Tools and Techniques for Prototyping Haptic Interfaces

Sensors and Sensor Processing

Katherine J. Kuchenbecker, Ph.D.

General Robotics, Automation, Sensing, and Perception Lab (GRASP)
MEAM Department, SEAS, University of Pennsylvania

Haptics Symposium 2012 Workshop

March 4, 2012

My definition of a haptic interface:

My definition of a haptic interface:

Senses a physical quantity from the user,
such as motion or force

My definition of a haptic interface:

Senses a physical quantity from the user,
such as motion or force

Physically acts on the user via a variable actuator

My definition of a haptic interface:

Senses a physical quantity from the user, such as motion or force

Physically acts on the user via a variable actuator

Connects sensing to acting with fast processing

My definition of a haptic interface:

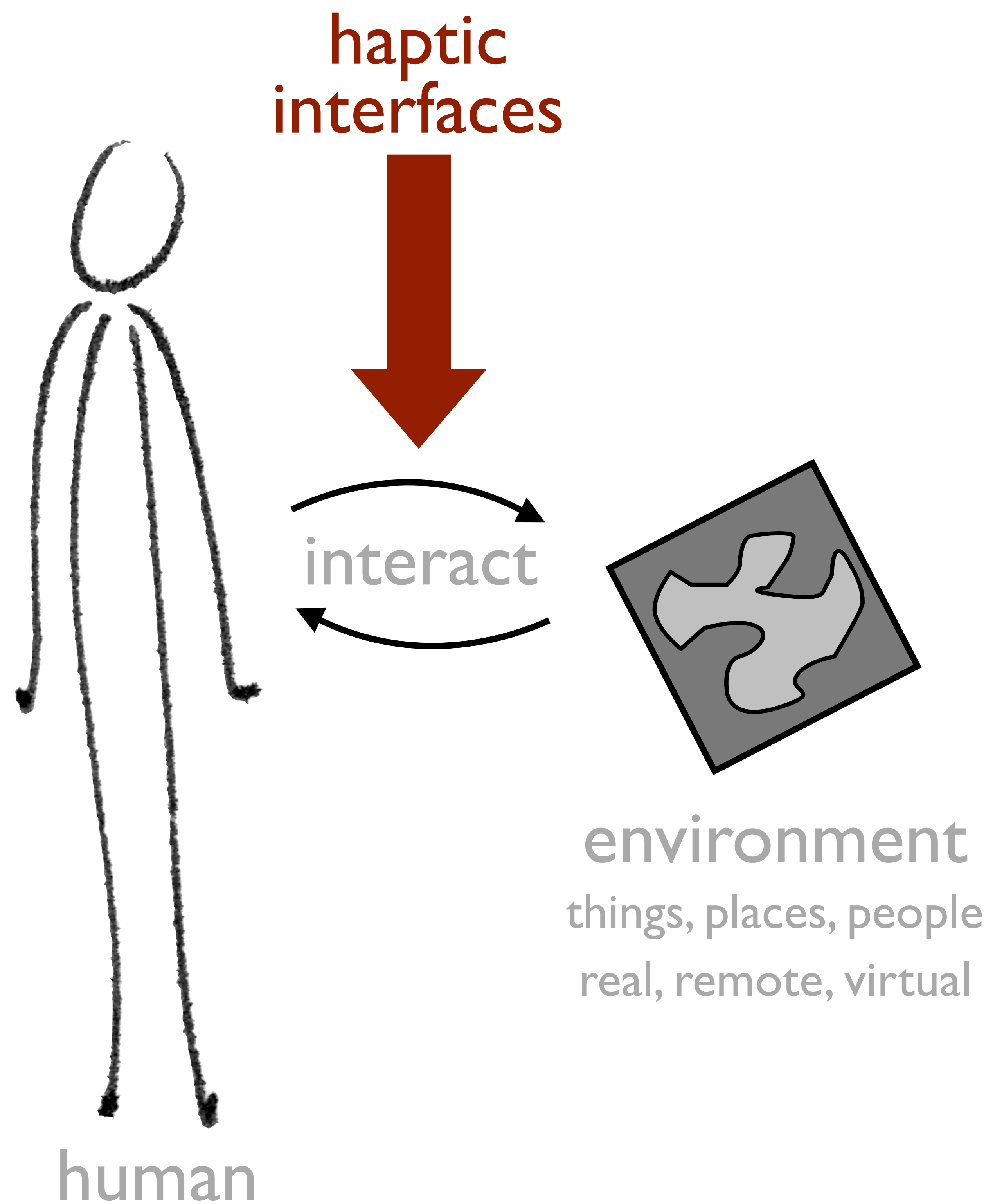
Senses a physical quantity from the user, such as motion or force

Physically acts on the user via a variable actuator

Connects sensing to acting with fast processing

it's all about physical interaction with a user...

it’s all about physical interaction with a user…

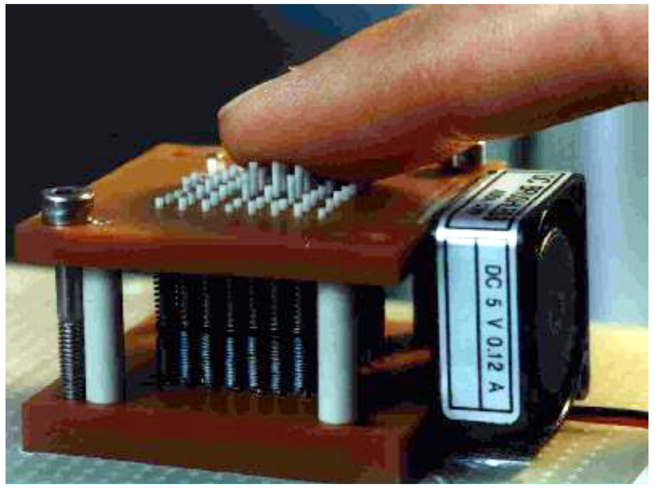


it's all about physical interaction with a user...

Haptic Interfaces

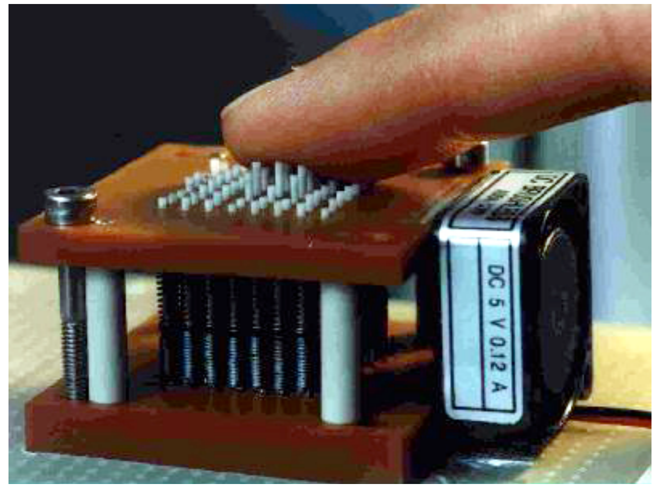
Haptic Interfaces

Tactile Devices



Stimulate skin to create contact sensations

Haptic Interfaces



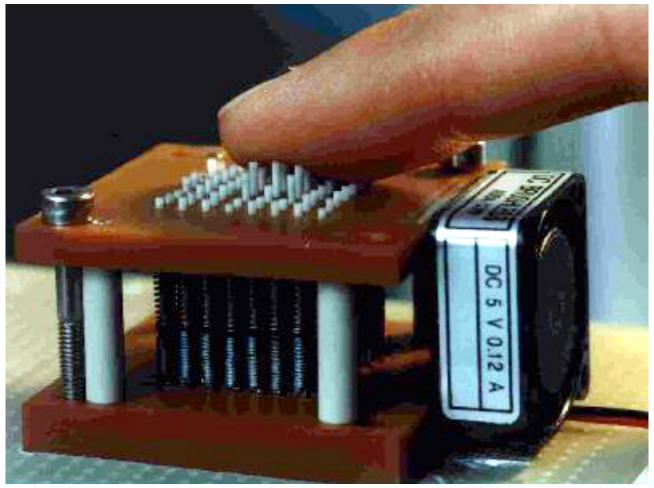
Tactile Devices

Stimulate skin to create contact sensations

Kinesthetic Devices

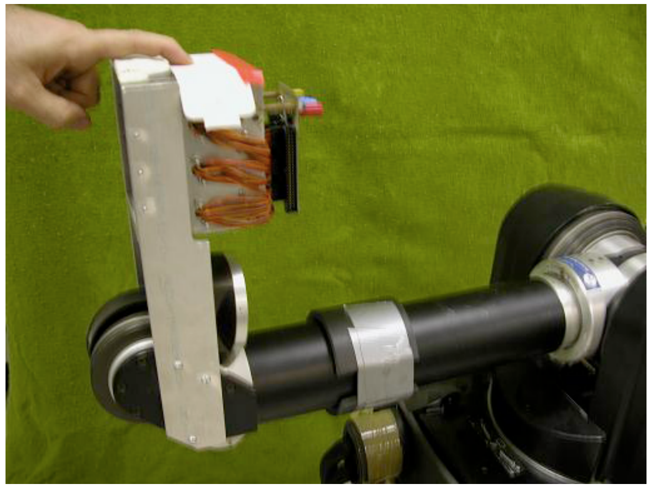
Apply forces to guide or inhibit body movement

Haptic Interfaces



Tactile Devices

Stimulate skin to create contact sensations



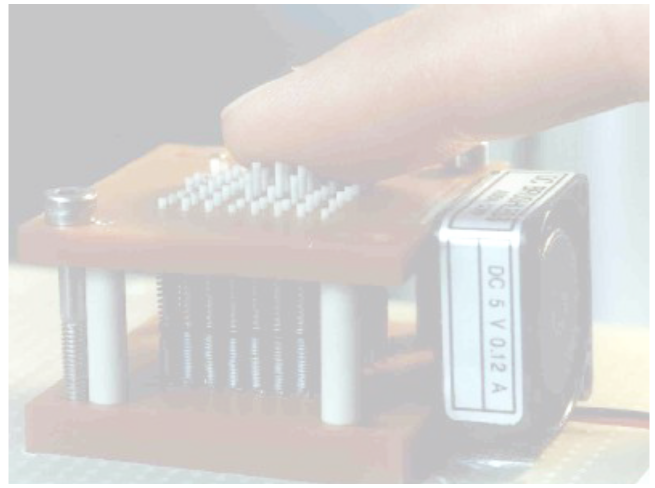
Hybrid Devices

Attempt to combine tactile and kinesthetic feedback

Kinesthetic Devices

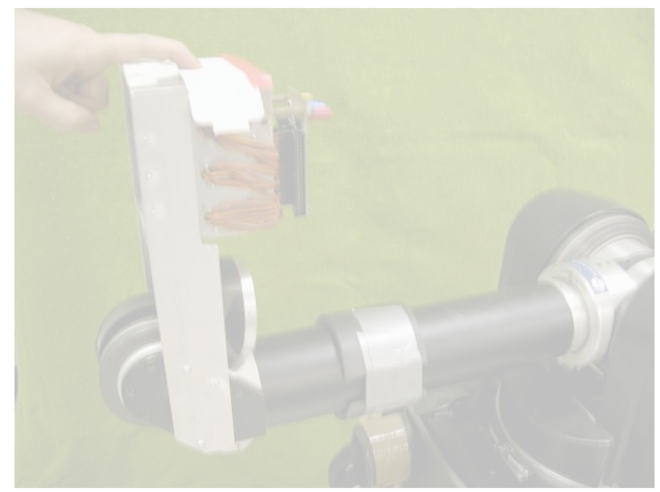
Apply forces to guide or inhibit body movement

Haptic Interfaces



Tactile Devices

Stimulate skin to create contact sensations



Hybrid Devices

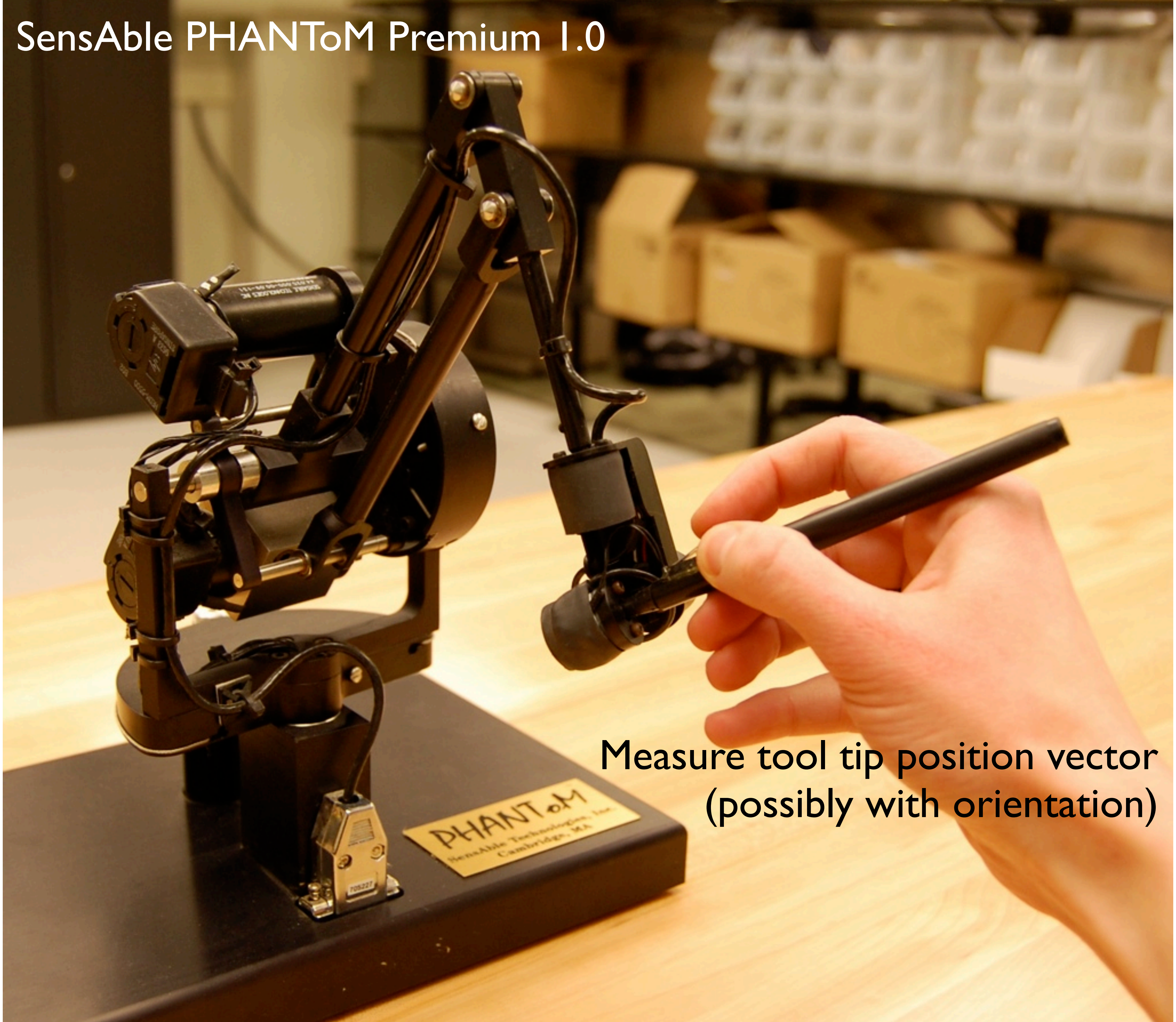
Attempt to combine tactile and kinesthetic feedback

Kinesthetic Devices

Apply forces to guide or inhibit body movement

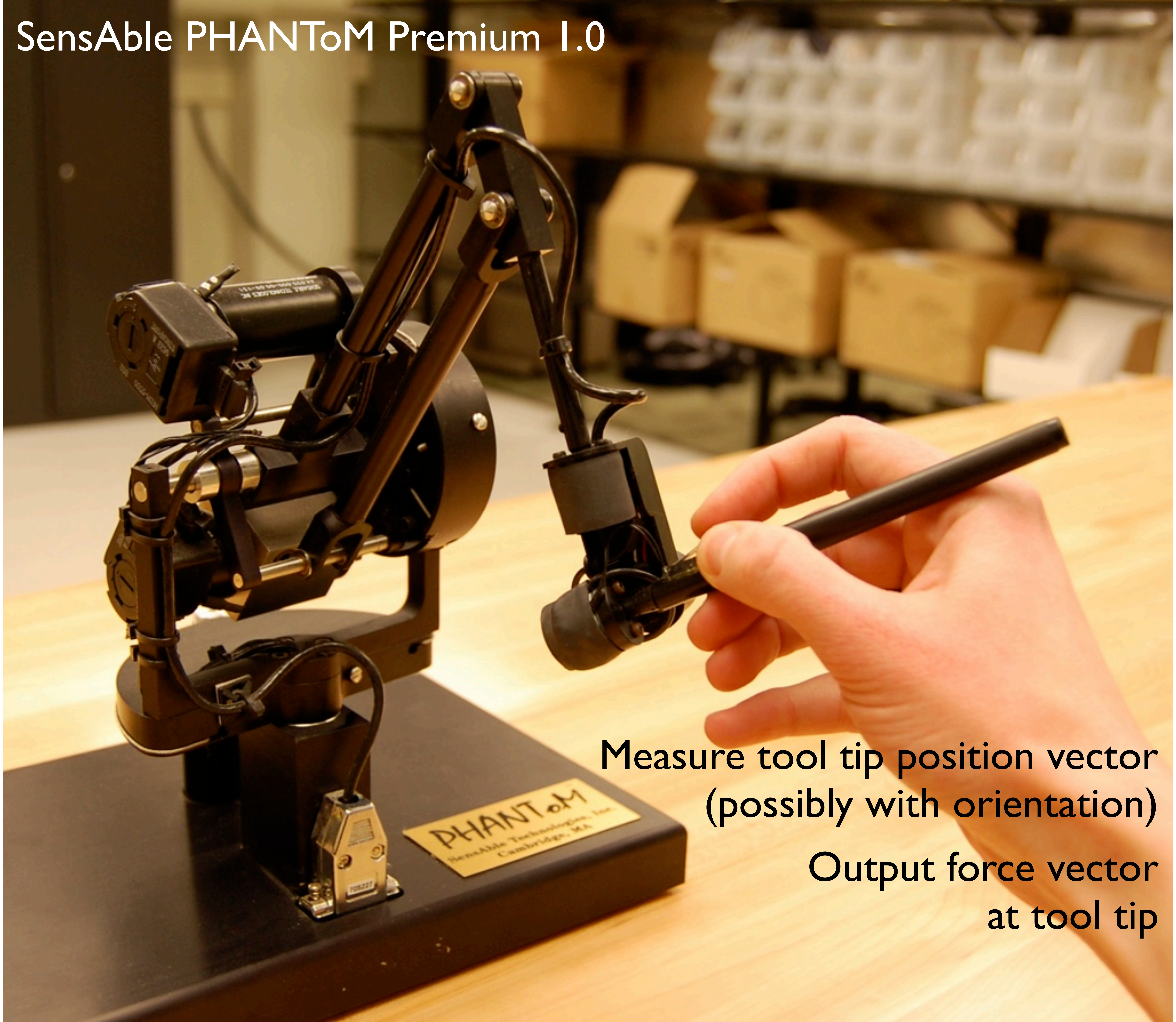
SensAble PHANToM Premium 1.0

SensAble PHANToM Premium 1.0



Measure tool tip position vector
(possibly with orientation)

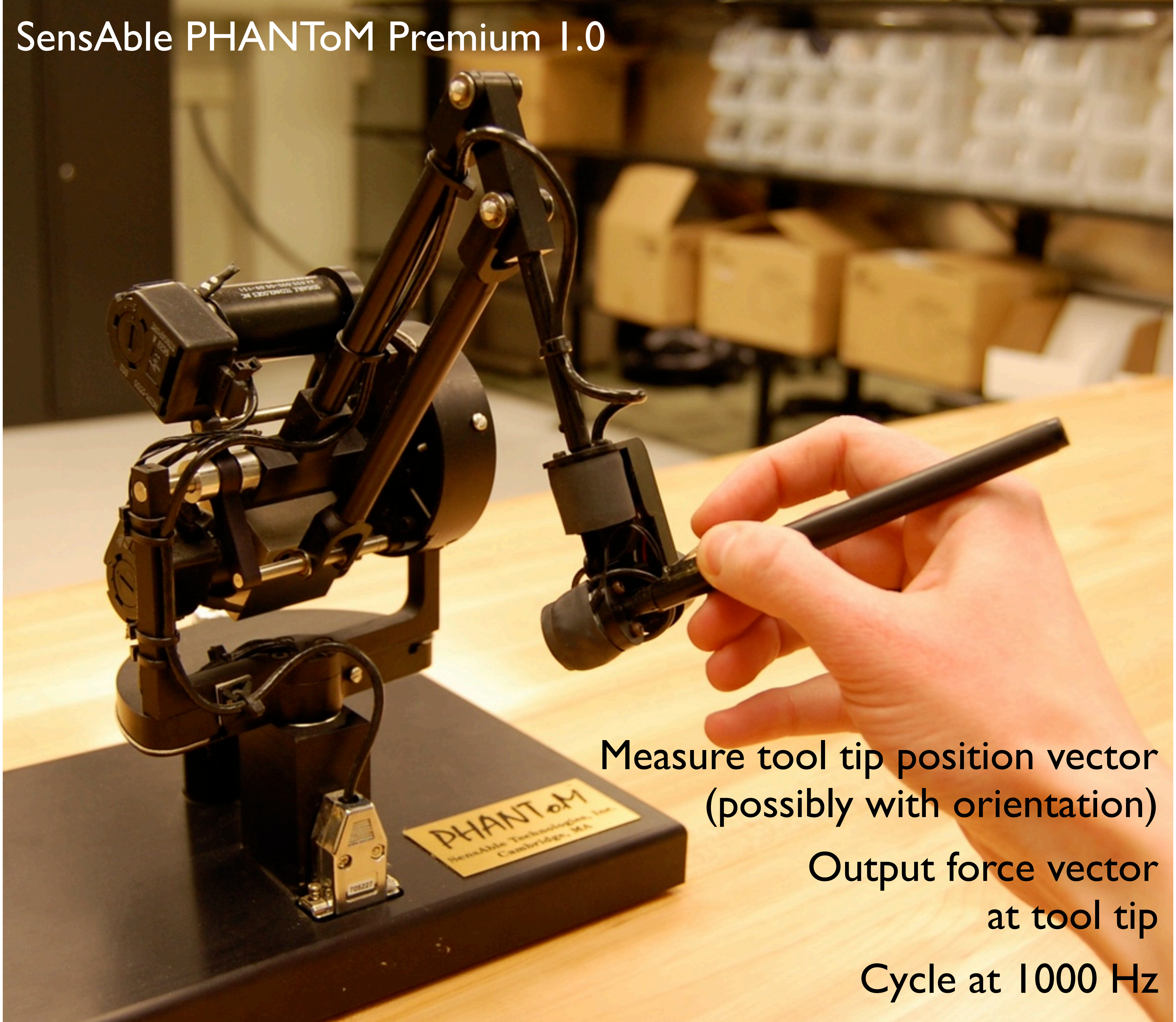
SensAble PHANToM Premium 1.0



Measure tool tip position vector
(possibly with orientation)

Output force vector
at tool tip

SensAble PHANToM Premium 1.0



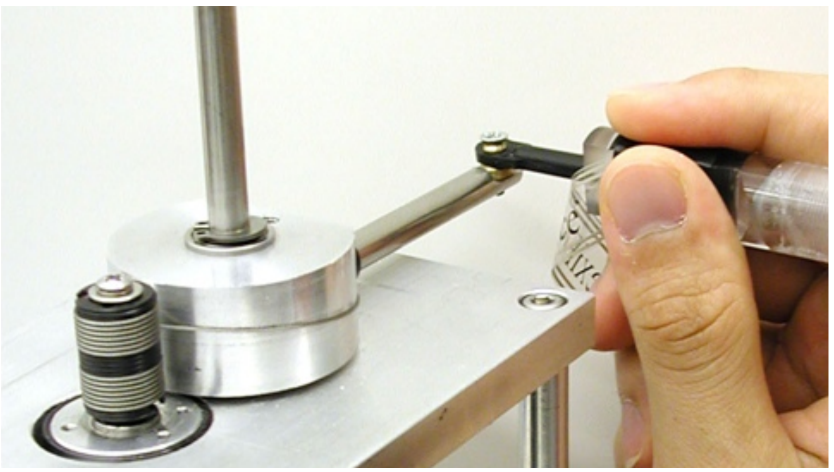
Measure tool tip position vector (possibly with orientation)

Output force vector at tool tip

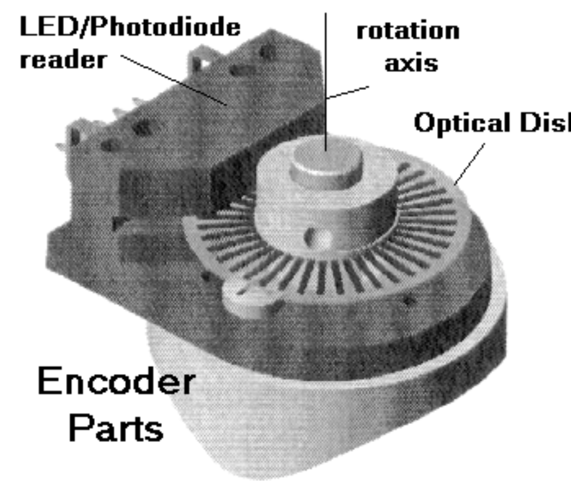
Cycle at 1000 Hz

Typical Components of Kinesthetic Haptic Interfaces

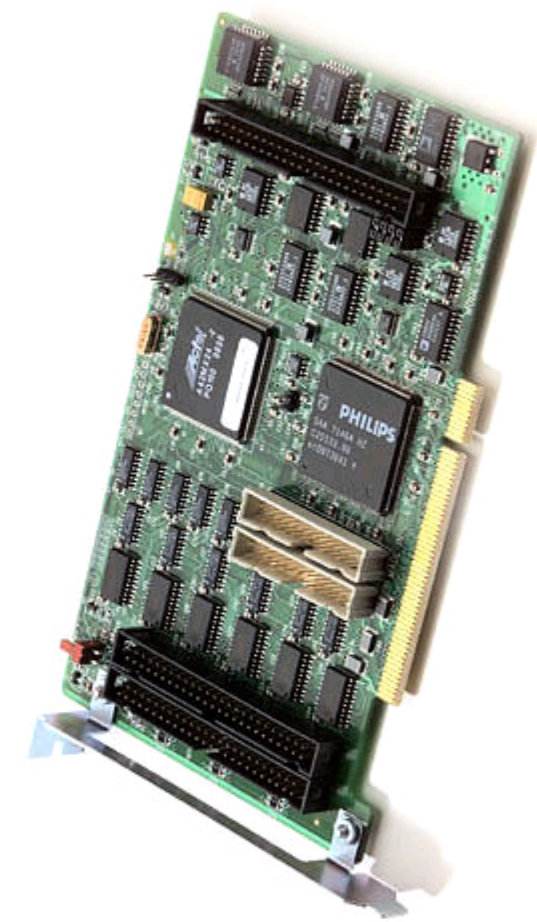
Typical Components of Kinesthetic Haptic Interfaces



Capstan & Cable Drive
Stiff Metal Linkages

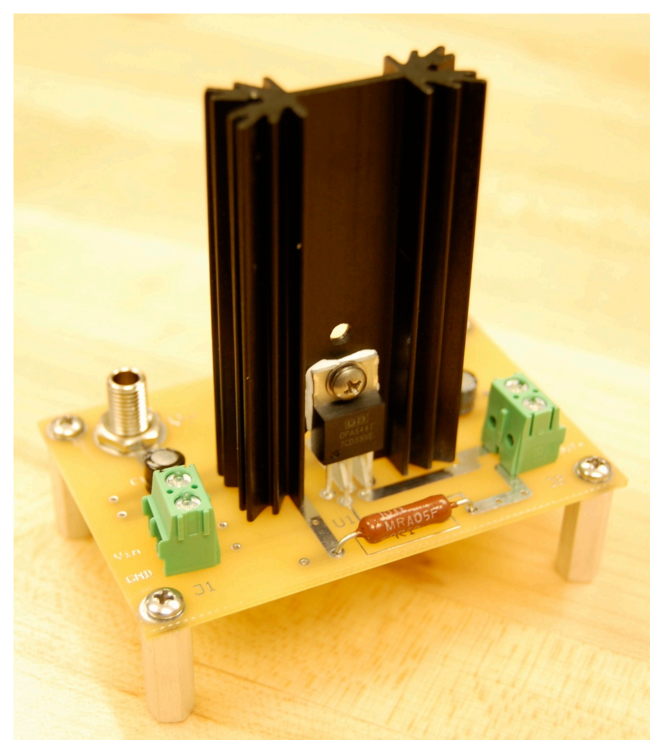


Incremental
Optical Encoder

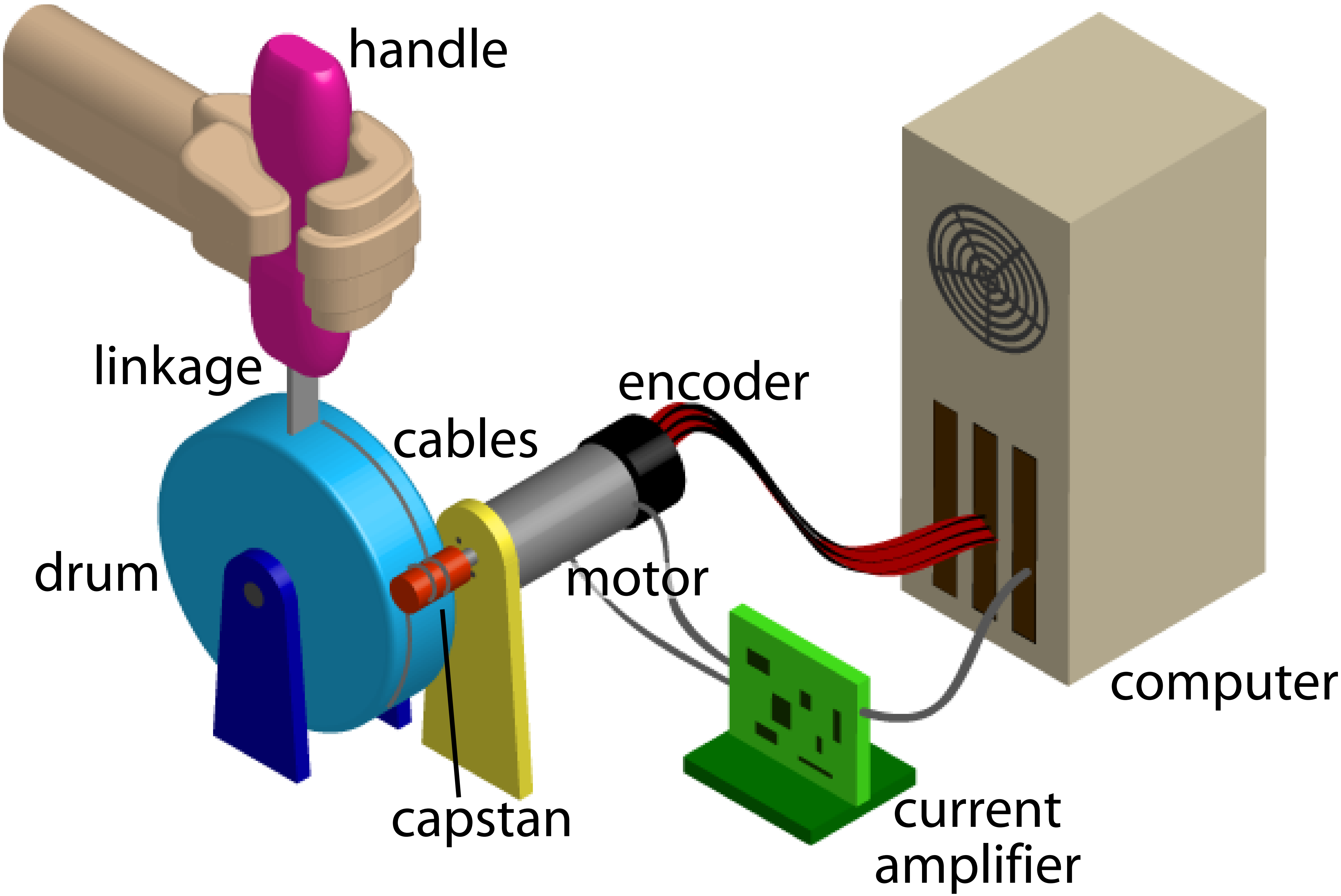


Computer Interface Card

Brushed Permanent Magnet
Direct Current Motor



Current Amplifier



K. J. Kuchenbecker and G. Niemeyer. Modeling induced master motion in force-reflecting teleoperation. In Proc. IEEE International Conference on Robotics and Automation, pages 348–353, Apr. 2005.

Elements of Haptic Interfaces

Katherine J. Kuchenbecker

Department of Mechanical Engineering and Applied Mechanics
University of Pennsylvania
kuchenbe@seas.upenn.edu

Course Notes for MEAM 625, University of Pennsylvania

Adapted from Section 3.1 of Professor Kuchenbecker's Ph.D. thesis [3].

A haptic interface plays the important role of connecting the user to the controller during interactions with remote and virtual objects. Such systems incorporate mechanical, electrical, and computational elements, which all interact to create the touch-based sensations experienced by the user. This document is concerned specifically with actuated impedance-type interfaces, which currently dominate the field due to their excellent free-space characteristics and their widespread use in a variety of applications. During an interaction, the controller of an impedance-type device must measure the user's hand motion and apply an appropriate force in response. Impedance-type haptic interfaces vary in design, but they usually include a series of electrical and mechanical elements between the handle and the computer, as described below.

Overview

Haptic interfaces typically provide two or three degrees of freedom in position, sensing the user's motion and applying feedback forces within this workspace. Many devices also permit changes in the orientation of the end effector; these rotational degrees of freedom can be unsensed, sensed but not actuated, or sensed and actuated. The remainder of this discussion will focus on translation rather than orientation, though the described design features can be applied to either.

Figure 1 illustrates the chain of elements typically present in each axis of a haptic interface. For clarity, the illustration depicts a device with a single degree of freedom, but typical systems combine several degrees of freedom in parallel or series to allow unrestricted translation and/or orientation. Although differences exist, individual position axes of most mechanisms can be represented by such an arrangement. The terms "haptic interface" and "master" are often used interchangeably to represent all electrical and mechanical elements depicted in Figure 1, extending from the amplifier and encoder to the handle.

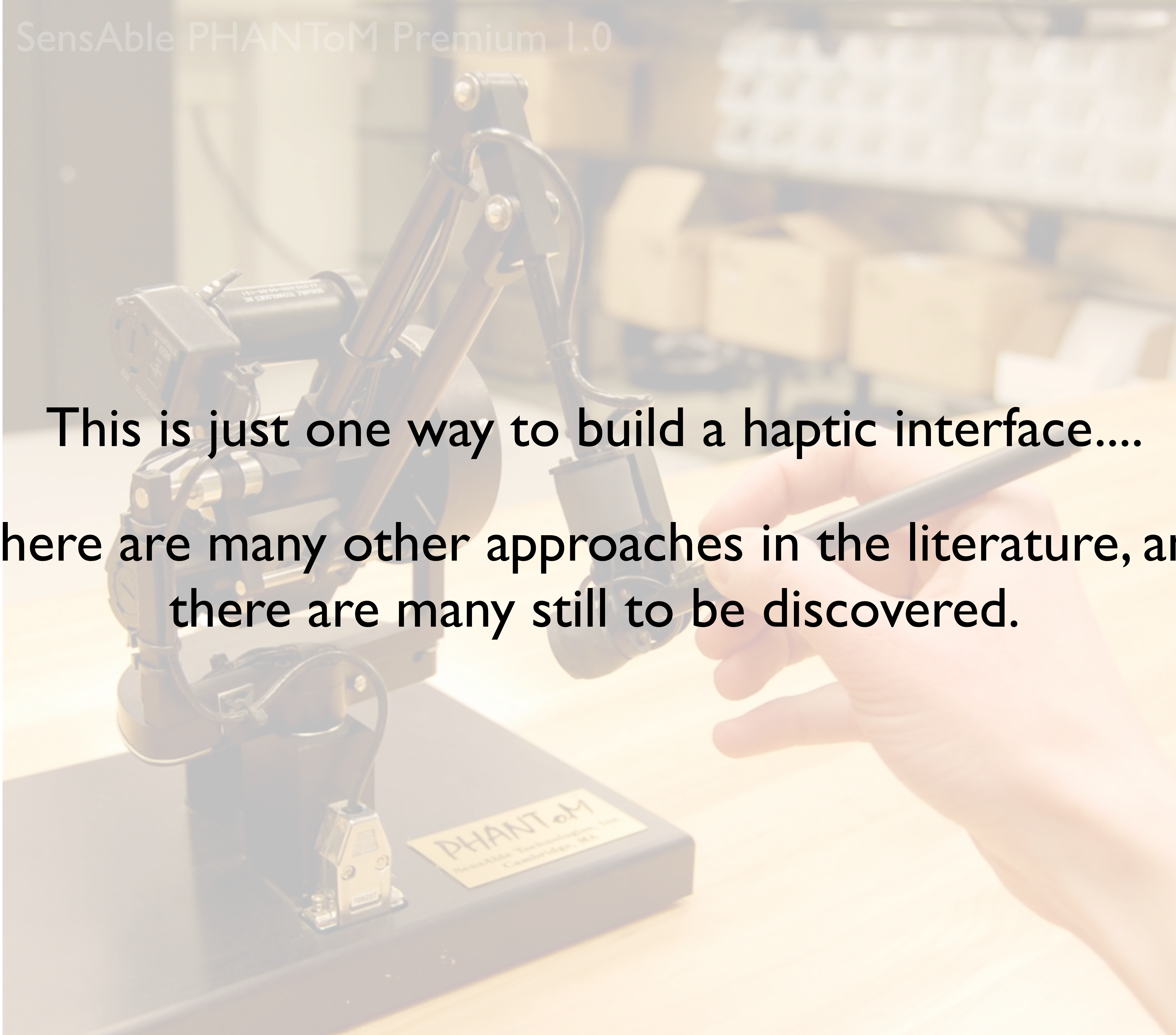
SensAble PHANToM Premium 1.0

SensAble PHANToM Premium 1.0

SensAble PHANToM Premium 1.0

This is just one way to build a haptic interface....

SensAble PHANToM Premium 1.0



This is just one way to build a haptic interface....

There are many other approaches in the literature, and there are many still to be discovered.

My definition of a haptic interface:

Senses a physical quantity from the user, such as motion or force

Physically acts on the user via a variable actuator

Connects sensing to acting with fast processing

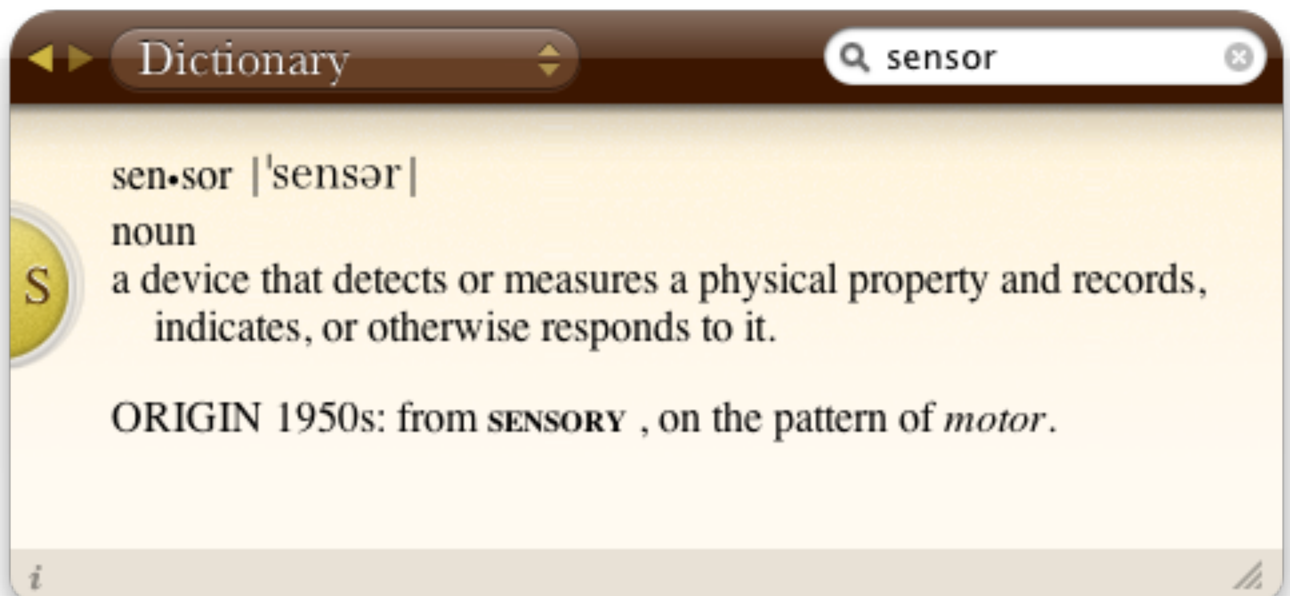
My definition of a haptic interface:

Senses a physical quantity from the user, such as motion or force

Physically acts on the user via a variable actuator

Connects sensing to acting with fast processing

Sensors

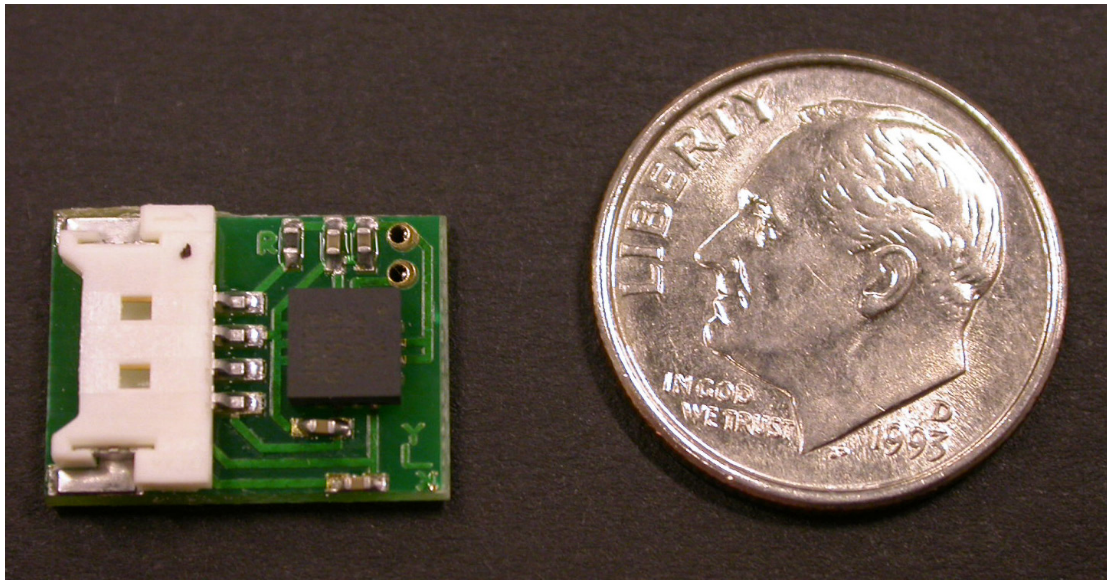

Dictionary
sensor
sen•sor |ˈsensər|
noun
a device that detects or measures a physical property and records, indicates, or otherwise responds to it.
ORIGIN 1950s: from SENSORY , on the pattern of motor.
S
i

Sensors

- Sensor Specifications
- Sensor Types
- Read the Datasheet

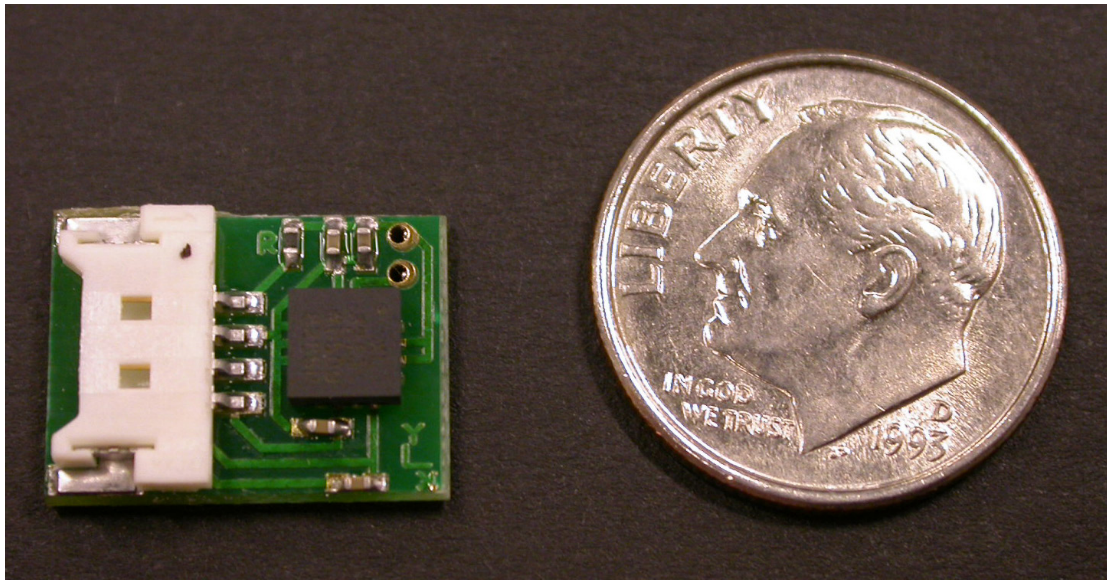
Sensors

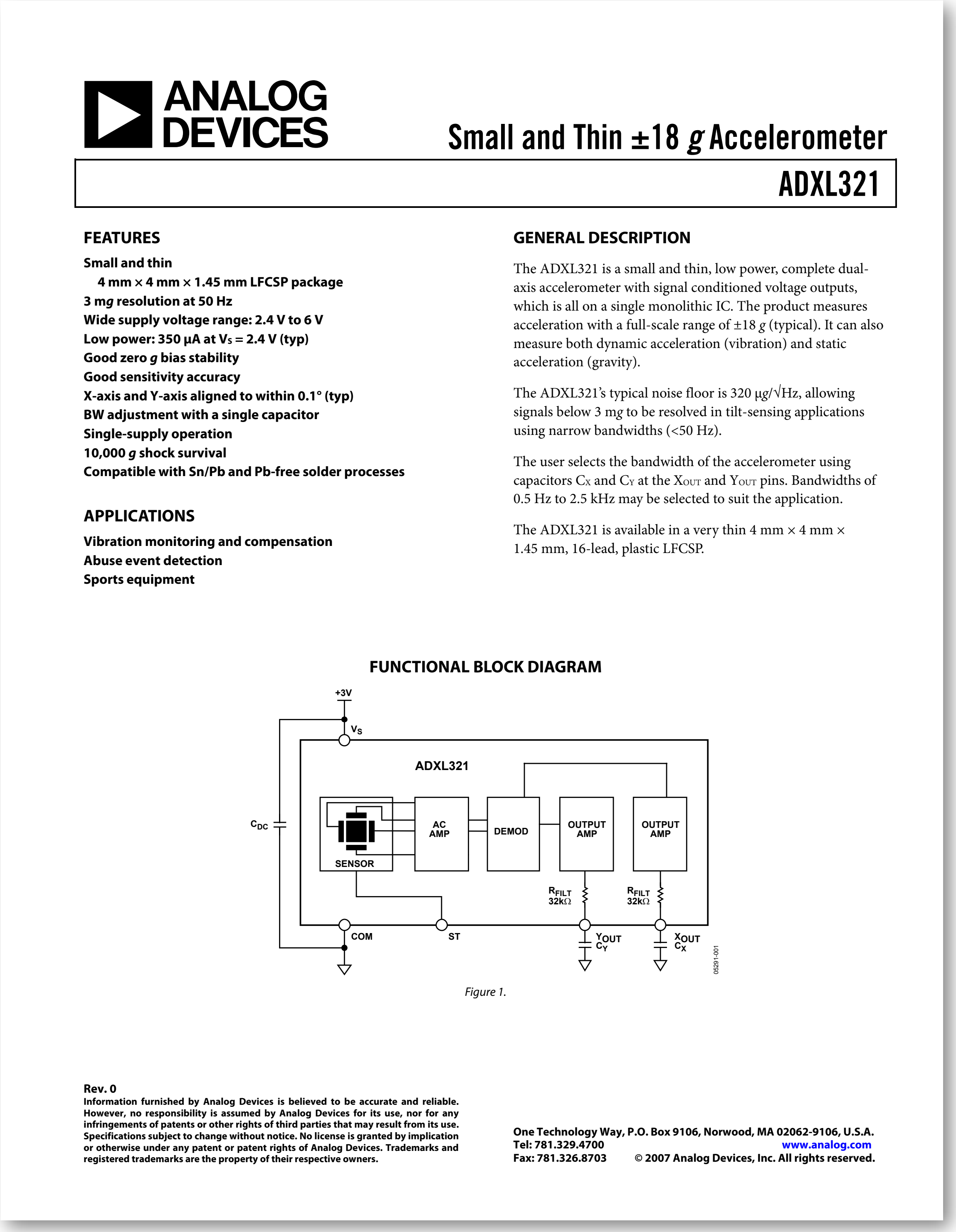
- Sensor Specifications
- Sensor Types
- Read the Datasheet



Sensors

- Sensor Specifications
- Sensor Types
- Read the Datasheet





ANALOG DEVICES

Small and Thin ±18 g Accelerometer

ADXL321

FEATURES

Small and thin
4 mm × 4 mm × 1.45 mm LFCSP package
3 mg resolution at 50 Hz
Wide supply voltage range: 2.4 V to 6 V
Low power: 350 μA at V_S = 2.4 V (typ)
Good zero g bias stability
Good sensitivity accuracy
X-axis and Y-axis aligned to within 0.1° (typ)
BW adjustment with a single capacitor
Single-supply operation
10,000 g shock survival
Compatible with Sn/Pb and Pb-free solder processes

APPLICATIONS

Vibration monitoring and compensation
Abuse event detection
Sports equipment

GENERAL DESCRIPTION

The ADXL321 is a small and thin, low power, complete dual-axis accelerometer with signal conditioned voltage outputs, which is all on a single monolithic IC. The product measures acceleration with a full-scale range of ±18 g (typical). It can also measure both dynamic acceleration (vibration) and static acceleration (gravity).

