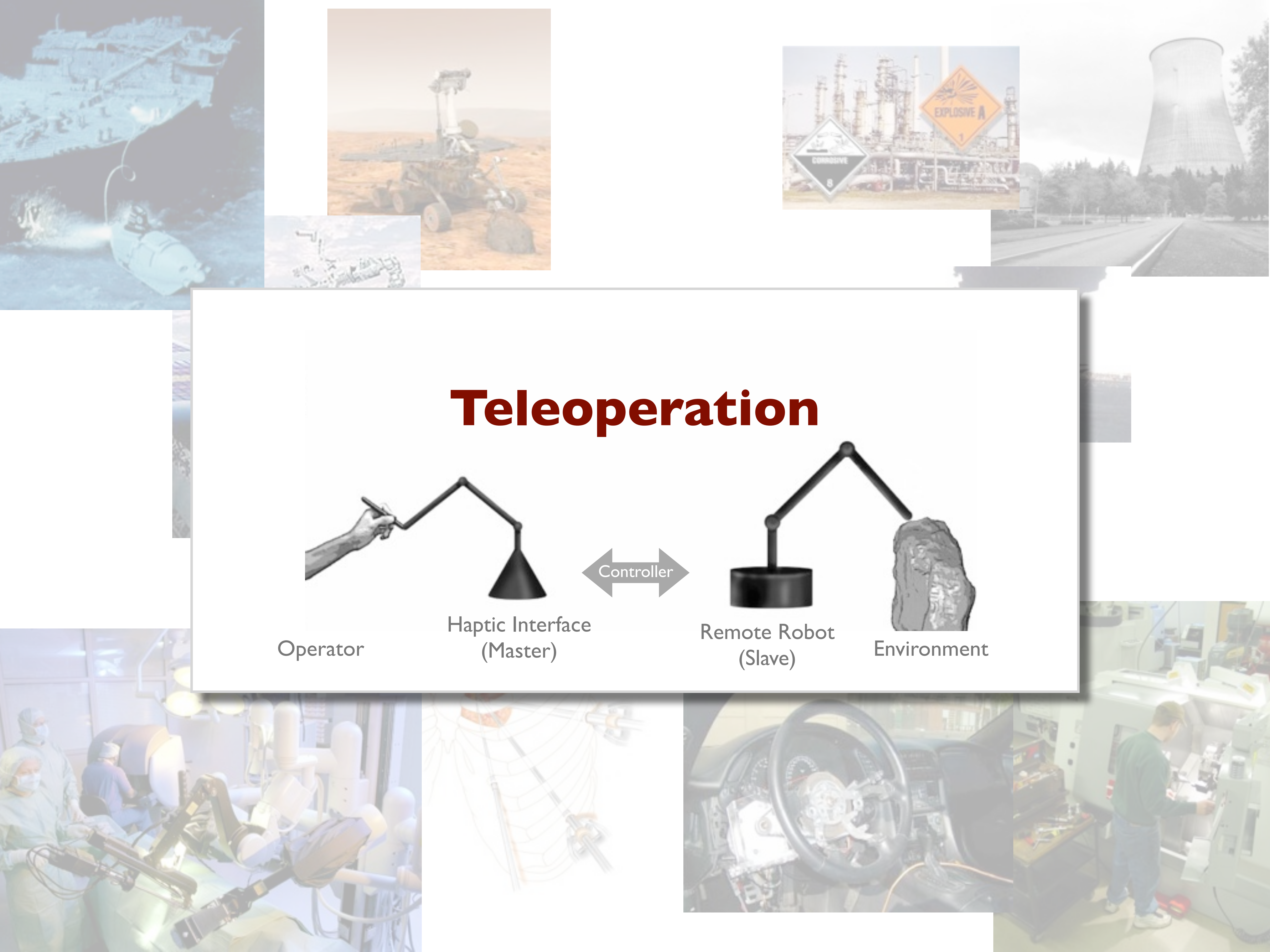



Teleoperation

extends the reach
of the human hand

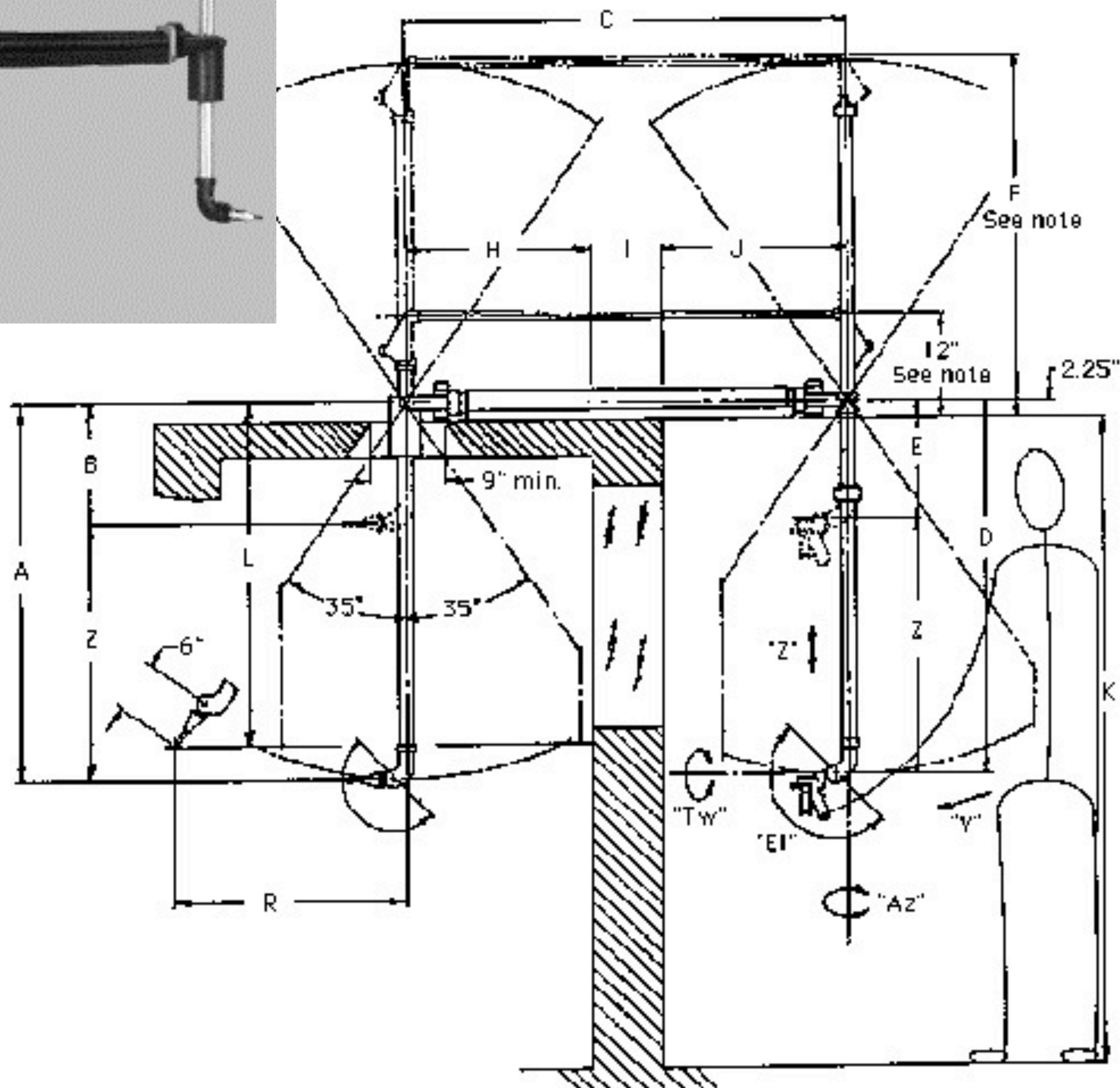




Teleoperation
Controller
Operator
Haptic Interface
(Master)
Remote Robot
(Slave)
Environment
EXPLOSIVE A
CORROSIVE

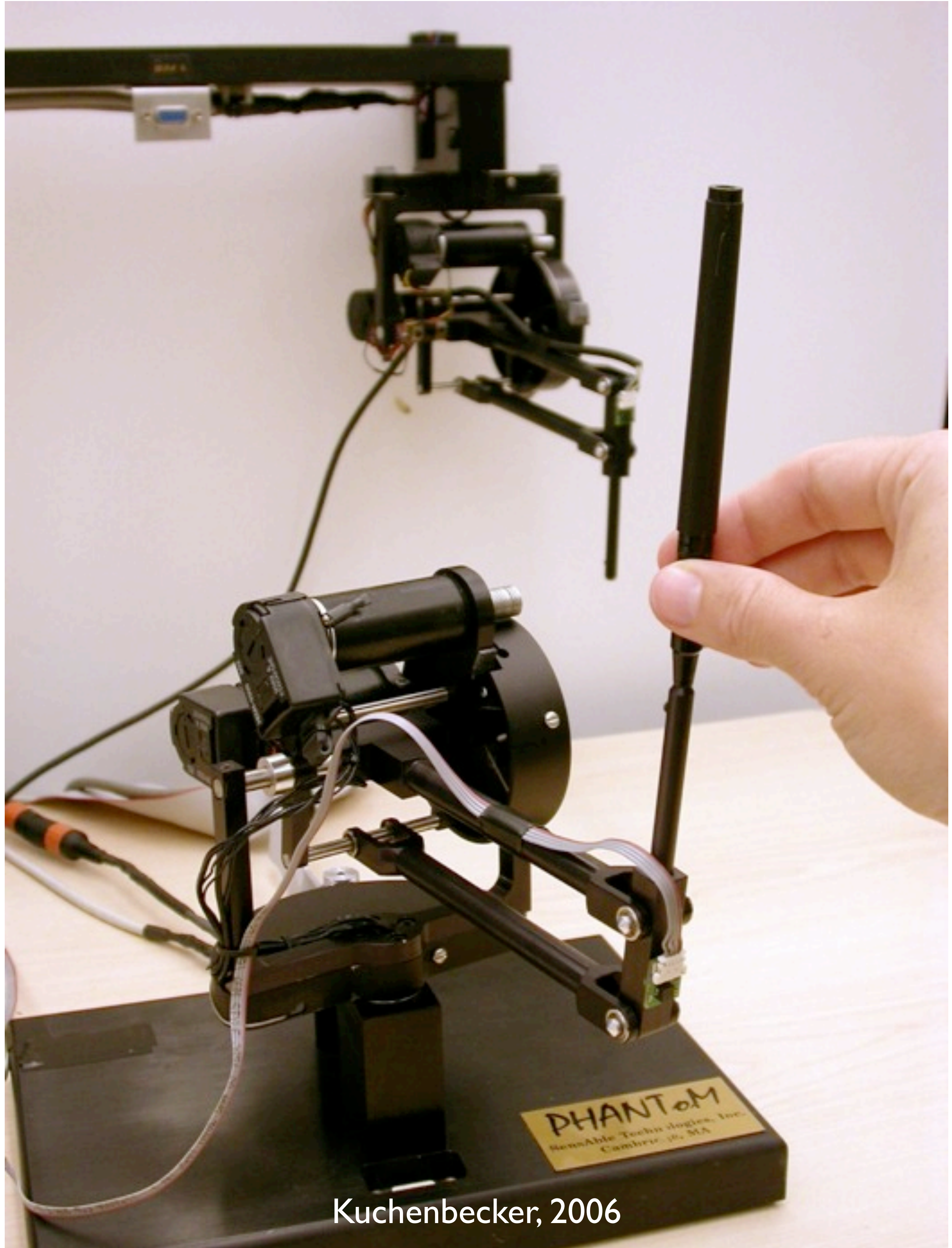
Mechanical Teleoperation

Goertz, 1952

Mechanical Teleoperation



Modern Teleoperation



Kuchenbecker, 2006

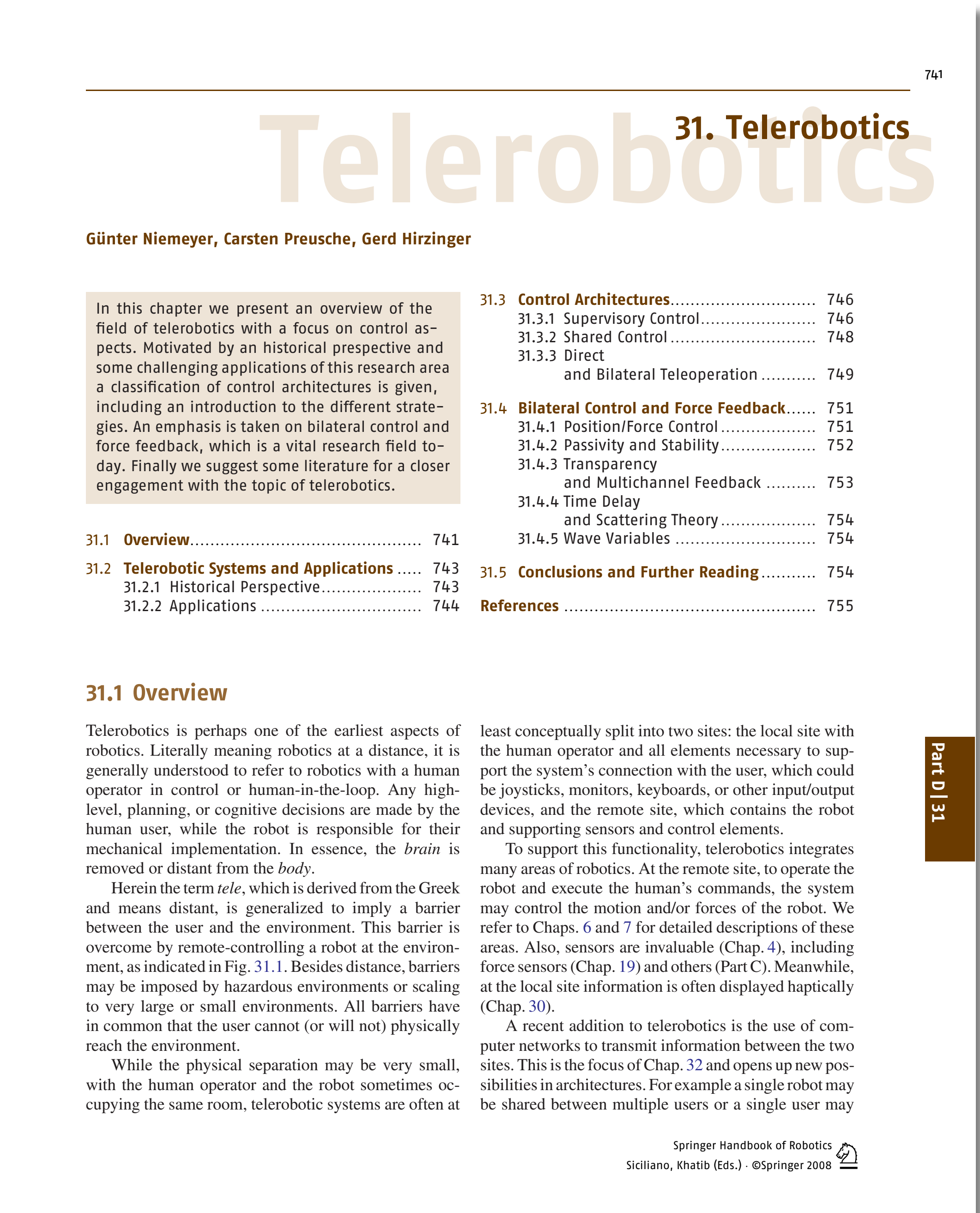
Robot-Assisted Minimally Invasive Surgery

INTUITIVE

SURGICAL

(Intuitive Surgical, Inc., 1998)

Teleoperation Reading



741

31. Telerobotics

Telerobotics

Günter Niemeyer, Carsten Preusche, Gerd Hirzinger

In this chapter we present an overview of the field of telerobotics with a focus on control aspects. Motivated by an historical prespective and some challenging applications of this research area a classification of control architectures is given, including an introduction to the different strategies. An emphasis is taken on bilateral control and force feedback, which is a vital research field today. Finally we suggest some literature for a closer engagement with the topic of telerobotics.

31.1 Overview 741
31.2 Telerobotic Systems and Applications 743
31.2.1 Historical Perspective 743
31.2.2 Applications 744
31.3 Control Architectures 746
31.3.1 Supervisory Control 746
31.3.2 Shared Control 748
31.3.3 Direct and Bilateral Teleoperation 749
31.4 Bilateral Control and Force Feedback 751
31.4.1 Position/Force Control 751
31.4.2 Passivity and Stability 752
31.4.3 Transparency and Multichannel Feedback 753
31.4.4 Time Delay and Scattering Theory 754
31.4.5 Wave Variables 754
31.5 Conclusions and Further Reading 754
References 755

31.1 Overview

Telerobotics is perhaps one of the earliest aspects of robotics. Literally meaning robotics at a distance, it is generally understood to refer to robotics with a human operator in control or human-in-the-loop. Any high-level, planning, or cognitive decisions are made by the human user, while the robot is responsible for their mechanical implementation. In essence, the *brain* is removed or distant from the *body*.

Herein the term *tele*, which is derived from the Greek and means distant, is generalized to imply a barrier between the user and the environment. This barrier is overcome by remote-controlling a robot at the environment, as indicated in Fig. 31.1. Besides distance, barriers may be imposed by hazardous environments or scaling to very large or small environments. All barriers have in common that the user cannot (or will not) physically reach the environment.