The ADXL321's typical noise floor is 320 μg/√Hz, allowing signals below 3 mg to be resolved in tilt-sensing applications using narrow bandwidths (<50 Hz).

The user selects the bandwidth of the accelerometer using capacitors C_X and C_Y at the X_{OUT} and Y_{OUT} pins. Bandwidths of 0.5 Hz to 2.5 kHz may be selected to suit the application.

The ADXL321 is available in a very thin 4 mm × 4 mm × 1.45 mm, 16-lead, plastic LFCSP.

FUNCTIONAL BLOCK DIAGRAM

Figure 1.

Rev. 0
Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

SPECIFICATIONS[1]

$T_A = 25°C$, $V_S = 3$ V, $C_X = C_Y = 0.1$ µF, Acceleration = 0 *g*, unless otherwise noted.

Table 1.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range			±18		*g*
Nonlinearity	% of full scale		±0.2		%
Package Alignment Error			±1		Degrees
Alignment Error	X sensor to Y sensor		±0.1		Degrees
Cross Axis Sensitivity			±2		%
SENSITIVITY (RATIOMETRIC)[2]	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}	$V_S = 3$ V	51	57	63	mV/*g*
Sensitivity Change due to Temperature[3]	$V_S = 3$ V		0.01		%/°C
ZERO *g* BIAS LEVEL (RATIOMETRIC)	Each axis				
0 *g* Voltage at X_{OUT}, Y_{OUT}	$V_S = 3$ V	1.4	1.5	1.6	V
0 *g* Offset vs. Temperature			±2		m*g*/°C
NOISE PERFORMANCE					
Noise Density	@ 25°C		320		µ*g*/√Hz rms
FREQUENCY RESPONSE[4]					
C_X, C_Y Range[5]		0.002		10	µF
R_{FILT} Tolerance			32 ± 15%		kΩ
Sensor Resonant Frequency			5.5		kHz
SELF-TEST[6]					
Logic Input Low			0.6		V
Logic Input High			2.4		V
ST Input Resistance to Ground			50		kΩ
Output Change at X_{OUT}, Y_{OUT}	Self-test 0 to 1		18		mV
OUTPUT AMPLIFIER					
Output Swing Low	No load		0.3		V
Output Swing High	No load		2.6		V
POWER SUPPLY					
Operating Voltage Range		2.4		6	V
Quiescent Supply Current			0.49		mA
Turn-On Time[7]			20		ms
TEMPERATURE					
Operating Temperature Range		−20		+70	°C

[1] All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.
[2] Sensitivity is essentially ratiometric to V_S.
[3] Defined as the change from ambient-to-maximum temperature or ambient-to-minimum temperature.
[4] Actual frequency response controlled by user-supplied external capacitor (C_X, C_Y).
[5] Bandwidth = $1/(2 \times \pi \times 32\ k\Omega \times C)$. For C_X, C_Y = 0.002 µF, bandwidth = 2500 Hz. For C_X, C_Y = 10 µF, bandwidth = 0.5 Hz. Minimum/maximum values are not tested.
[6] Self-test response changes cubically with V_S.
[7] Larger values of C_X, C_Y increase turn-on time. Turn-on time is approximately $160 \times C_X$ or C_Y + 4 ms, where C_X, C_Y are in µF.

SPECIFICATIONS[1]

$T_A = 25°C$, $V_S = 3$ V, $C_X = C_Y = 0.1$ µF, Acceleration = 0 *g*, unless otherwise noted.

Table 1.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range			±18		*g*
Nonlinearity	% of full scale		±0.2		%
Package Alignment Error			±1		Degrees
Alignment Error	X sensor to Y sensor		±0.1		Degrees
Cross Axis Sensitivity			±2		%
SENSITIVITY (RATIOMETRIC)[2]	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}	$V_S = 3$ V	51	57	63	mV/*g*
Sensitivity Change due to Temperature[3]	$V_S = 3$ V		0.01		%/°C
ZERO *g* BIAS LEVEL (RATIOMETRIC)	Each axis				
0 *g* Voltage at X_{OUT}, Y_{OUT}	$V_S = 3$ V	1.4	1.5	1.6	V
0 *g* Offset vs. Temperature			±2		m*g*/°C
NOISE PERFORMANCE					
Noise Density	@ 25°C		320		µ*g*/√Hz rms
FREQUENCY RESPONSE[4]					
C_X, C_Y Range[5]		0.002		10	µF
R_{FILT} Tolerance			32 ± 15%		kΩ
Sensor Resonant Frequency			5.5		kHz
SELF-TEST[6]					
Logic Input Low			0.6		V
Logic Input High			2.4		V
ST Input Resistance to Ground			50		kΩ
Output Change at X_{OUT}, Y_{OUT}	Self-test 0 to 1		18		mV
OUTPUT AMPLIFIER					
Output Swing Low	No load		0.3		V
Output Swing High	No load		2.6		V
POWER SUPPLY					
Operating Voltage Range		2.4		6	V
Quiescent Supply Current			0.49		mA
Turn-On Time[7]			20		ms
TEMPERATURE					
Operating Temperature Range		−20		+70	°C

SENSOR INPUT	Each axis				
Measurement Range			±18		*g*
Nonlinearity	% of full scale		±0.2		%
Package Alignment Error			±1		Degrees
Alignment Error	X sensor to Y sensor		±0.1		Degrees
Cross Axis Sensitivity			±2		%
SENSITIVITY (RATIOMETRIC)[2]	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}	$V_S = 3$ V	51	57	63	mV/*g*
Sensitivity Change due to Temperature[3]	$V_S = 3$ V		0.01		%/°C
ZERO *g* BIAS LEVEL (RATIOMETRIC)	Each axis				
0 *g* Voltage at X_{OUT}, Y_{OUT}	$V_S = 3$ V	1.4	1.5	1.6	V
0 *g* Offset vs. Temperature			±2		m*g*/°C
NOISE PERFORMANCE					
Noise Density	@ 25°C		320		μ*g*/√Hz rms
FREQUENCY RESPONSE[4]					
C_X, C_Y Range[5]		0.002		10	μF
R_{FILT} Tolerance			32 ± 15%		kΩ
Sensor Resonant Frequency			5.5		kHz
SELF-TEST[6]					
Logic Input Low			0.6		V
Logic Input High			2.4		V
ST Input Resistance to Ground			50		kΩ
Output Change at X_{OUT}, Y_{OUT}	Self-test 0 to 1		18		mV
OUTPUT AMPLIFIER					
Output Swing Low	No load		0.3		V
Output Swing High	No load		2.6		V
POWER SUPPLY					
Operating Voltage Range		2.4		6	V
Quiescent Supply Current			0.49		mA
Turn-On Time[7]			20		ms
TEMPERATURE					
Operating Temperature Range		−20		+70	°C

[1] All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.
[2] Sensitivity is essentially ratiometric to V_S.
[3] Defined as the change from ambient-to-maximum temperature or ambient-to-minimum temperature.
[4] Actual frequency response controlled by user-supplied external capacitor (C_X, C_Y).
[5] Bandwidth = $1/(2 \times \pi \times 32\ k\Omega \times C)$. For C_X, C_Y = 0.002 μF, bandwidth = 2500 Hz. For C_X, C_Y = 10 μF, bandwidth = 0.5 Hz. Minimum/maximum values are not tested.
[6] Self-test response changes cubically with V_S.
[7] Larger values of C_X, C_Y increase turn-on time. Turn-on time is approximately $160 \times C_X$ or $C_Y + 4$ ms, where C_X, C_Y are in μF.

Sensor Specifications

Static Measures

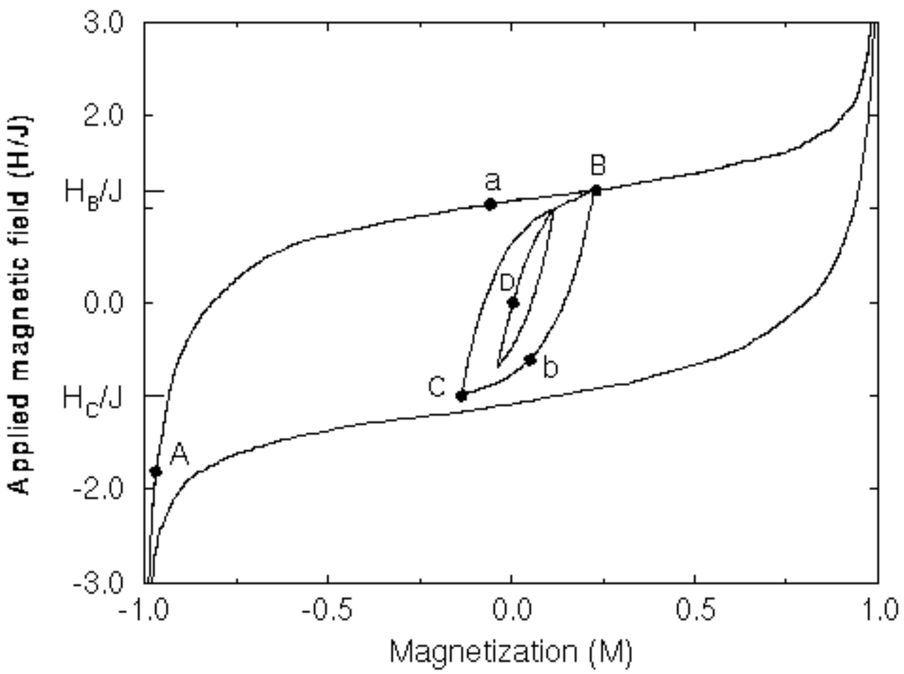
- Range: minimum to maximum of measurable physical quantity, e.g., 10-20 PSI
- Span: the limits between minimum and maximum value the sensor can measure
- Error: measured value minus true value, often associated with a specific cause
- Accuracy: total of the effects of all errors
- Sensitivity: Gain (output divided by input), may be ratiometric with supply voltage

Dynamic Measures

- Response Time - time to achieve 95% of final value
- Time Constant - time to achieve 63% of final value
- Rise time - time from 10% to 90% of final value
- Settling time - time to get and stay within 2% of final value

Imperfections

- Hysteresis: Sensitivity to direction of change
- Nonlinearity: deviation from linear relationship (constant gain)
- Repeatability: produces the same output for the same input (not the same as accuracy)
- Stability: holding the same value over a period of time
- Deadband/Dead Time: period or range of input where no output occurs
- Resolution: Smallest change in input that will cause a change in output
- Output Impedance: ability to deliver current, lower is better



Sensor Types

Sensor Types

- All convert a physical effect into an electrical signal

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  - Cost

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  - Cost
  - Size

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  - Cost
  - Size
  - Accuracy

Sensor Types

- All convert a physical effect into an electrical signal
  – Measure a voltage or current using capacitance, resistance, inductance
  – May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  – Cost
  – Size
  – Accuracy
  – Durability

Sensor Types

- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  - Cost
  - Size
  - Accuracy
  - Durability
  - Availability

Sensor Types

- All convert a physical effect into an electrical signal
  – Measure a voltage or current using capacitance, resistance, inductance
  – May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  – Cost
  – Size
  – Accuracy
  – Durability
  – Availability
  – Compatibility with rest of system (interference, communication)

Sensor Types

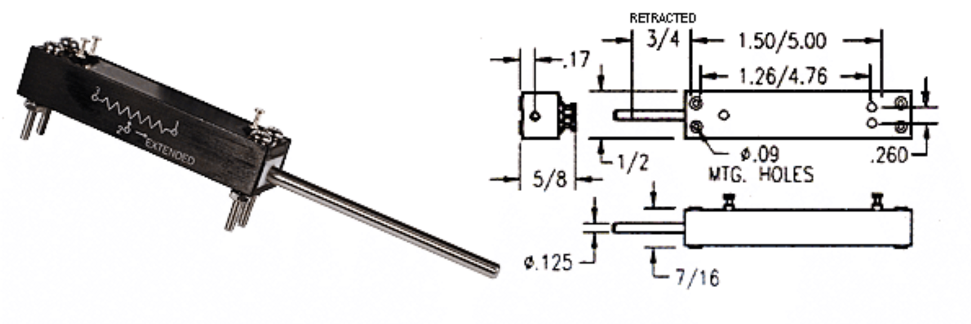
- All convert a physical effect into an electrical signal
  - Measure a voltage or current using capacitance, resistance, inductance
  - May output analog voltage, digital signal, serial comm., current, etc.
- Choosing a sensor is always about:
  - Cost
  - Size
  - Accuracy
  - Durability
  - Availability
  - Compatibility with rest of system (interference, communication)
  - Other concerns?

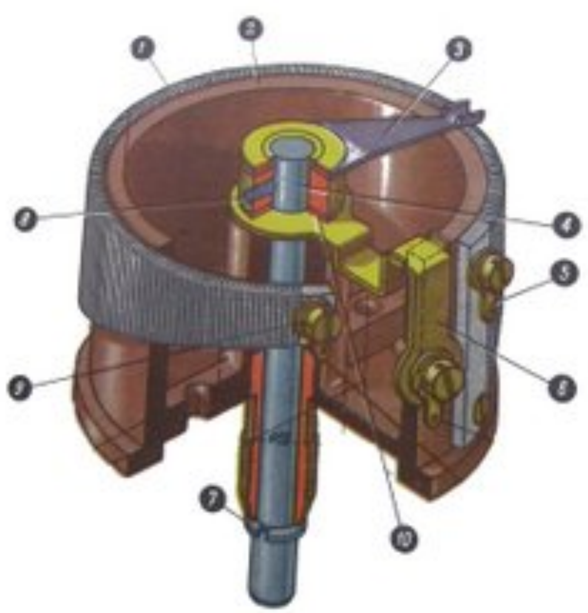
Mechanical Trackers

- Ground-based linkages most commonly used
- Position Sensors
  - Analog: potentiometers or Hall-effect (magnetic)
  - Digital: encoders (optical or MR)

Adapted from slides by Will Provancher

Potentiometers

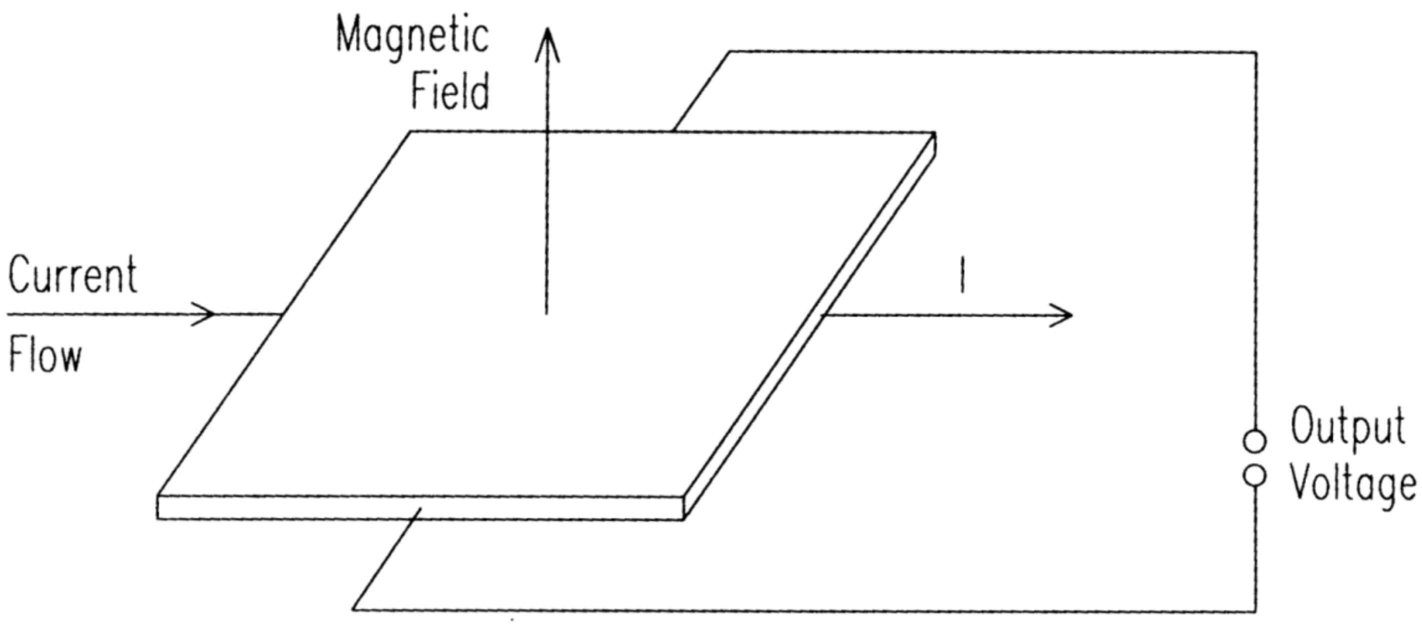




- Typically rotary, but linear exist.
- Cheap and easy.
- Moving parts means it can wear out.
- Hard to waterproof or dustproof.
- Has non-negligible friction.

Hall-Effect Sensors

- How do they work?
  - A small transverse voltage is generated across a current-carrying conductor in the presence of a magnetic field



(Discovery made in 1879, but not useful until the advent of semiconductor technology)

Adapted from slides by Will Provancher

Hall-Effect Sensors

V_h = Hall voltage
R_h = Hall coefficient
I = Current
B = Magnetic flux density
t = Element thickness

$$V_h = \frac{R_h I B}{t}$$

- Amount of voltage output related to the strength of magnetic field passing through.
- Linear over small range of motion
  - Need to be calibrated
- Affected by temperature, other magnetic objects in the environments

Adapted from slides by Will Provancher

Hall-Effect Sensors

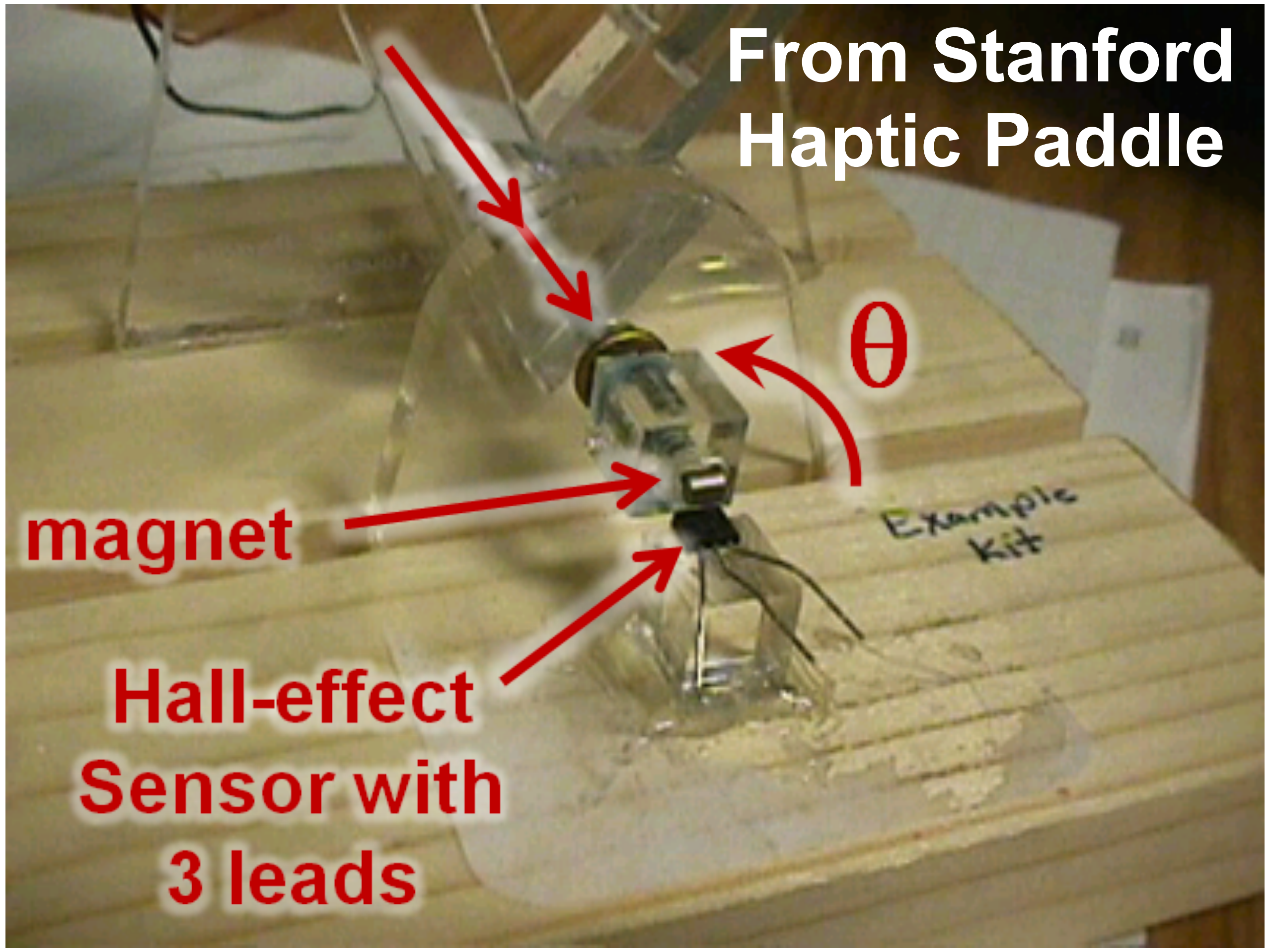
V_h = Hall voltage

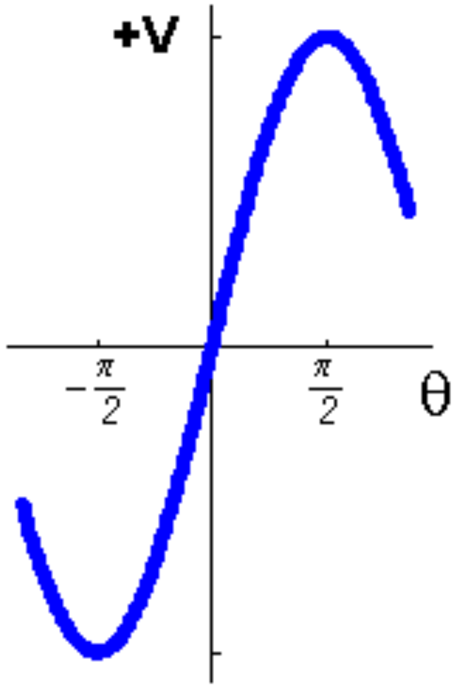
R_h = Hall coefficient

I = Current

B = Magnetic flux density

t = Element thickness

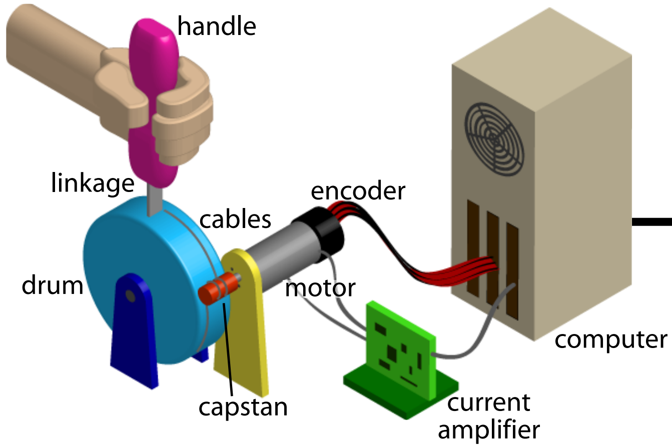




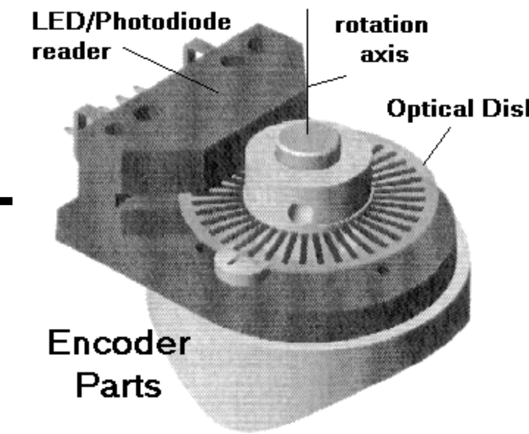
$$V_h = \frac{R_h I B}{t}$$

- The voltage varies sinusoidally with rotation angle

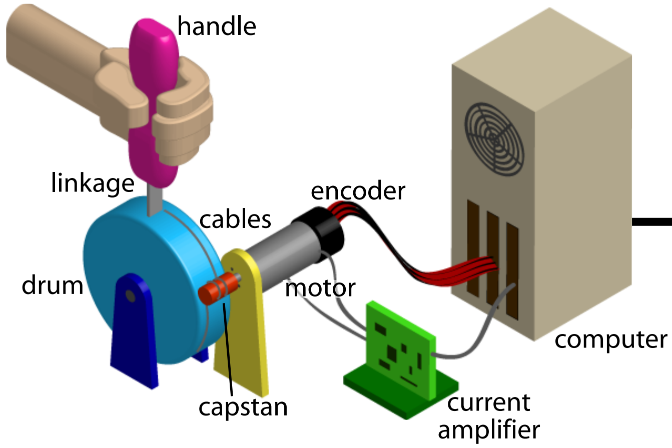
Adapted from slides by Will Provancher



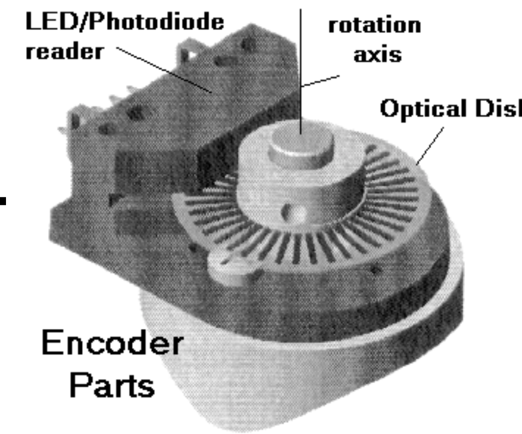
Encoder



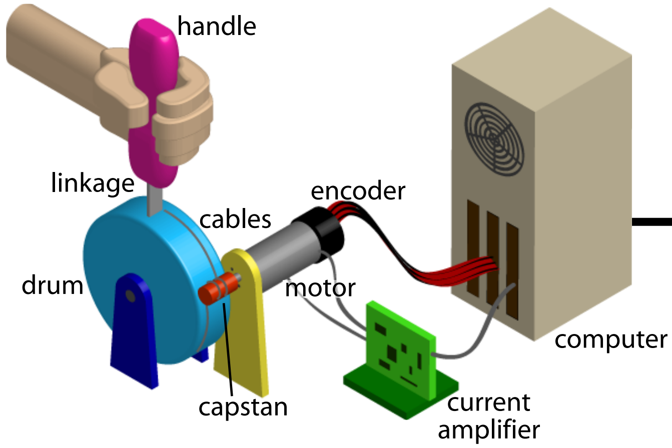
Some material adapted from slides by A. Okamura and W. Provancher



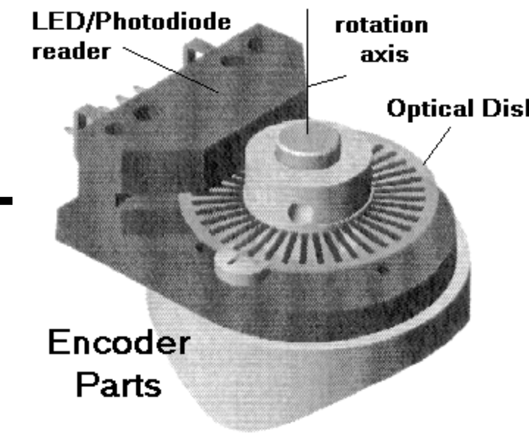
Encoder



The most common motion sensor in haptics is the incremental optical encoder, often by Agilent.

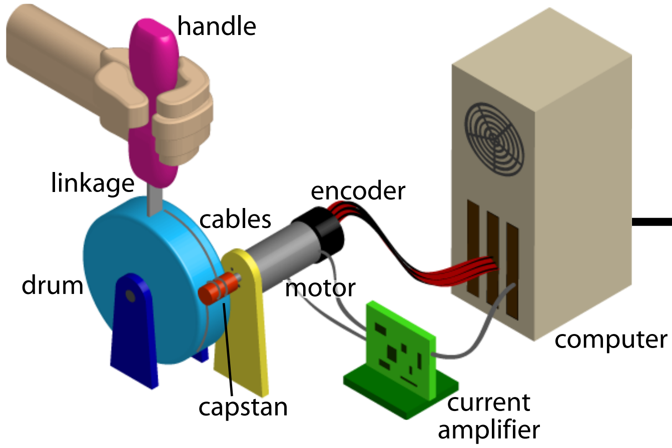


Encoder

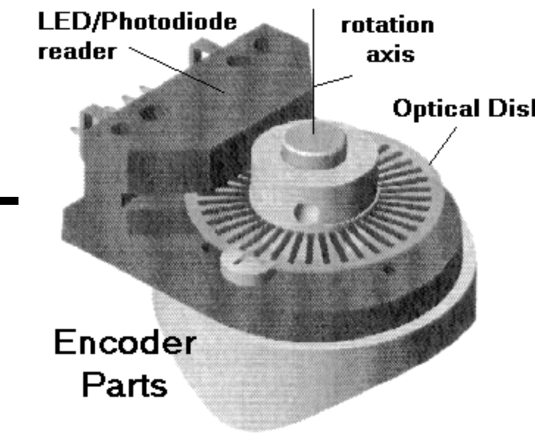


The most common motion sensor in haptics is the incremental optical encoder, often by Agilent.

- A thin disk is attached to the rotating shaft whose angle you want to measure, usually the motor.



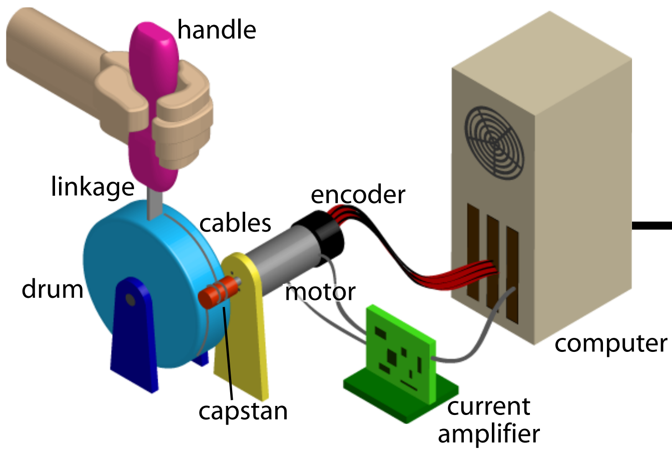
Encoder



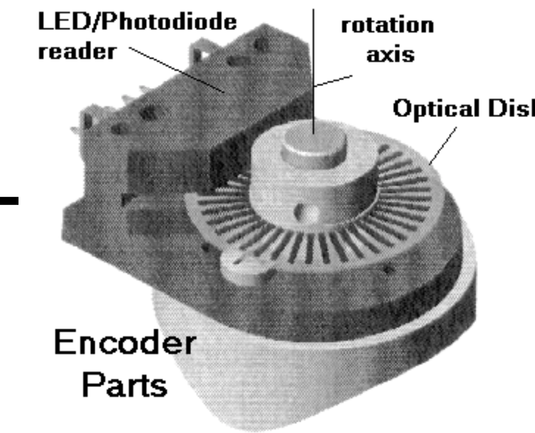
The most common motion sensor in haptics is the incremental optical encoder, often by Agilent.

- A thin disk is attached to the rotating shaft whose angle you want to measure, usually the motor.
- The disk has slits cut into it in a regular pattern.

Some material adapted from slides by A. Okamura and W. Provancher



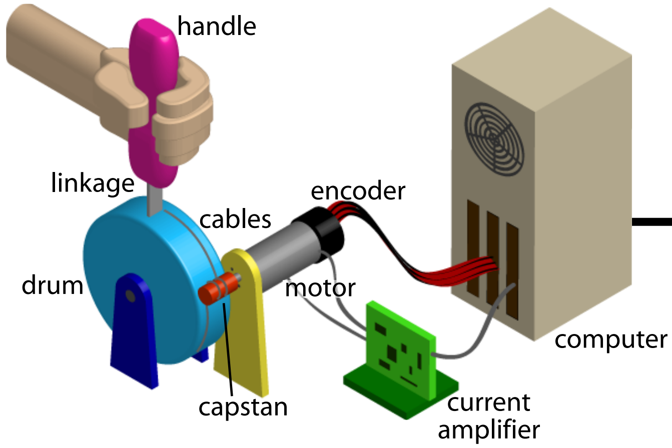
Encoder



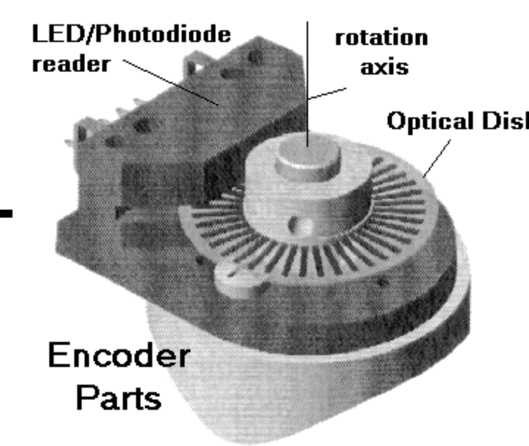
The most common motion sensor in haptics is the incremental optical encoder, often by Agilent.

- A thin disk is attached to the rotating shaft whose angle you want to measure, usually the motor.
- The disk has slits cut into it in a regular pattern.
- A light shines on the disk on one side, and photo sensors are located on the opposite side.

Some material adapted from slides by A. Okamura and W. Provancher



Encoder

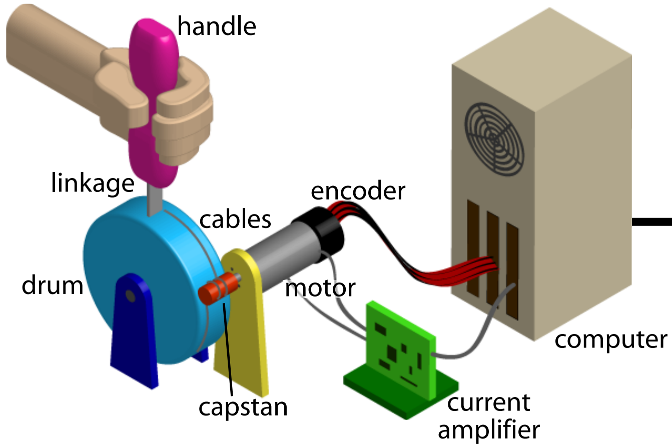


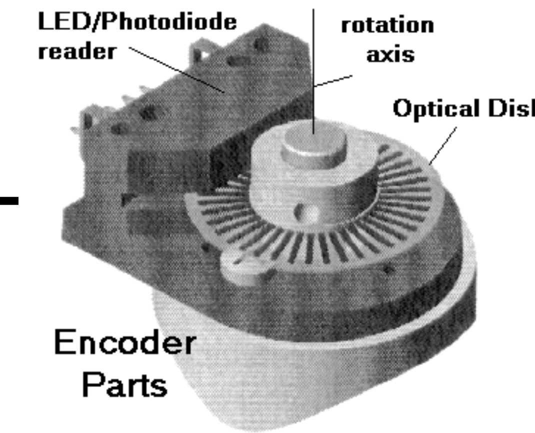
The most common motion sensor in haptics is the incremental optical encoder, often by Agilent.

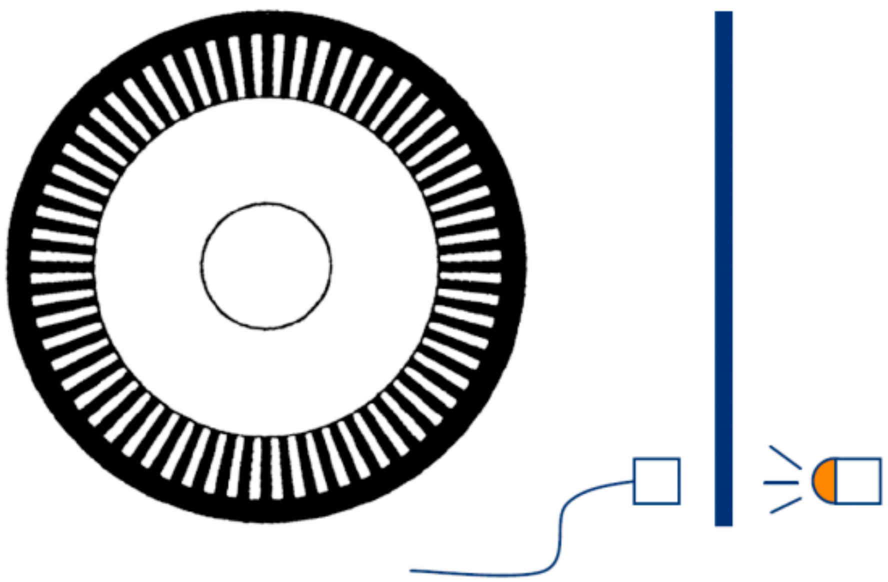
- A thin disk is attached to the rotating shaft whose angle you want to measure, usually the motor.
- The disk has slits cut into it in a regular pattern.
- A light shines on the disk on one side, and photo sensors are located on the opposite side.
- Produces a number of pulses per revolution, with higher resolution being more expensive.