While the physical separation may be very small, with the human operator and the robot sometimes occupying the same room, telerobotic systems are often at least conceptually split into two sites: the local site with the human operator and all elements necessary to support the system's connection with the user, which could be joysticks, monitors, keyboards, or other input/output devices, and the remote site, which contains the robot and supporting sensors and control elements.

To support this functionality, telerobotics integrates many areas of robotics. At the remote site, to operate the robot and execute the human's commands, the system may control the motion and/or forces of the robot. We refer to Chaps. 6 and 7 for detailed descriptions of these areas. Also, sensors are invaluable (Chap. 4), including force sensors (Chap. 19) and others (Part C). Meanwhile, at the local site information is often displayed haptically (Chap. 30).

A recent addition to telerobotics is the use of computer networks to transmit information between the two sites. This is the focus of Chap. 32 and opens up new possibilities in architectures. For example a single robot may be shared between multiple users or a single user may

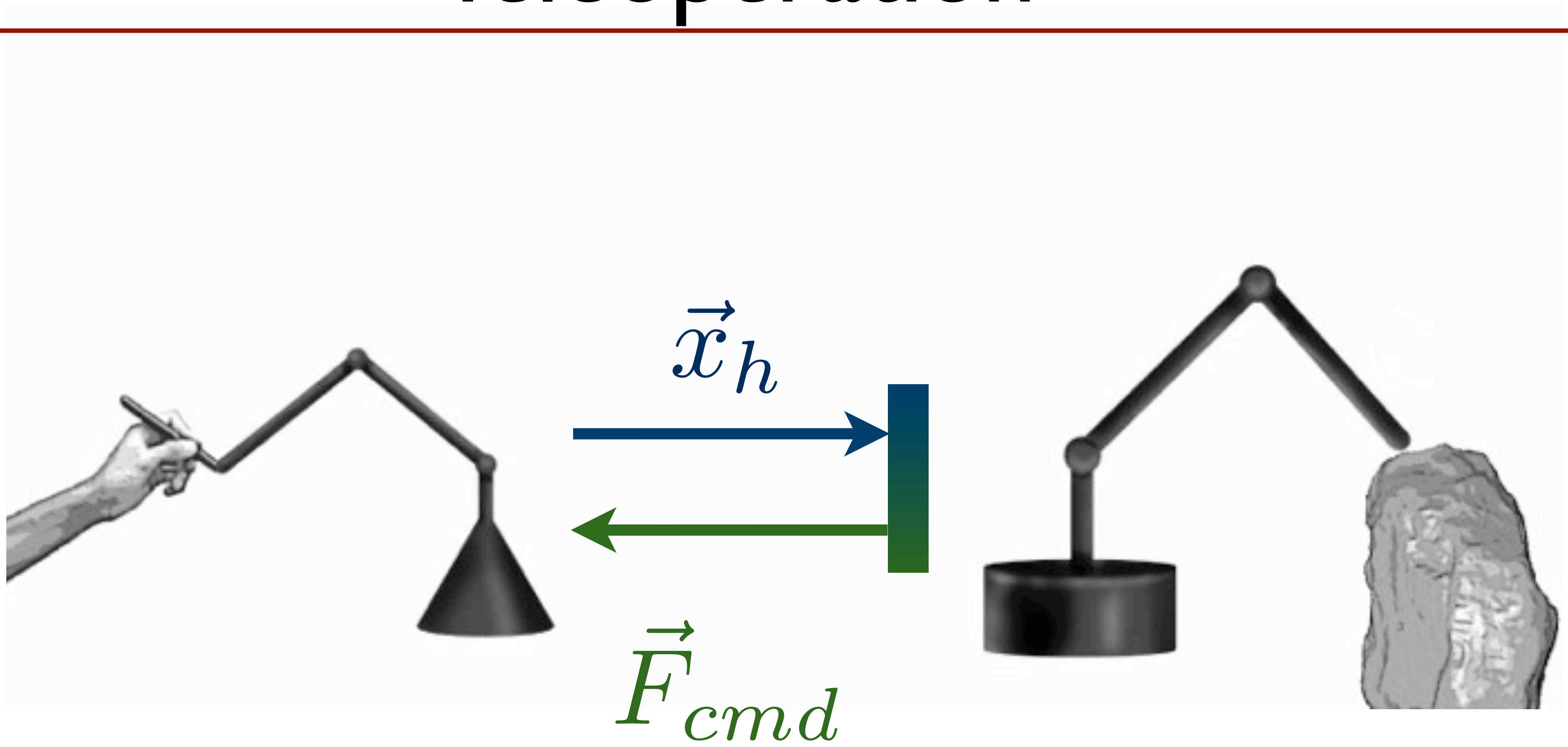
Part D | 31

Springer Handbook of Robotics
Siciliano, Khatib (Eds.) · ©Springer 2008

G. Niemeyer, C. Preusche, and G. Hirzinger. Telerobotics. Chapter 31 in *Springer Handbook of Robotics*, Siciliano and Khatib, Eds., pp. 741–757. 2008.

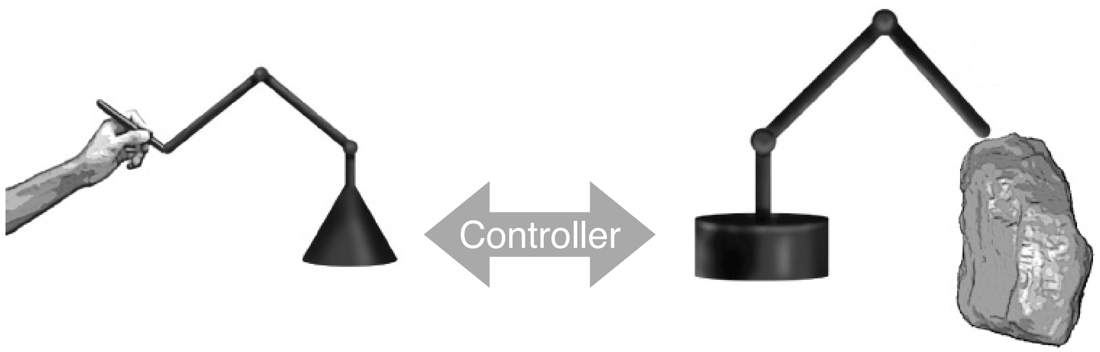
Provides a good introduction to the topic of teleoperation, including discussions of varying levels of remote robot autonomy and different control schemes for achieving force feedback.

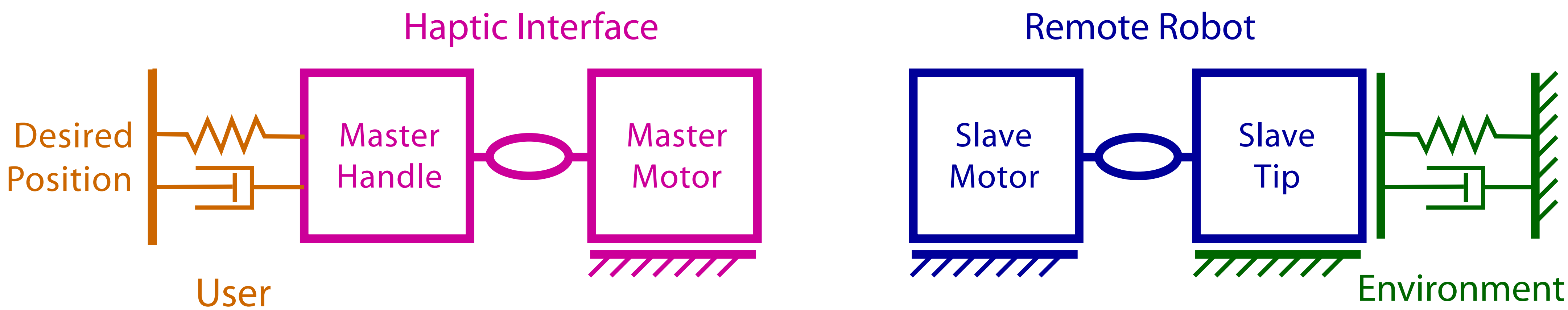
Teleoperation



- Teleoperation has always been tightly intertwined with robotics, especially manipulators.
- Control system design is a primary concern:
  - Stability
  - Transparency

Teleoperation

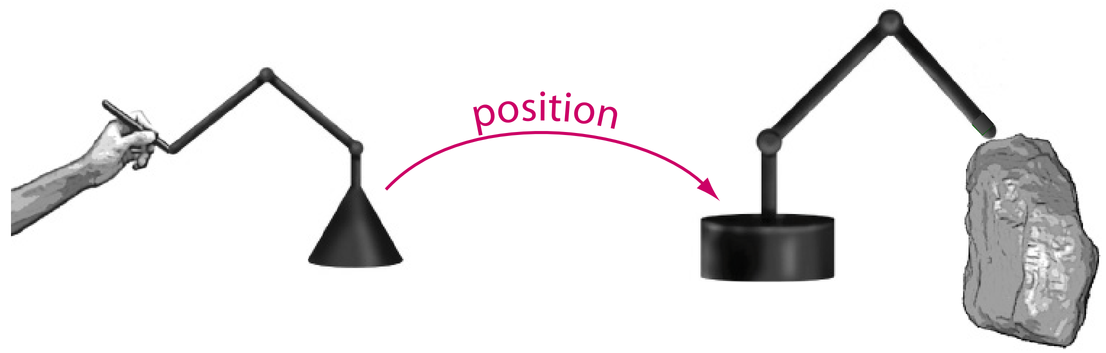


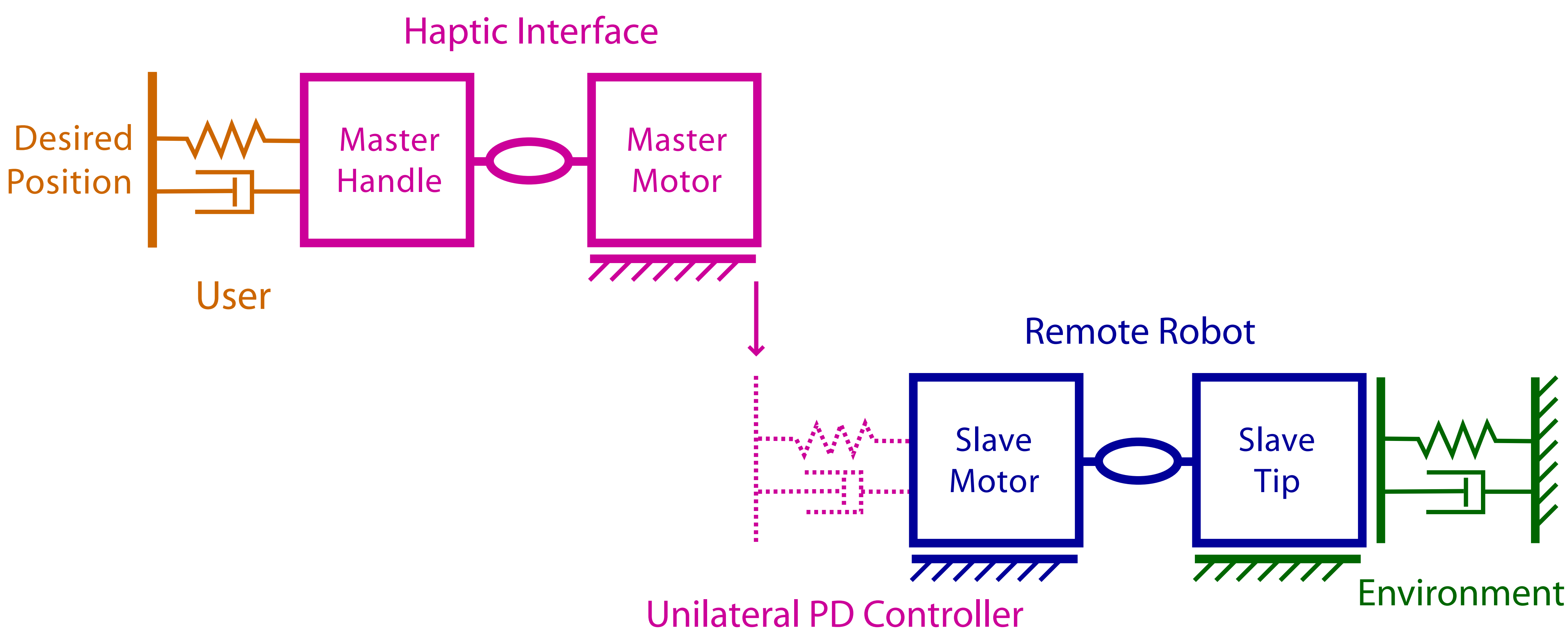


How do we want this system to behave?

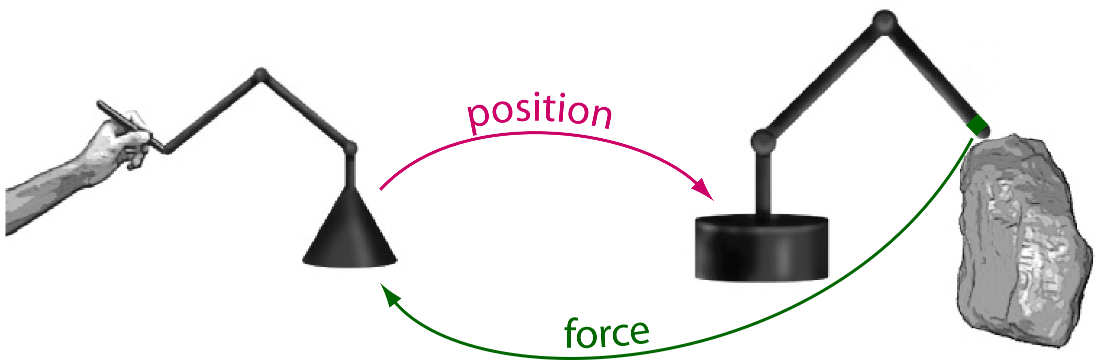
How should we connect the sensors and actuators of the master and slave to make the system behave well?