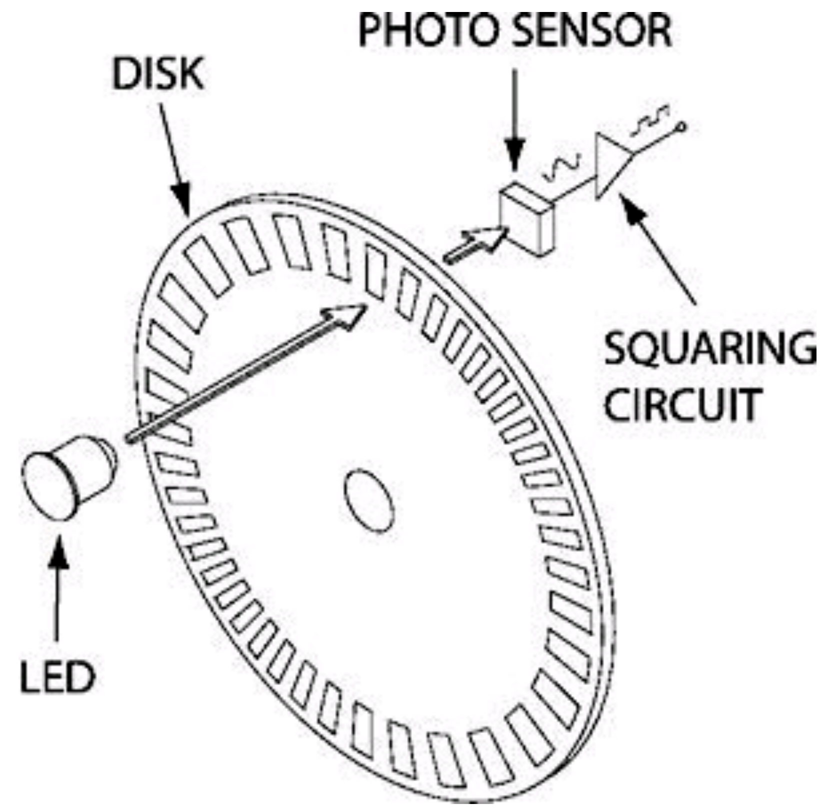
Some material adapted from slides by A. Okamura and W. Provancher

Encoder



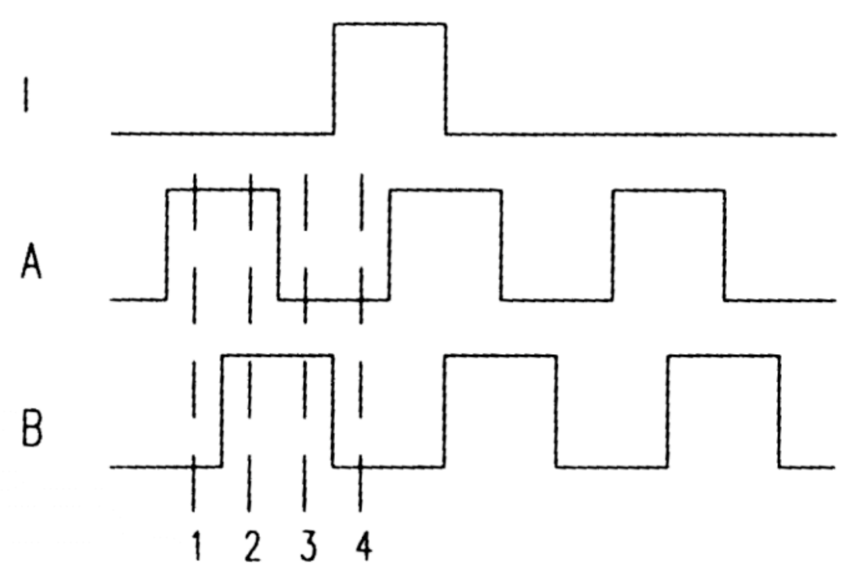






$$\Delta = \frac{2\pi}{4n}$$

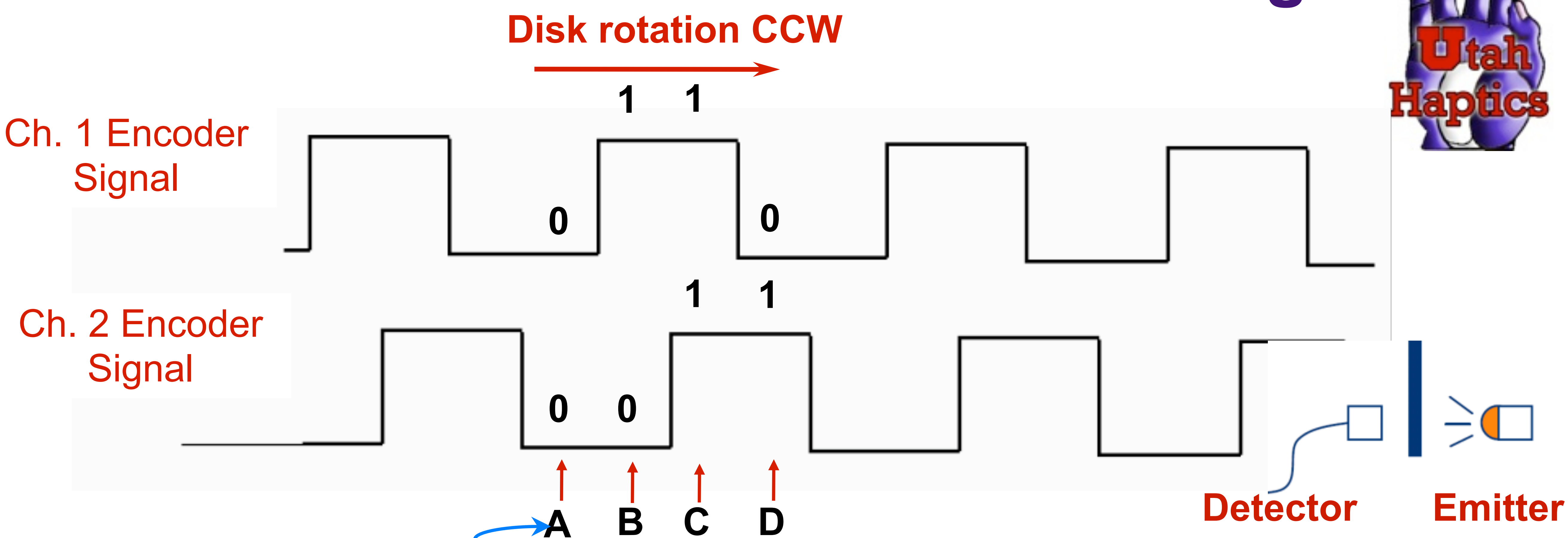
Two channels of pulses, 90 degrees out of phase: quadrature



State	Ch A	Ch B
S_1	High	Low
S_2	High	High
S_3	Low	High
S_4	Low	Low

Some material adapted from slides by A. Okamura and W. Provancher

Quadrature Encoder States & Decoding



Encoder States

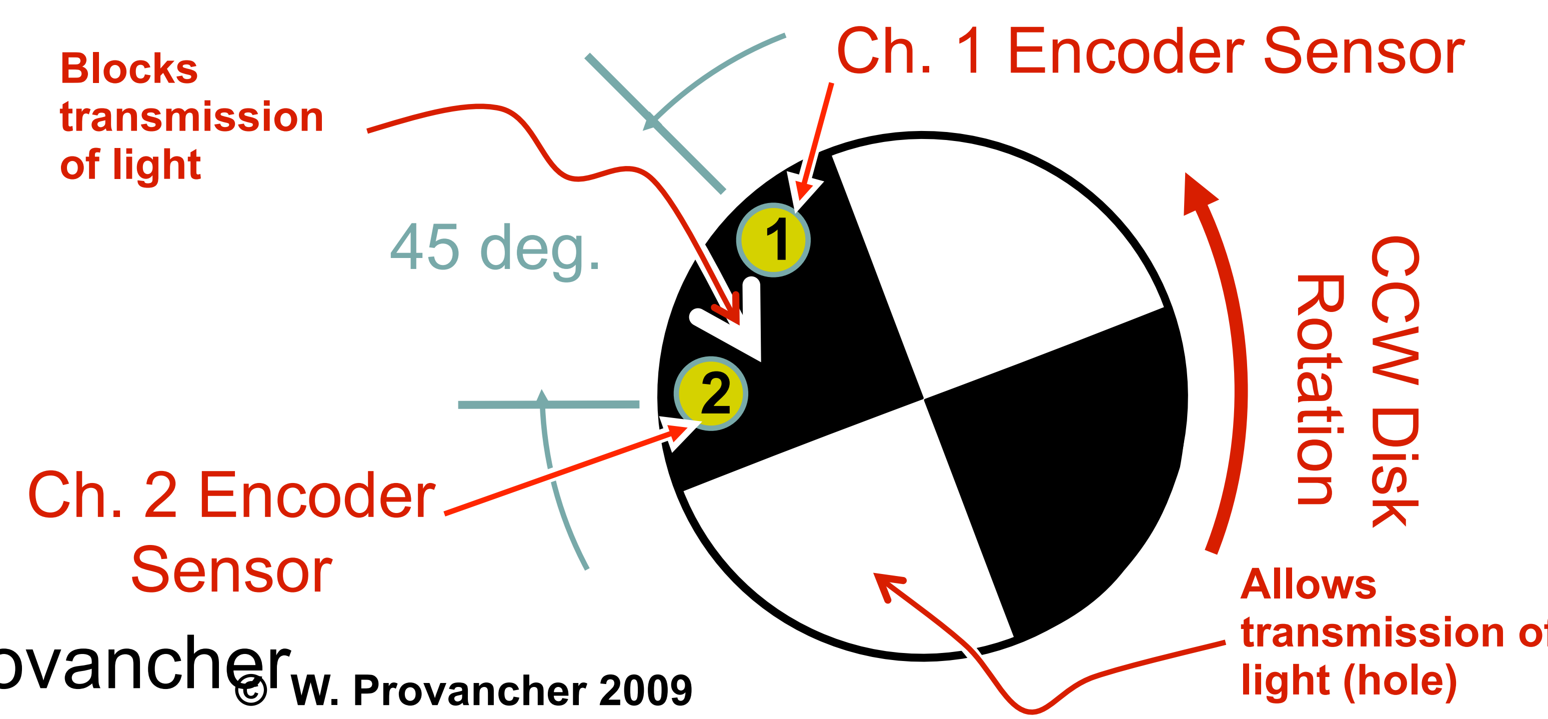
Disk rotation CCW

	A	B	C	D
Ch. 1	0	1	1	0
Ch. 2	0	0	1	1
	A	B	C	D

Disk rotation CW

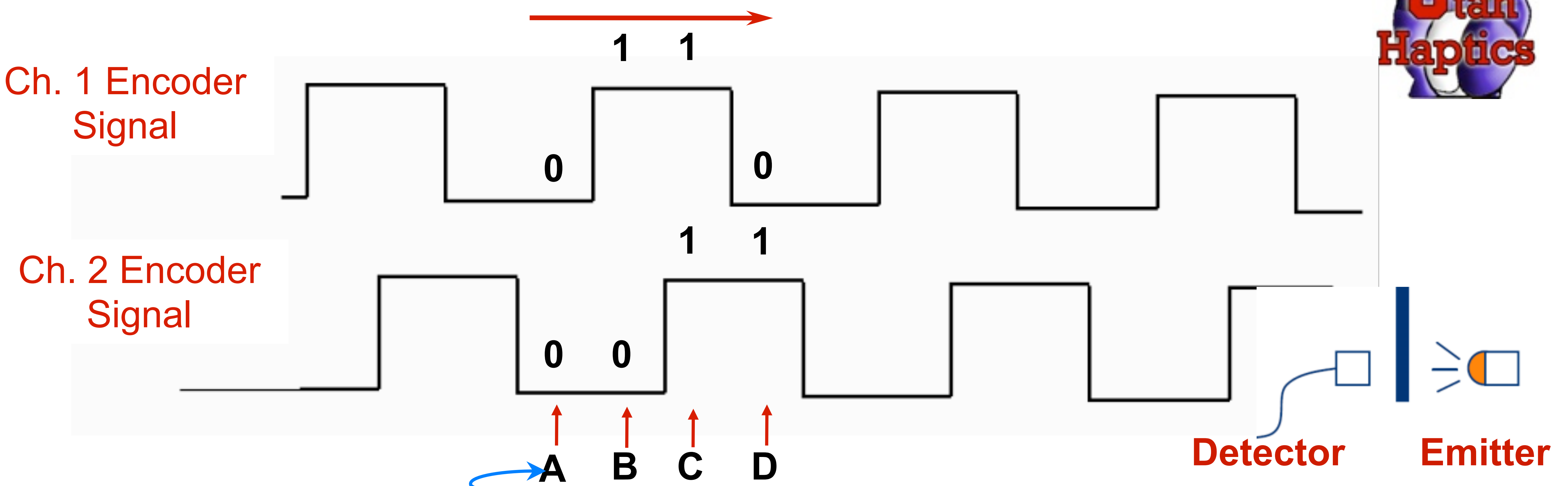
Simplified Encoder Disk

(4 CPR, 16 PPR) (shown in state A)



Adapted from slides by Will Provancher

Quadrature Encoder States & Decoding



Encoder States

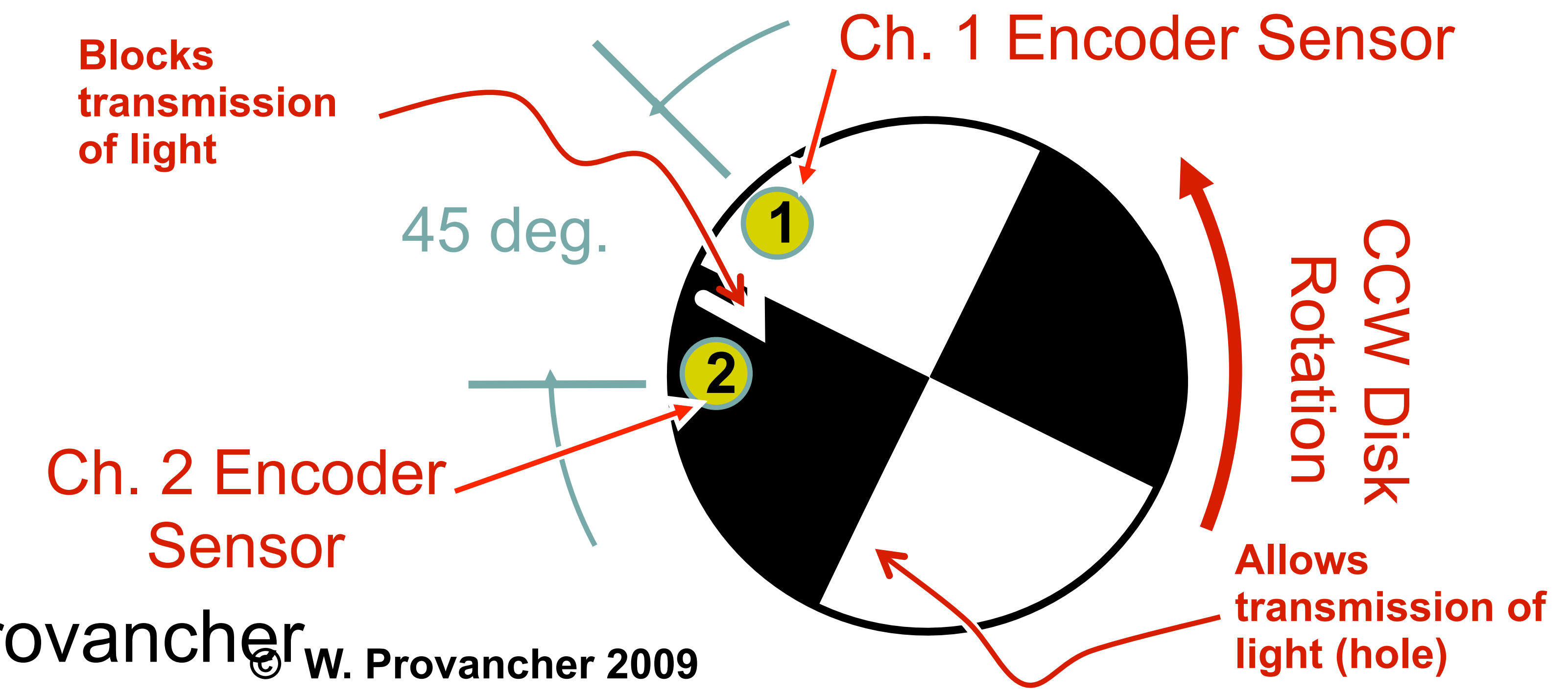
Disk rotation CCW →

	A	B	C	D
Ch. 1	0	1	1	0
Ch. 2	0	0	1	1

← Disk rotation CW

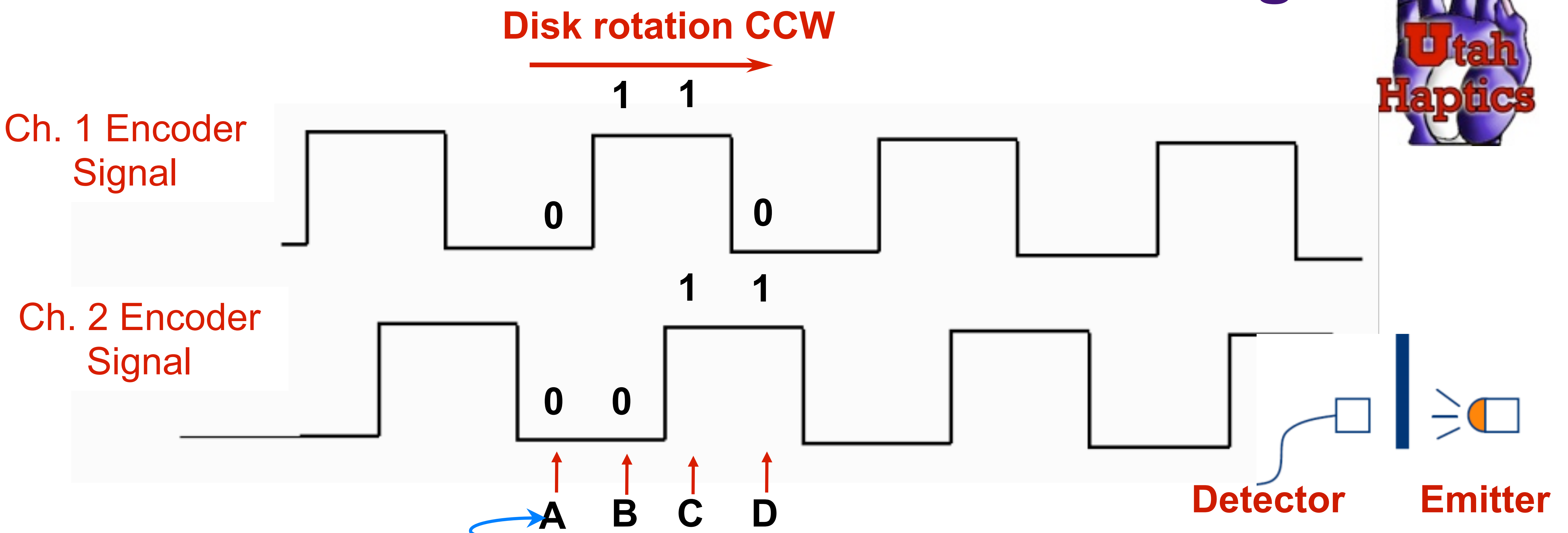
Simplified Encoder Disk

(4 CPR, 16 PPR) (shown in state A)



Adapted from slides by Will Provancher

Quadrature Encoder States & Decoding



Encoder States

Disk rotation CCW →

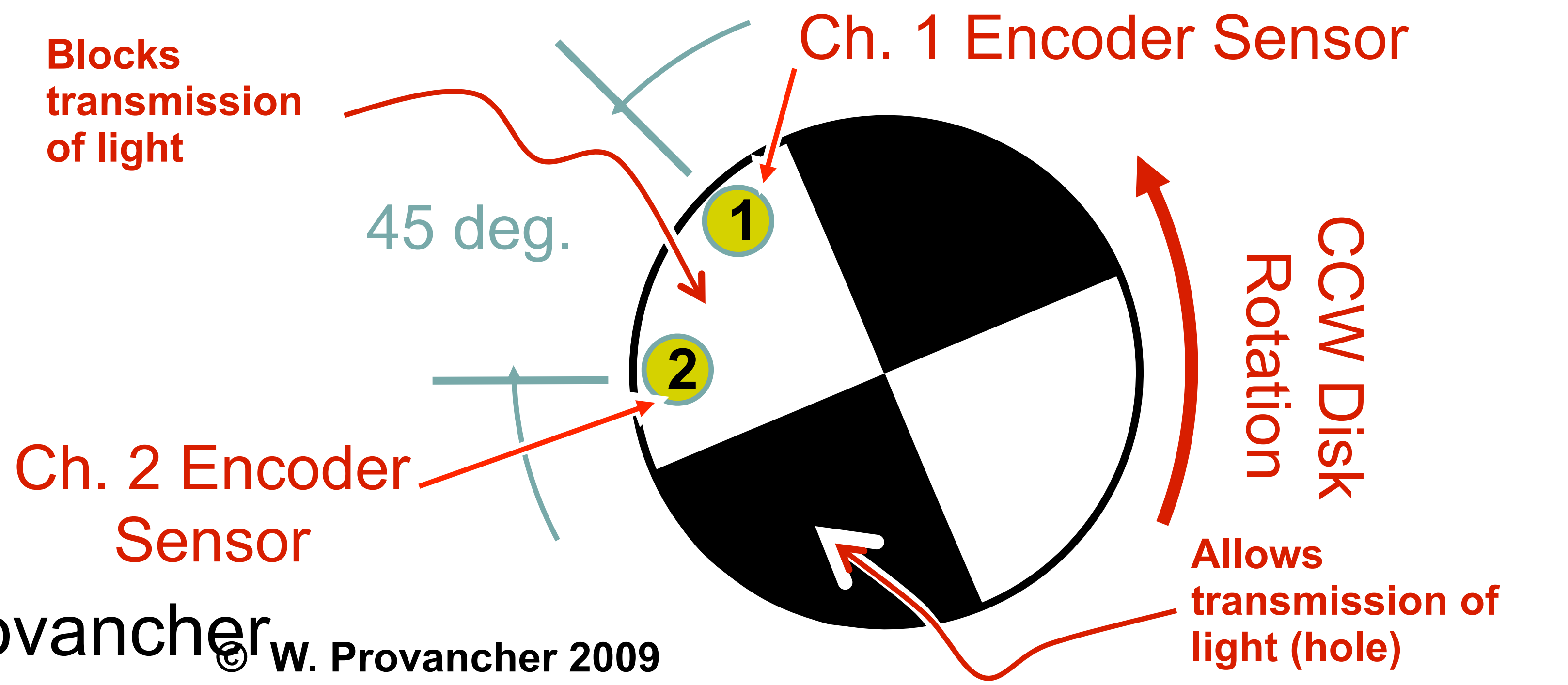
	A	B	C	D
Ch. 1	0	1	1	0
Ch. 2	0	0	1	1

A B C D

← Disk rotation CW

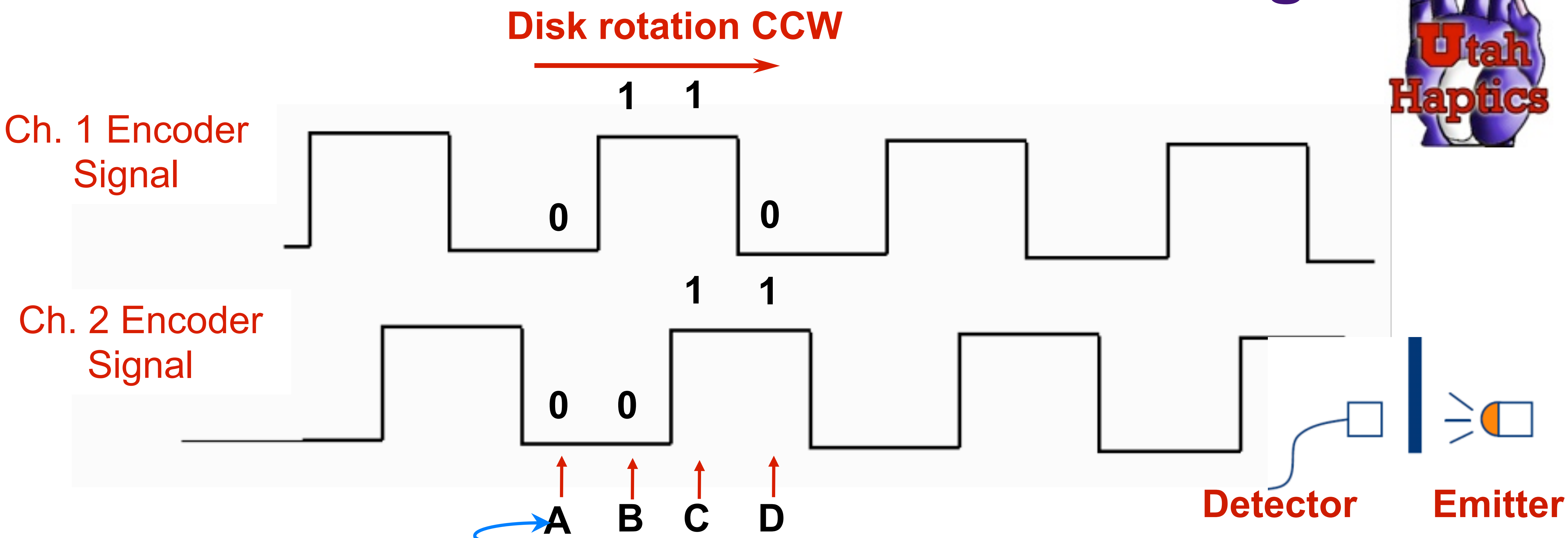
Simplified Encoder Disk

(4 CPR, 16 PPR) (shown in state A)



Adapted from slides by Will Provancher

Quadrature Encoder States & Decoding



Encoder States

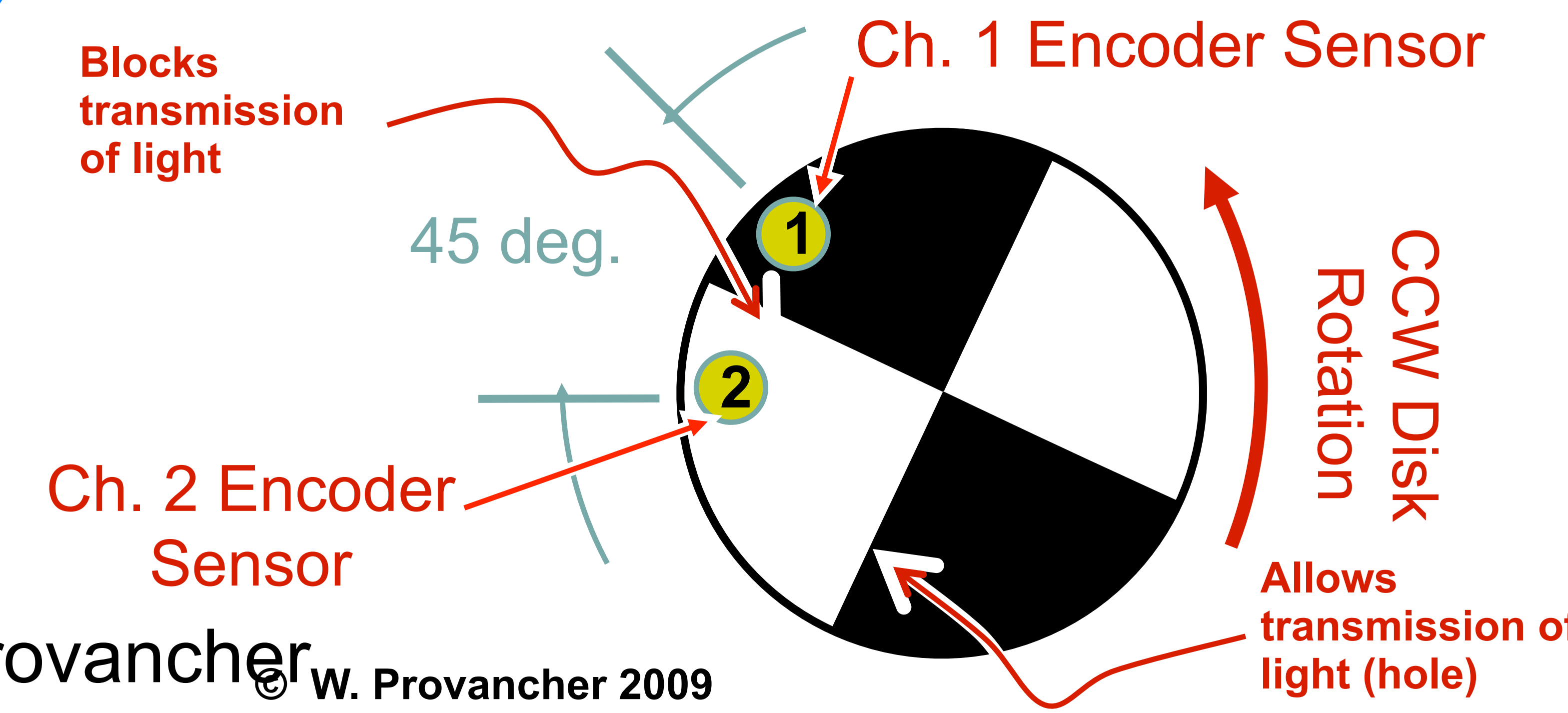
Disk rotation CCW →

	A	B	C	D
Ch. 1	0	1	1	0
Ch. 2	0	0	1	1

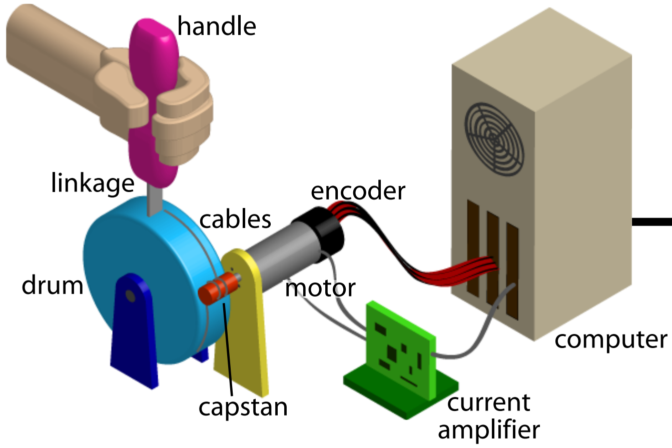
← Disk rotation CW

Simplified Encoder Disk

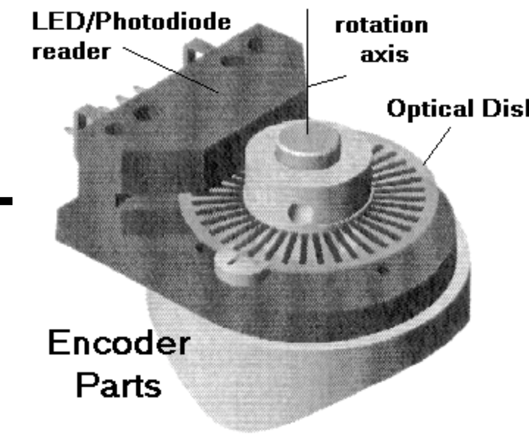
(4 CPR, 16 PPR) (shown in state A)



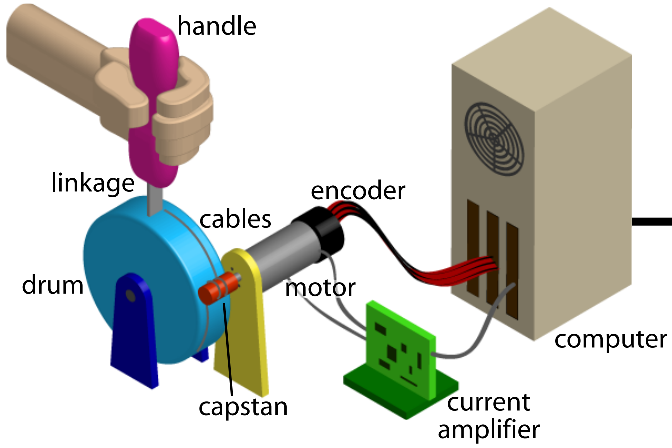
Adapted from slides by Will Provancher



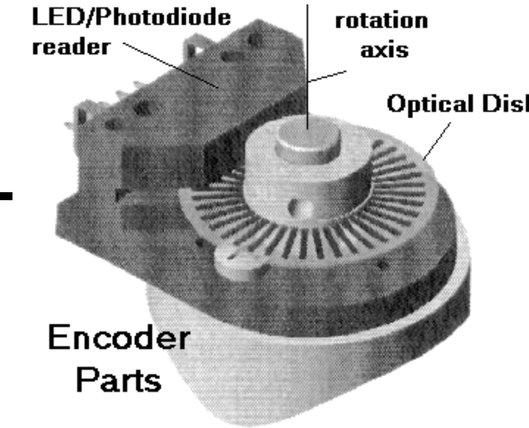
Encoder



Some material adapted from slides by A. Okamura and W. Provancher

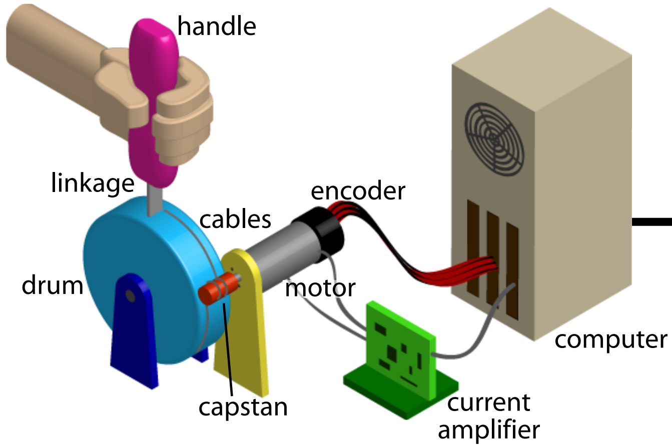


Encoder

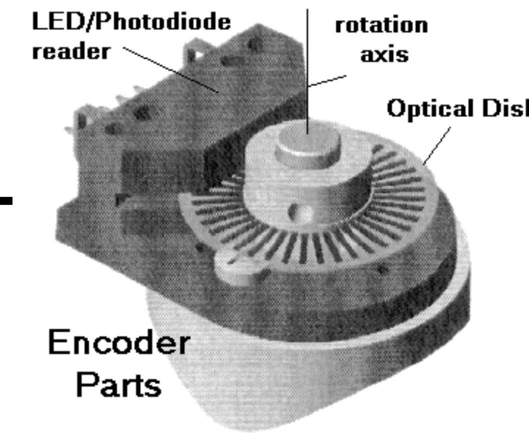


Ramifications of using incremental of optical encoders:

Some material adapted from slides by A. Okamura and W. Provancher

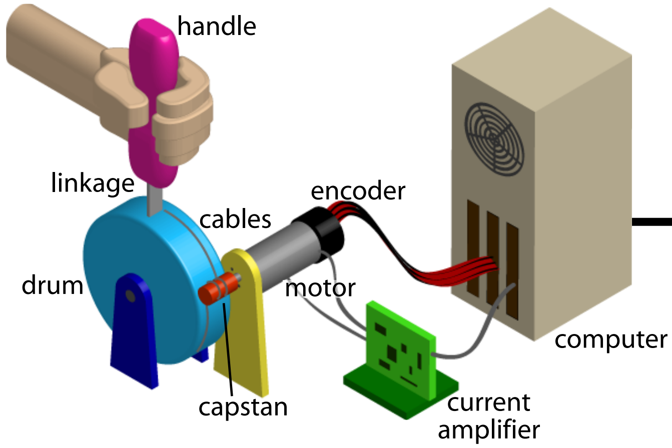


Encoder

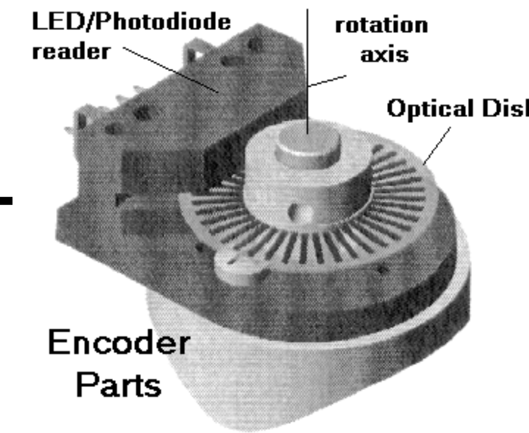


Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.

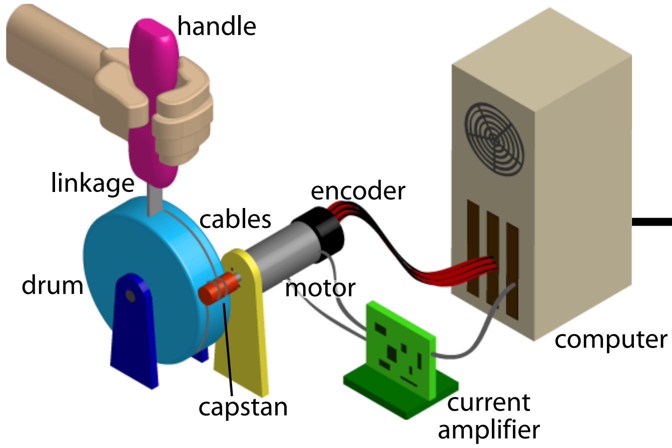


Encoder

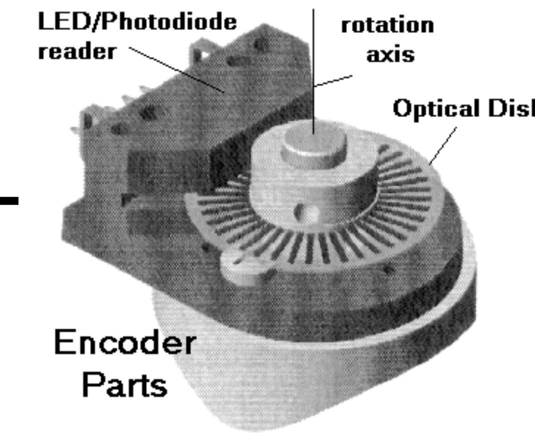


Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.
- How can you solve this?

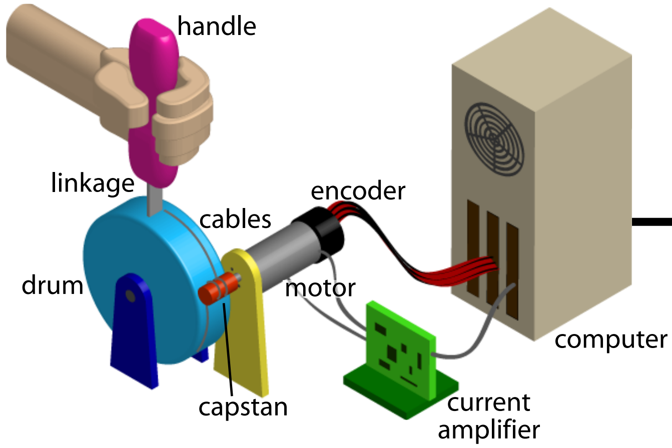


Encoder

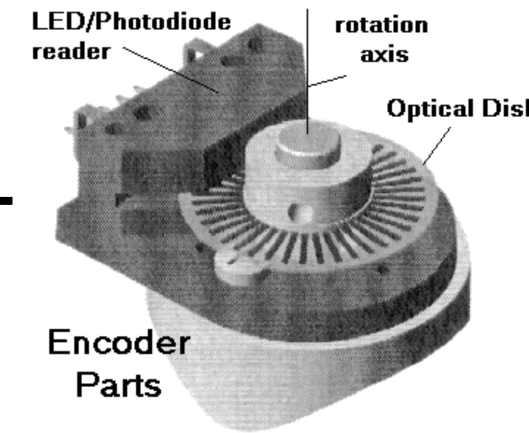


Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.
- How can you solve this?
  - Calibration pose (SensAble)

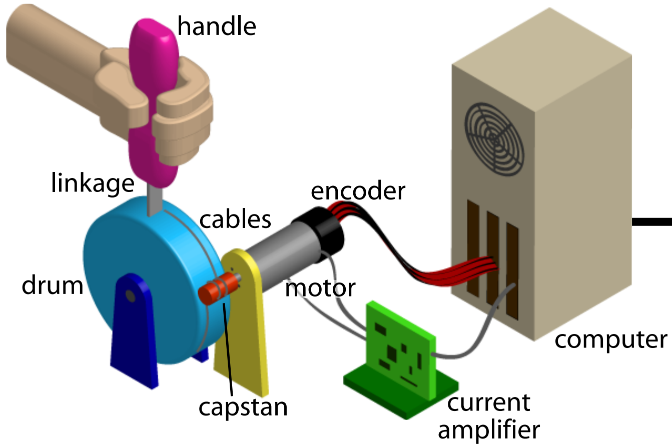


Encoder

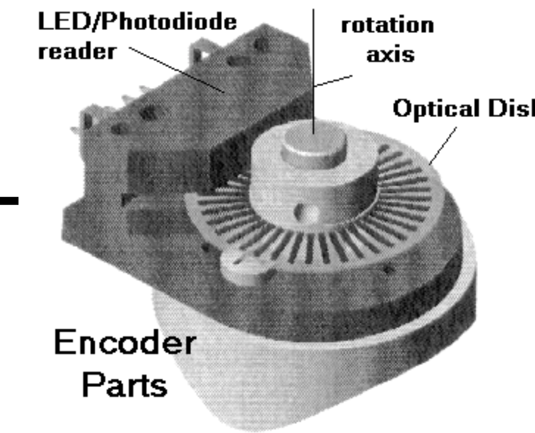


Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.
- How can you solve this?
  - Calibration pose (SensAble)
  - Secondary sensors with absolute readings (da Vinci)



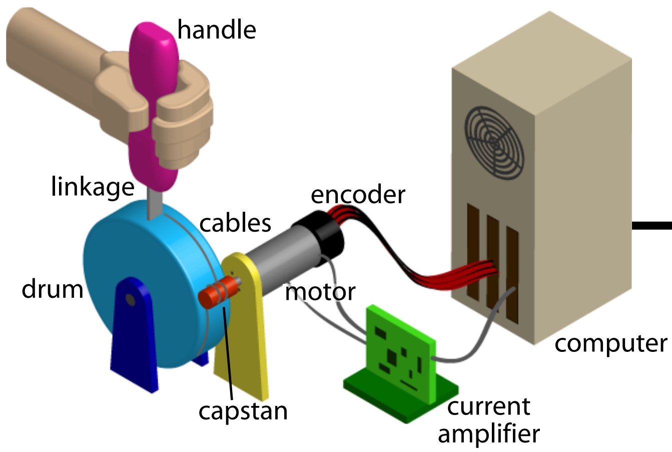
Encoder

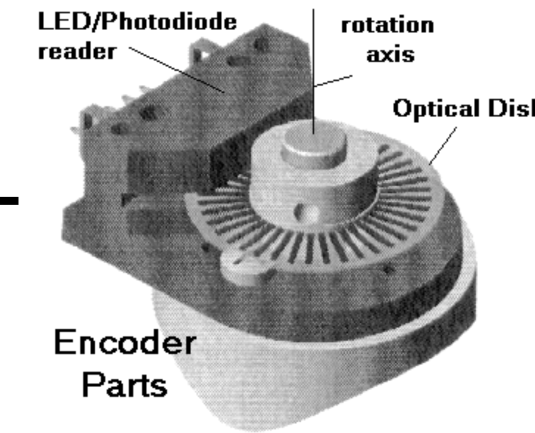


Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.
- How can you solve this?
  - Calibration pose (SensAble)
  - Secondary sensors with absolute readings (da Vinci)
- Sometimes problems occur at high velocities.

Encoder

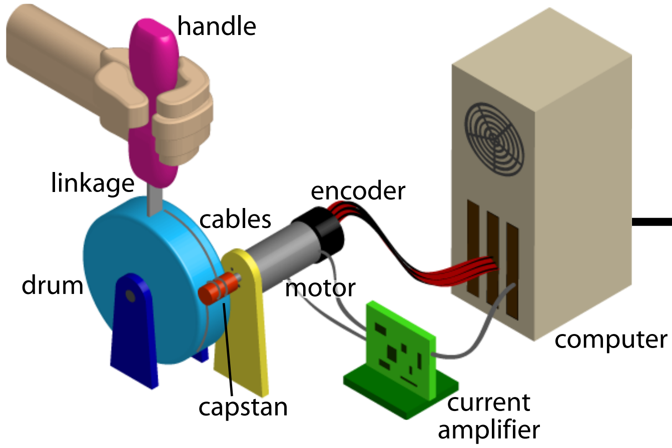




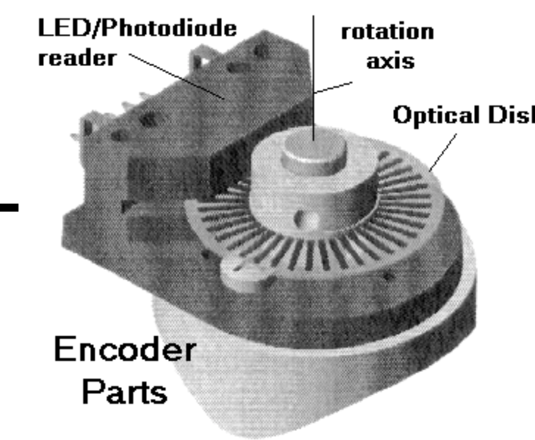
Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.
- How can you solve this?
  - Calibration pose (SensAble)
  - Secondary sensors with absolute readings (da Vinci)
- Sometimes problems occur at high velocities.
- No noise on position, but uncertainty due to resolution, and significant noise on velocity.

Some material adapted from slides by A. Okamura and W. Provancher



Encoder



Ramifications of using incremental of optical encoders:

- The system has no knowledge of absolute position, because it's always just counting pulses.
- How can you solve this?
  - Calibration pose (SensAble)
  - Secondary sensors with absolute readings (da Vinci)

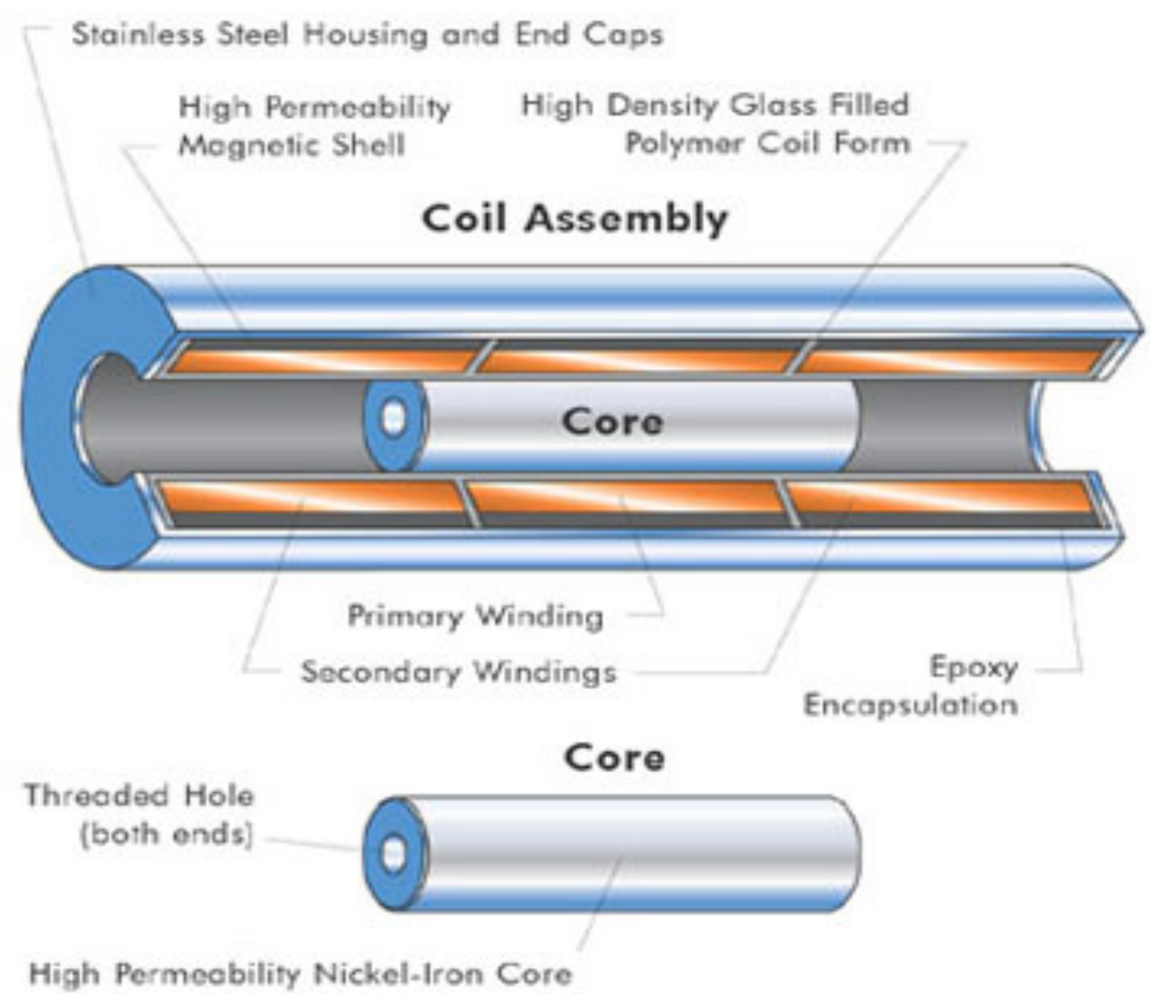
$$\theta_m = \Delta(Q - Q_{zero})$$

- Sometimes problems occur at high velocities.
- No noise on position, but uncertainty due to resolution, and significant noise on velocity.