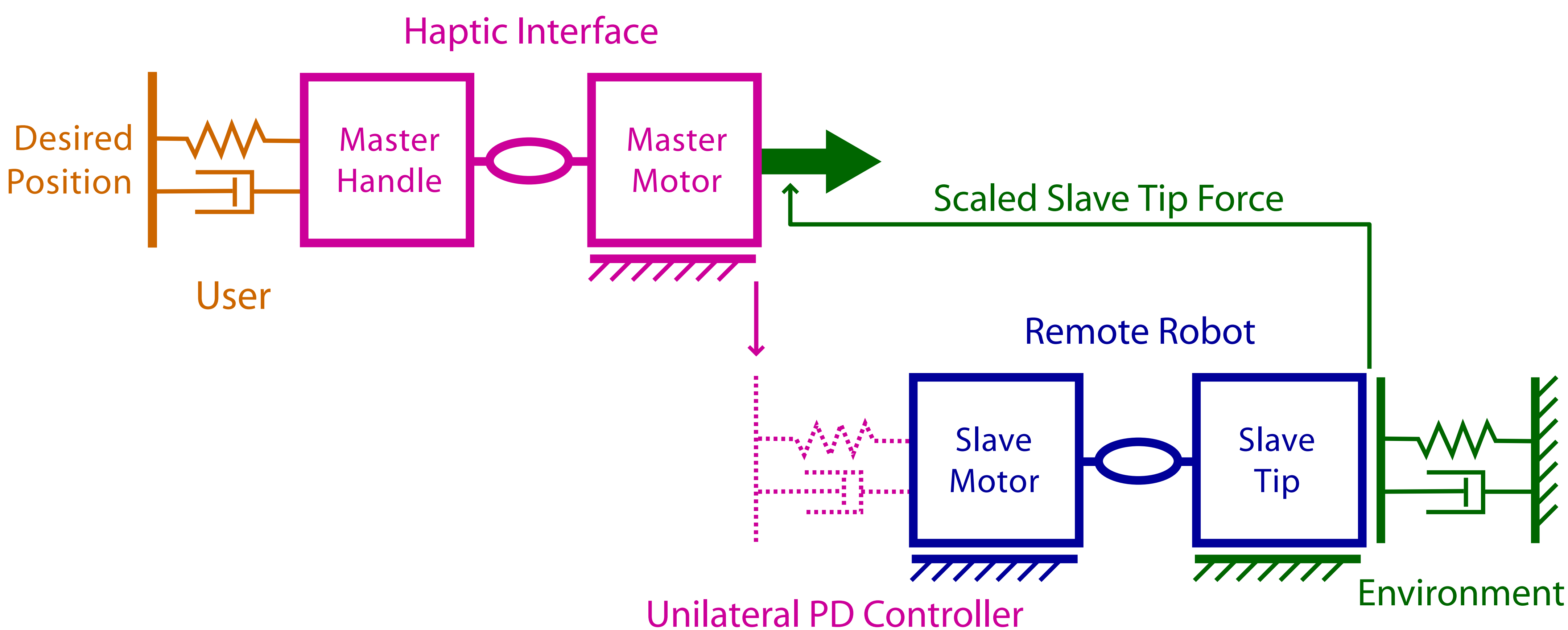
Position-Forward Control



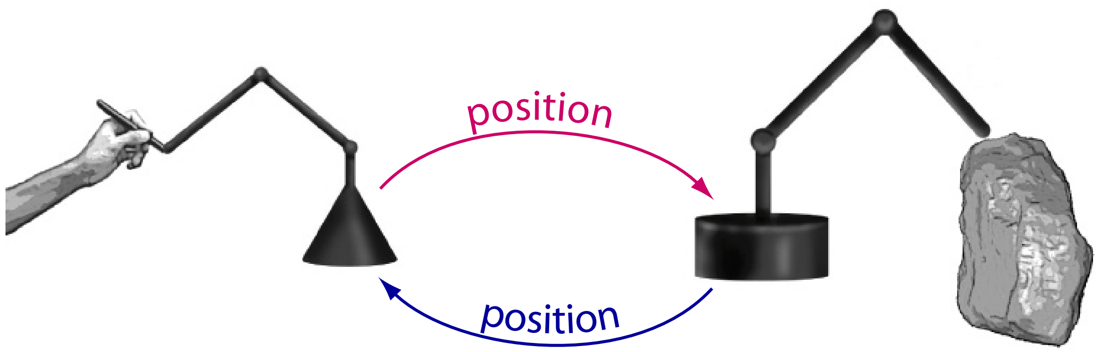


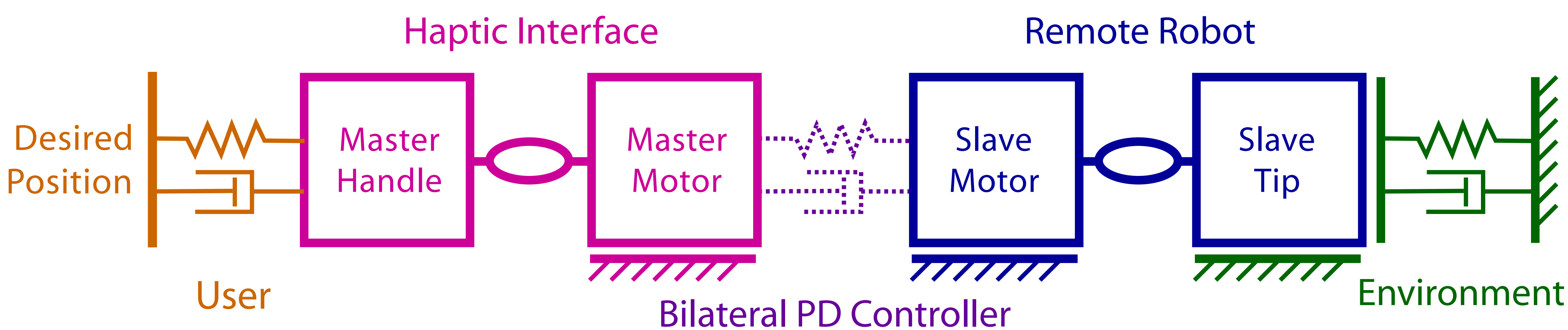
Position-Force Control



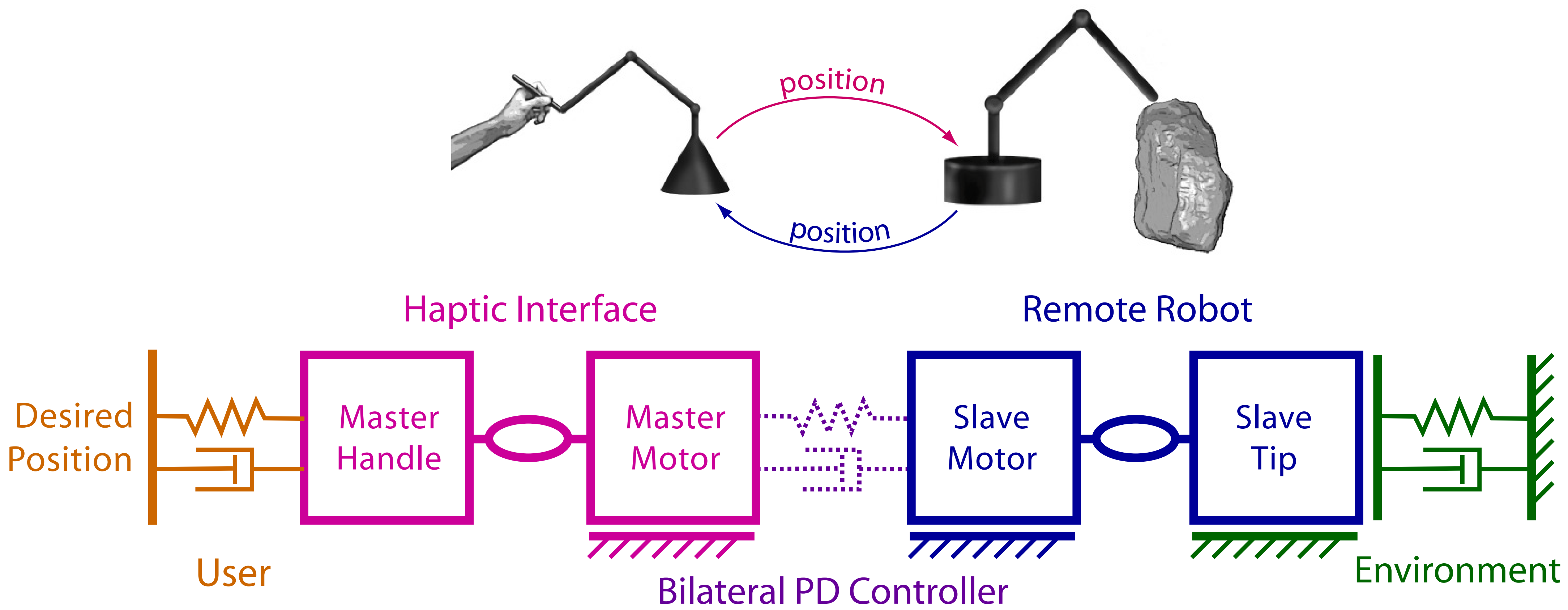


Position-Position Control





Position-Position Control



- With two impedance-type (backdrivable) devices, the most common controller is position-position, also known as position exchange.
- Each device has a desired state (position and velocity), which is computed from measured states.
- Separate controllers try to make each device achieve its desired state by using the motors to output forces.

Name ____________________________________

Midterm Exam

MEAM 520, Introduction to Robotics
University of Pennsylvania
Katherine J. Kuchenbecker, Ph.D.

November 8, 2012

You must take this exam independently, without assistance from anyone else. You may bring in a calculator and two 8.5"×11" sheets of notes for reference. Aside from these two pages of notes, you may not consult any outside references, such as the textbook or the Internet. Any suspected violations of Penn's Code of Academic Integrity will be reported to the Office of Student Conduct for investigation.

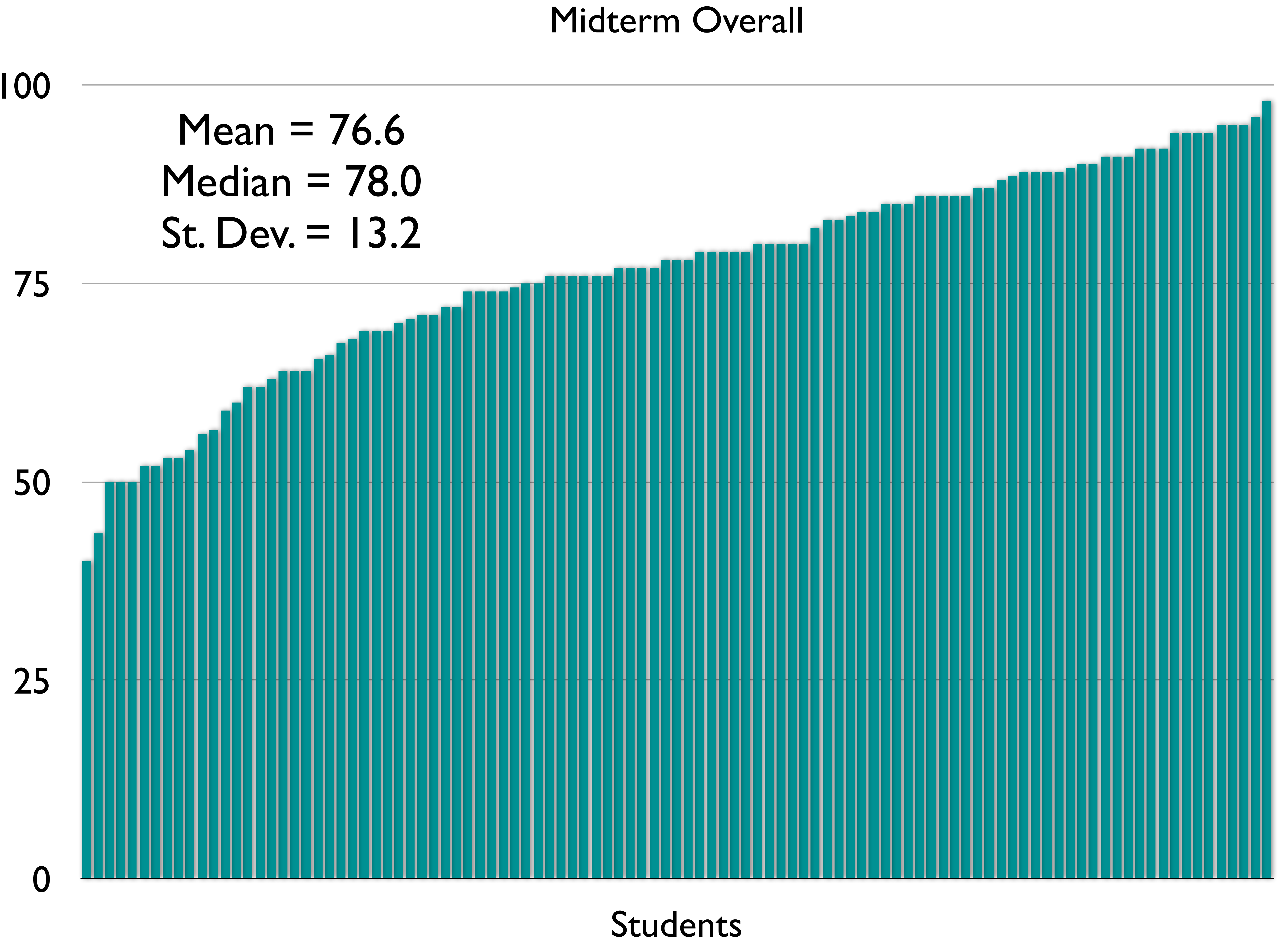
This exam consists of several problems. We recommend you look at all of the problems before starting to work. If you need clarification on any question, please ask a member of the teaching team. When you work out each problem, please show all steps and box your answer. On problems involving actual numbers, please keep your solution symbolic for as long as possible; this will make your work easier to follow and easier to grade. The exam is worth a total of 100 points, and partial credit will be awarded for the correct approach even when you do not arrive at the correct answer.