Some material adapted from slides by A. Okamura and W. Provancher

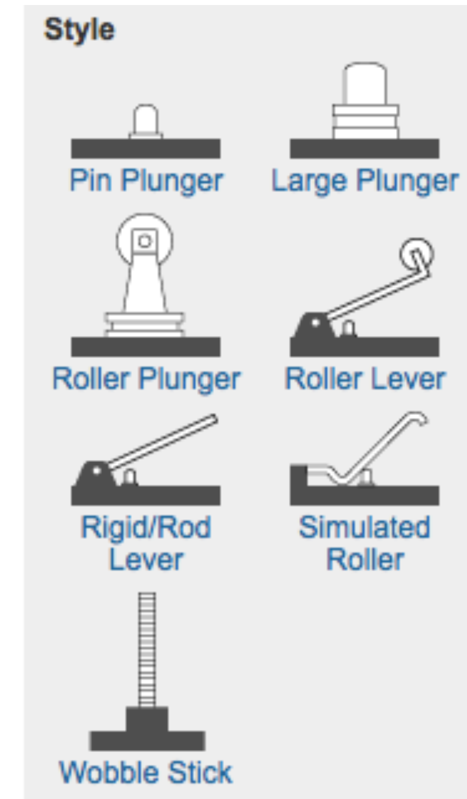
LVDT

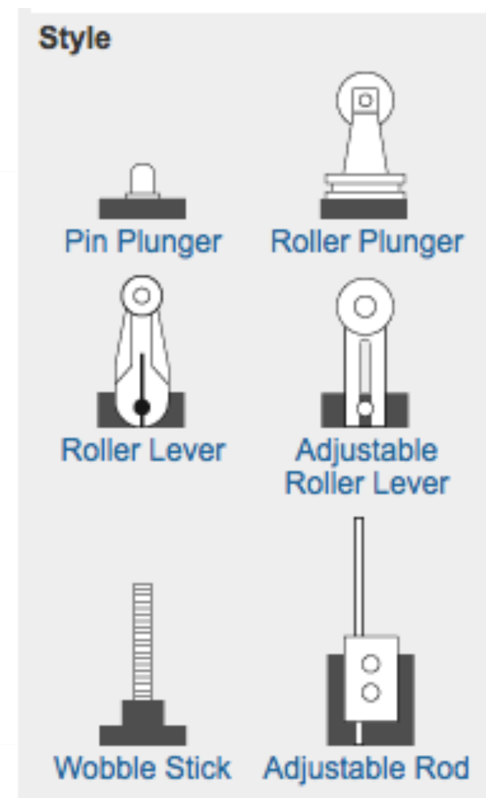
- Linear variable displacement transducer
- Very accurate
- More complex to support than potentiometers or encoders – need multiple AC voltage sources
- Inherently analog



Proximity Sensors

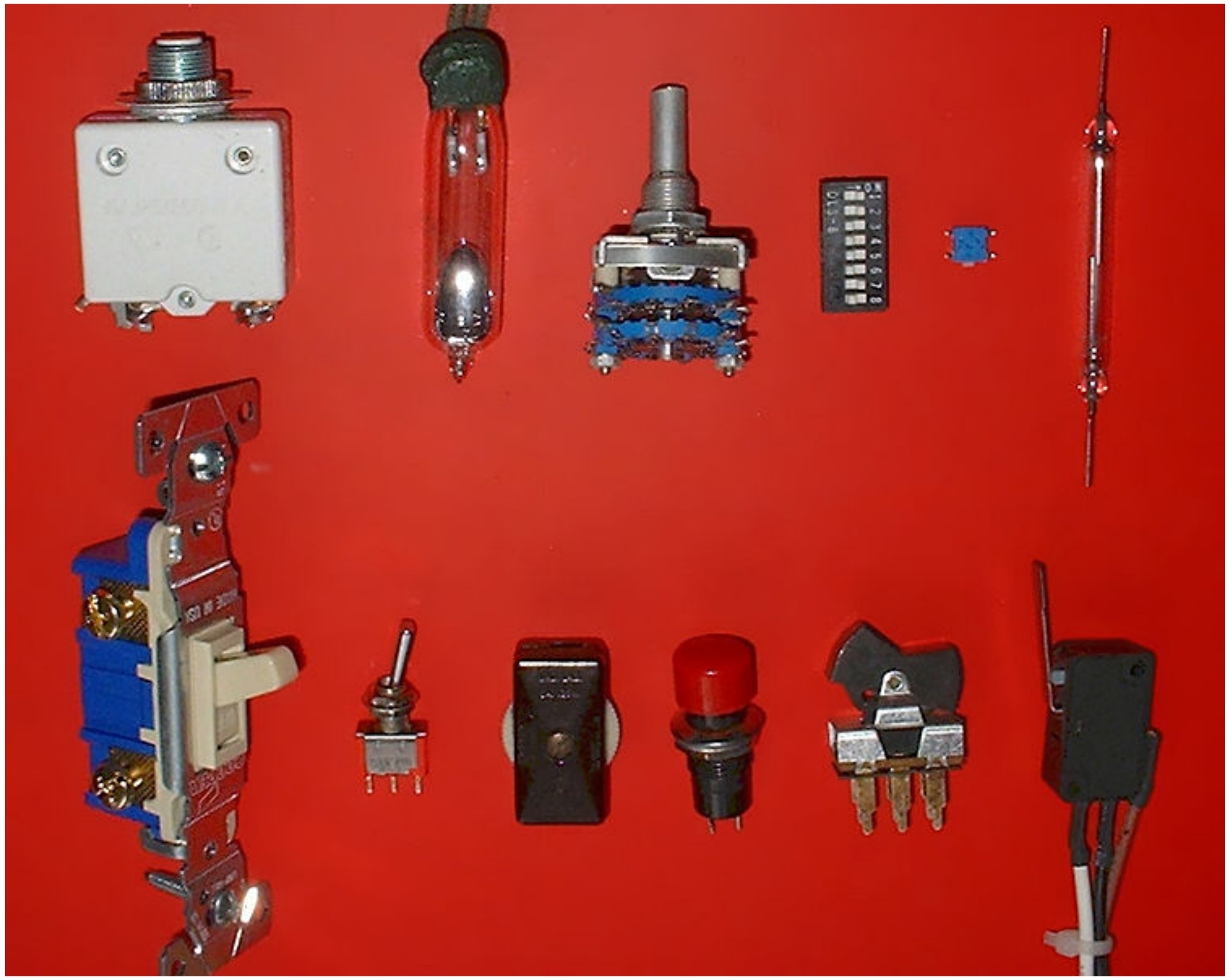
- Implies that one only wants data when something is close – don't care when it is "far away"
  - Eddy current proximity sensor – lower accuracy
  - Inductive proximity sensor – senses metallic objects
  - Limit switch – rugged sealed on/off designed for repeated contact with moving parts
  - Infrared emitter-detector pair – reflected IR from source gives indication of distance
  - Ultrasonic range sensor – reflected ultrasonic signal from source gives indication of distance





Switches

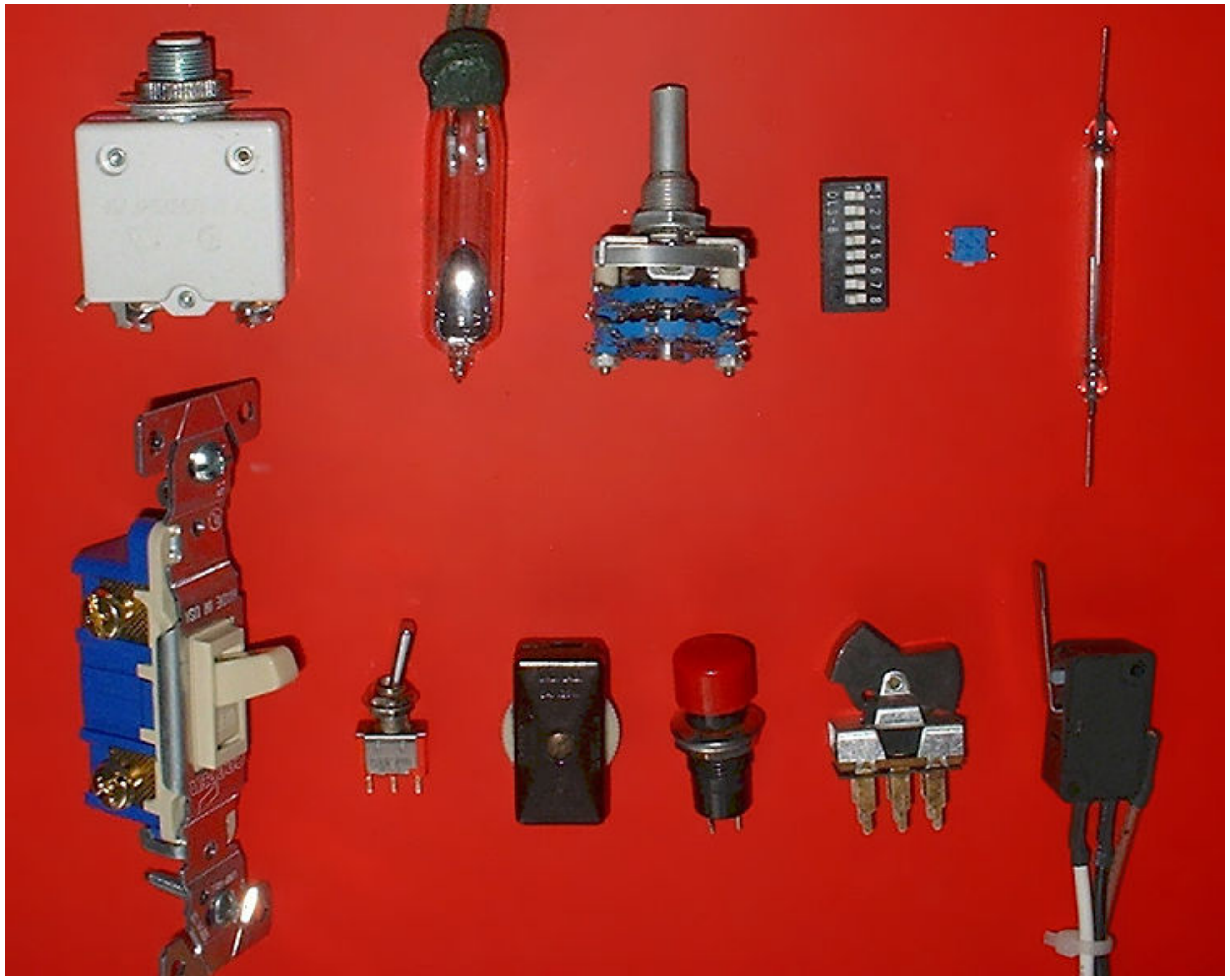
- Rotary, pushbutton, slide, toggle, tilt …
- Momentary vs. Persistent ON/OFF
- Many, many types
- Bounce



Videos from http://video_demos.colostate.edu/mechatronics/

Switches

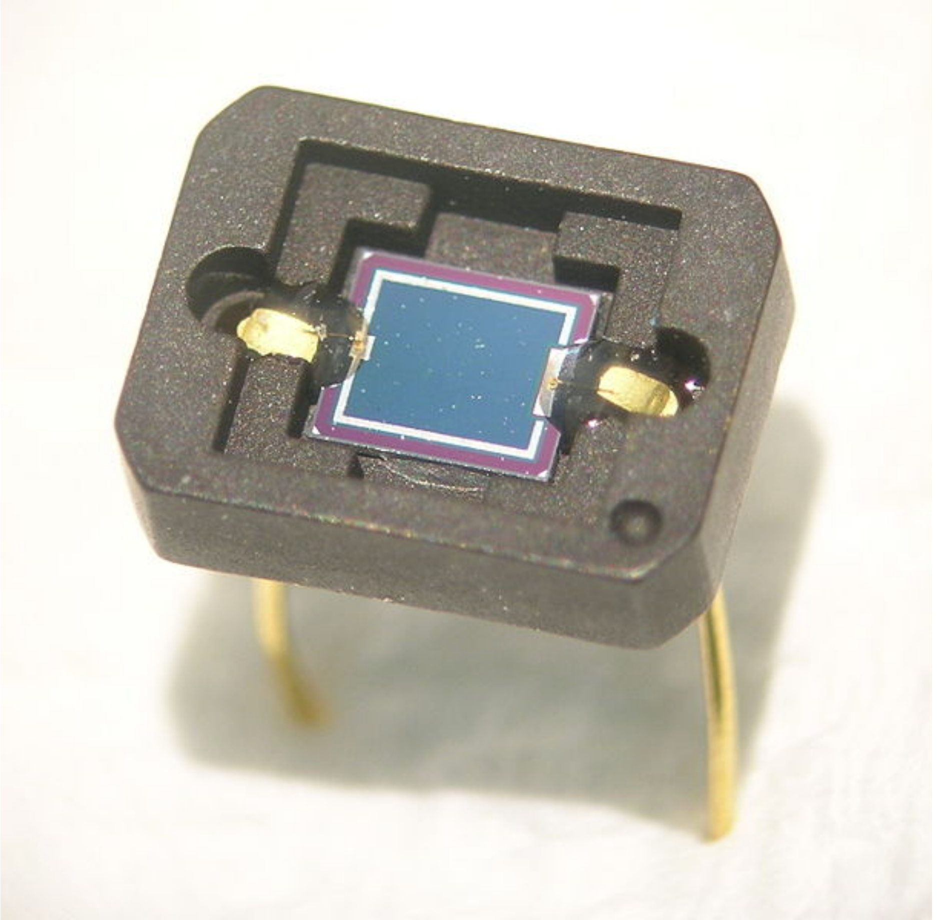
- Rotary, pushbutton, slide, toggle, tilt …
- Momentary vs. Persistent ON/OFF
- Many, many types
- Bounce

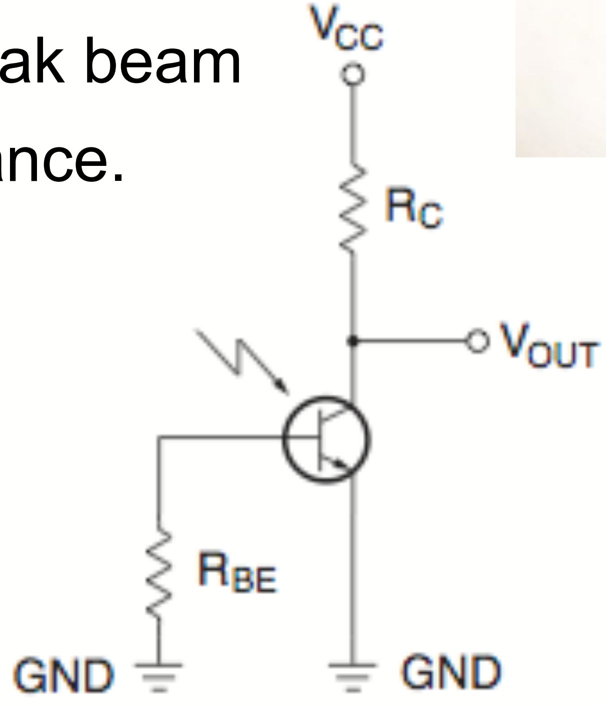


Videos from http://video_demos.colostate.edu/mechatronics/

Light

- Photodiodes, phototransistors, photoresistors
  - Small, easy to fit into electronic systems.
  - Require electrical engineering knowledge to implement properly.
  - Can set up in a variety of ways, including distance reflectance, gray-scale reflectance, distance intensity, break beam
  - Ambient light can impact performance.

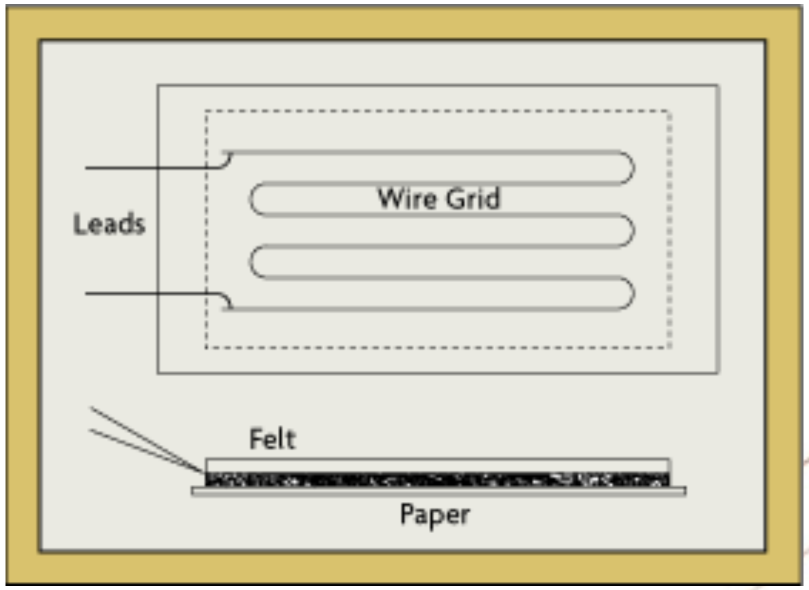


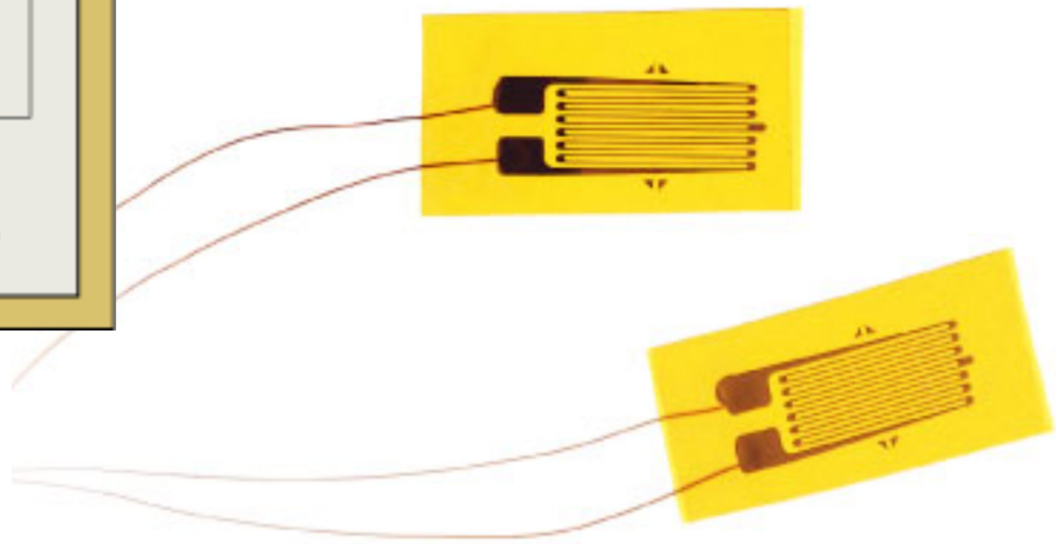


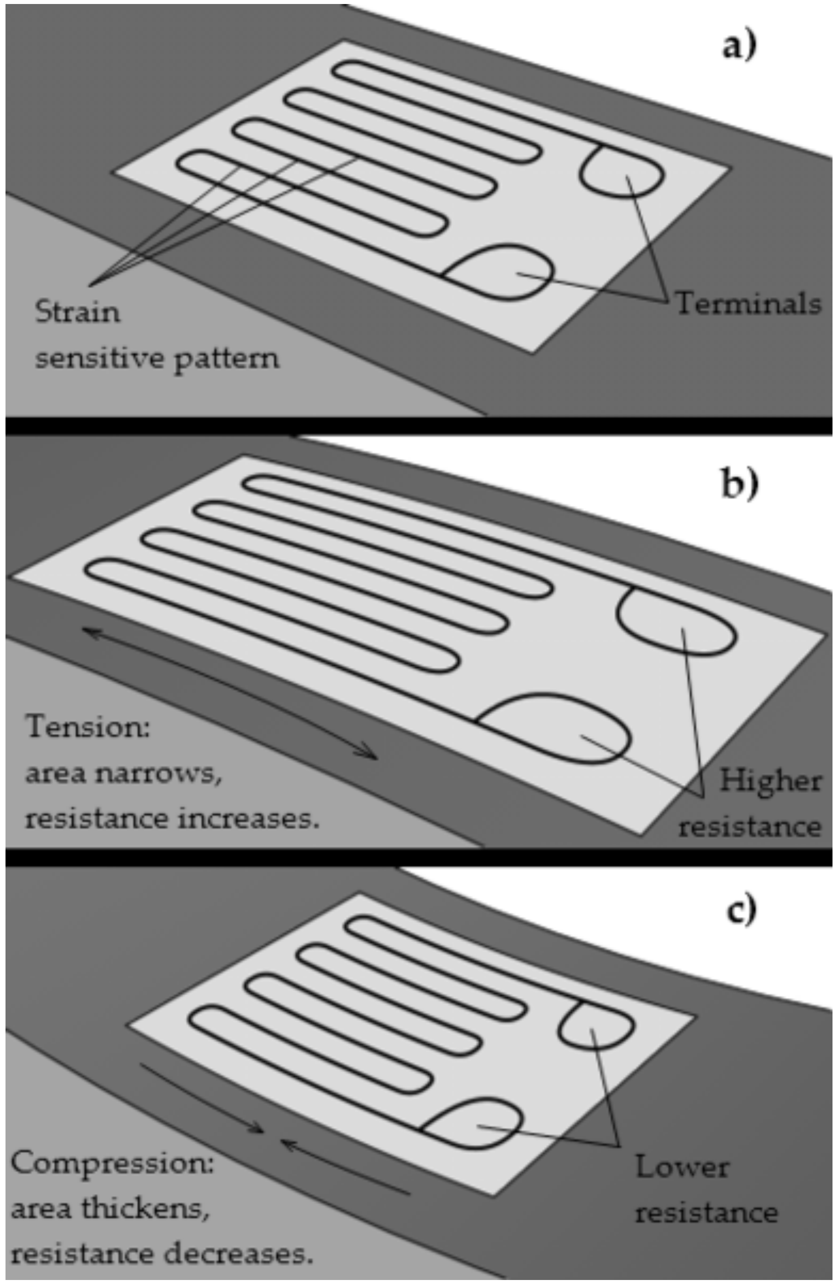
Strain Gauges

- Change in length changes resistance
- Temperature also changes resistance

$$\frac{\Delta R}{R} = G\varepsilon$$

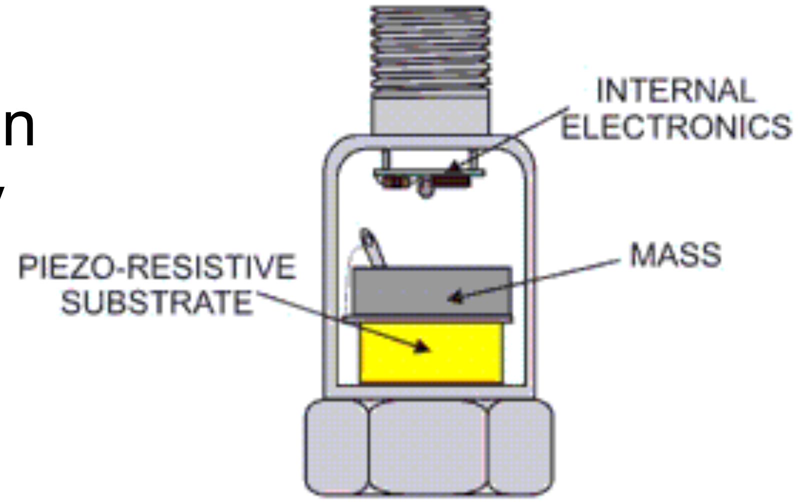






Force Sensing

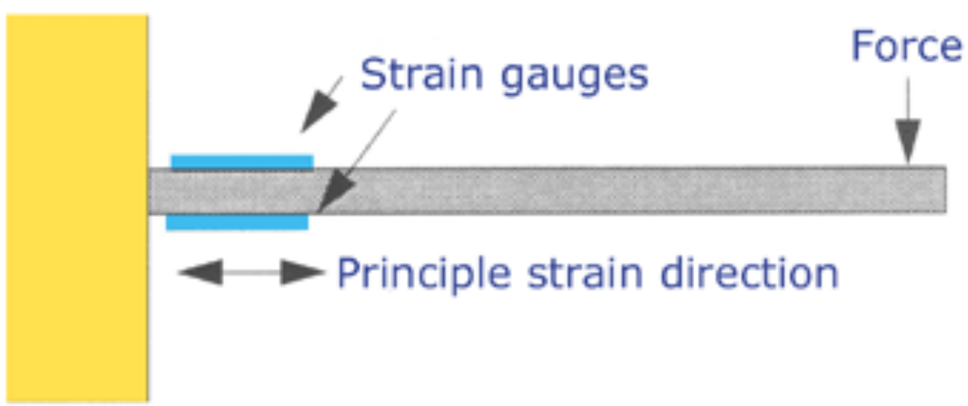
- Measure the position change on an elastic element
- Strain gauge load cell – clever layout of gauges & material shape to create accurate system
- Fluid pressure on a diaphragm – measure change in length/size of a diaphragm with strain gauges
- Piezo electric crystals – generate electrical signals when a strain is generated – typically used for high frequency force changes (accelerations)

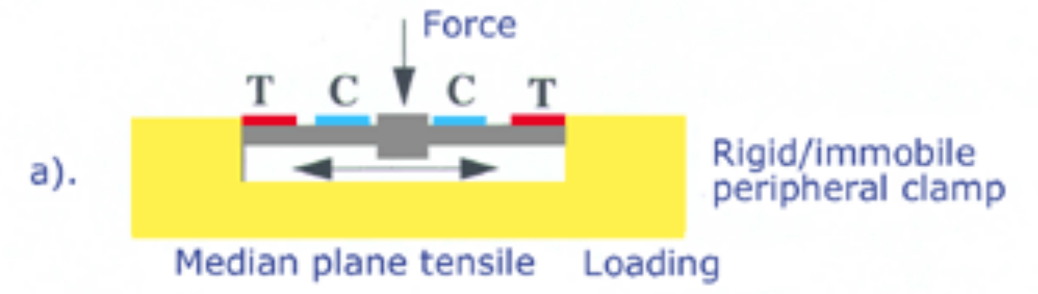


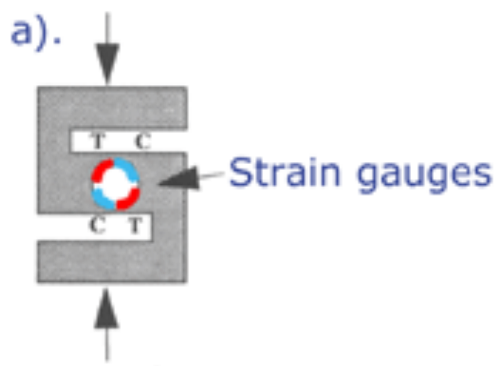
A single ended compression accelerometer

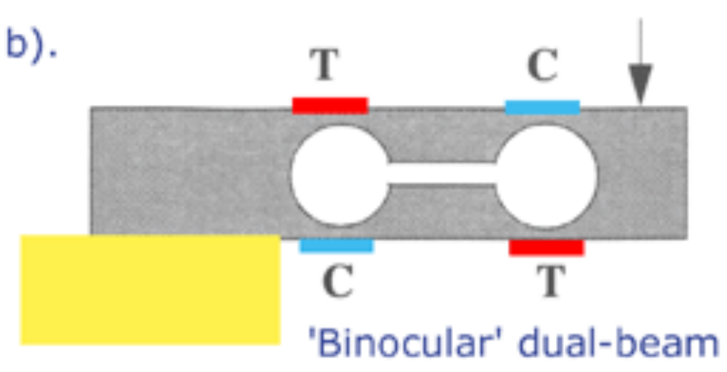
A load cell

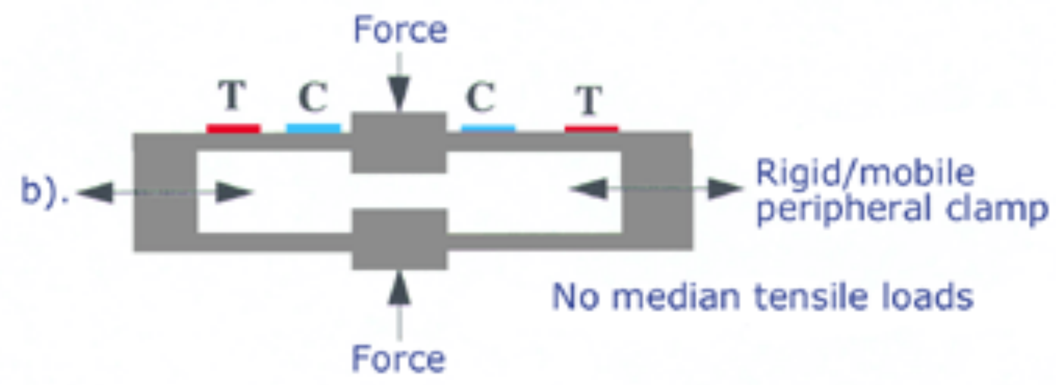
- A piece of metal that is designed to deform in a predictable way to create a measurement of load (torque and/or force)







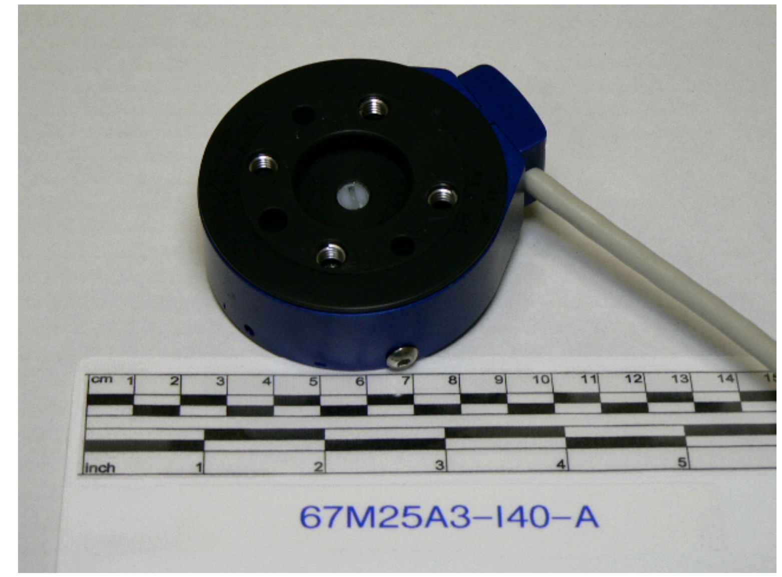




Force Sensors

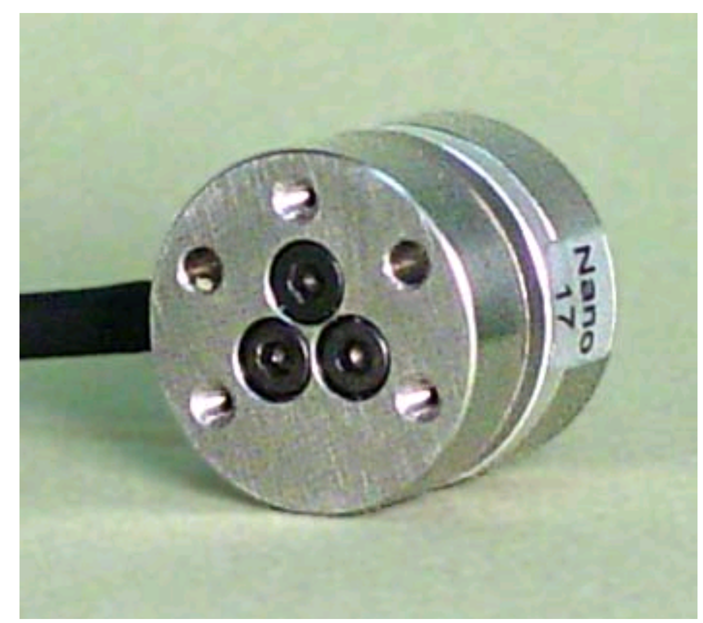
- How do they work?
  - Typically a flexure + a strain gage (sometimes also piezoelectric sensors, but these tend to drift)
- A good quality 6-axis force-torque sensor is ~$6000
  - Mechanically delicate - do not drop or hit
  - Sensitive to temperature fluctuations

6-Axis JR3 Force Torque Load Cell



JR3.com

ATI Nano17 Transducer



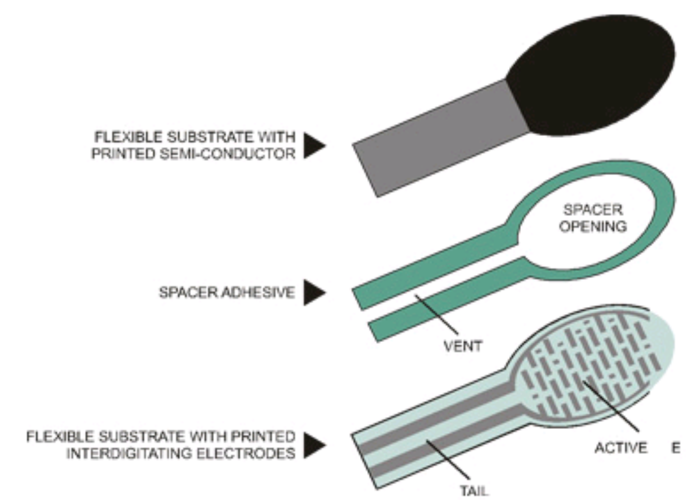
www.ati-ia.com

Adapted from slides by Will Provancher

Force Sensing Resistors

- Known as FSRs
  - Piezoresistive ink
  - Tons of sensor drift and hysteresis, sensitivity to contact location
  - Very thin!
  - Cheap ~$10
  - Use drive circuit recommended by manufacturer to get voltage output that is approximately linear with force

Interlink FSR



www.interlinkelectronics.com

Tekscan Flexiforce FSR

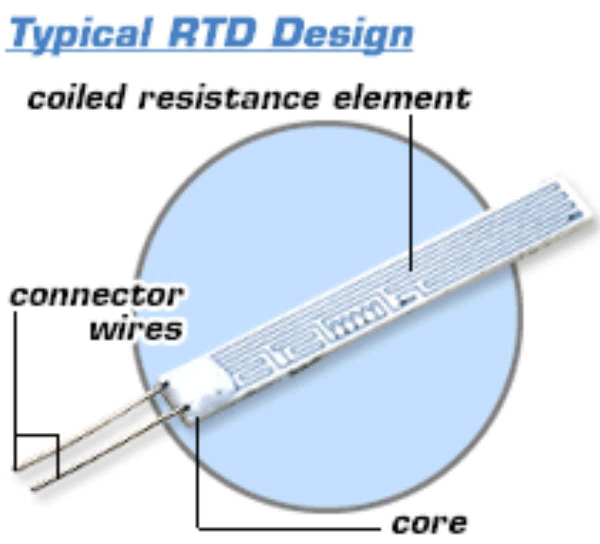
www.tekscan.com/flexiforce

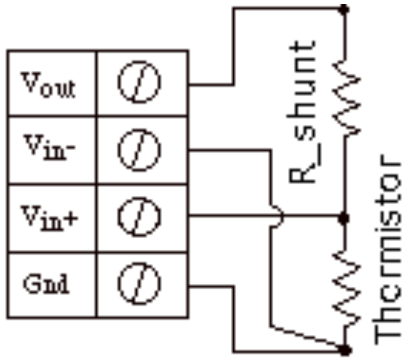
Adapted from slides by Will Provancher

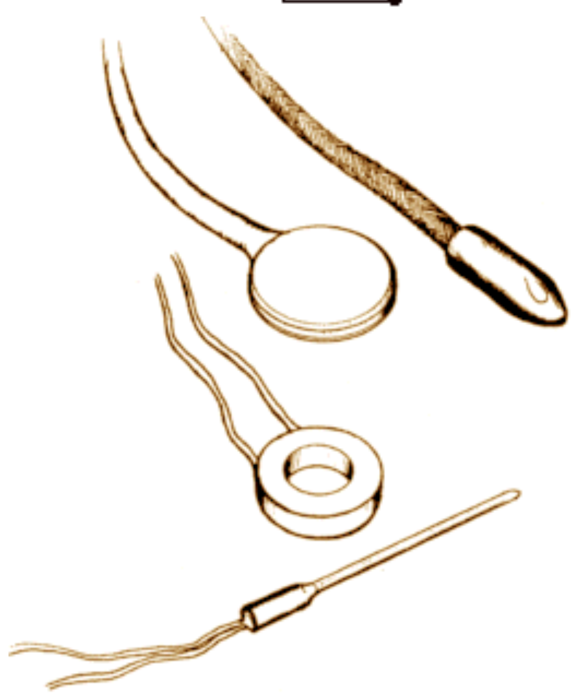
Temperature

- Thermistor
- RTD
- Thermocouples

- Equations vary but they are all nonlinear and require some “figuring” to get the right answer



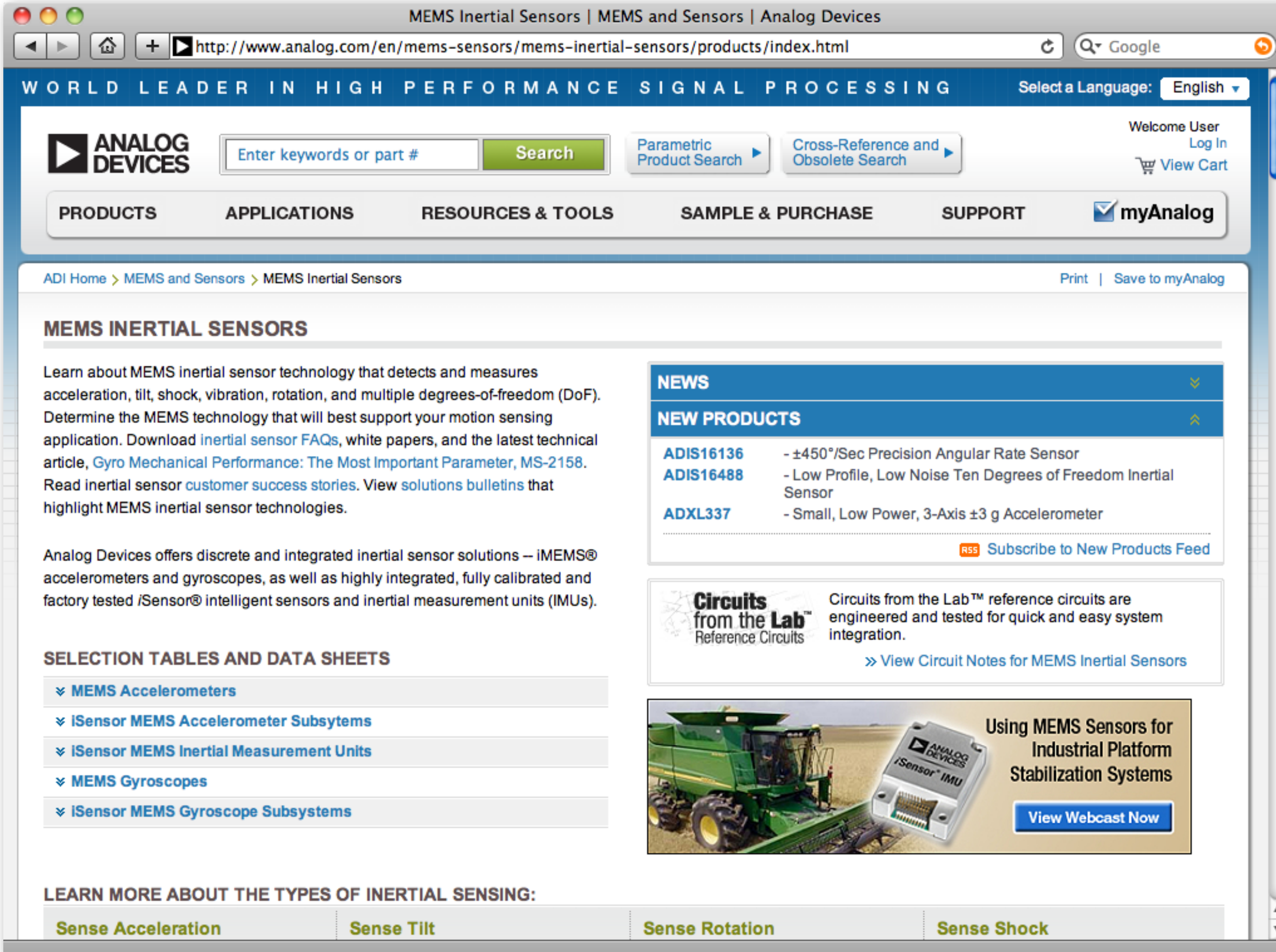




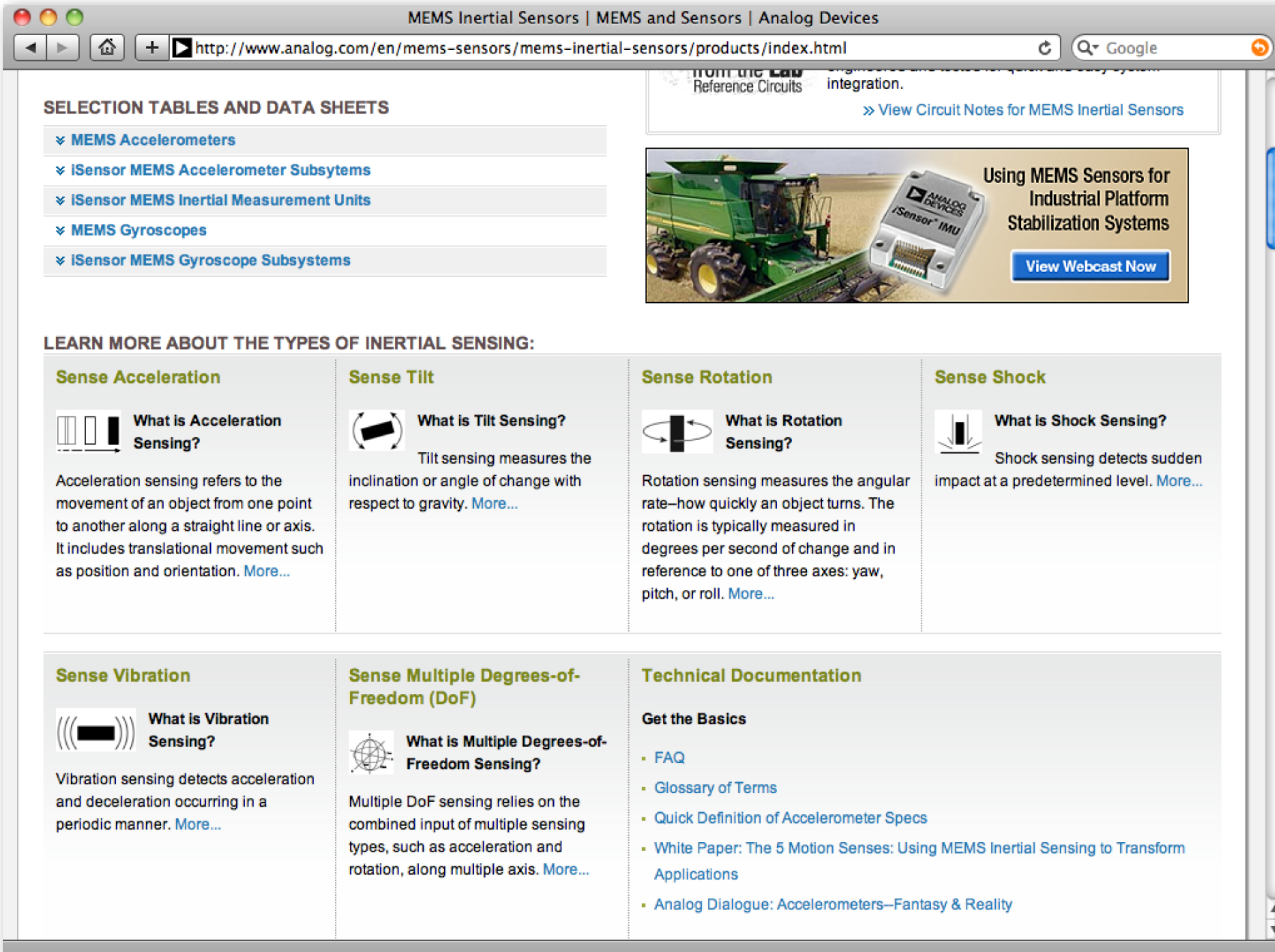
Adapted from slides by John Morrell

Inertial Sensing

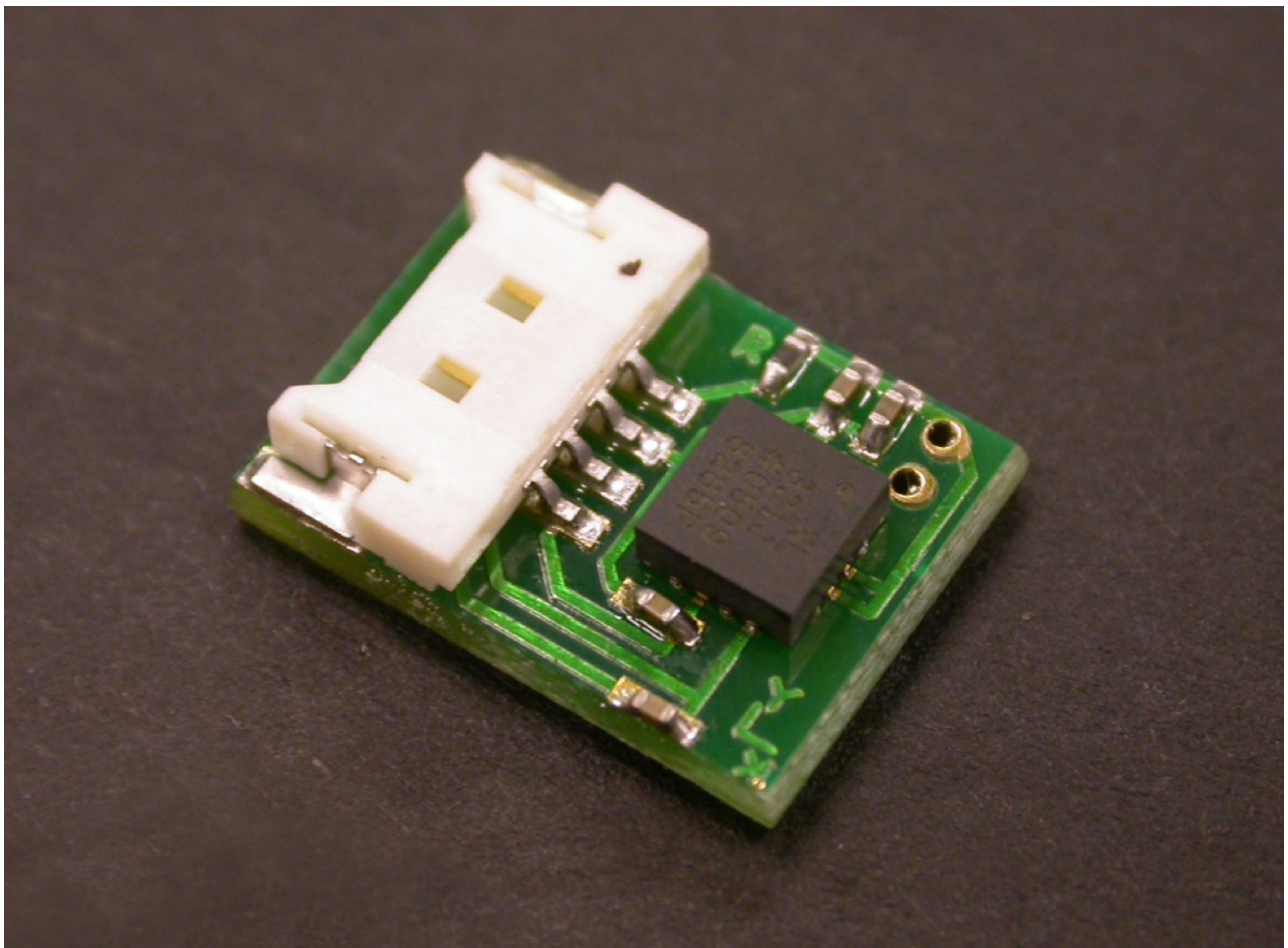
Inertial Sensing



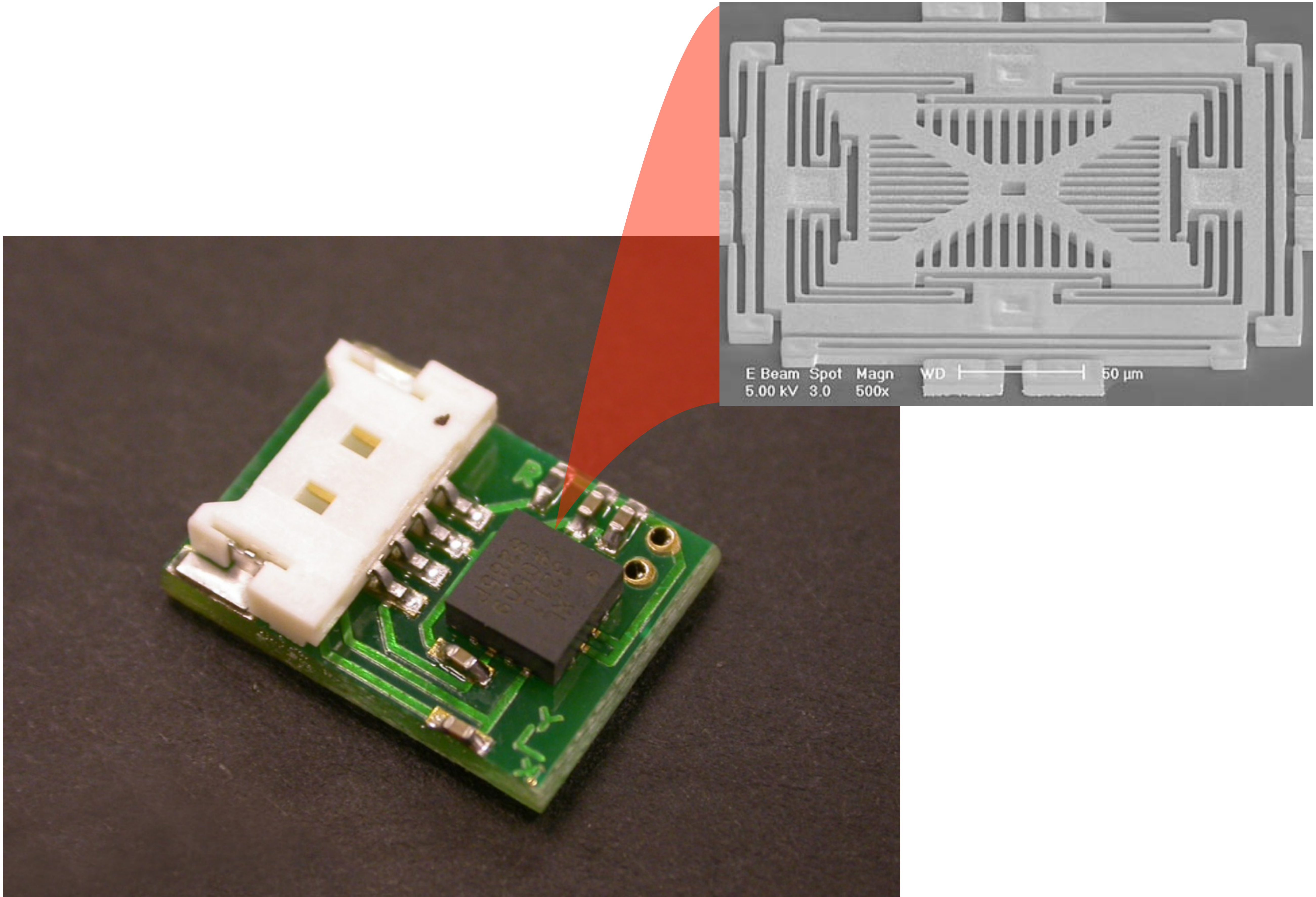
Inertial Sensing

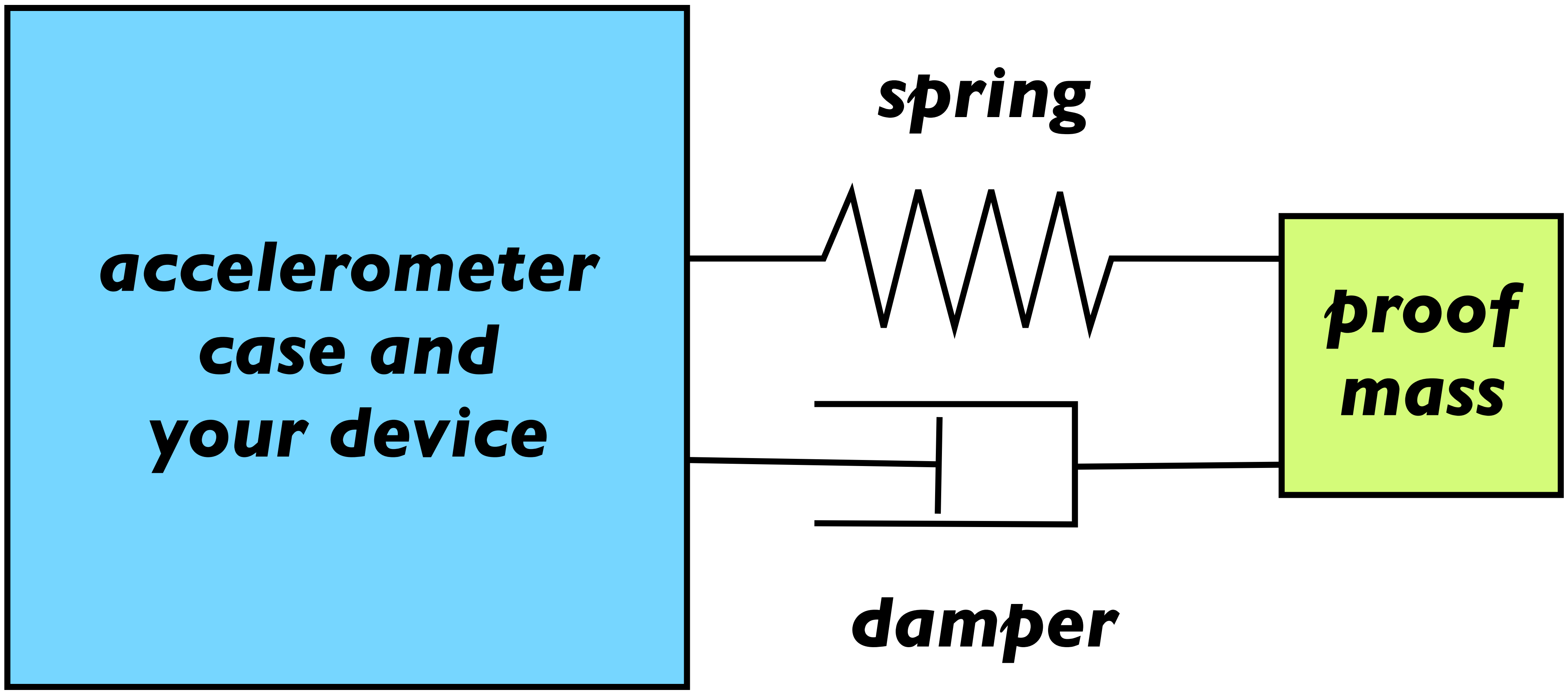


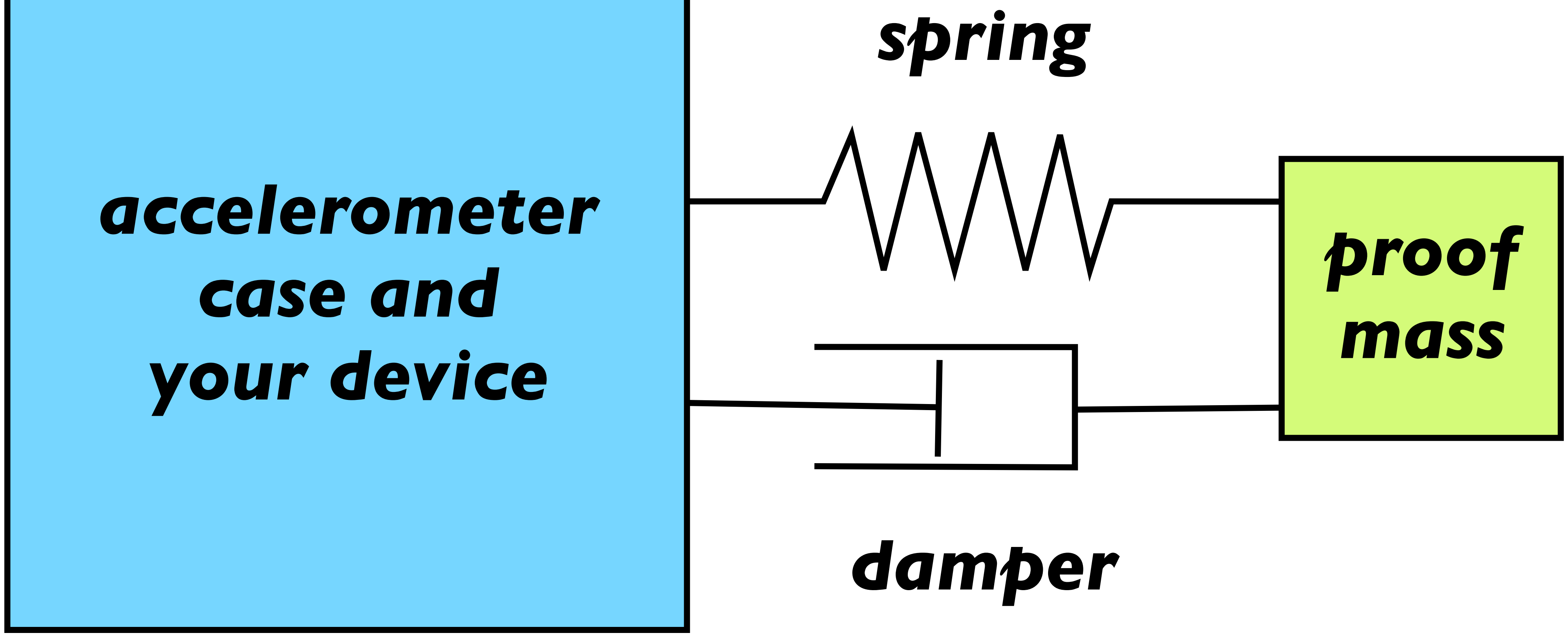
MEMS-based Accelerometers



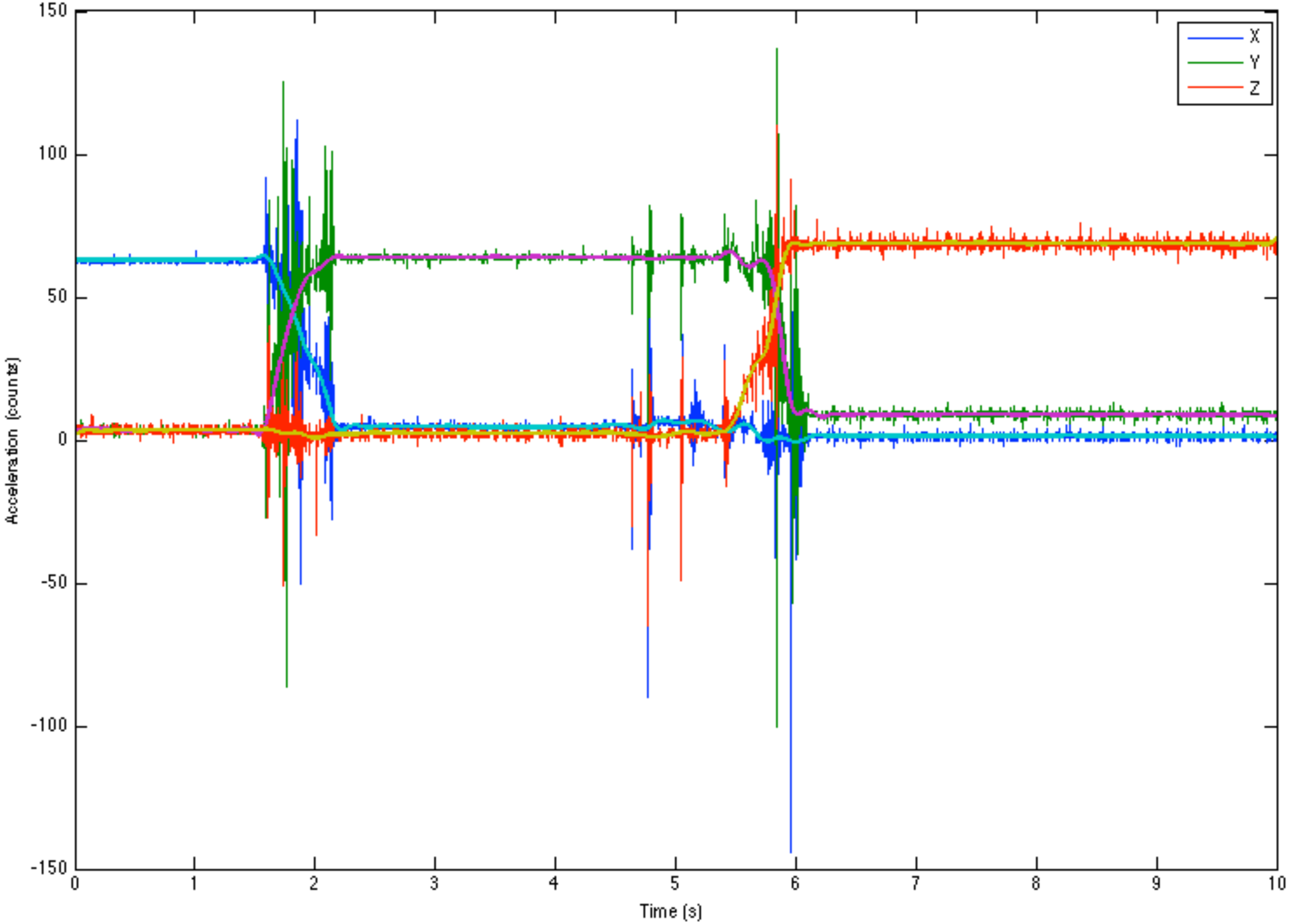
MEMS-based Accelerometers




accelerometer case and your device
spring
damper
proof mass

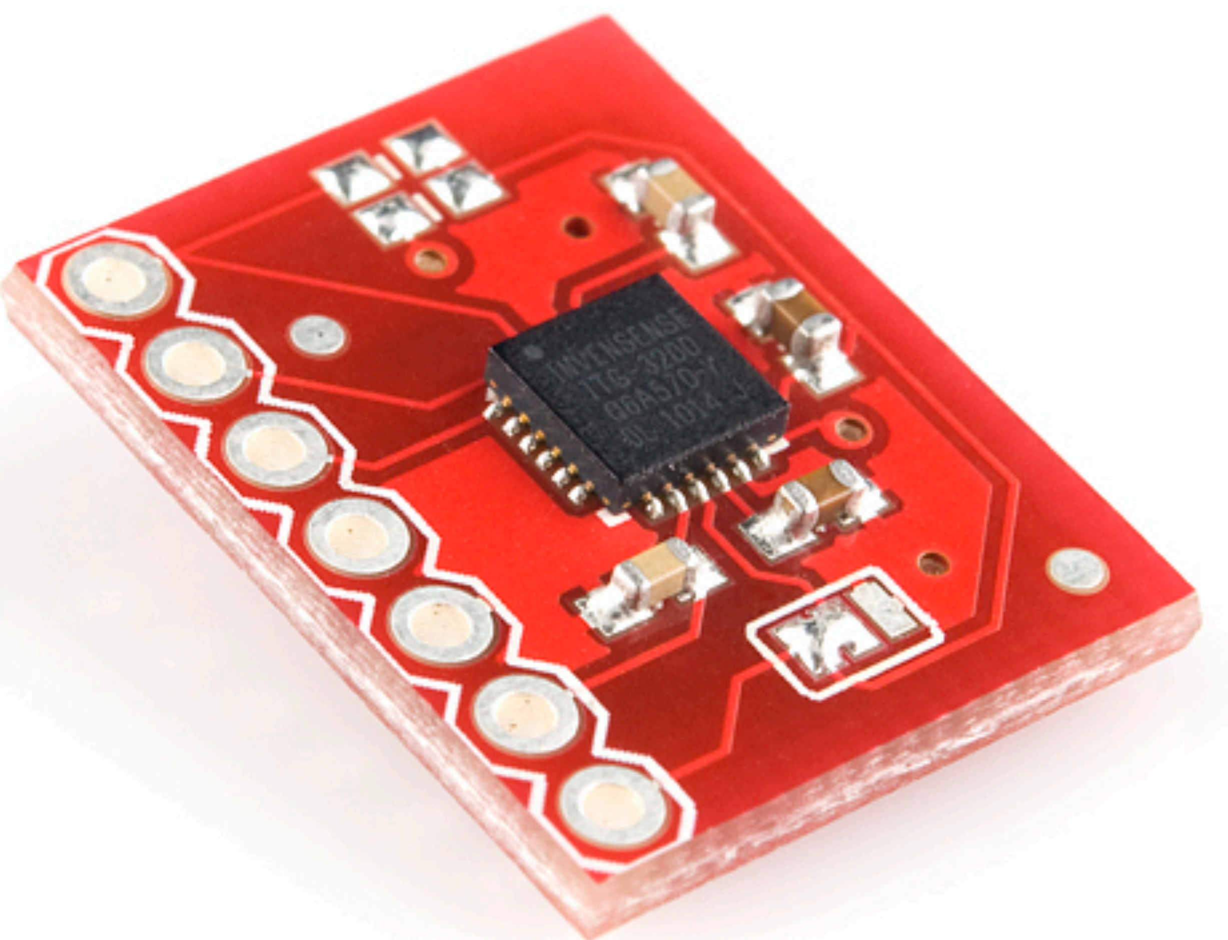


Measures acceleration and gravity

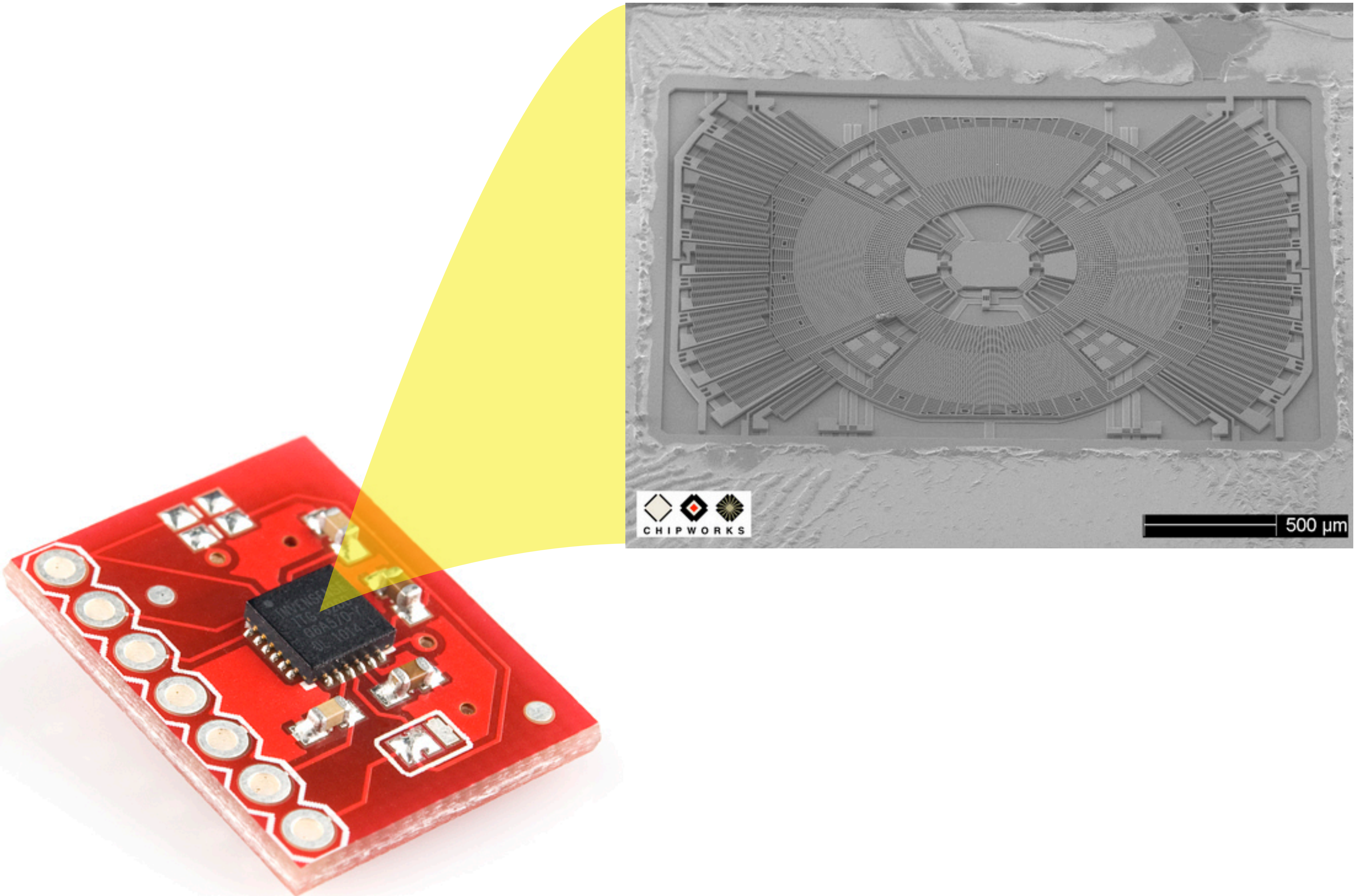


X
Y
Z
150
100
50
0
-50
-100
-150
Acceleration (counts)
0
1
2
3
4
5
6
7
8
9
10
Time (s)

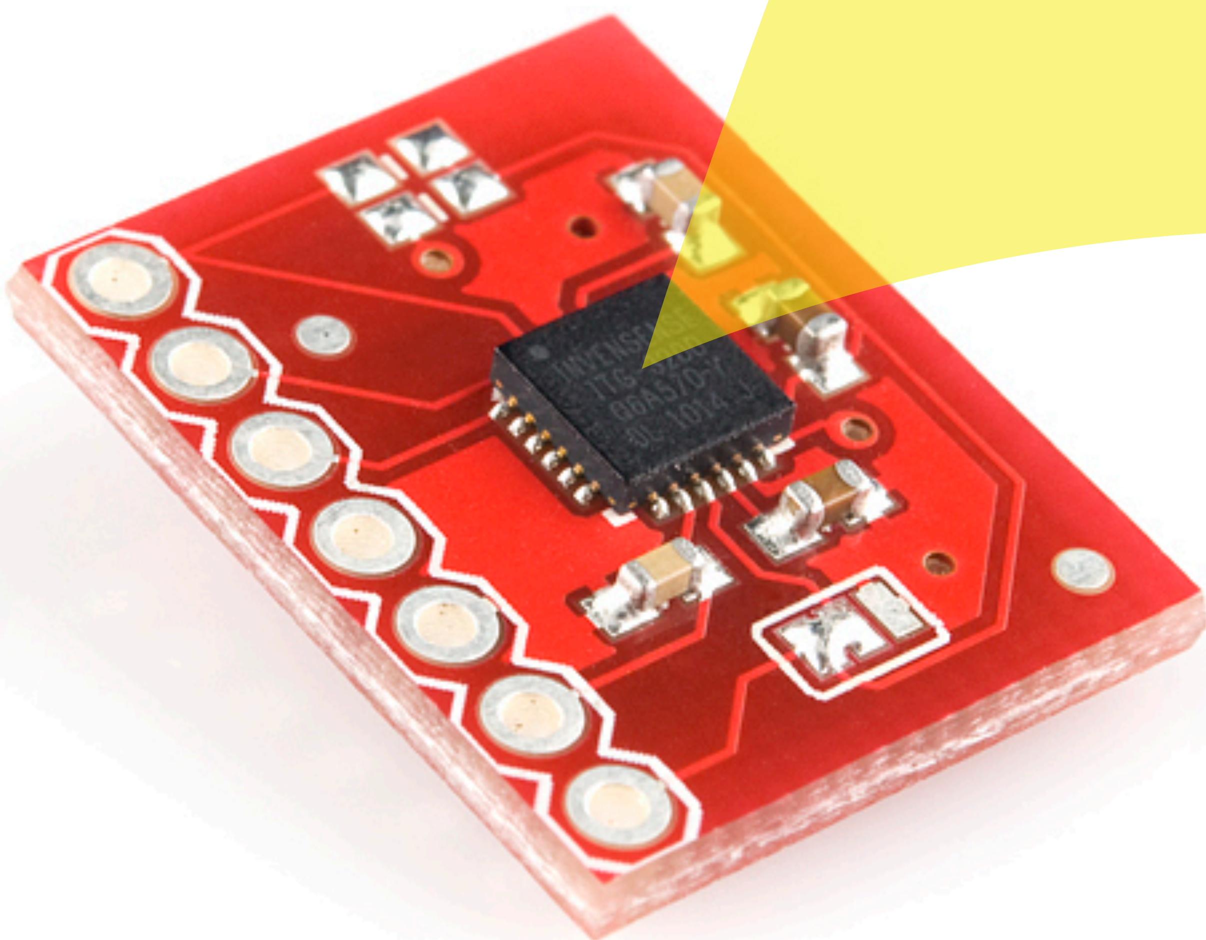
MEMS-based Rate Gyroscopes

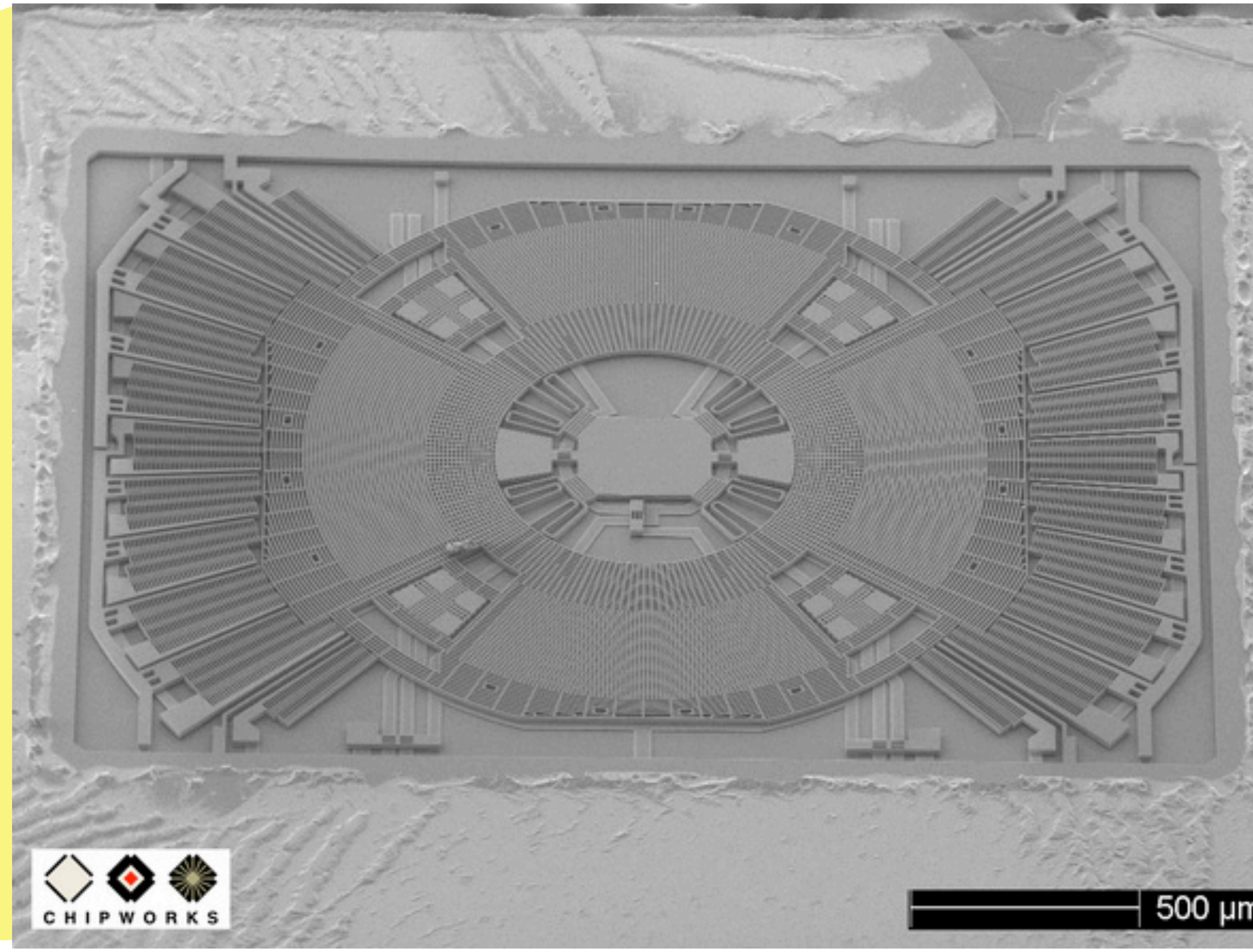


MEMS-based Rate Gyroscopes



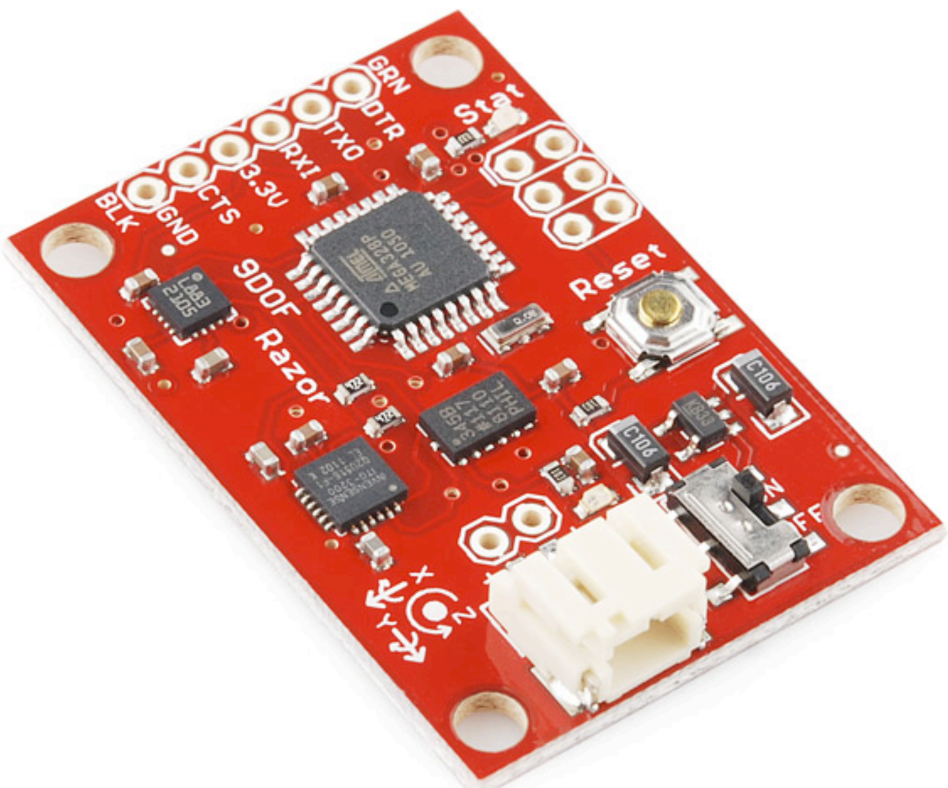
MEMS-based Rate Gyroscopes





Measures angular velocity

Inertial Measurement Units (IMUs)



9 Degrees of Freedom on a single, flat board for $125:

ITG-3200 - triple-axis digital-output gyroscope

ADXL345 - 13-bit resolution, ±16g, triple-axis accelerometer

HMC5883L - triple-axis, digital magnetometer

Outputs of all sensors processed by on-board ATmega328 and sent out via a serial stream

Inertial Measurement Units (IMUs)

Same sensors in a clean package with all processing and software done for you, estimates absolute heading, around $2000?

Magnetic Tracking

Field generator creates magnetic field; small wired sensors used to estimate position and orientation of tracked item; bad interference from metal and electromagnetic actuators; price ranges from ~$300 (Razer Hydra) to ~$16,000 (precise, medical use).

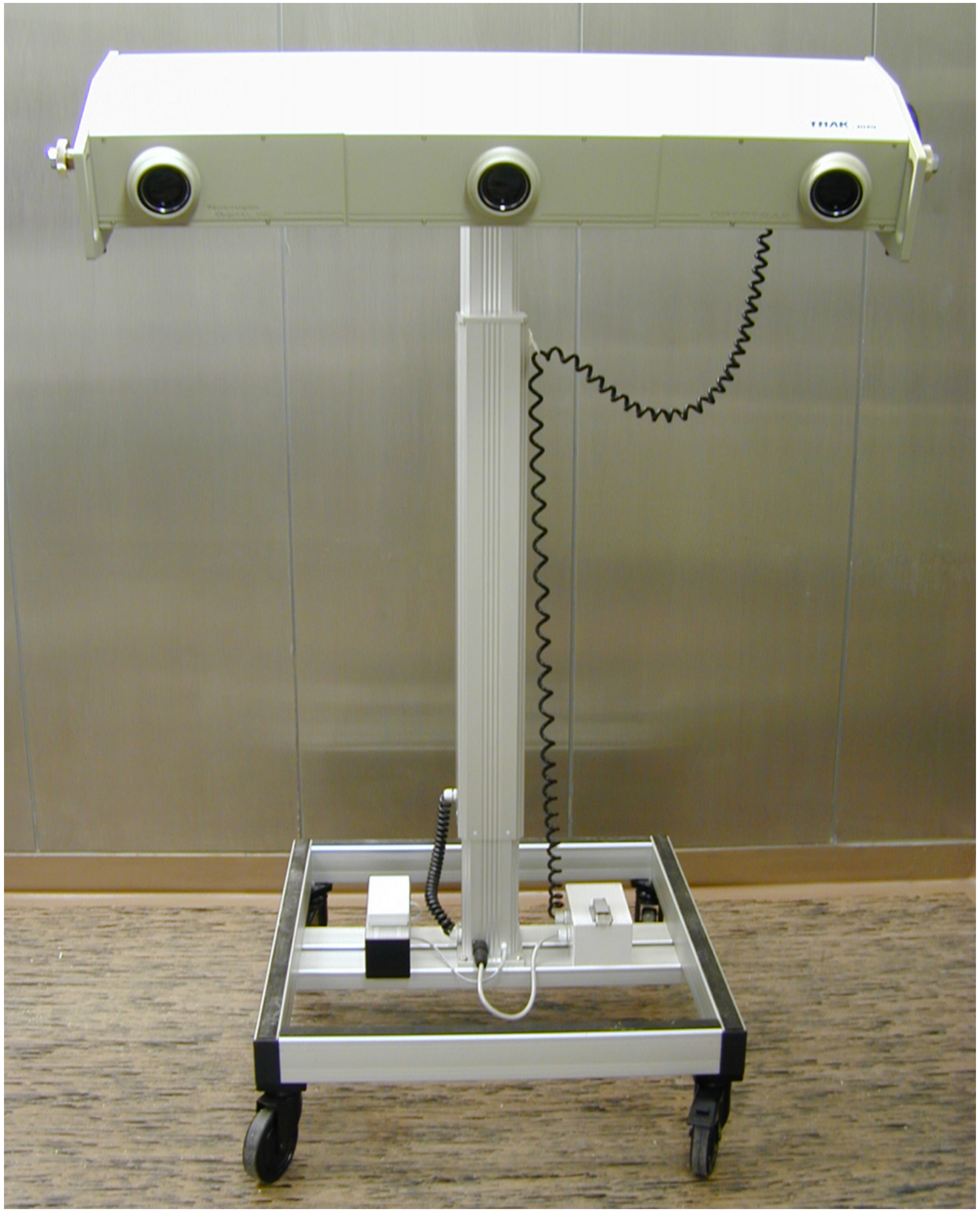
Optical Tracking

Custom camera system with blob detection in 2D; limited by camera frame rate, processing time

Optical Tracking

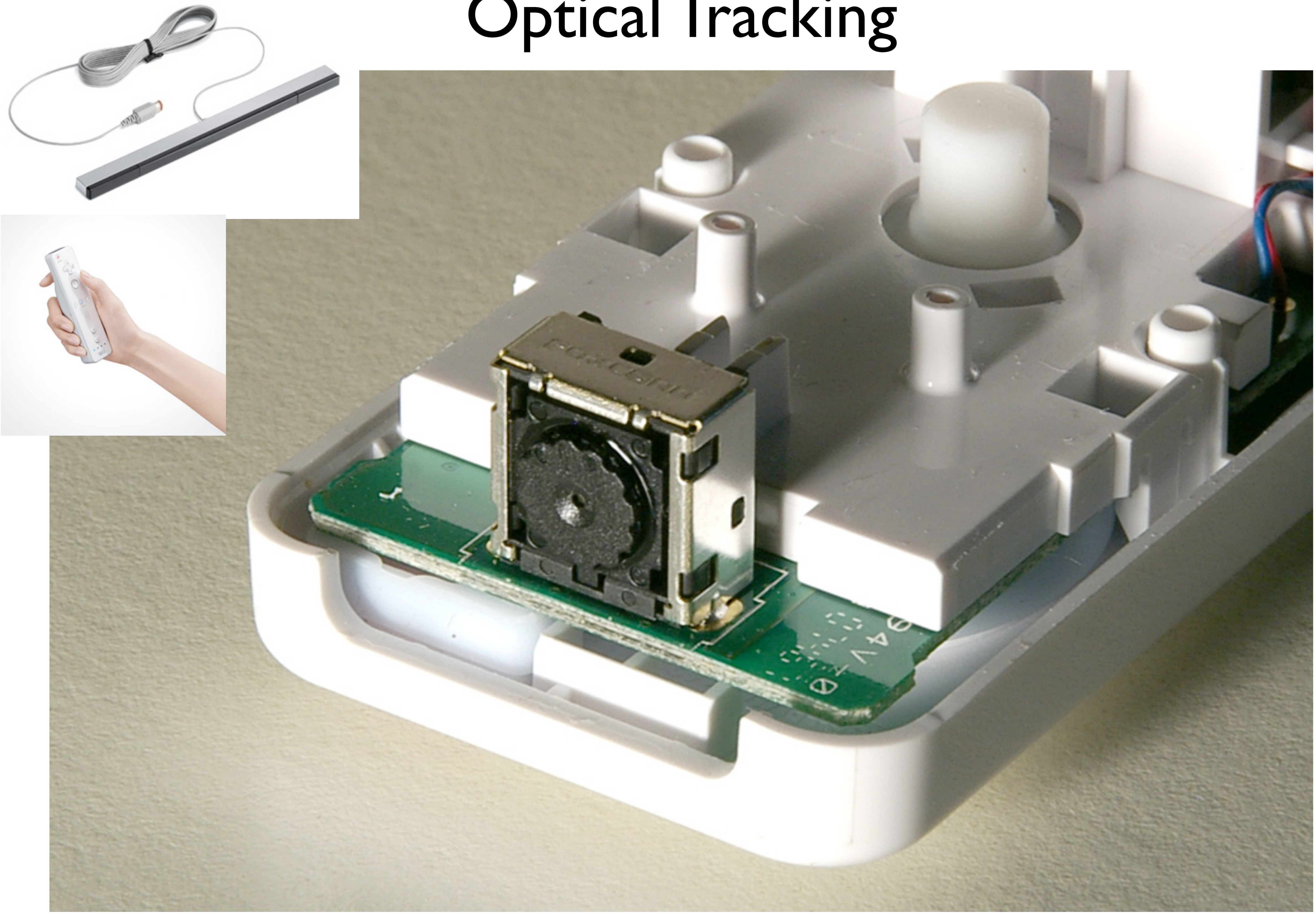
VICON system, many cameras, passive markers, >$100k

Optical Tracking



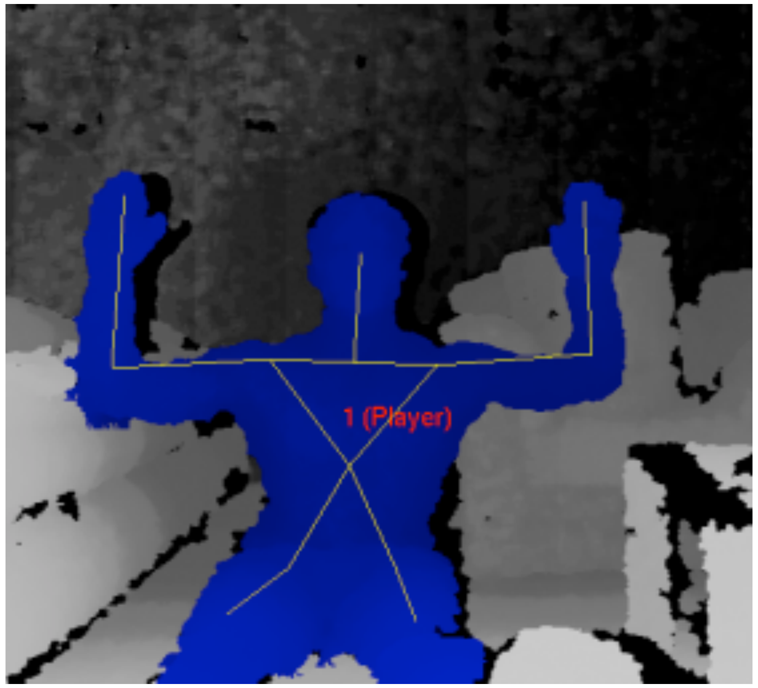
OptoTrak system: 3 cameras, active markers, ~$50k?

Optical Tracking

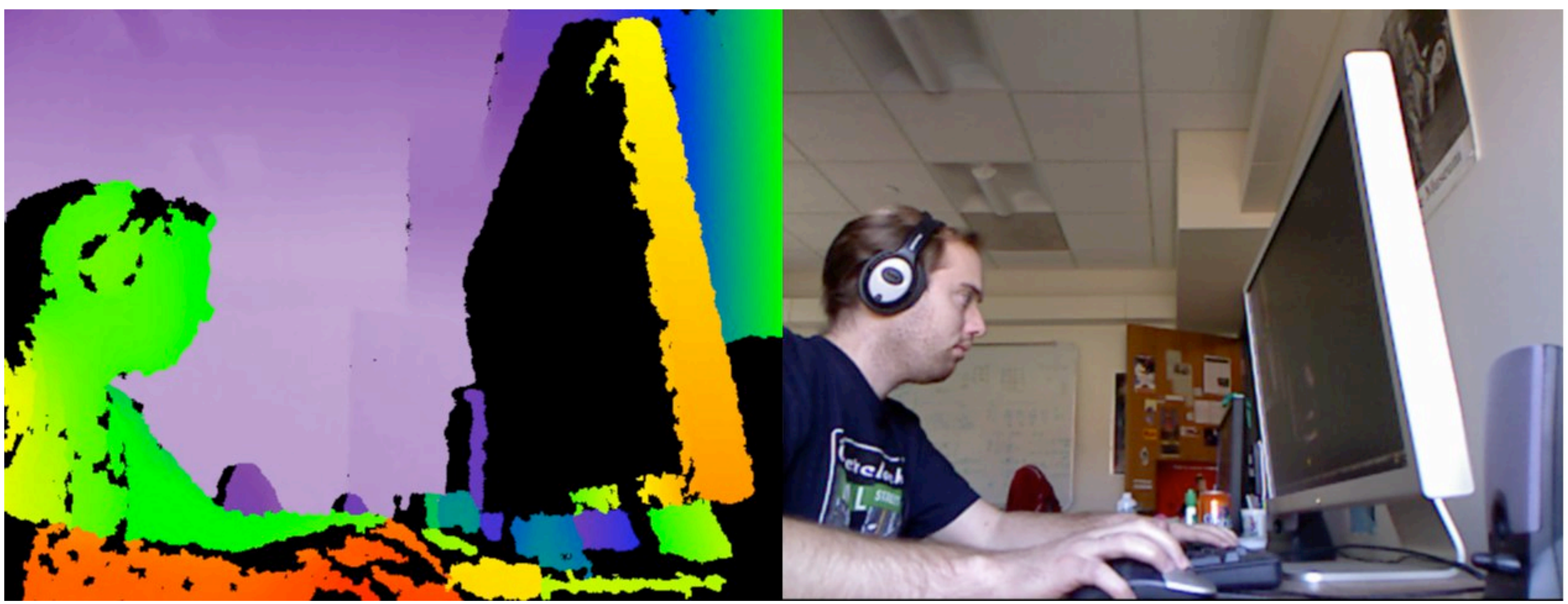


WiiMote camera: finds 4 brightest IR spots, ~$40

Optical Tracking

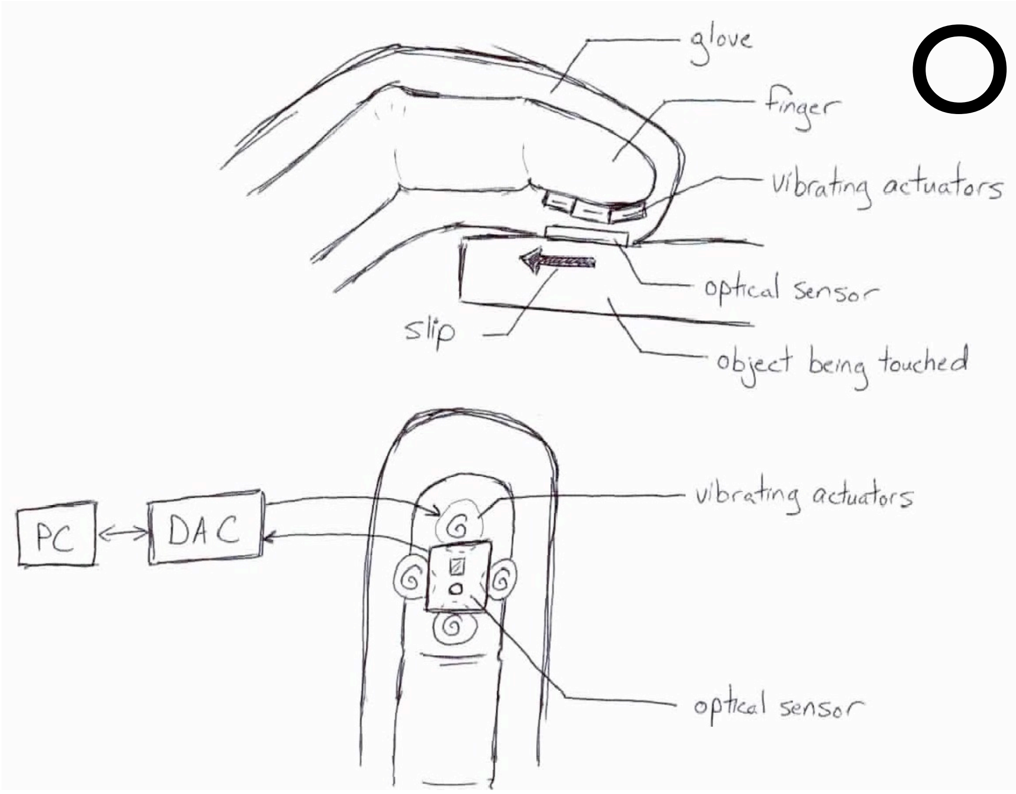


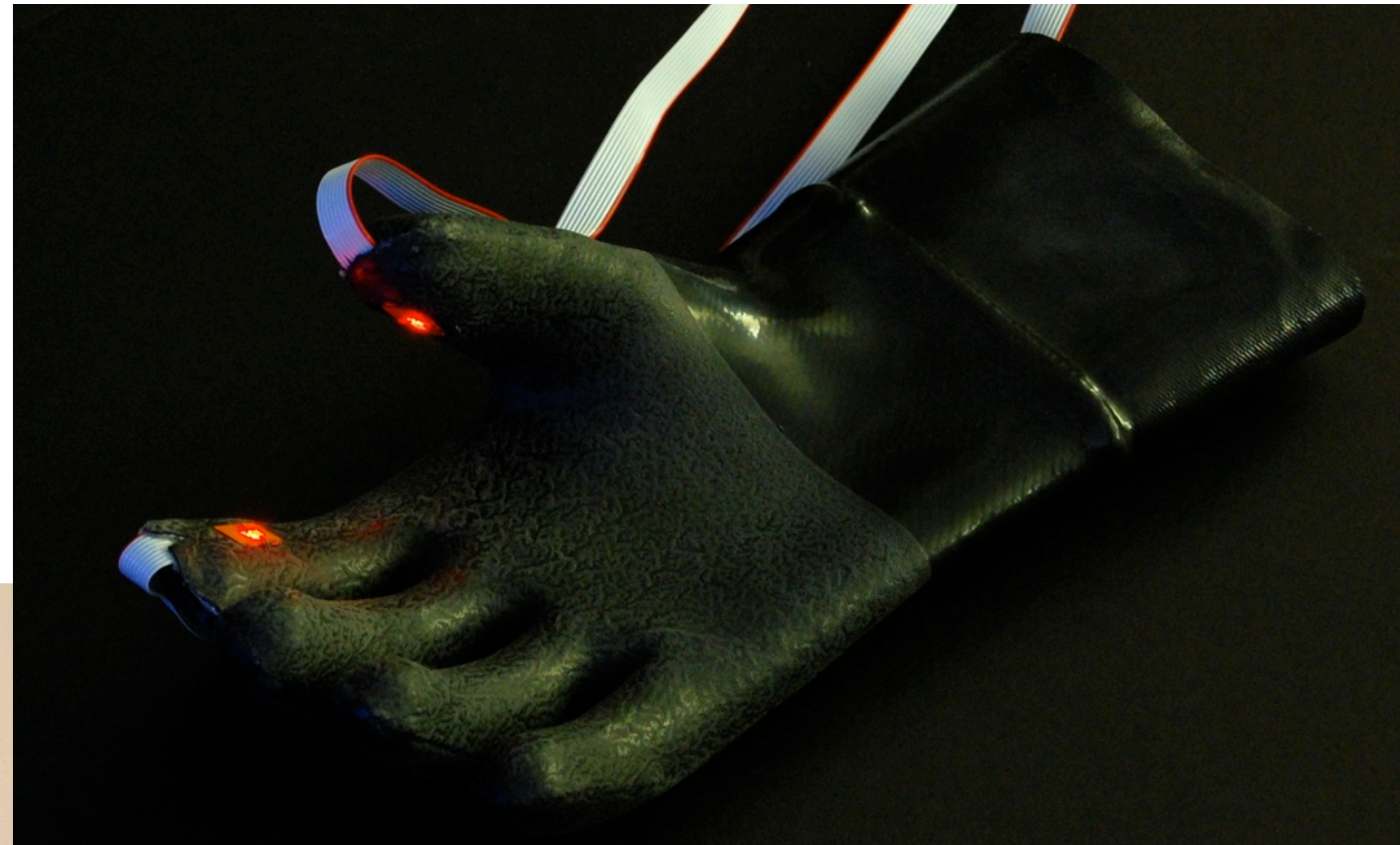
KINECT for XBOX 360

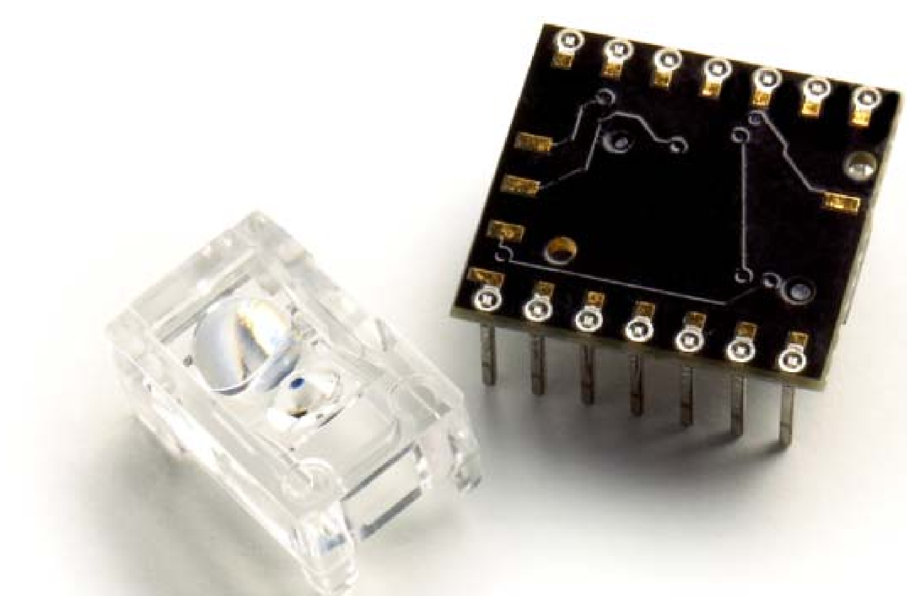


Kinect: color camera with depth, tracks humans, ~$200

Optical Tracking







Mouse Sensor: senses optical flow in 2D, digital comm.