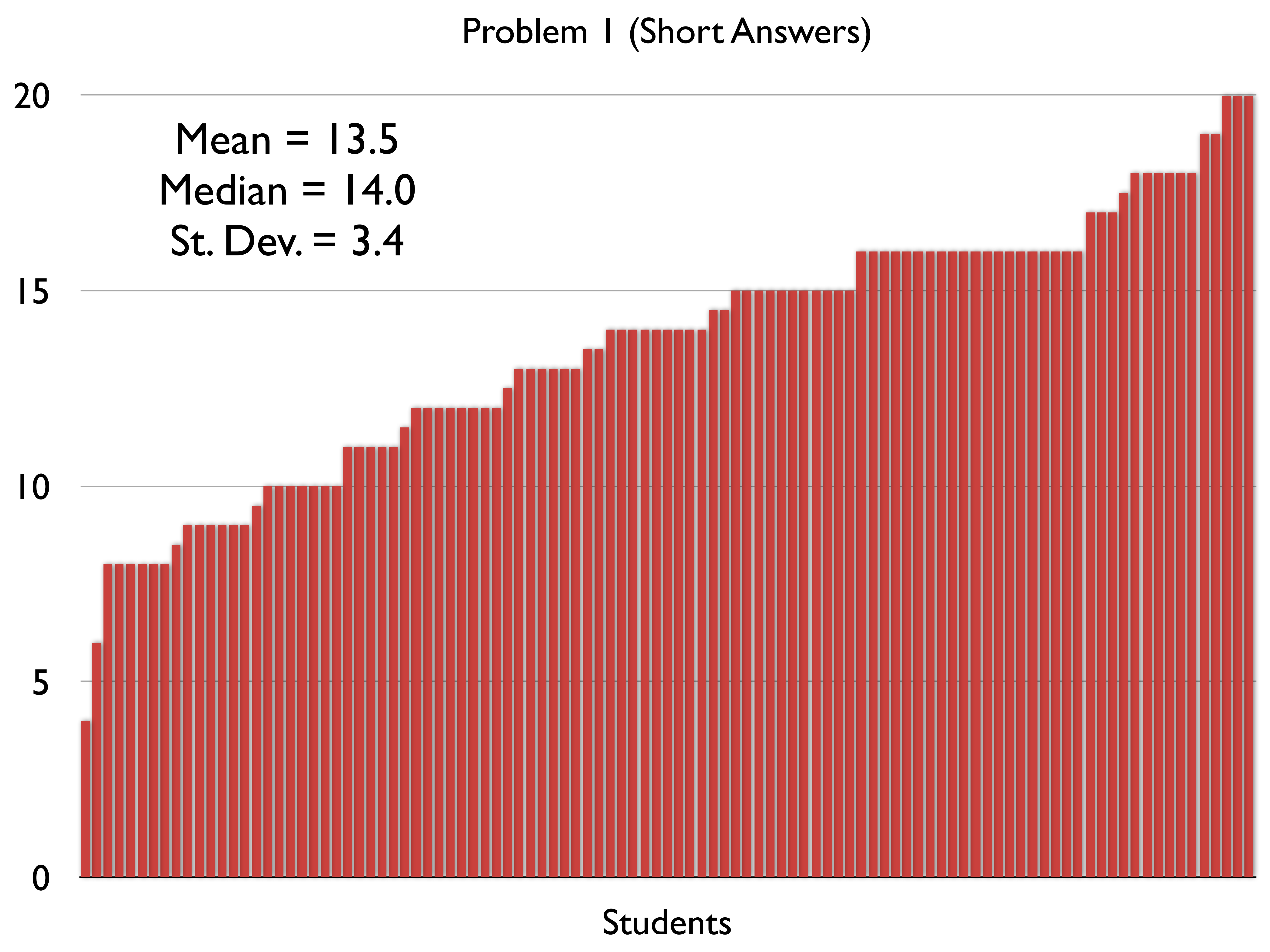
	Points	Score
Problem 1	20	
Problem 2	20	
Problem 3	15	
Problem 4	20	
Problem 5	25	
Total	100	

I agree to abide by the University of Pennsylvania Code of Academic Integrity during this exam. I pledge that all work is my own and has been completed without the use of unauthorized aid or materials.

Signature ____________________________________

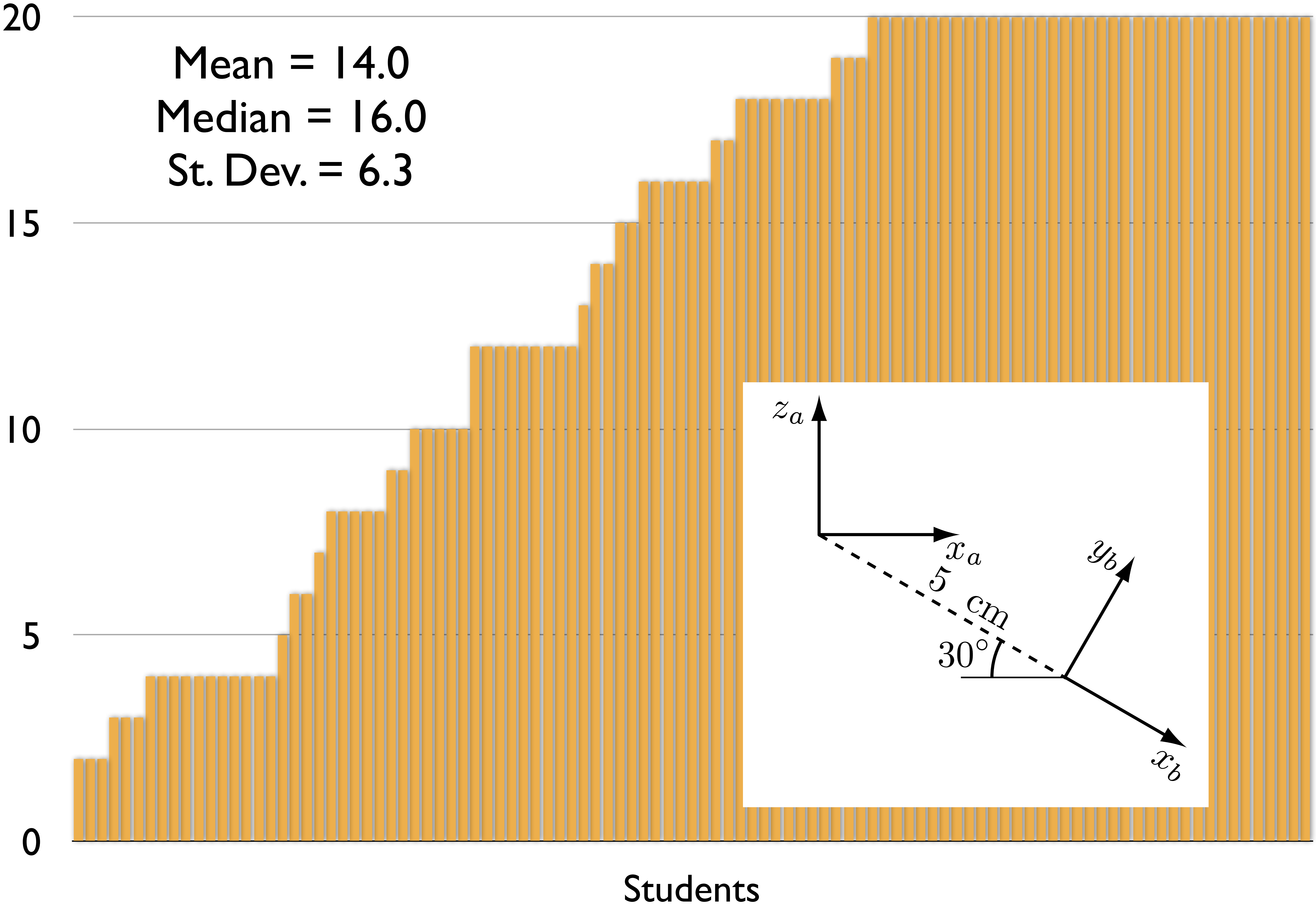
Date ____________________________________