Sensor Videos

Video Demonstrations of Mechatronics Principles

http://video_demos.colostate.edu/mechatronics/

Video Demonstrations of Mechatronic Devices and Principles

NOTE: The video clips below are Windows Media files.
They are **best viewed using Internet Explorer** and the Windows Media Player.
Click on the icon below if you wish to download the latest free player.

FREE Windows Media Player

- **actuators**
- **circuit examples**
- **data acquisition**
- **electrical components and measurements**
- **mechatronic system examples**
- **PIC microcontroller examples**
- **PIC microcontroller student design projects**
- **power transmission**
- **sensors**

See also:

- control system demonstrations
- vibration and sound demonstrations (various examples of spectra, frequency response, and sampling)
- robotics demonstrations

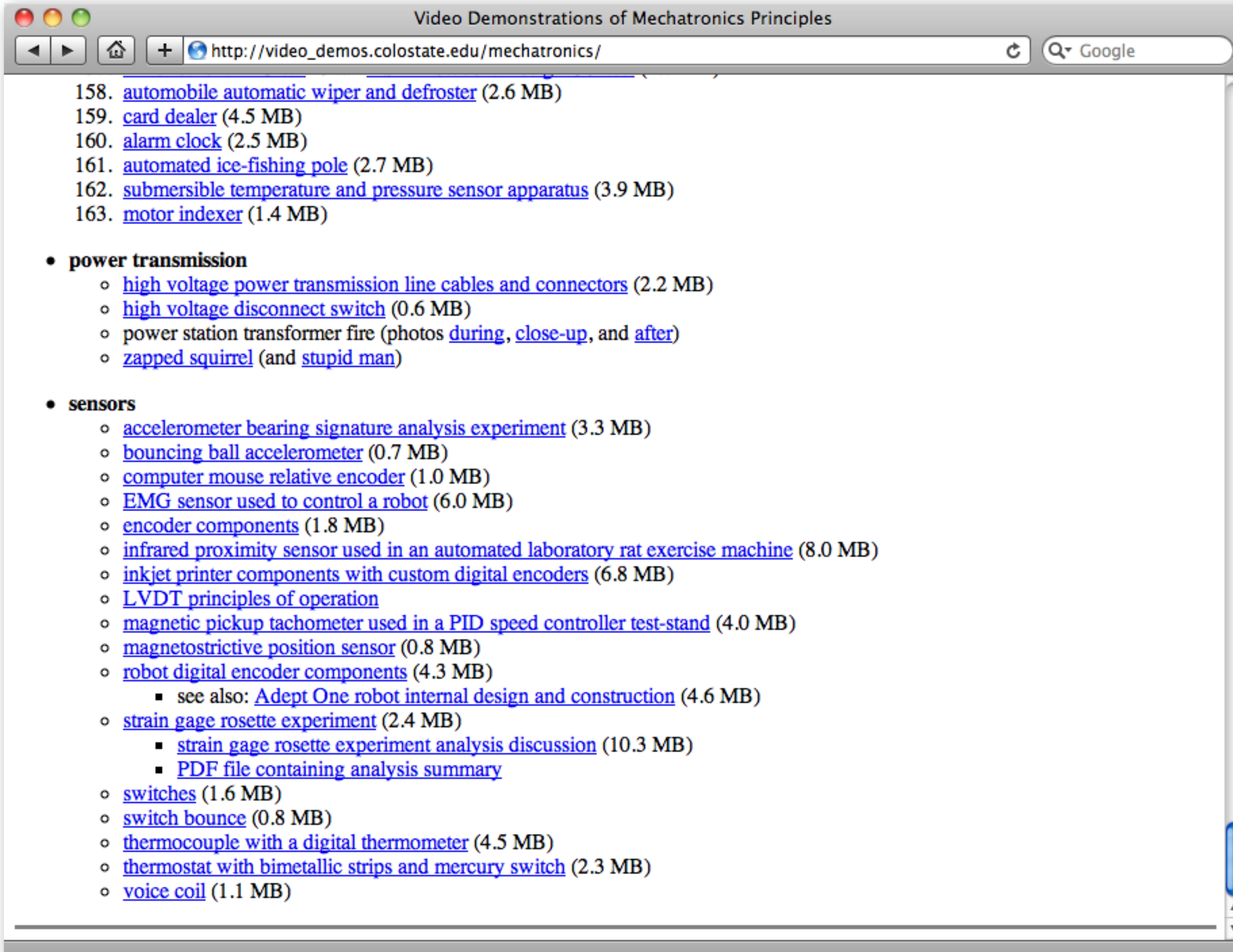
For more information about mechatronics and measurement systems, see:

- Introduction to Mechatronics and Measurement Systems

- **actuators**
  - ac induction motor (single phase) (2.3 MB)
  - ac induction motor variable frequency drive for a building air handler unit (6.8 MB)
  - ac induction motor with a soft start for a water pump (1.2 MB)
  - ac generator, motor, and load experiment (6.9 MB)
  - brushless dc motor from a computer fan (1.7 MB)
  - brushless dc motor gear pump (2.5 MB)

Videos from http://video_demos.colostate.edu/mechatronics/

Sensor Videos


Video Demonstrations of Mechatronics Principles

http://video_demos.colostate.edu/mechatronics/

158. automobile automatic wiper and defroster (2.6 MB)
159. card dealer (4.5 MB)
160. alarm clock (2.5 MB)
161. automated ice-fishing pole (2.7 MB)
162. submersible temperature and pressure sensor apparatus (3.9 MB)
163. motor indexer (1.4 MB)

- **power transmission**
  - high voltage power transmission line cables and connectors (2.2 MB)
  - high voltage disconnect switch (0.6 MB)
  - power station transformer fire (photos during, close-up, and after)
  - zapped squirrel (and stupid man)
- **sensors**
  - accelerometer bearing signature analysis experiment (3.3 MB)
  - bouncing ball accelerometer (0.7 MB)
  - computer mouse relative encoder (1.0 MB)
  - EMG sensor used to control a robot (6.0 MB)
  - encoder components (1.8 MB)
  - infrared proximity sensor used in an automated laboratory rat exercise machine (8.0 MB)
  - inkjet printer components with custom digital encoders (6.8 MB)
  - LVDT principles of operation
  - magnetic pickup tachometer used in a PID speed controller test-stand (4.0 MB)
  - magnetostrictive position sensor (0.8 MB)
  - robot digital encoder components (4.3 MB)
    - see also: Adept One robot internal design and construction (4.6 MB)
  - strain gage rosette experiment (2.4 MB)
    - strain gage rosette experiment analysis discussion (10.3 MB)
    - PDF file containing analysis summary
  - switches (1.6 MB)
  - switch bounce (0.8 MB)
  - thermocouple with a digital thermometer (4.5 MB)
  - thermostat with bimetallic strips and mercury switch (2.3 MB)
  - voice coil (1.1 MB)

Videos from http://video_demos.colostate.edu/mechatronics/

Quick Quiz What sensors do you see?

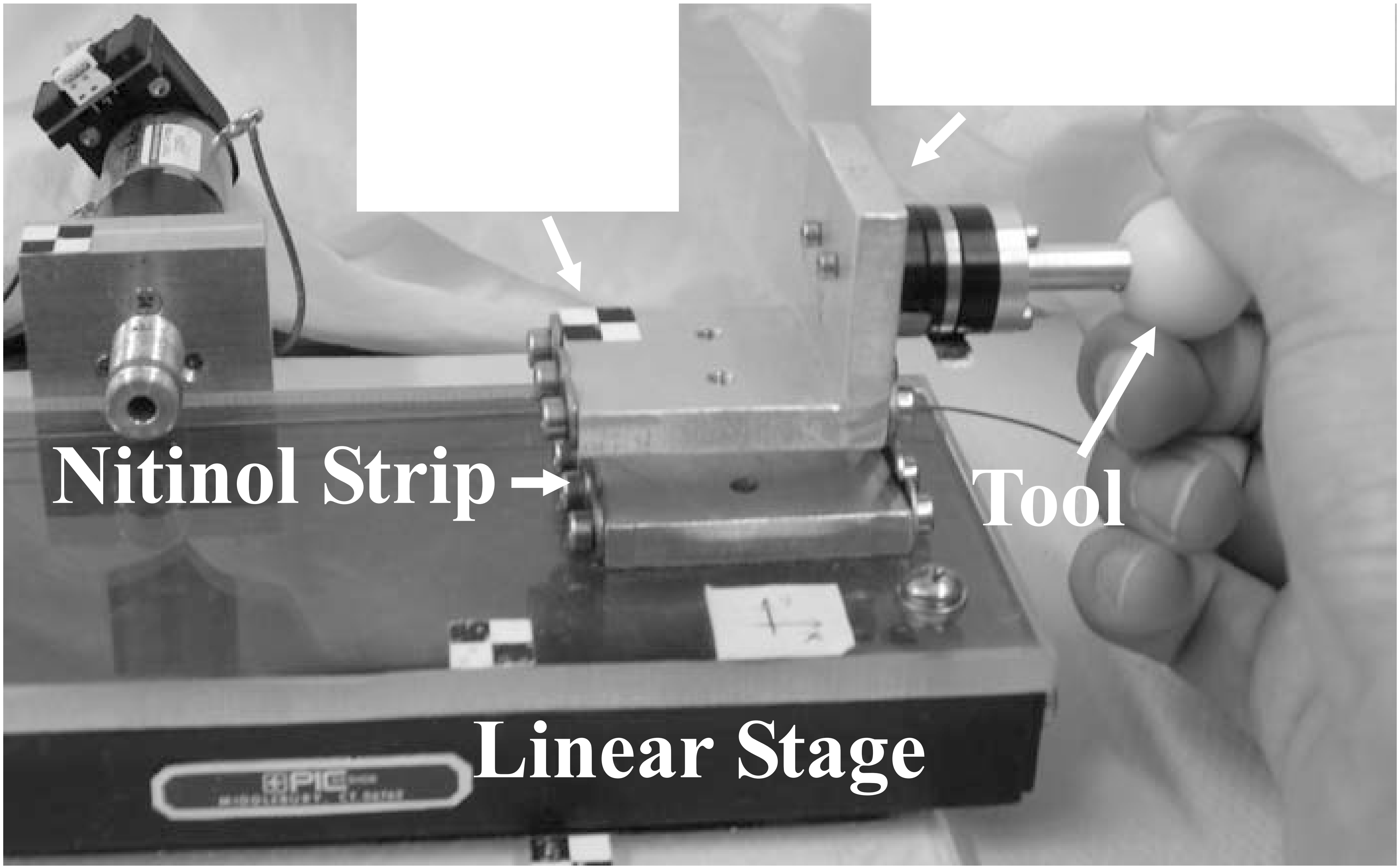


Image from Marayong, Na, and Okamura (ICRA 2007)

Quick Quiz What sensors do you see?

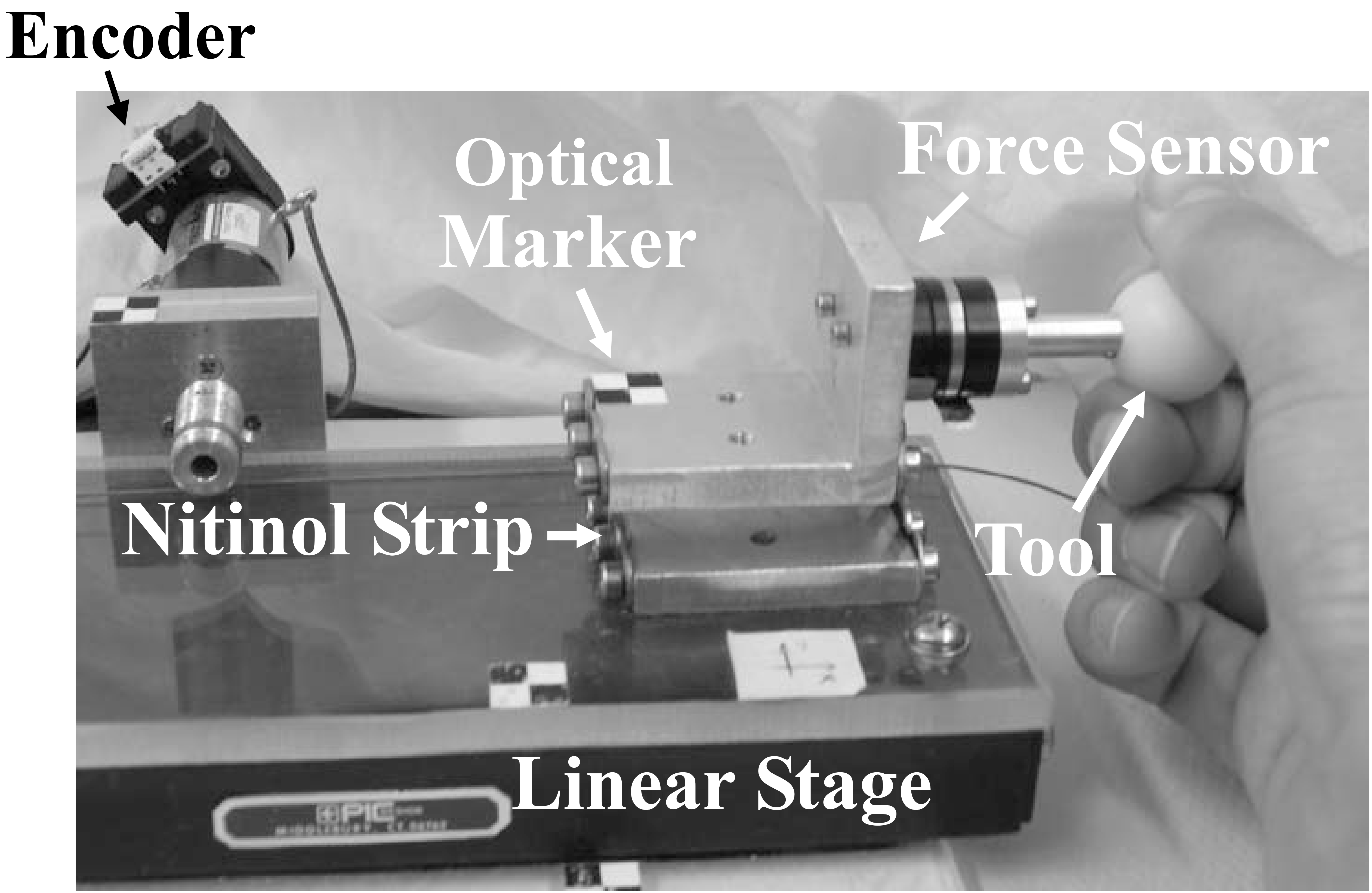


Image from Marayong, Na, and Okamura (ICRA 2007)

Any other types of sensors you are wondering about?

Sensor Processing

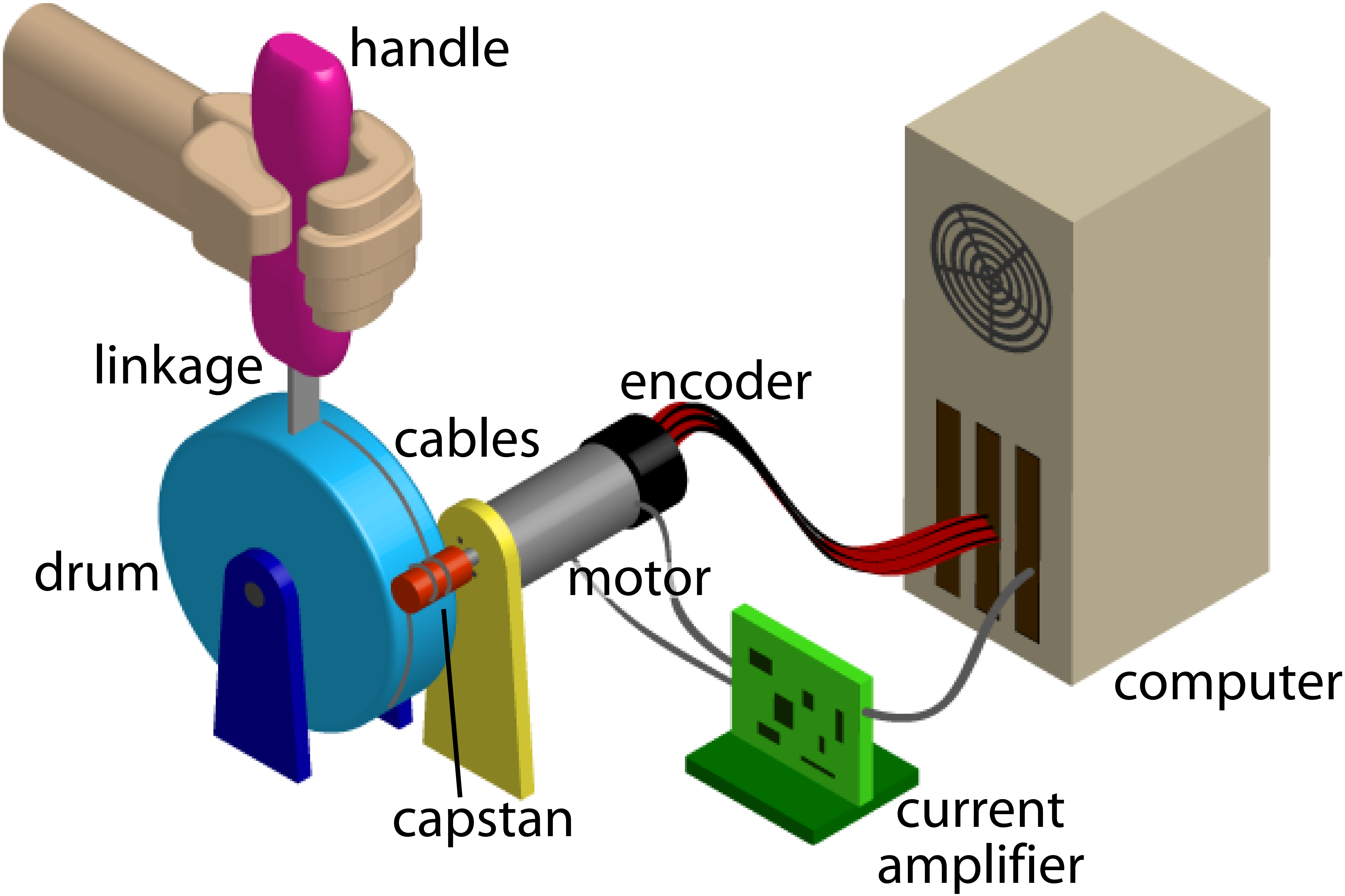


Illustration from K. J. Kuchenbecker and G. Niemeyer, "Induced Master Motion in Force-Reflecting Teleoperation." ASME Journal of Dynamic Systems, Measurement, and Control. Volume 128(4):800-810, December 2006.

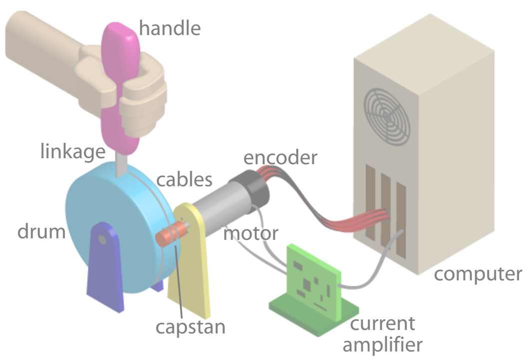
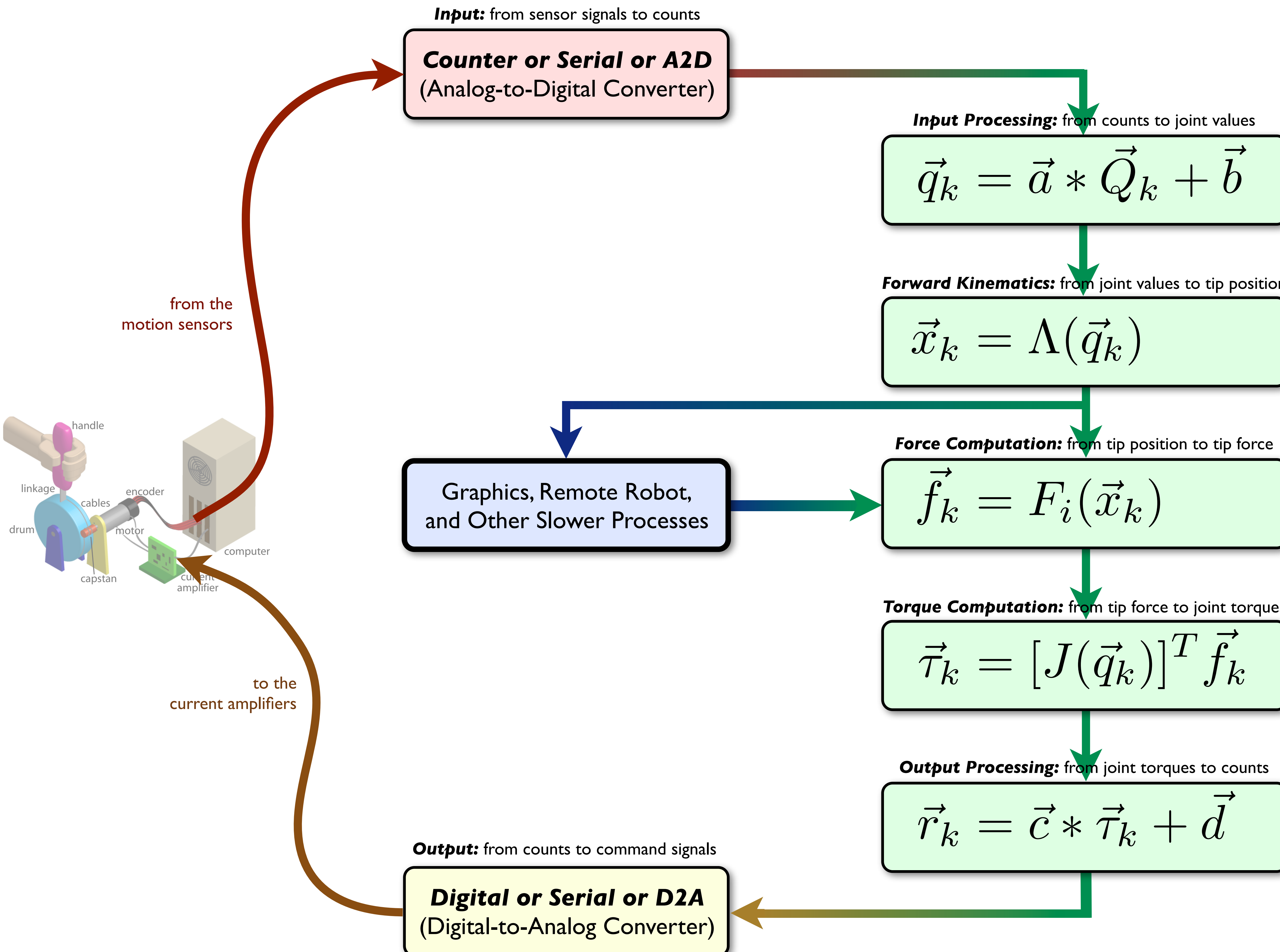


Illustration from K. J. Kuchenbecker and G. Niemeyer, "Induced Master Motion in Force-Reflecting Teleoperation."
ASME Journal of Dynamic Systems, Measurement, and Control. Volume 128(4):800-810, December 2006.

Typical Software Configuration



D/A and A/D

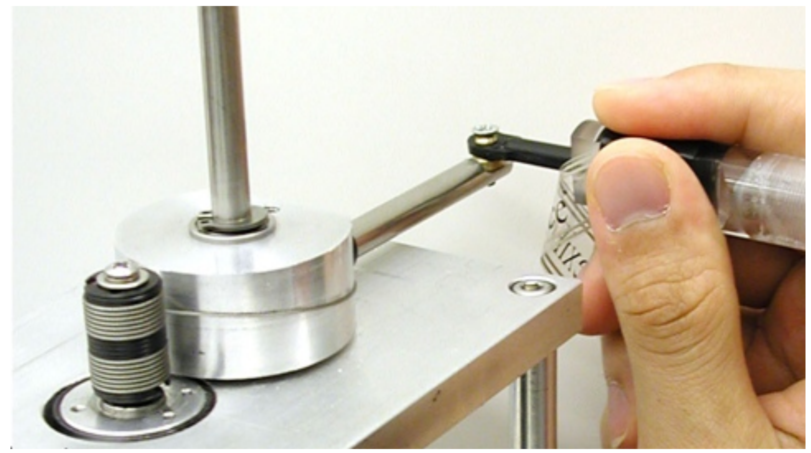
- Converts between voltages and counts
- Computer stores information digitally, and communicates with the outside world using signed voltage signals
  - e.g., for 8-bit 0-5V ADC
    2.5V = 10000000

MSB LSB

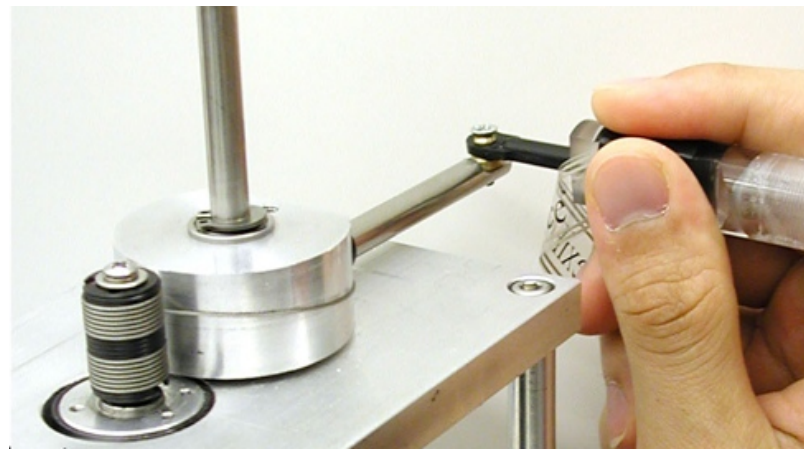
Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Adapted from slides by Will Provancher

Input Processing Steps

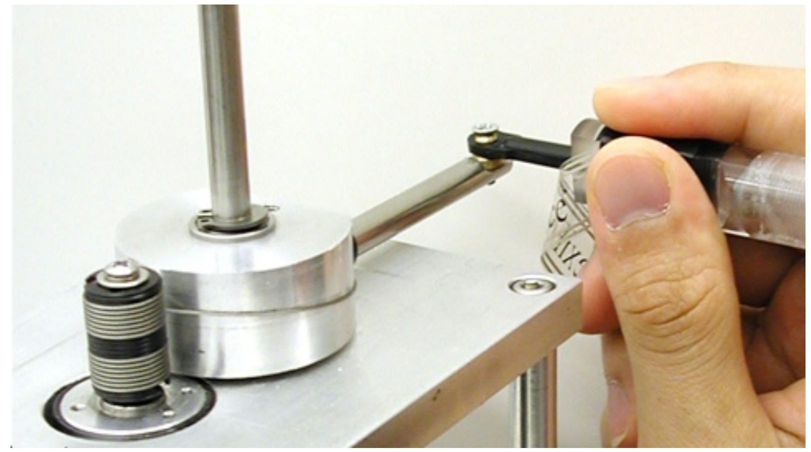


Input Processing Steps



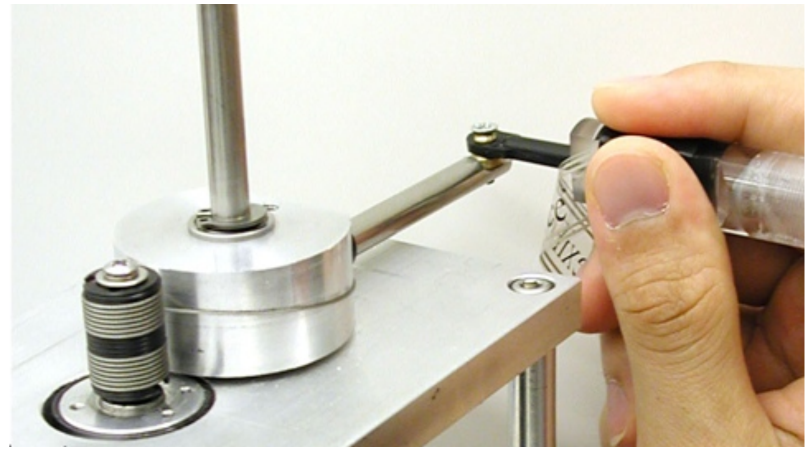
- Get counts $\boldsymbol{Q_j}$ from encoder counters, serial communications, or analog-to-digital conversions.

Input Processing Steps



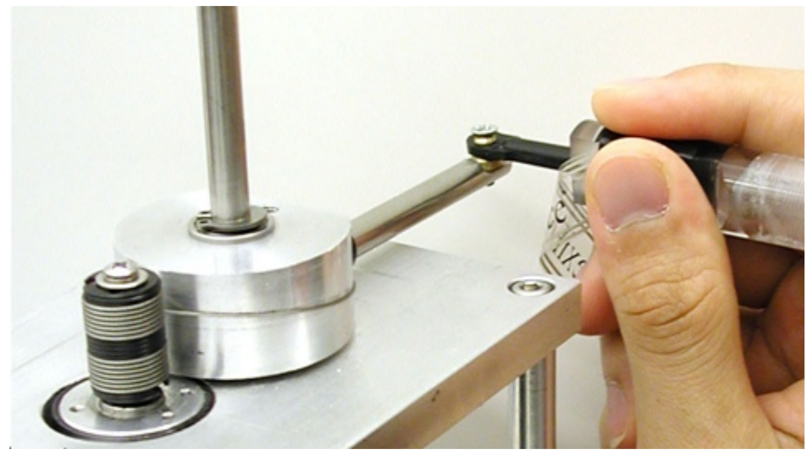
- Get counts Q_j from encoder counters, serial communications, or analog-to-digital conversions.
- Convert counts to sensor shaft angles θ_{sj} or sensor displacements d_{sj} using knowledge of the sensor's characteristics.

Input Processing Steps



- Get counts $\boldsymbol{Q_j}$ from encoder counters, serial communications, or analog-to-digital conversions.
- Convert counts to sensor shaft angles θ_{sj} or sensor displacements $\boldsymbol{d_{sj}}$ using knowledge of the sensor's characteristics.
- Convert sensor angles to joint coordinates $\boldsymbol{q_j}$ (joint angles θ_j or joint displacements $\boldsymbol{d_j}$) using the gear ratio. In this process, use a negative sign to flip the joint angle direction if desired.

Input Processing Steps

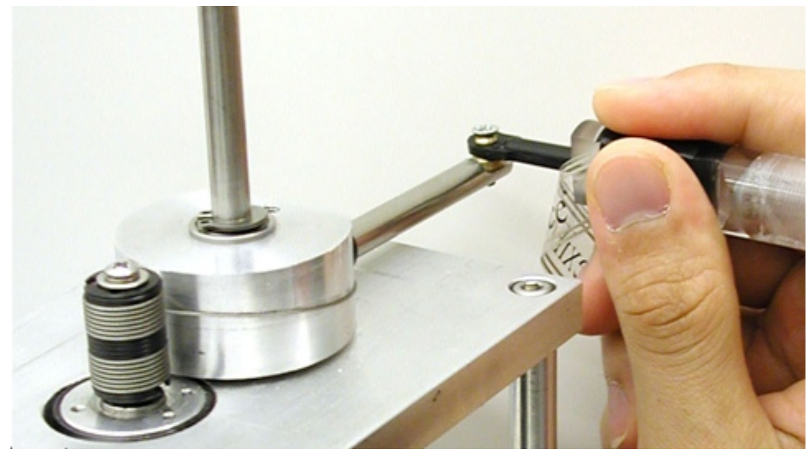


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Get counts Q_j from encoder counters, serial communications, or analog-to-digital conversions.
- Convert counts to sensor shaft angles θ_{sj} or sensor displacements d_{sj} using knowledge of the sensor's characteristics.
- Convert sensor angles to joint coordinates q_j (joint angles θ_j or joint displacements d_j) using the gear ratio. In this process, use a negative sign to flip the joint angle direction if desired.

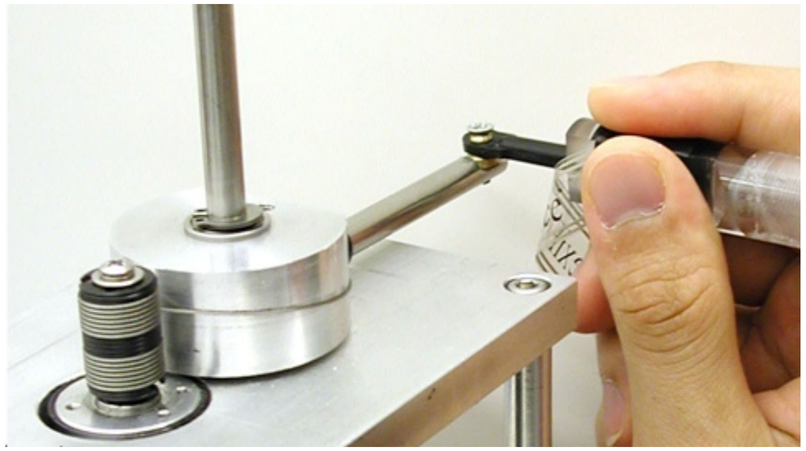
Input Processing Steps



Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

Input Processing Steps

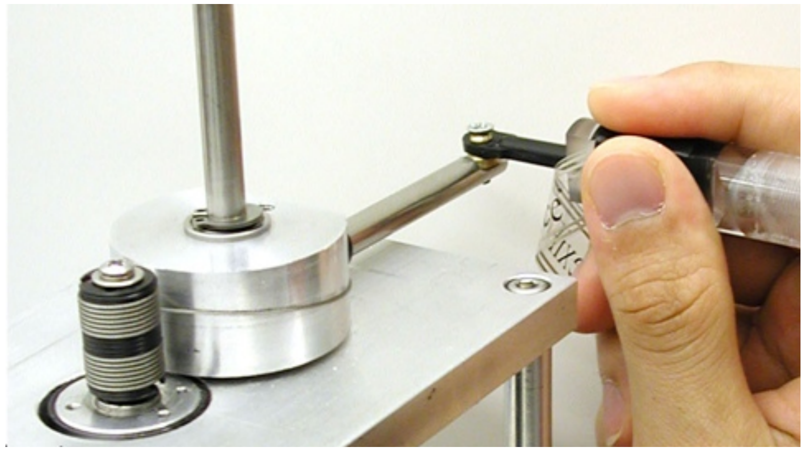


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Check your work along the way.

Input Processing Steps

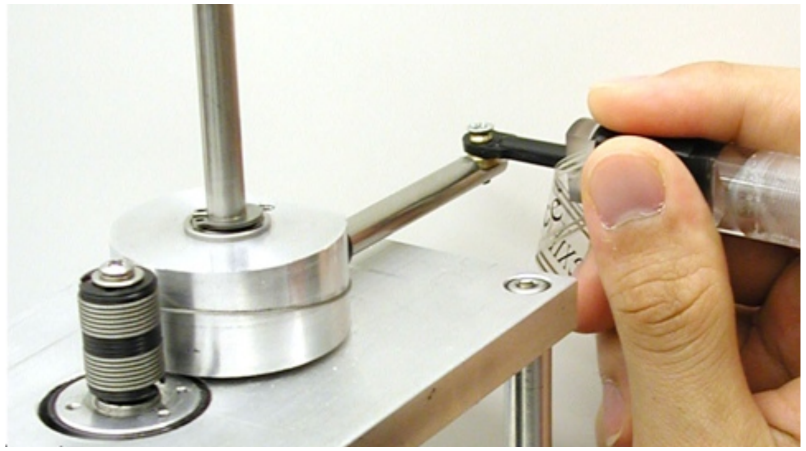


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Check your work along the way.
- How?

Input Processing Steps

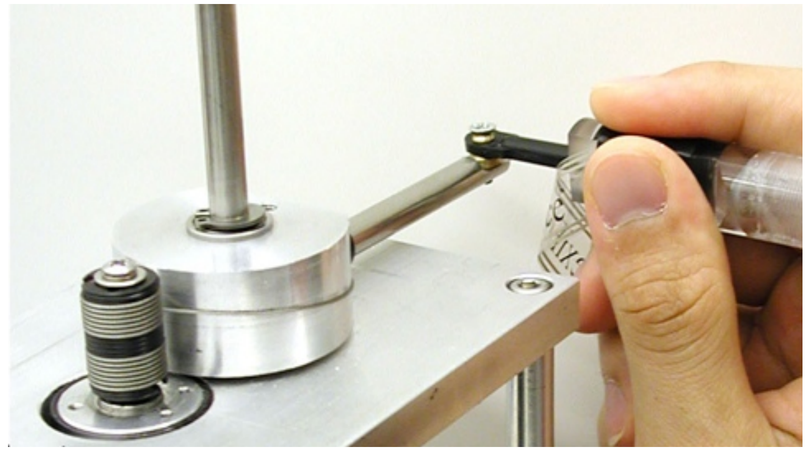


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Check your work along the way.
- How?
  - Units

Input Processing Steps

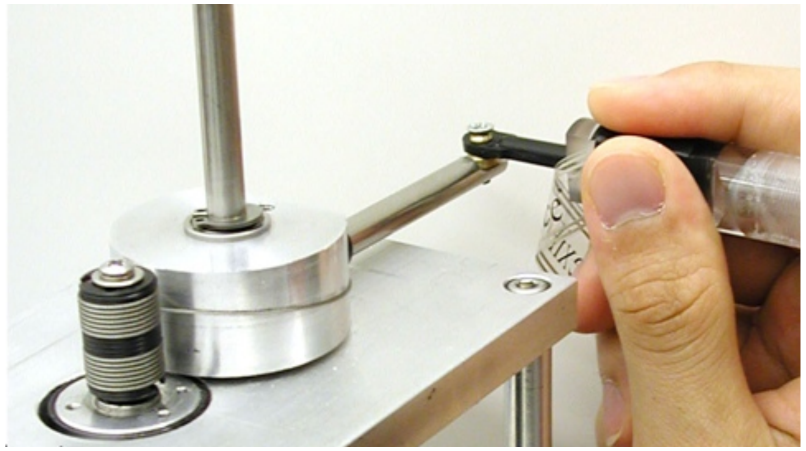


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Check your work along the way.
- How?
  - Units
  - Known configurations

Input Processing Steps

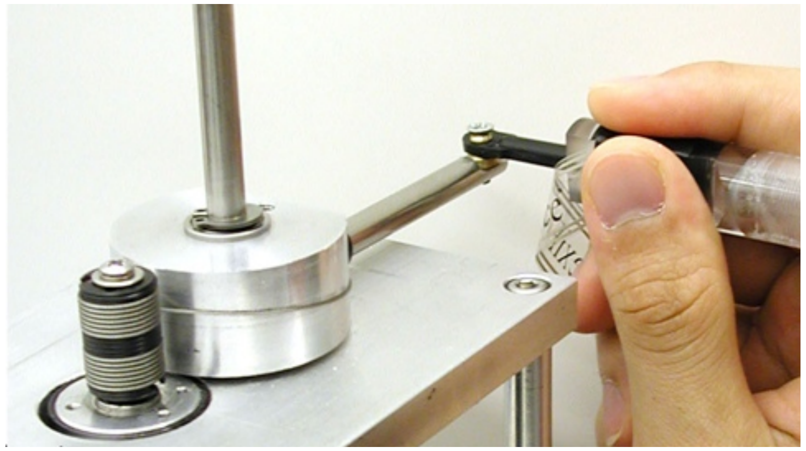


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Check your work along the way.
- How?
  - Units
  - Known configurations
  - Ranges

Input Processing Steps

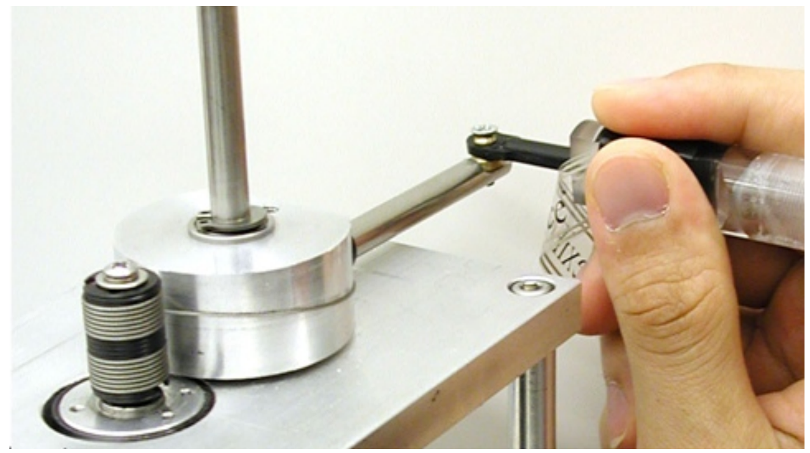


Input Processing: from counts to joint values

$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

- Check your work along the way.
- How?
  - Units
  - Known configurations
  - Ranges
  - Record and graph

Input Processing Steps



Input Processing: from counts to joint values

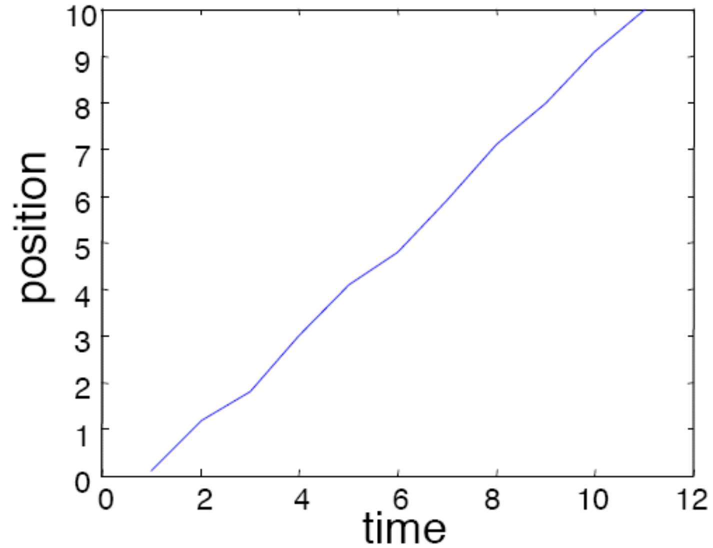
$$\vec{q}_k = \vec{a} * \vec{Q}_k + \vec{b}$$

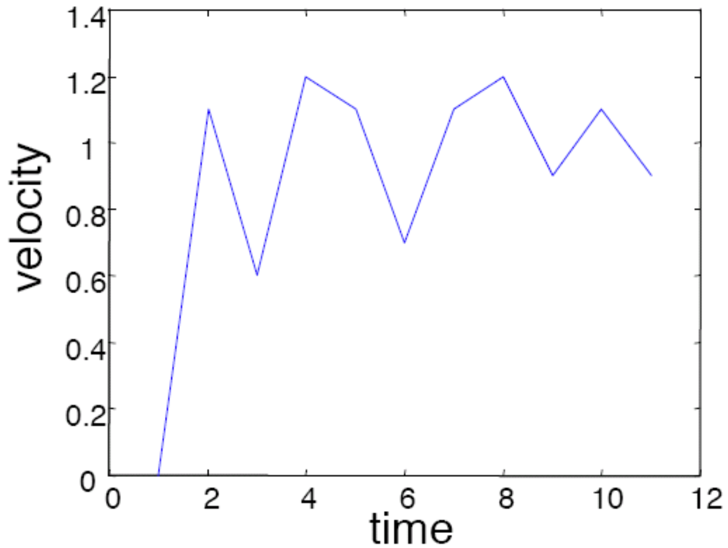
- Check your work along the way.
- How?
  - Units
  - Known configurations
  - Ranges
  - Record and graph
- Check *before* you use the movement information to output forces.

Digital differentiation

- Many different methods
- Simple Example:
  - Position reading at time 1 = P1
  - Position reading at time 2 = P2
  - t is the period of the servo loop (in sec. or counts)
    - The position is typically sampled on a fixed interval
- Differentiation increases noise

$$V = \frac{P2 - P1}{t}$$





Adapted from slides by Will Provancher

Noisy Velocity readings

- Noise on velocity signal can create jitter on your haptic device when your controller has velocity feedback (virtual damping)
- Common solutions
  - Use a Tach/Generator
    - Voltage goes with speed (same source as back-EMF)
    - Resolution is set by your A/D converter
  - Integrate the signal from an accelerometer
  - Measure time per tick rather than ticks per time
    - Use a special chip that measures time between ticks
    - Especially good to do at slow speeds
    - Fares poorly at high velocities
  - Filtering (conventional to smooth or Kalman filtering to combine sensor signals)

Adapted from slides by Will Provancher

Calculating Velocity

Calculating Velocity

```
/*******************************************************************************
 hapticCallback()
 Main callback that sets the force that the user will feel.  It gets the current
 position and velocity of the device.
 This is what you want to edit to change the system's haptic feedback.
*******************************************************************************/
HDCallbackCode HDCALLBACK hapticCallback(void *data)
{
  // Local variables.
  hduVector3Dd position;
  hduVector3Dd velocity;
  hduVector3Dd force;
  hduVector3Dd extraForce;
  hduVector3Dd proxyPosition;
  HDint currentButtonState;
  HDint lastButtonState;
  double stiffness = 0.25;  // Units are newtons per millimeter.
```

Calculating Velocity

```
/*******************************************************************************
 hapticCallback()
 Main callback that sets the force that the user will feel.  It gets the current
 position and velocity of the device.
 This is what you want to edit to change the system's haptic feedback.
*******************************************************************************/
HDCallbackCode HDCALLBACK hapticCallback(void *data)
{
  // Local variables.
  hduVector3Dd position;
  hduVector3Dd velocity;
  hduVector3Dd force;
  hduVector3Dd extraForce;
  hduVector3Dd proxyPosition;
  HDint currentButtonState;
  HDint lastButtonState;
  double stiffness = 0.25;  // Units are newtons per millimeter.

  // Local variables for custom velocity calculation.
  static bool firstTime = true;
  static hduVector3Dd lastPosition; // mm
  hduVector3Dd rawVelocity; // mm/s
  static hduVector3Dd filteredVelocity(0.0, 0.0, 0.0);  // mm/s
  float filterWeight = 0.03;
  float dampingCoefficient = 0.01; // N/(mm/s)
  hduVector3Dd dampingForce; // N
```

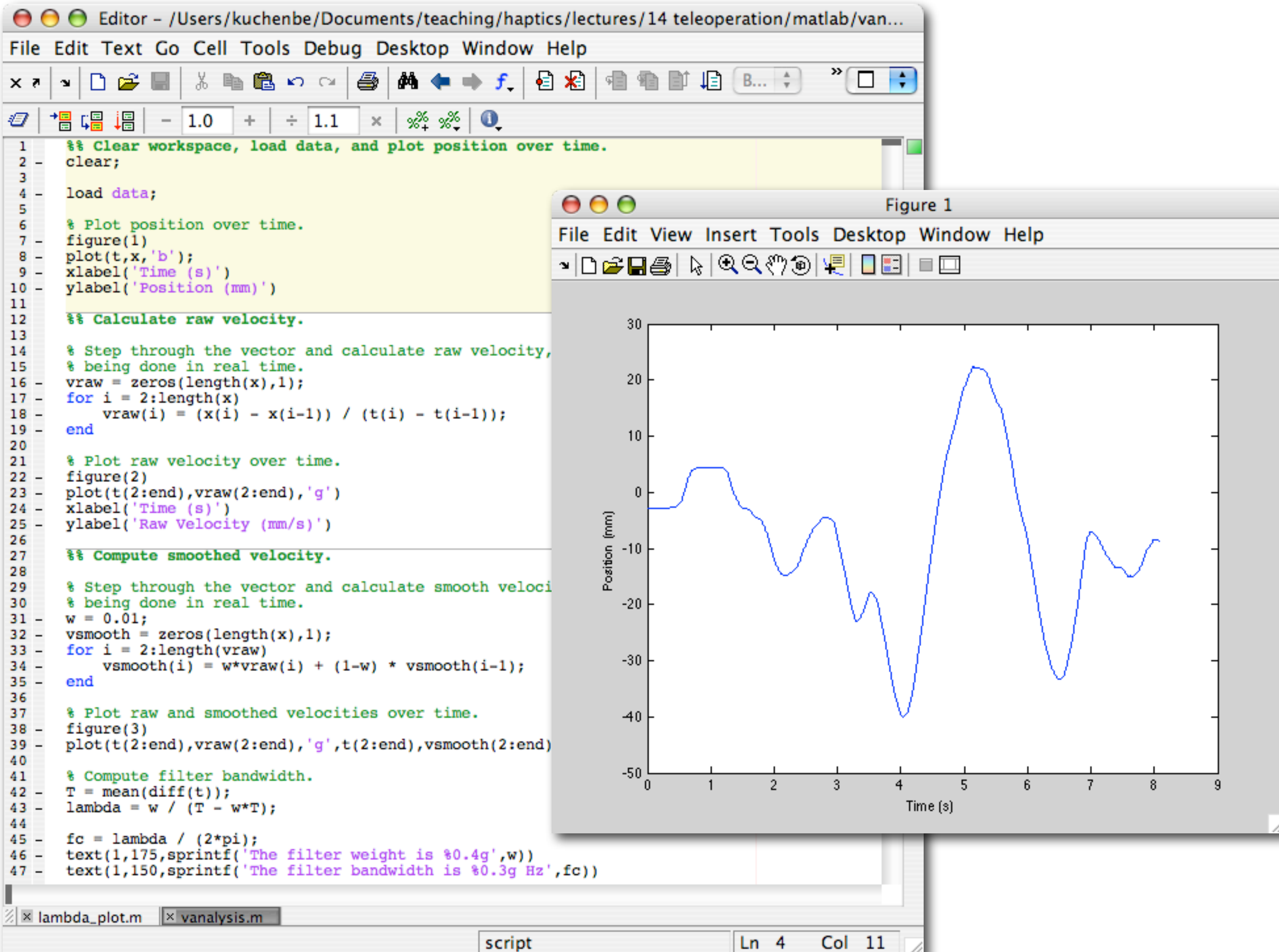
Calculating Velocity

```
// Local variables for custom velocity calculation.
static bool firstTime = true;
static hduVector3Dd lastPosition; // mm
hduVector3Dd rawVelocity; // mm/s
static hduVector3Dd filteredVelocity(0.0, 0.0, 0.0);  // mm/s
float filterWeight = 0.03;
float dampingCoefficient = 0.01; // N/(mm/s)
hduVector3Dd dampingForce; // N

// Get the handle for the current haptic device.
HHD hHD = hdGetCurrentDevice();

// Begin the haptic frame for this device.
hdBeginFrame(hHD);

// Get its position and velocity and store them in hduVector3Dd variables.
hdGetDoublev(HD_CURRENT_POSITION, position);  // Units are millimeters.
hdGetDoublev(HD_CURRENT_VELOCITY, velocity);  // Units are millimeters per second.

// Fill lastPosition with current position if this is the first function call.
if (firstTime) {
  lastPosition = position;
  firstTime = false;
}

// Calculate the raw velocity from this position and lastPosition.
rawVelocity = (position - lastPosition) / DELTAT;

// Low-pass filter this raw velocity signal using a first-order IIR filter.
filteredVelocity = filterWeight * rawVelocity + (1 - filterWeight) * filteredVelocity;
```

Calculating Velocity

```
hdGetDoublev(HD_CURRENT_POSITION, position);  // Units are millimeters.
hdGetDoublev(HD_CURRENT_VELOCITY, velocity);  // Units are millimeters per second.

// Fill lastPosition with current position if this is the first function call.
if (firstTime) {
  lastPosition = position;
  firstTime = false;
}

// Calculate the raw velocity from this position and lastPosition.
rawVelocity = (position - lastPosition) / DELTAT;

// Low-pass filter this raw velocity signal using a first-order IIR filter.
filteredVelocity = filterWeight * rawVelocity + (1 - filterWeight) * filteredVelocity;

// Store current position as lastPosition for next time.
lastPosition = position;

// Use the custom filtered velocity rather than SensAble's velocity?
// Comment out this line if you want to use the standard velocity.
velocity = filteredVelocity;

// Compute an extra damping force to add to the force the user feels,
// just so you can test the velocity.
dampingForce = -dampingCoefficient * velocity;

// Other code....

// Compute the force.
force = stiffness * (proxyPosition - position) + dampingForce + extraForce;
```

$$v_{smooth}(k) = w \cdot v_{raw}(k) + (1 - w) \cdot v_{smooth}(k - 1)$$


Editor - /Users/kuchenbe/Documents/teaching/haptics/lectures/14 teleoperation/matlab/van...
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
%% Clear workspace, load data, and plot position over time.
clear;
load data;
% Plot position over time.
figure(1)
plot(t,x,'b');
xlabel('Time (s)')
ylabel('Position (mm)')
%% Calculate raw velocity.
% Step through the vector and calculate raw velocity,
% being done in real time.
vraw = zeros(length(x),1);
for i = 2:length(x)
vraw(i) = (x(i) - x(i-1)) / (t(i) - t(i-1));
end
% Plot raw velocity over time.
figure(2)
plot(t(2:end),vraw(2:end),'g')
xlabel('Time (s)')
ylabel('Raw Velocity (mm/s)')
%% Compute smoothed velocity.
% Step through the vector and calculate smooth veloci
% being done in real time.
w = 0.01;
vsmooth = zeros(length(x),1);
for i = 2:length(vraw)
vsmooth(i) = w*vraw(i) + (1-w) * vsmooth(i-1);
end
% Plot raw and smoothed velocities over time.
figure(3)
plot(t(2:end),vraw(2:end),'g',t(2:end),vsmooth(2:end)
% Compute filter bandwidth.
T = mean(diff(t));
lambda = w / (T - w*T);
fc = lambda / (2*pi);
text(1,175,sprintf('The filter weight is %0.4g',w))
text(1,150,sprintf('The filter bandwidth is %0.3g Hz',fc))
lambda_plot.m
vanalysis.m
script
Ln 4
Col 11
Figure 1
File Edit View Insert Tools Desktop Window Help
30
20
10
0
-10
-20
-30
-40
-50
Position (mm)
0 1 2 3 4 5 6 7 8 9
Time (s)

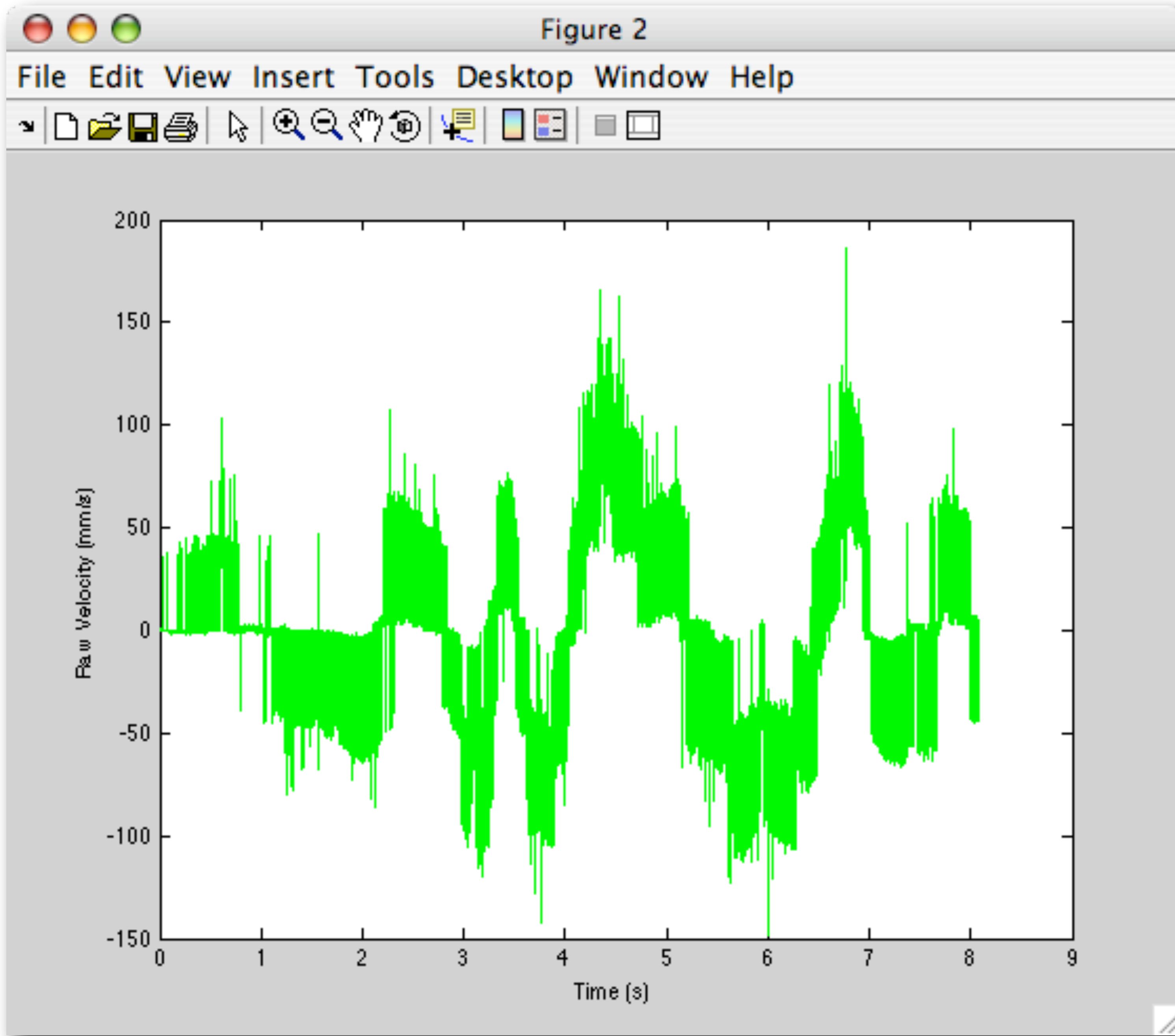


Figure 2
File Edit View Insert Tools Desktop Window Help
200
150
100
50
0
-50
-100
-150
Raw Velocity (mm/s)
0 1 2 3 4 5 6 7 8 9
Time (s)

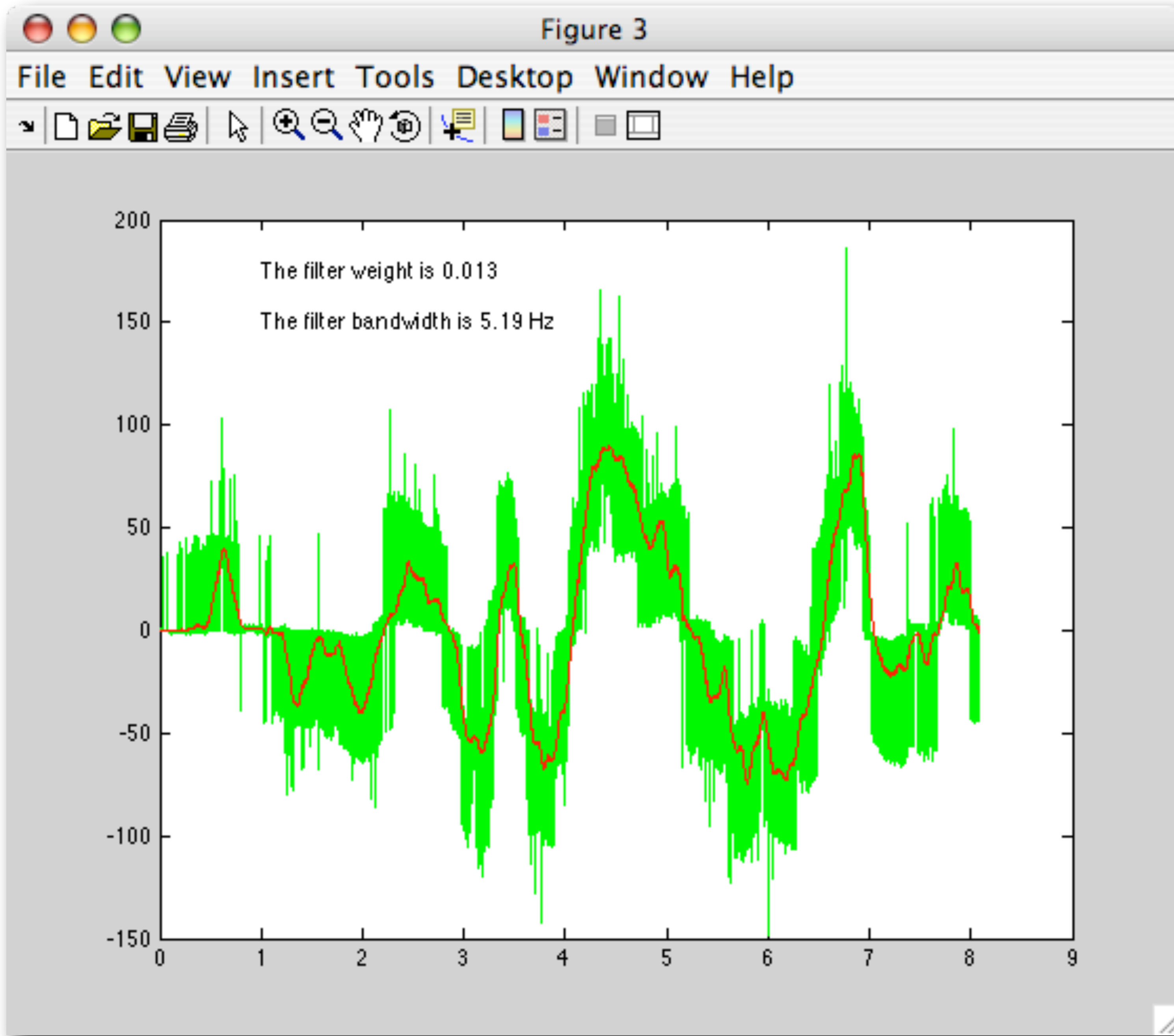


Figure 3
File Edit View Insert Tools Desktop Window Help
The filter weight is 0.013
The filter bandwidth is 5.19 Hz
200
150
100
50
0
-50
-100
-150
0
1
2
3
4
5
6
7
8
9

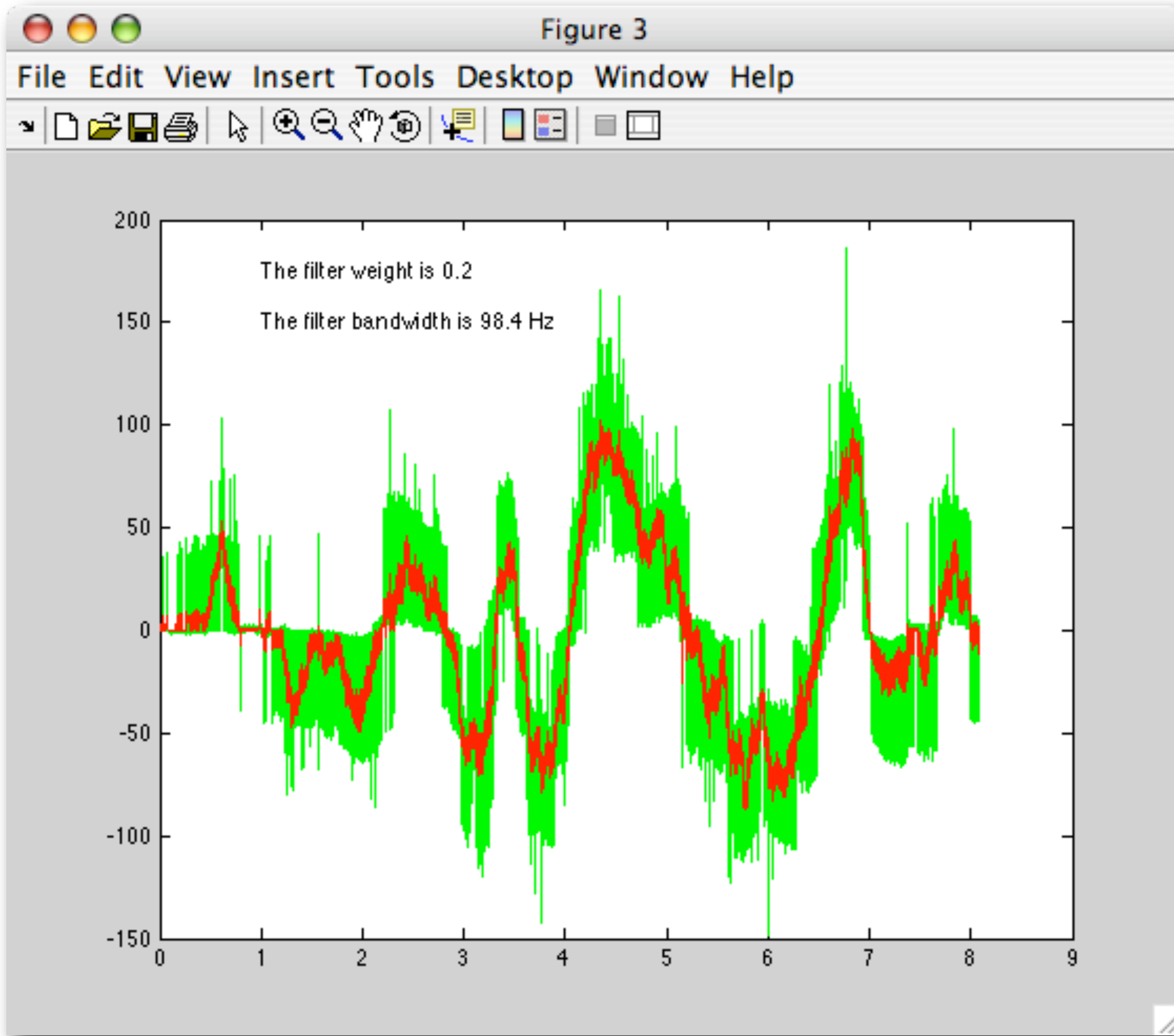


Figure 3
File Edit View Insert Tools Desktop Window Help
The filter weight is 0.2
The filter bandwidth is 98.4 Hz
200
150
100
50
0
-50
-100
-150
0 1 2 3 4 5 6 7 8 9

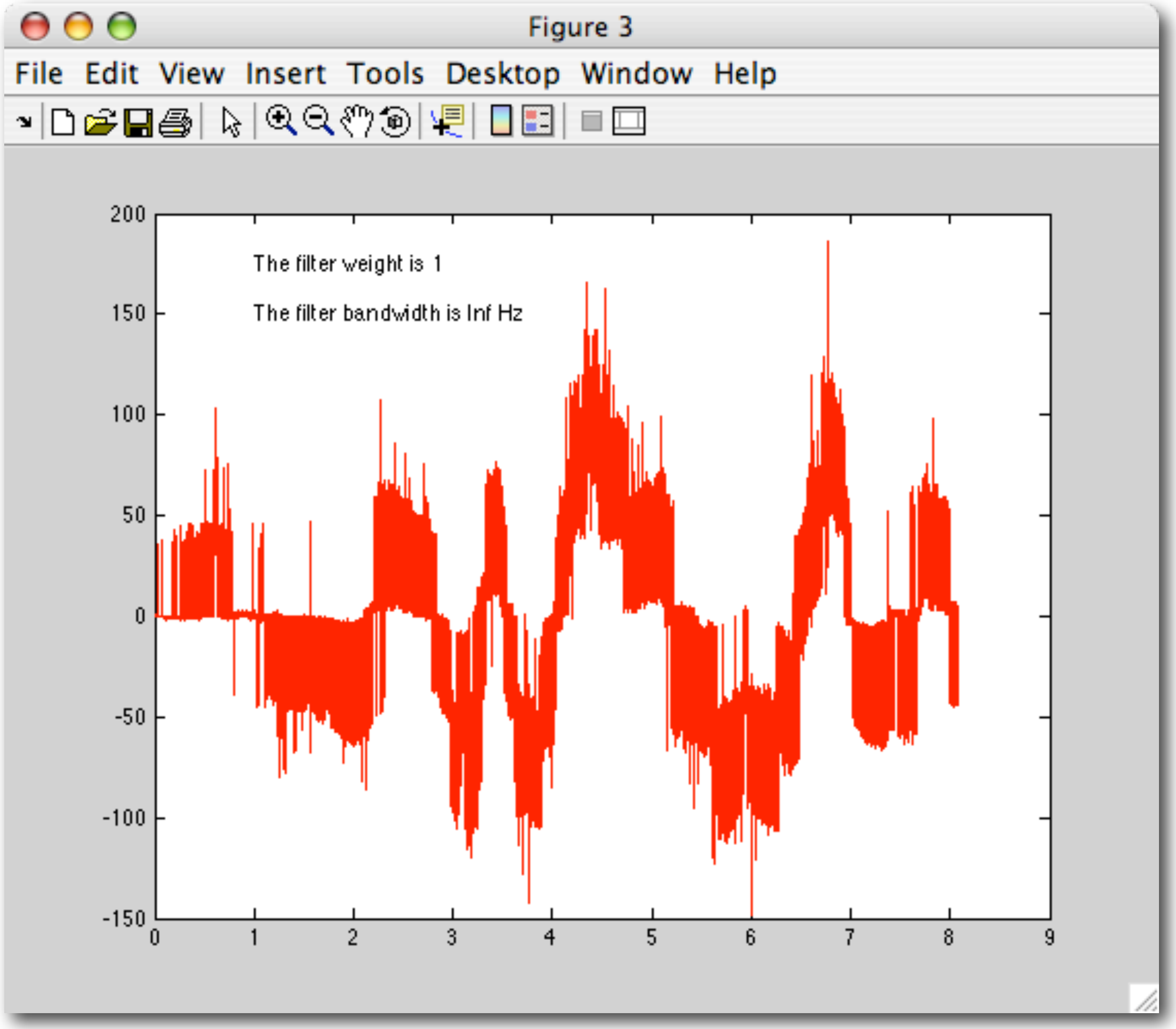


Figure 3
File Edit View Insert Tools Desktop Window Help
The filter weight is 1
The filter bandwidth is Inf Hz
200
150
100
50
0
-50
-100
-150
0
1
2
3
4
5
6
7
8
9

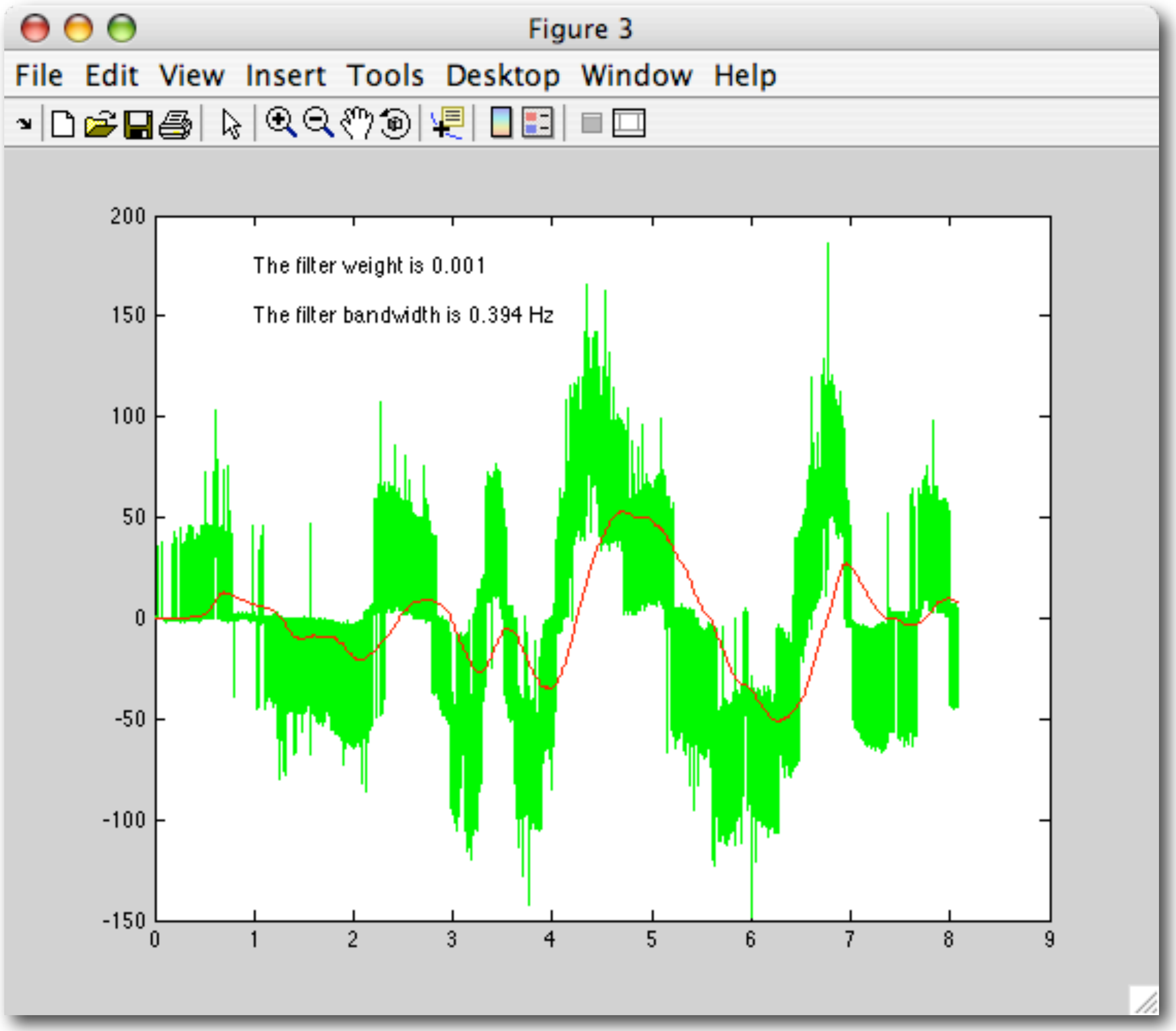


Figure 3
File Edit View Insert Tools Desktop Window Help
The filter weight is 0.001
The filter bandwidth is 0.394 Hz
200
150
100
50
0
-50
-100
-150
0
1
2
3
4
5
6
7
8
9

KJK

24 Jan 2007

Begin with a first-order continuous-time low-pass filter, where Y(s) is the Laplace transform of the filtered output and X(s) is that of our signal.

$$\frac{Y(s)}{X(s)} = \frac{\lambda}{s+\lambda}$$

output → Y(s); input ↗ X(s)

← gain = 1 @ s=0

↑ one pole at $-\lambda$, λ is filter cutoff in rad/s

Convert from continuous time (smooth derivatives) to discrete time (sampled at intervals of T seconds, $f=\frac{1}{T}$). This requires us to choose a method for approximating the derivative. Other options would work too, but the simplest is backward differencing:

$$s = \frac{(1 - z^{-1})}{T}$$

makes sense: this value minus last value divided by T.

z transform acts like a shift operator.

$Y(z) * z^{-1}$ is previous y value

$Y(z) * z^{0} = Y(z)$ is this y value

$Y(z) * z$ is next y value

substitute this in for s in the above eqn.

$$\frac{Y(z)}{X(z)} = \frac{\lambda}{\frac{(1-z^{-1})}{T} + \lambda}$$

$$Y(z)\left[\frac{(1-z^{-1})}{T} + \lambda\right] = \lambda X(z)$$

$$Y(z) - Y(z) * z^{-1} + \lambda T Y(z) = \lambda T X(z)$$

$$(1+\lambda T) Y(z) = \lambda T X(z) + z^{-1} * Y(z)$$

$$Y(z) = \frac{\lambda T}{1+\lambda T} X(z) + \frac{1}{1+\lambda T} z^{-1} Y(z)$$

do inverse z transform

$$y(k) = \frac{\lambda T}{1+\lambda T} x(k) + \frac{1}{1+\lambda T} y(k-1)$$

index ↗ (k)

$$y(k) = w \cdot x(k) + (1-w) y(k-1)$$

↑ filter_weight $= \frac{\lambda T}{1+\lambda T}$

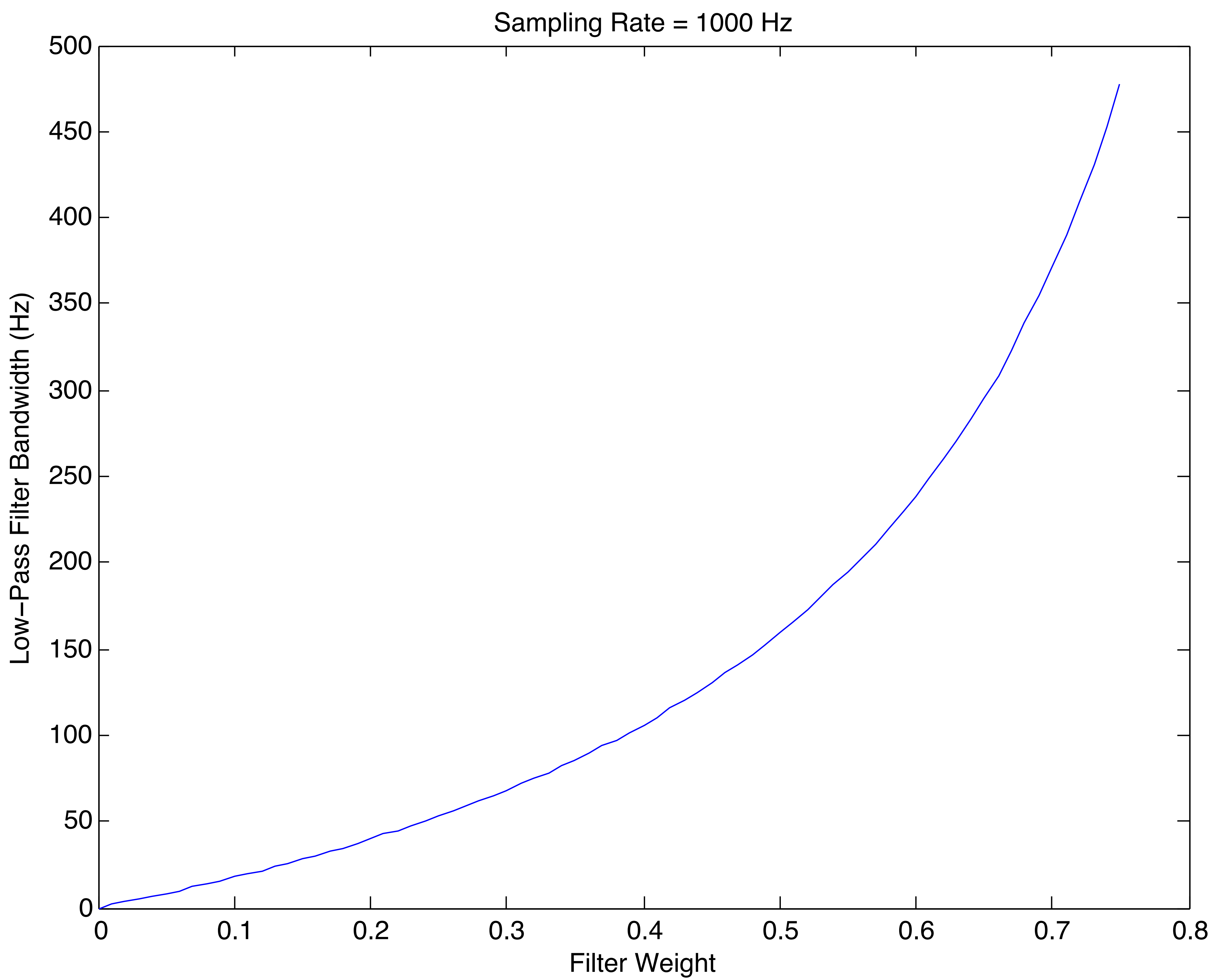
$$\lambda = \frac{w}{T(1-w)}$$

$$\lambda = \frac{w}{T(1-w)}$$

$$f = \lambda \cdot \frac{1\ \text{cycle}}{2\pi\ \text{rad}} = \frac{w}{T(1-w)} \cdot \frac{1\ \text{cycle}}{2\pi\ \text{rad}}$$

$$\lambda = \frac{w}{T(1-w)}$$

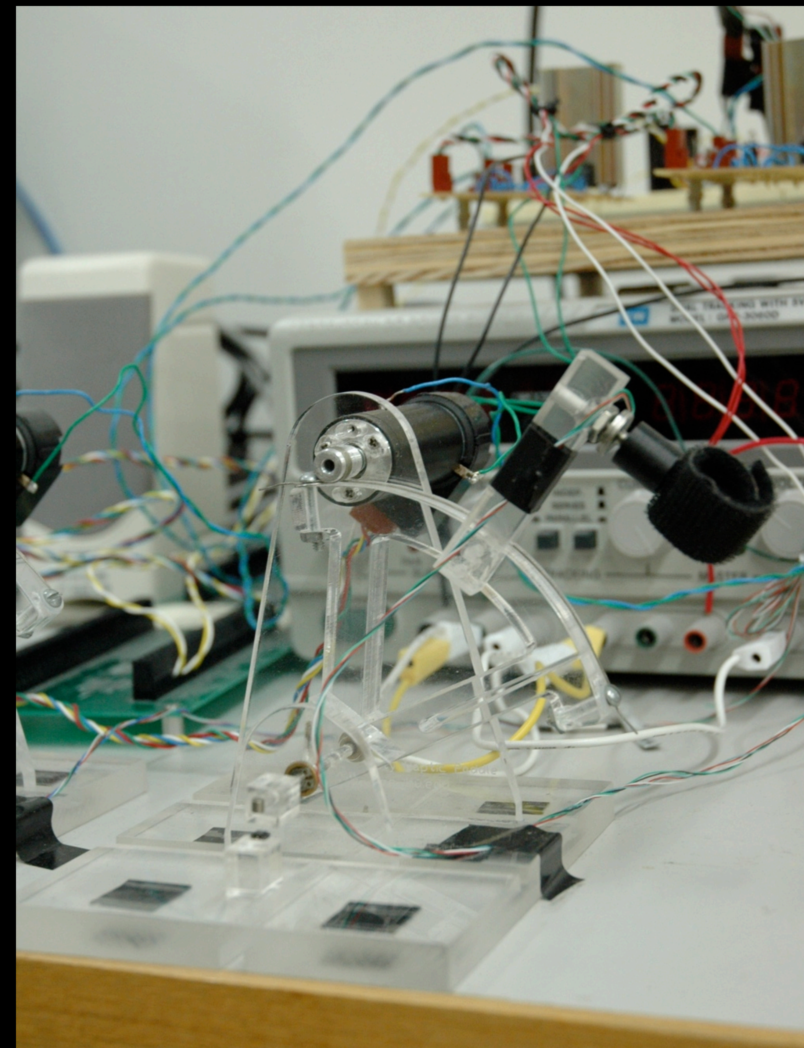
$$f = \lambda \cdot \frac{1 \text{ cycle}}{2\pi \text{ rad}} = \frac{w}{T(1-w)} \cdot \frac{1 \text{ cycle}}{2\pi \text{ rad}}$$

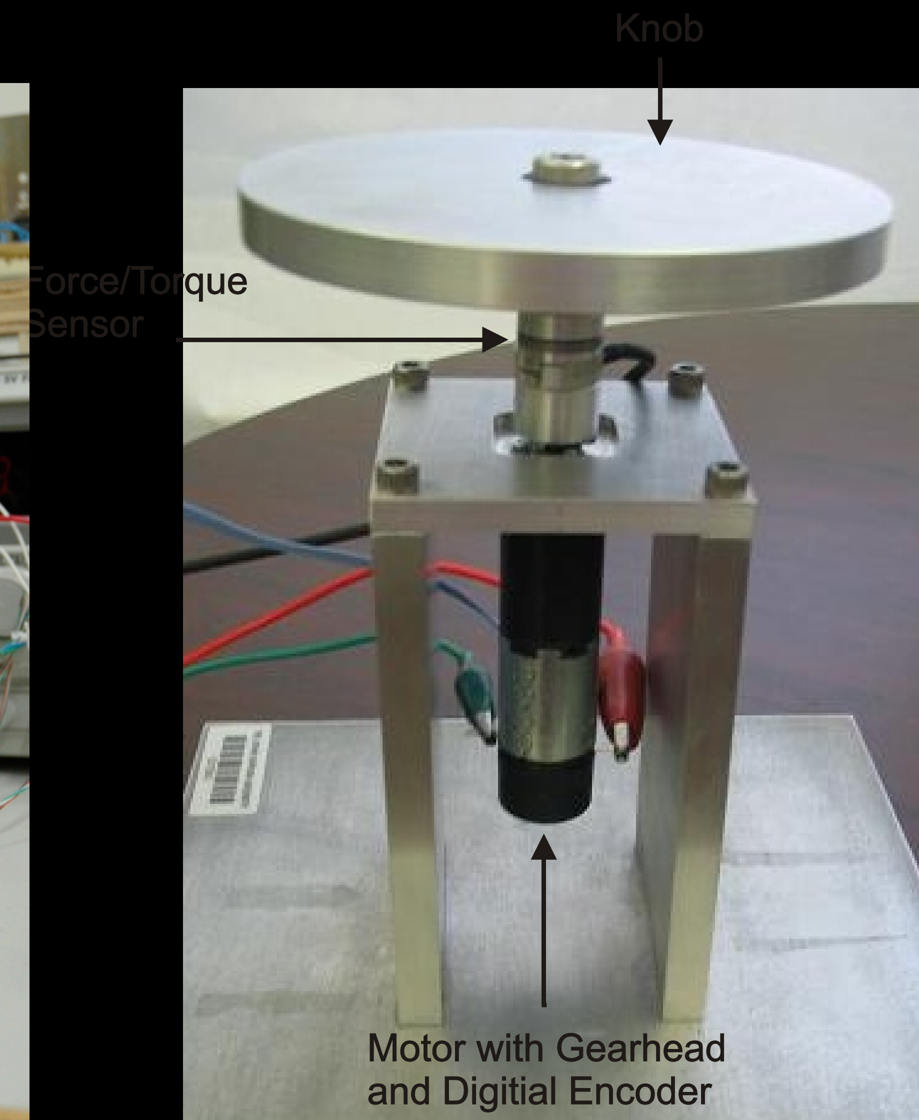


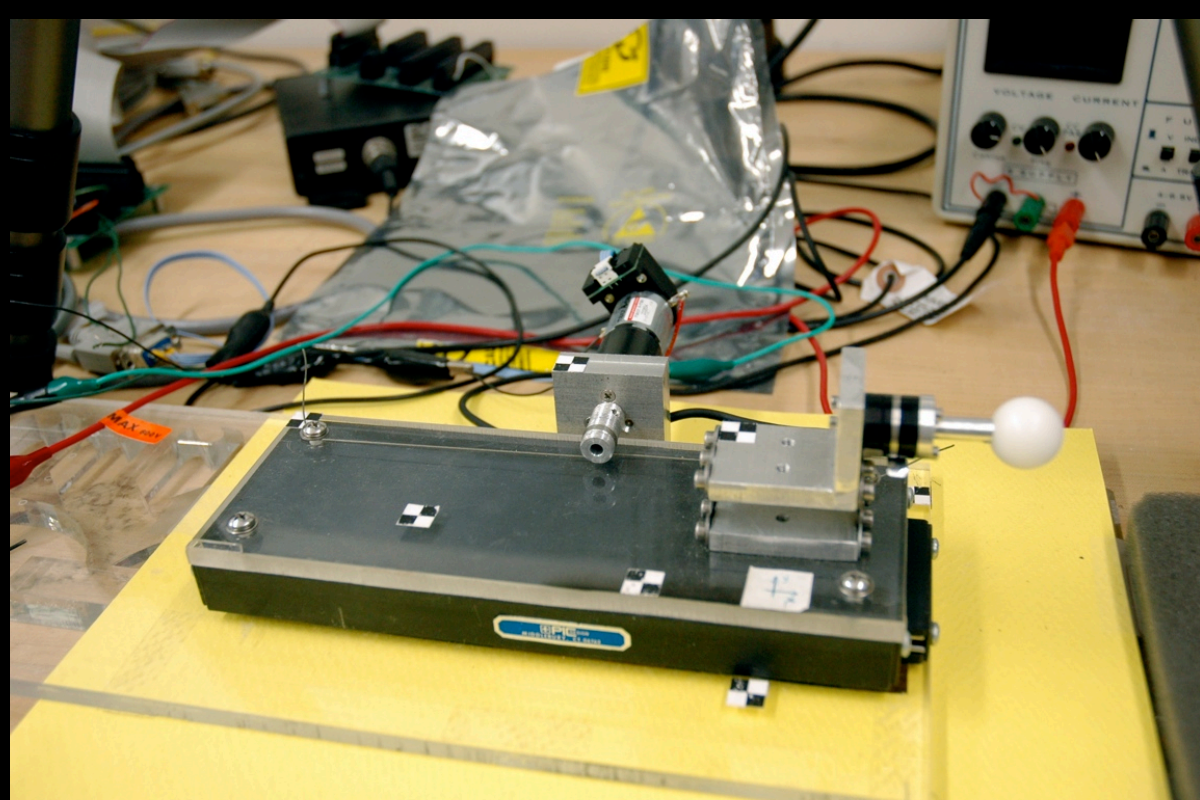
Calculating Velocity

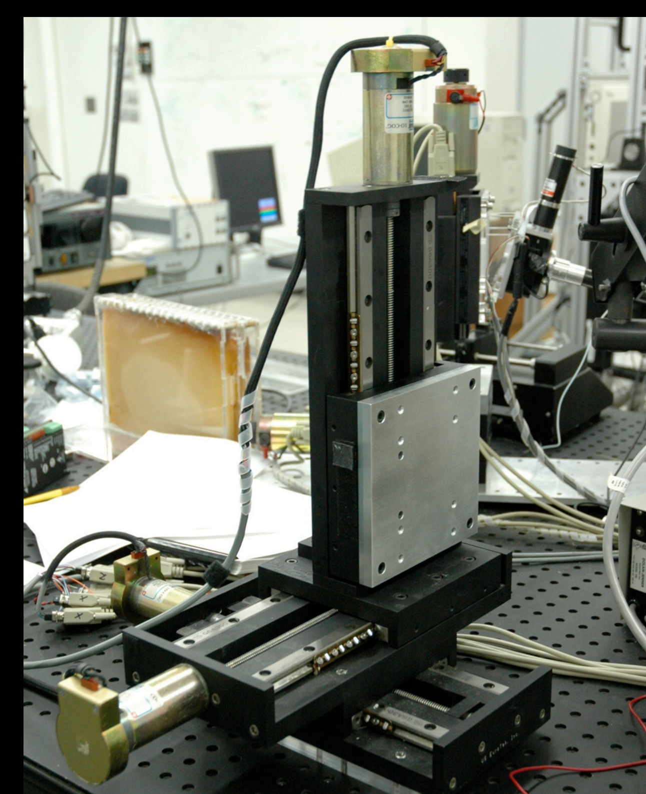
```
/*******************************************************************************
 hapticCallback()
 Main callback that sets the force that the user will feel.  It gets the current
 position and velocity of the device.
 This is what you want to edit to change the system's haptic feedback.
*******************************************************************************/
HDCallbackCode HDCALLBACK hapticCallback(void *data)
{
  // Local variables.
  hduVector3Dd position;
  hduVector3Dd velocity;
  hduVector3Dd force;
  hduVector3Dd extraForce;
  hduVector3Dd proxyPosition;
  HDint currentButtonState;
  HDint lastButtonState;
  double stiffness = 0.25;  // Units are newtons per millimeter.

  // Local variables for custom velocity calculation.
  static bool firstTime = true;
  static hduVector3Dd lastPosition; // mm
  hduVector3Dd rawVelocity; // mm/s
  static hduVector3Dd filteredVelocity(0.0, 0.0, 0.0);  // mm/s
  float filterWeight = 0.03;
  float dampingCoefficient = 0.01; // N/(mm/s)
  hduVector3Dd dampingForce; // N
```