Midterm Overall
100
75
50
25
0
Mean = 76.6
Median = 78.0
St. Dev. = 13.2
Students

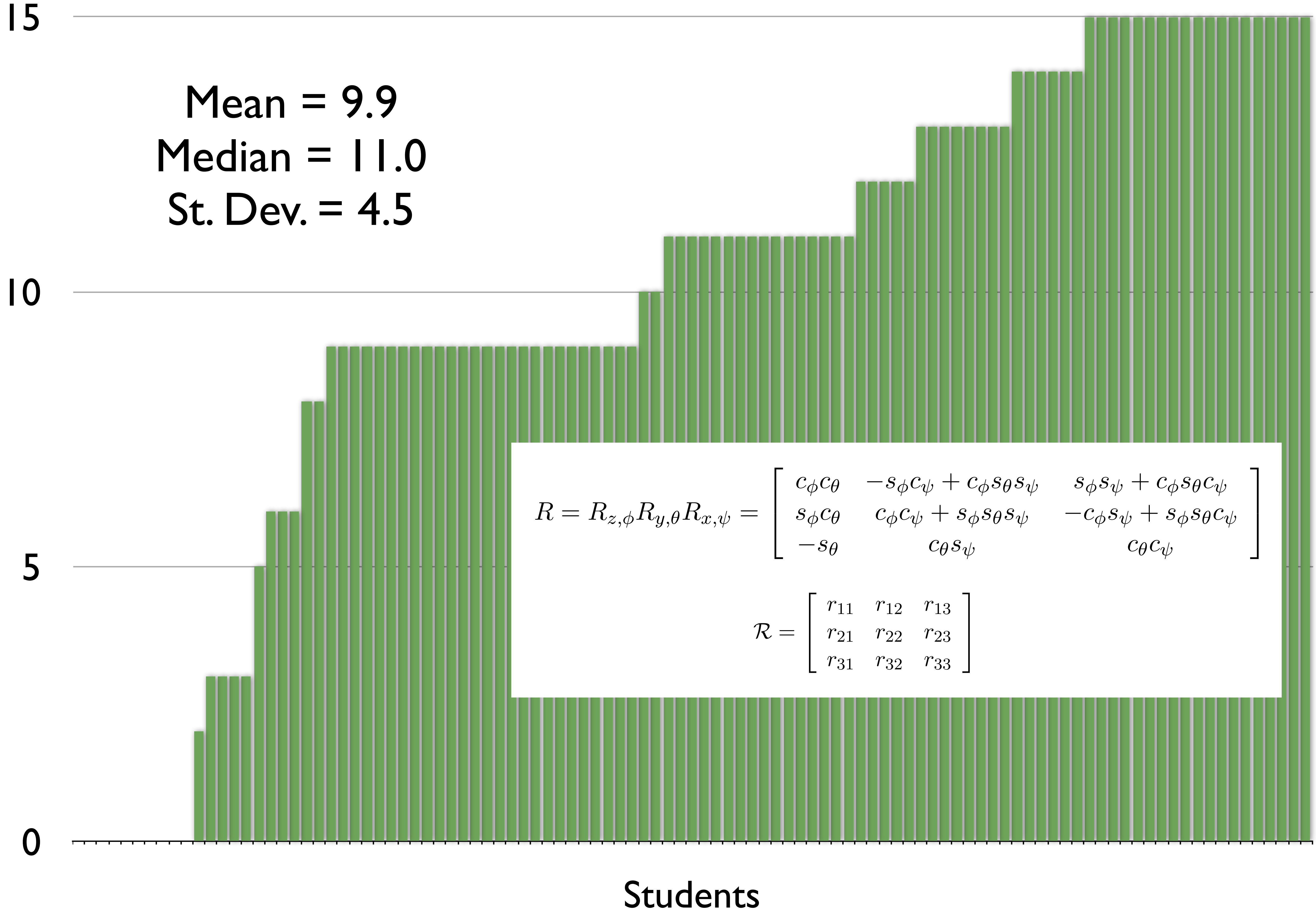


Problem 1 (Short Answers)
Mean = 13.5
Median = 14.0
St. Dev. = 3.4
20
15
10
5
0
Students

Problem 2 (Homogeneous Transformations)

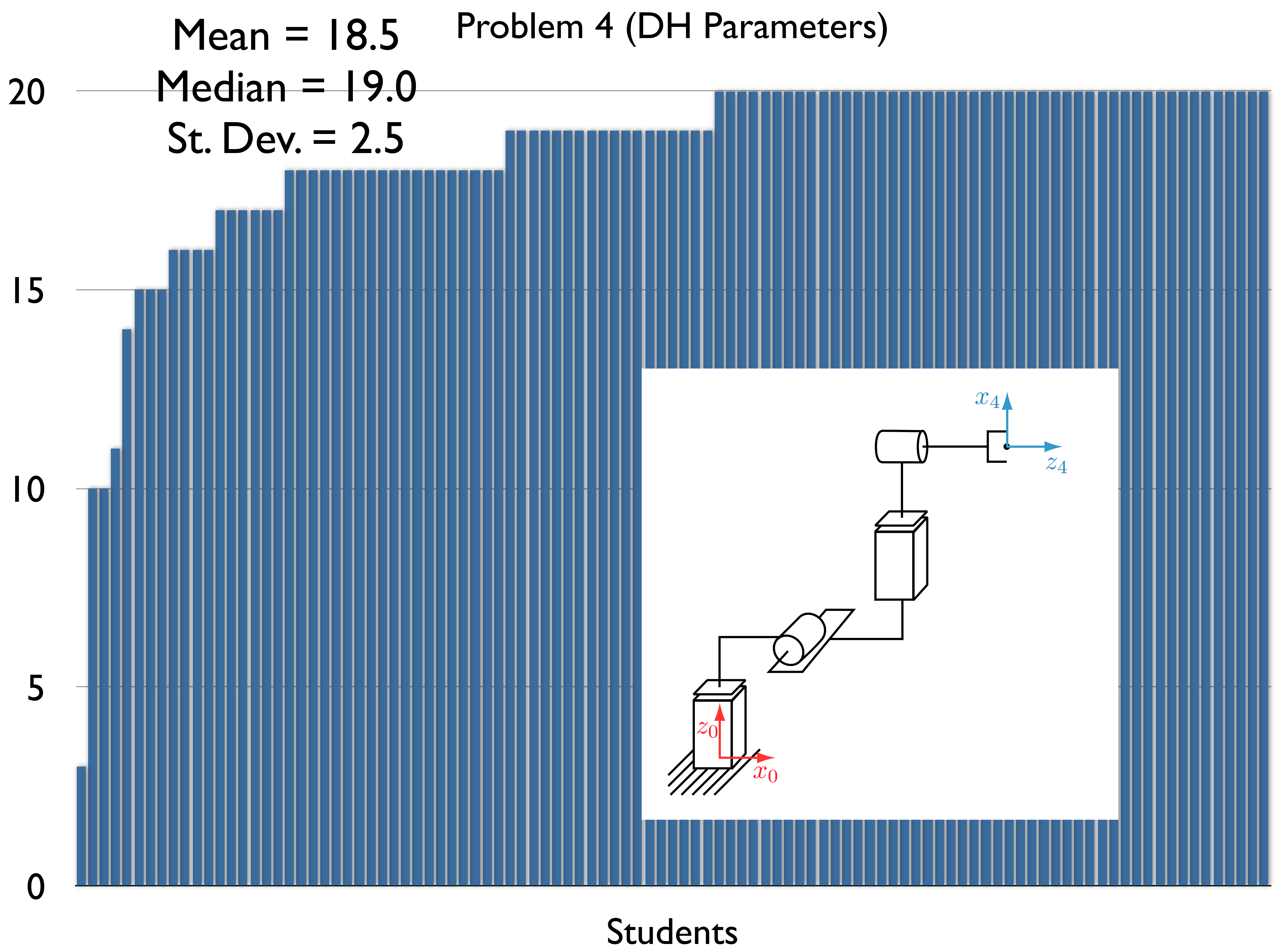


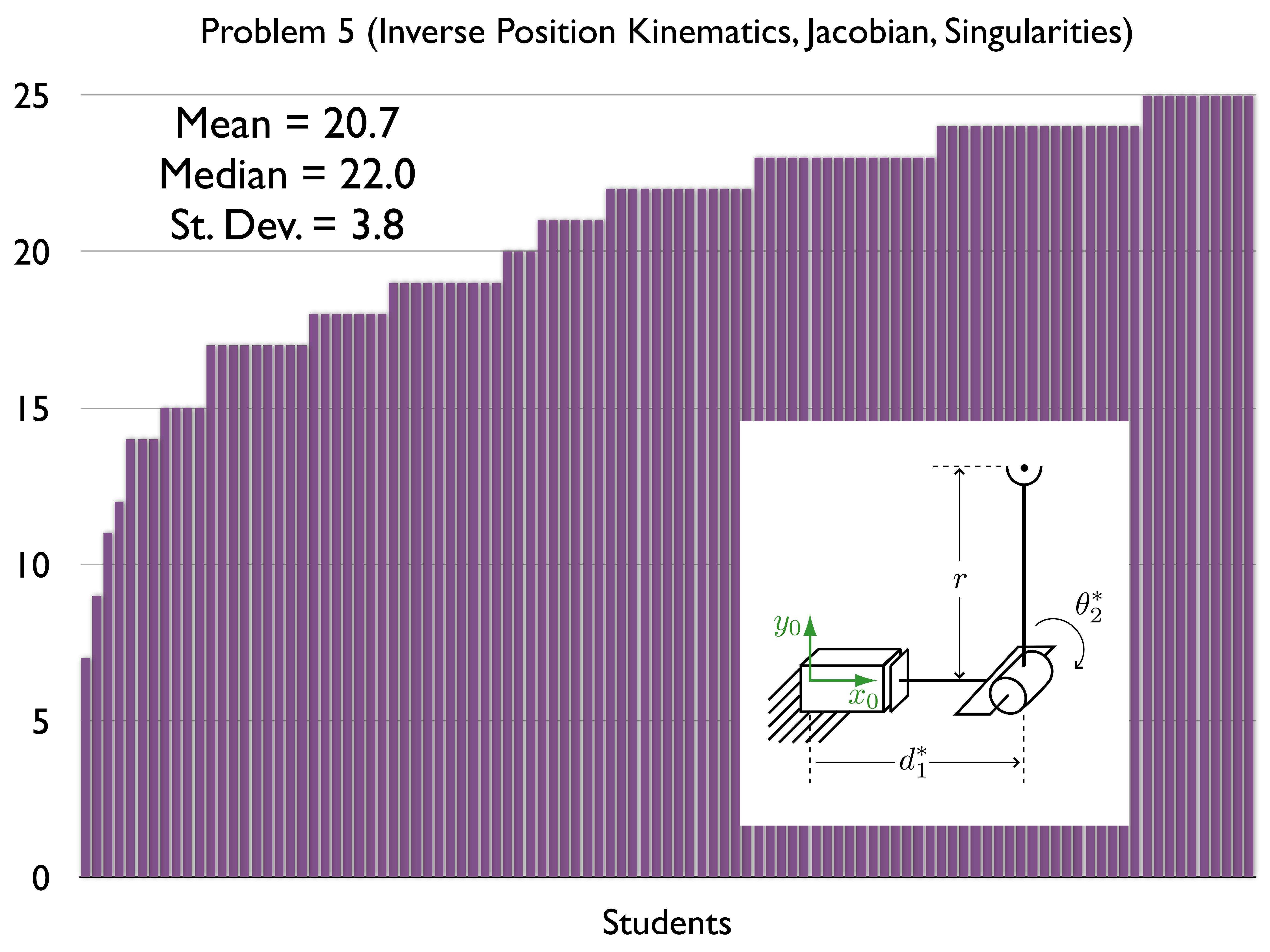
Problem 3 (Inverse Orientation Kinematics)



$$R = R_{z,\phi} R_{y,\theta} R_{x,\psi} = \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}$$

$$\mathcal{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$


Problem 4 (DH Parameters)
Mean = 18.5
Median = 19.0
St. Dev. = 2.5
20
15
10
5
0
Students
x_4
z_4
z_0
x_0



Problem 5 (Inverse Position Kinematics, Jacobian, Singularities)
Mean = 20.7
Median = 22.0
St. Dev. = 3.8
25
20
15
10
5
0
Students
y_0
x_0
r
θ_2^*
d_1^*

MEAM.Design : MEAM520 – Introduction to Robotics

http://medesign.seas.upenn.edu/index.php/Courses/MEAM520

Google

MEAM.Design : MEAM520 - Introduction to Robotics

View Logout
Edit Upload

GENERAL
- Hall of Fame
- Laboratories
- Contact Info

COURSES
- MEAM 101
- MEAM 201
- MEAM 410/510
- MEAM 520
- IPD 501
- SAAST

GUIDES
- Materials
- Laser Cutting
- 3D Printing
- Machining
- ProtoTRAK
- PUMA 260
- PHANToM
- BeagleBoard
- MAEVARM
- Phidget
- Tap Chart

SOFTWARE
- SolidWorks
- Matlab
- NX
- Nastran

Calendar

	Date	Topic (Linked to Lecture Slides)	Reading	Assignments Due	Project Deadlines
01	Thu, 9/6	Course Logistics and Motivation			
02	Tue, 9/11	Rotation Matrices	B.1, 2.1-2.3		
03	Thu, 9/13	Homogenous Transformations	2.4-2.8		
04	Tue, 9/18	Manipulator Kinematics	1.1-1.3, 3.1	HW01 (Flying Box)	
05	Thu, 9/20	Denavit-Hartenberg (DH)	3.2		
06	Tue, 9/25	More Denavit-Hartenberg (DH)	3.2		
07	Thu, 9/27	Inverse Kinematics (IK)	3.3, 3.4	HW02 (SCARA Robot)	
08	Tue, 10/2	More Inverse Kinematics (IK)	3.3		
09	Thu, 10/4	PUMA 260 and Project 1			
10	Tue, 10/9	More Manipulator Kinematics	3.3	HW03 (PUMA FK + SCARA IK)	PUMA Light Painting: Teams
	Thu, 10/11	No lecture - project work time			
11	Tue, 10/16	Velocity Kinematics	4.6		PUMA Light Painting: IK
12	Thu, 10/18	More Velocity Kinematics	4.6, 4.9, 4.11, 4.12		
	Tue, 10/23	No lecture - fall break			
13	Thu, 10/25	From Simulation to Reality			PUMA Light Painting: Simulation
	Tue, 10/30	No lecture - hurricane			
14	Thu, 11/1	Robot Trajectories	5.1, 5.2	HW04 (Jacobians) *due Friday*	
15	Tue, 11/6	Robot Hardware	6.1, 6.2		~~PUMA Light Painting: Reality~~
	Thu, 11/8	Midterm Exam (Solution)			
16	Tue, 11/13	Haptic Interface Hardware	KJK, MS		
17	Thu, 11/15	Teleoperation			PUMA Light Painting: Reality
18	Tue, 11/20			HW05 (Phantom)	
	Thu, 11/15	No lecture - Thanksgiving			

(note: all items are due at 5:00 p.m. unless otherwise specified)

Resources

Piazza Forum

Name Solution

Midterm Exam

MEAM 520, Introduction to Robotics
University of Pennsylvania
Katherine J. Kuchenbecker, Ph.D.

November 8, 2012

You must take this exam independently, without assistance from anyone else. You may bring in a calculator and two 8.5"×11" sheets of notes for reference. Aside from these two pages of notes, you may not consult any outside references, such as the textbook or the Internet. Any suspected violations of Penn's Code of Academic Integrity will be reported to the Office of Student Conduct for investigation.

This exam consists of several problems. We recommend you look at all of the problems before starting to work. If you need clarification on any question, please ask a member of the teaching team. When you work out each problem, please show all steps and box your answer. On problems involving actual numbers, please keep your solution symbolic for as long as possible; this will make your work easier to follow and easier to grade. The exam is worth a total of 100 points, and partial credit will be awarded for the correct approach even when you do not arrive at the correct answer.

	Points	Score
Problem 1	20	
Problem 2	20	
Problem 3	15	
Problem 4	20	
Problem 5	25	
Total	100	

I agree to abide by the University of Pennsylvania Code of Academic Integrity during this exam. I pledge that all work is my own and has been completed without the use of unauthorized aid or materials.

Signature ______________________

Date ______________________

Look over your exam and compare with the solution.

If you think we made a mistake in grading your test, write out an explanation on a separate piece of paper.

Give your written inquiry and your test to Philip.

We will correct any grading mistakes.

Approximate grade breakdown

A+ 96 **A** 89 **A-** 83 **B+** 78 **B** 73 **B-** 66 **C+** 60 **C** 54 **C-**

Please make an appointment to talk with me
if you got less than a 55/100.