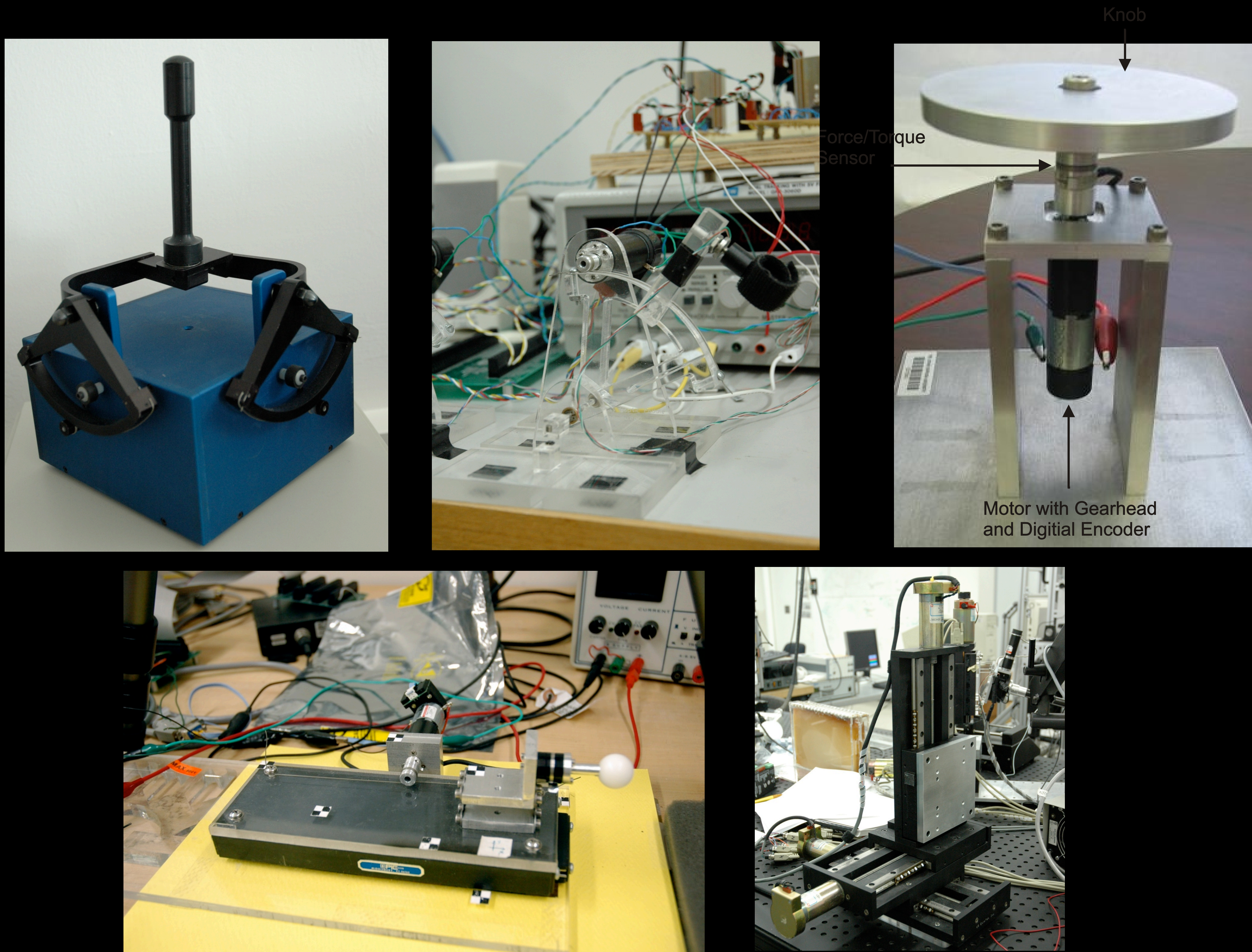
A sample custom haptic device

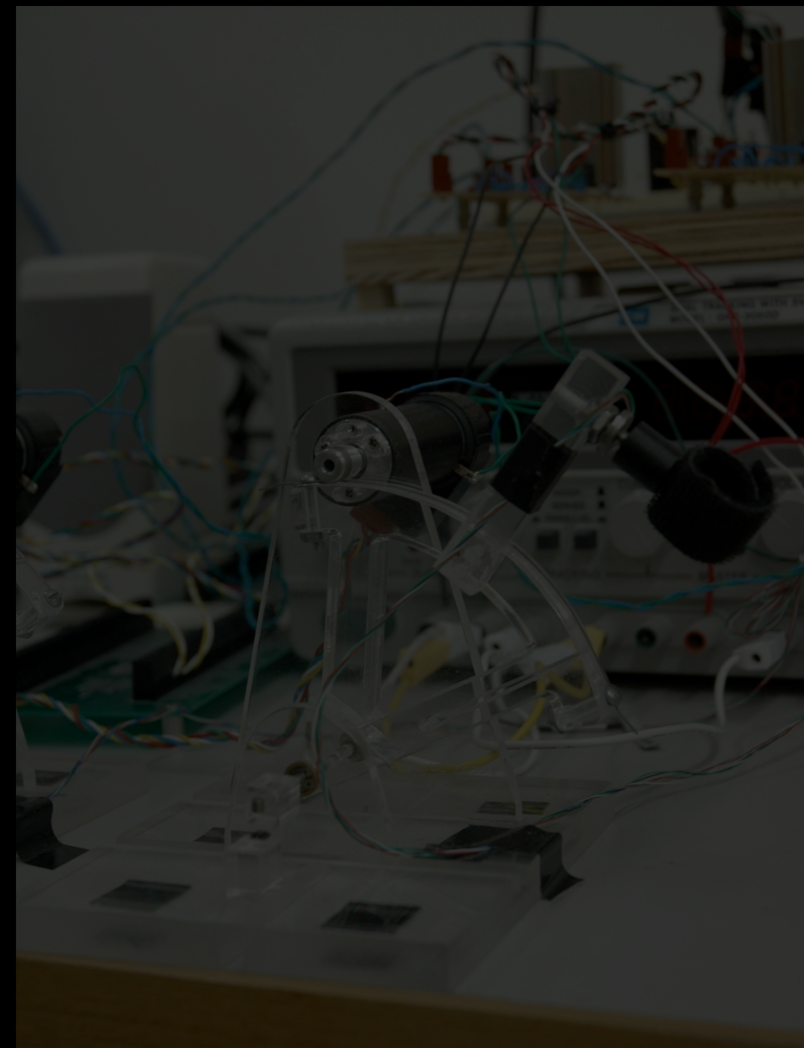


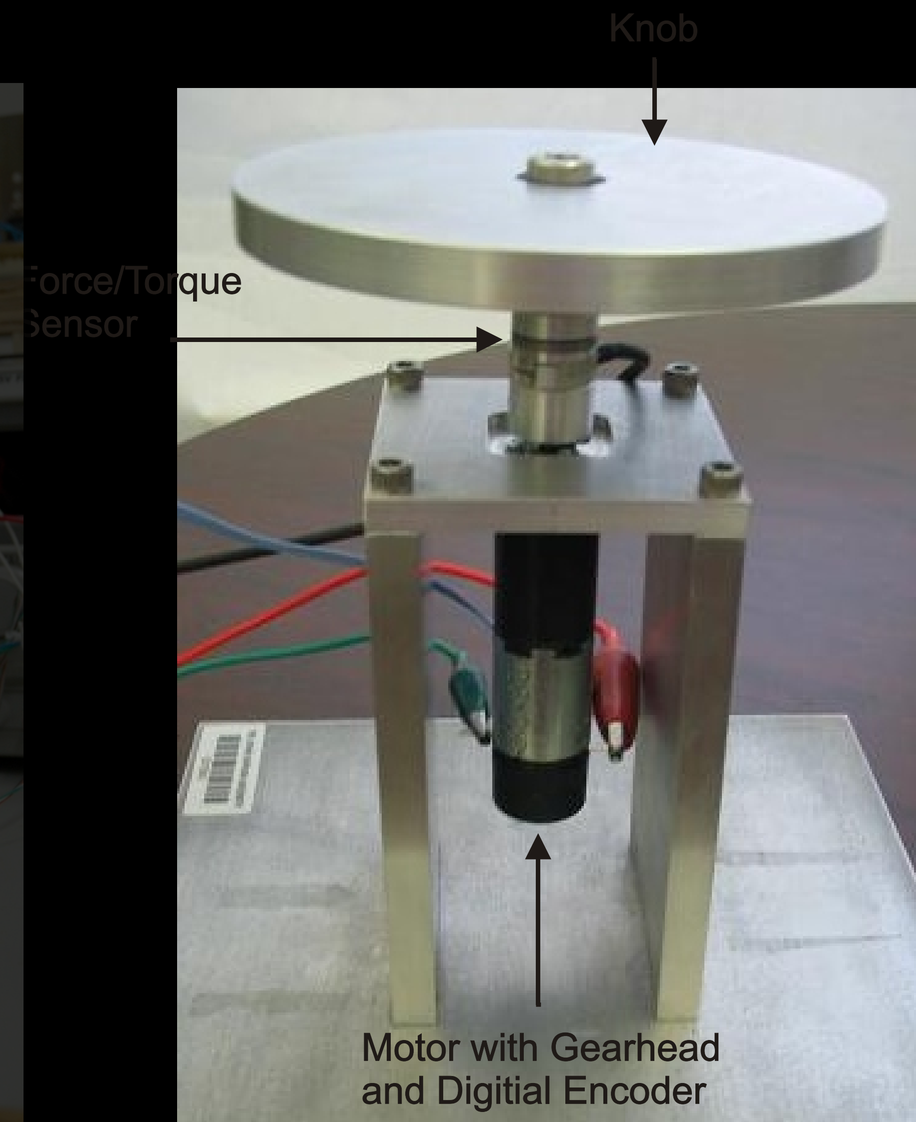

Knob
Force/Torque
Sensor
Motor with Gearhead
and Digitial Encoder

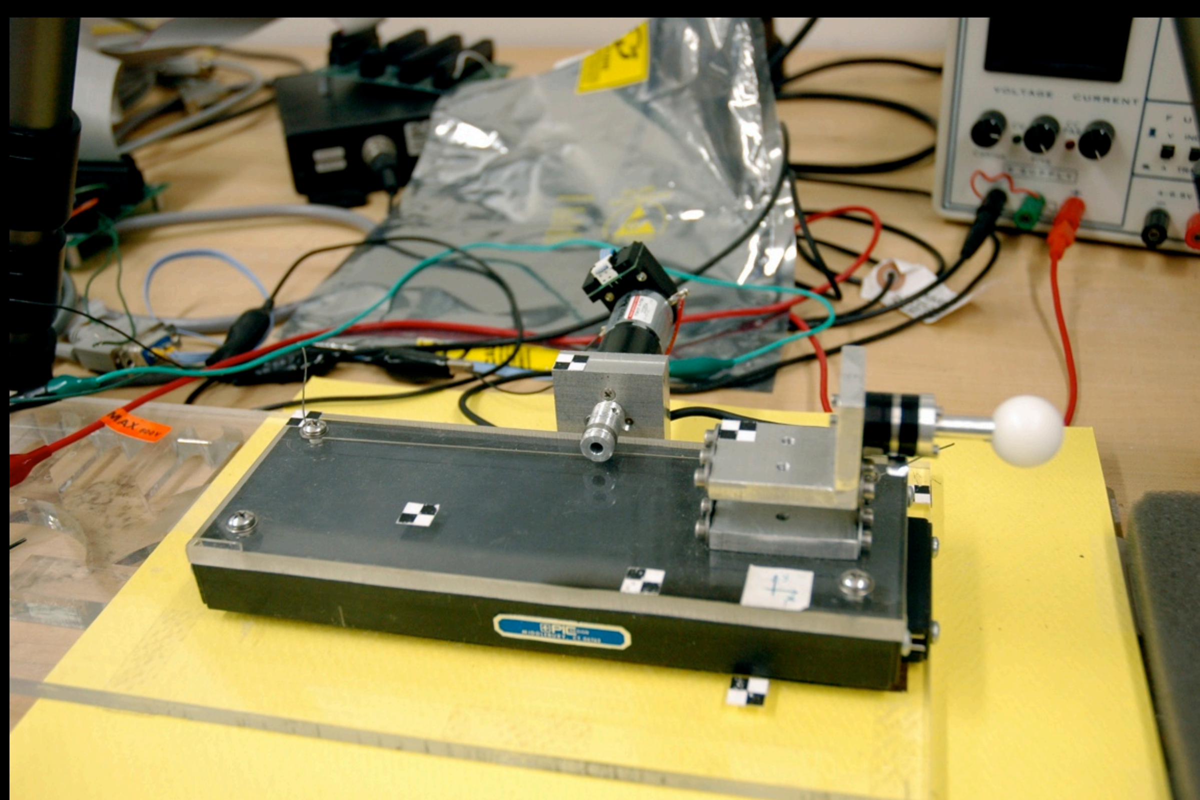


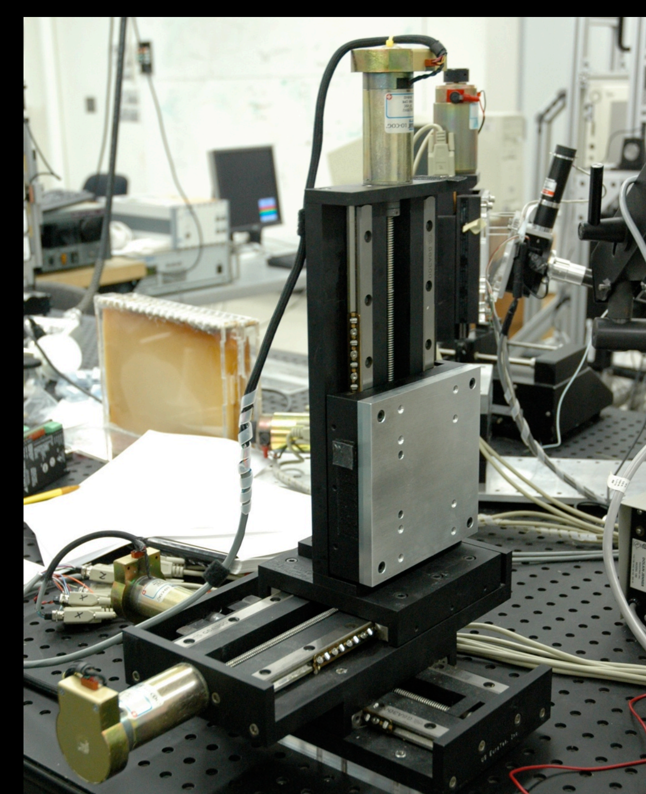


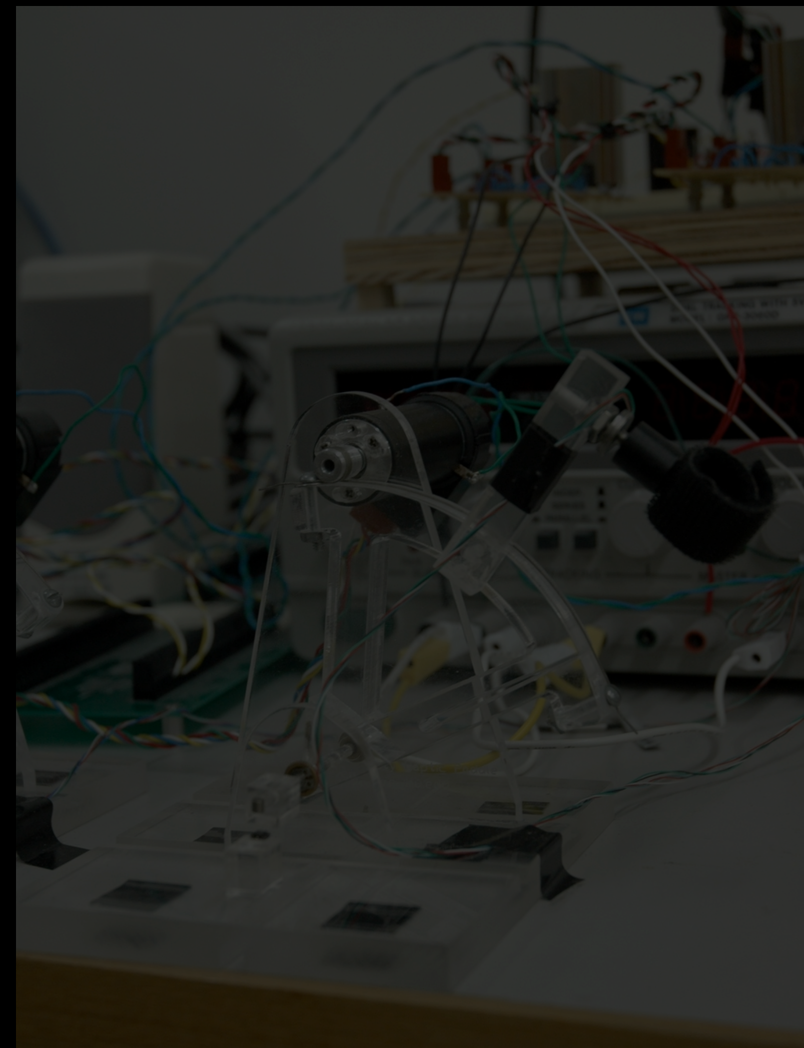

Knob
Force/Torque Sensor
Motor with Gearhead and Digitial Encoder

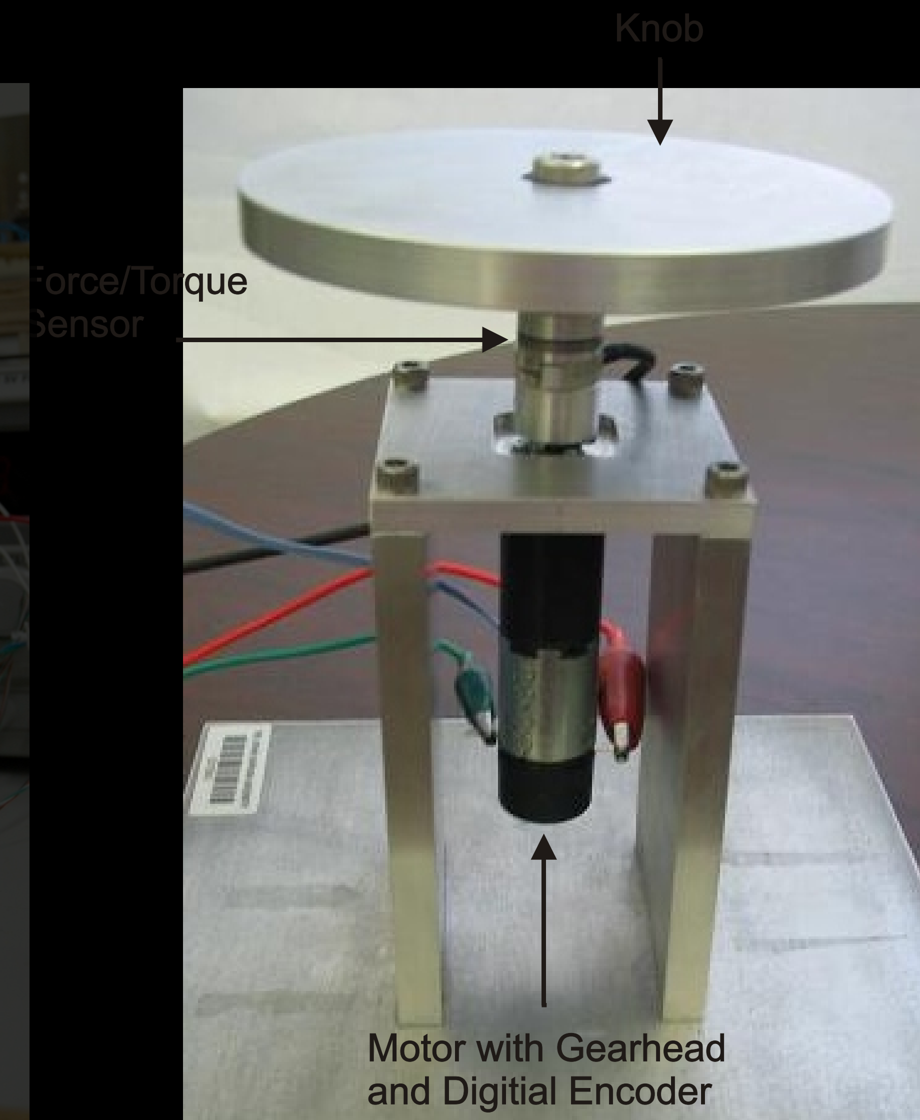


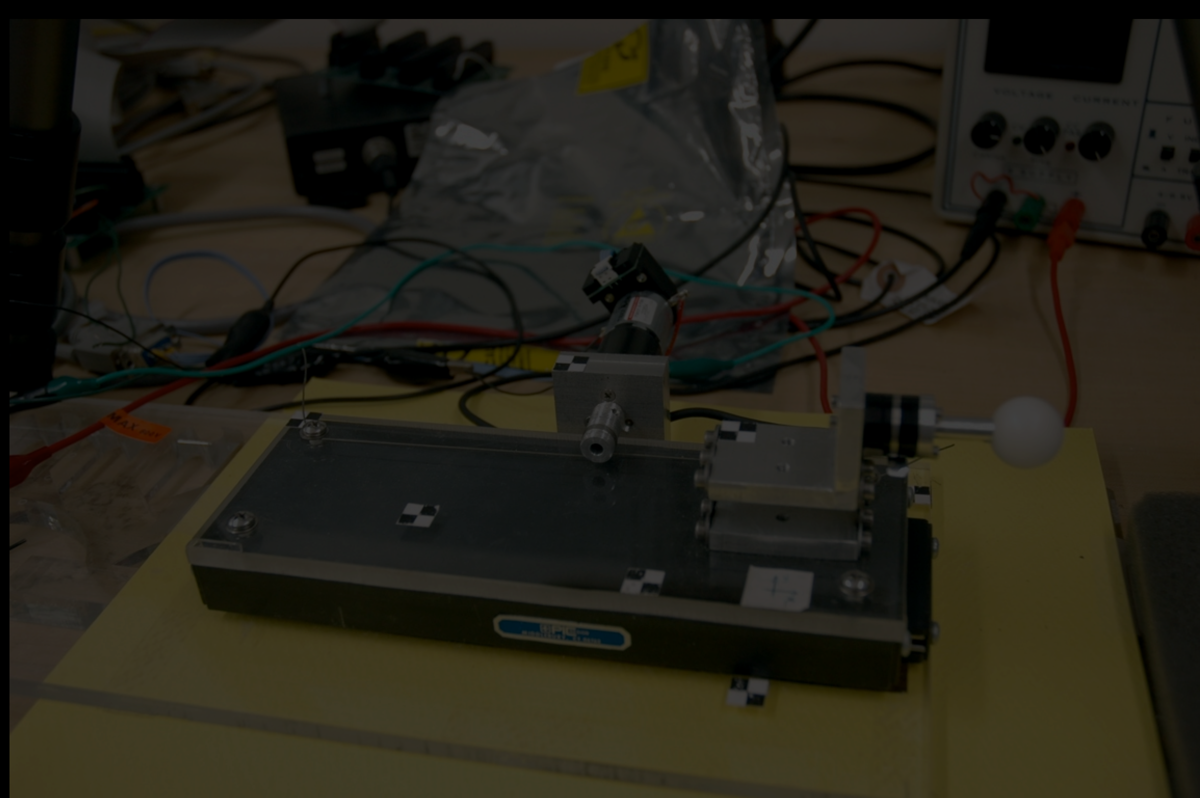

Knob
orce/Torque
ensor
Motor with Gearhead
and Digitial Encoder

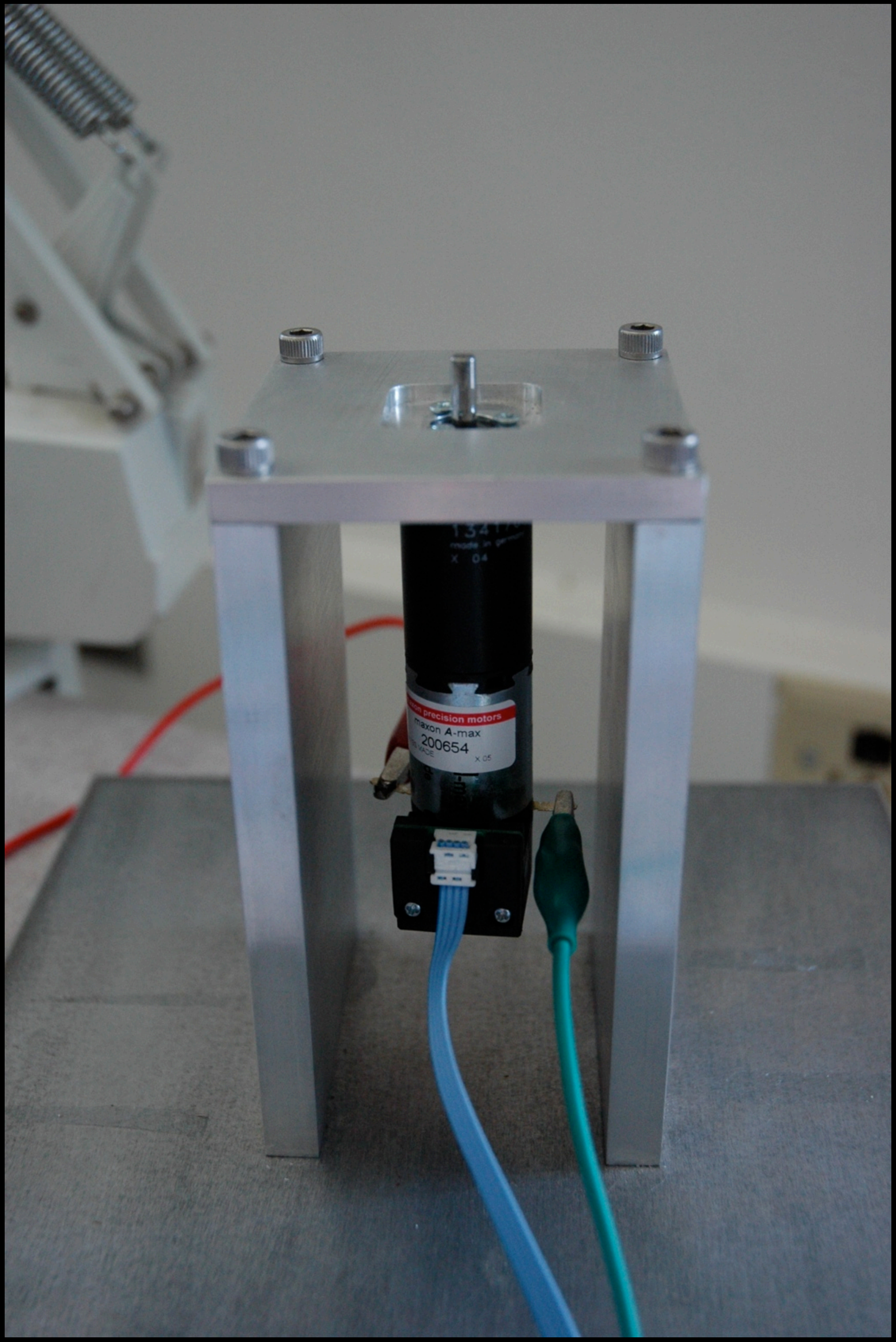


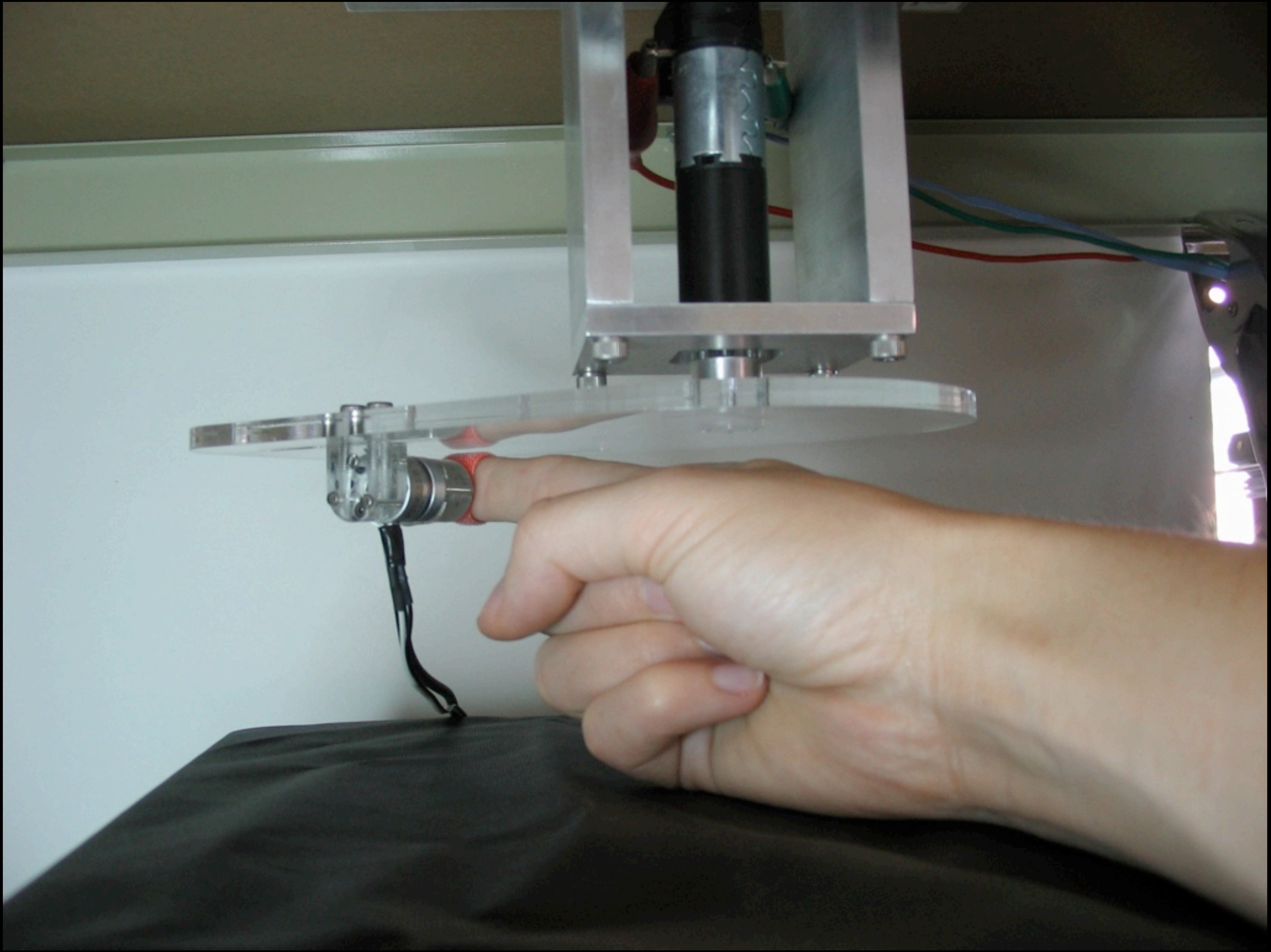





Knob
Force/Torque Sensor
Motor with Gearhead and Digitial Encoder

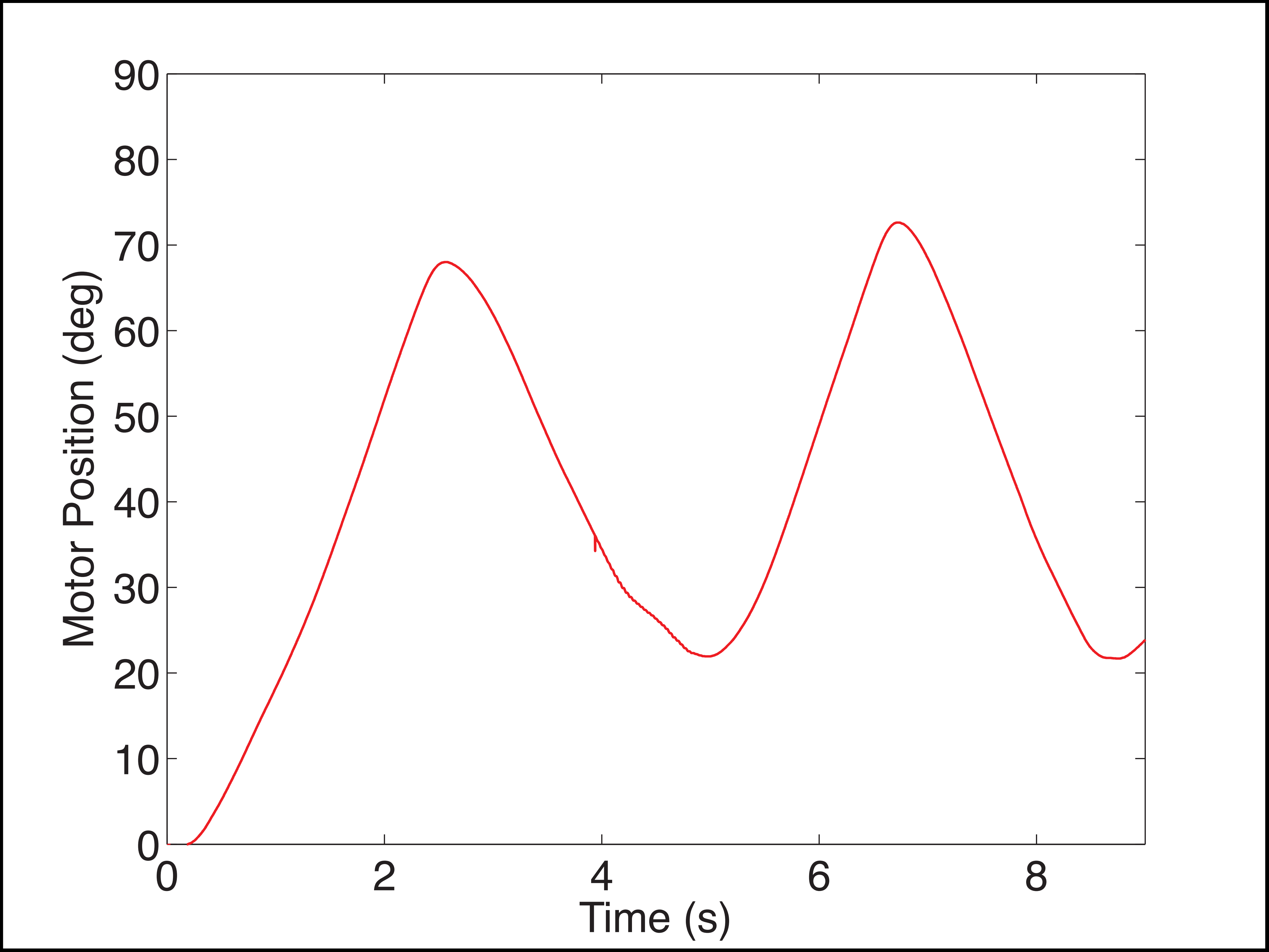



precision motors
maxon A-max
200654

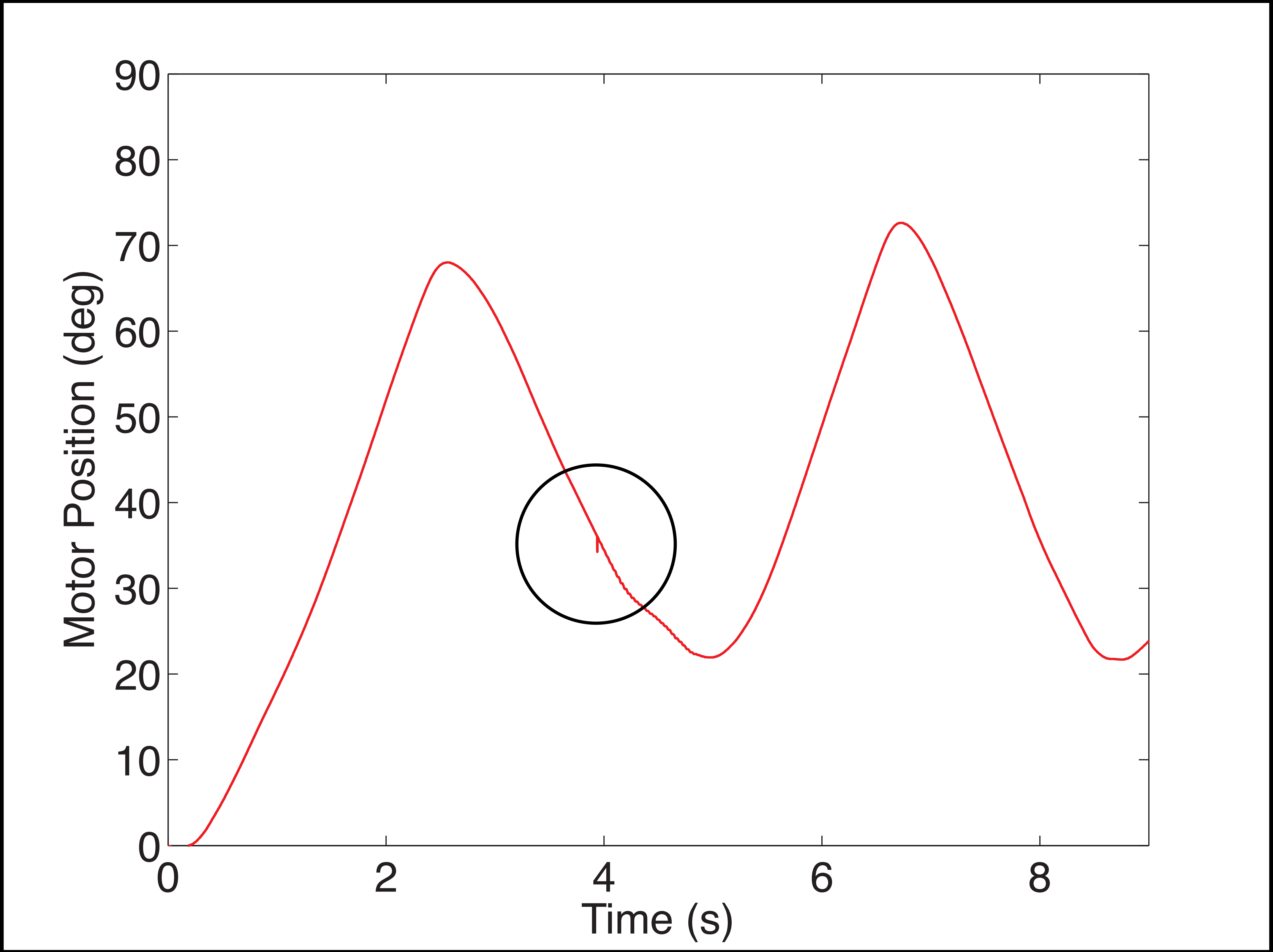


$$\tau_m = k_p(\theta_d - \theta_m) + k_d(\omega_d - \omega_m)$$

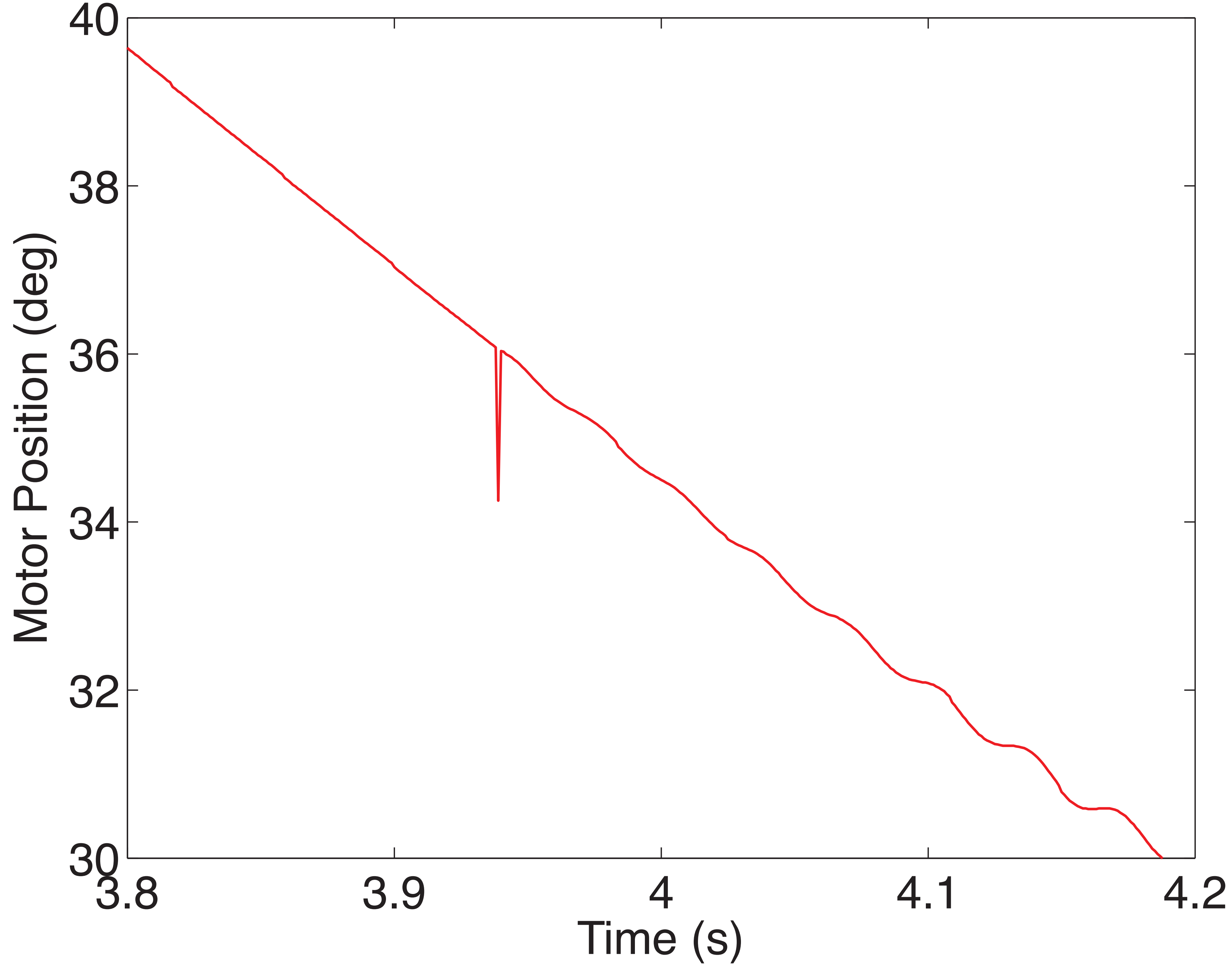
$$\tau_m = k_p(\theta_d - \underline{\theta_m}) + k_d(\omega_d - \underline{\omega_m})$$



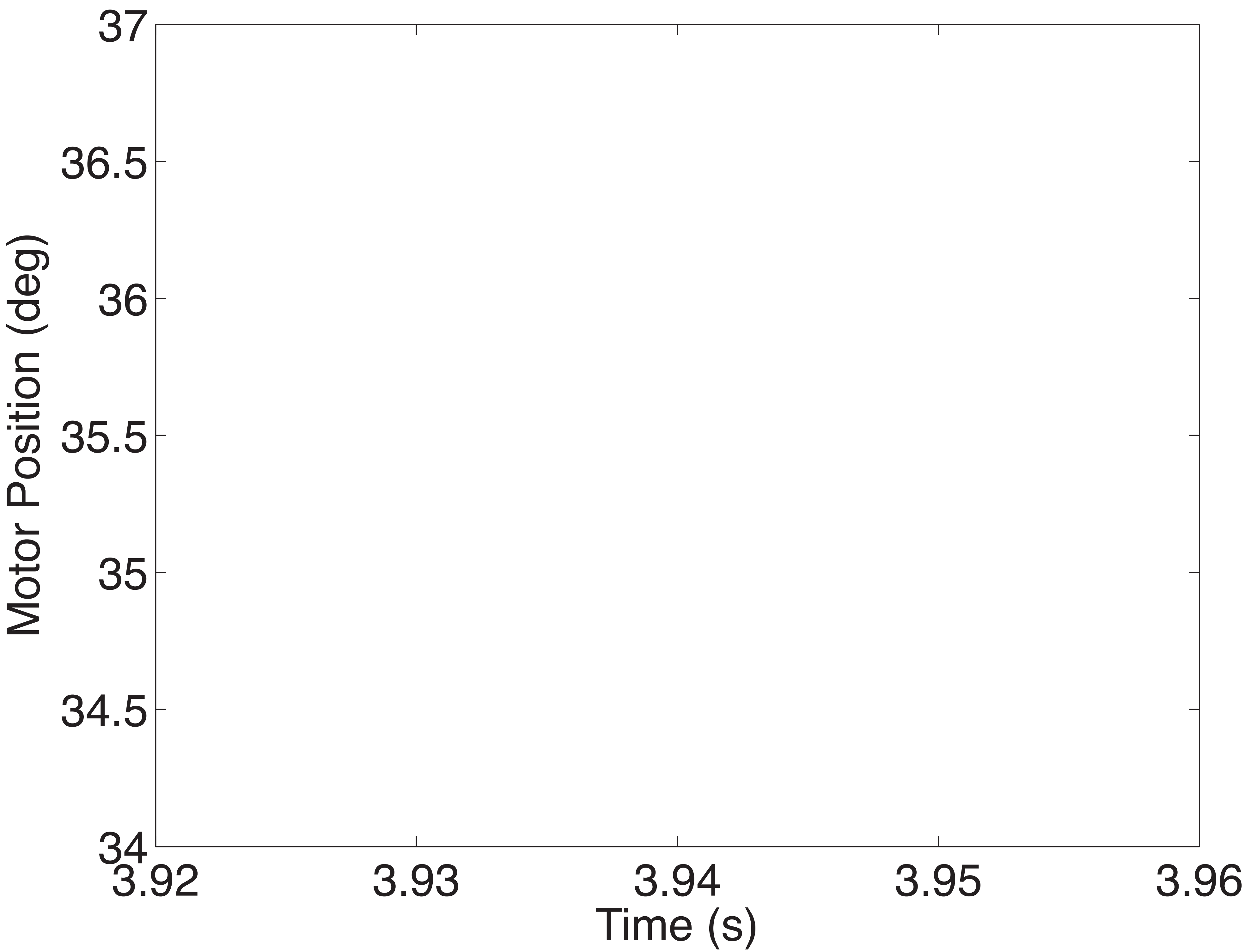
Motor Position (deg)
Time (s)
0
10
20
30
40
50
60
70
80
90
0
2
4
6
8

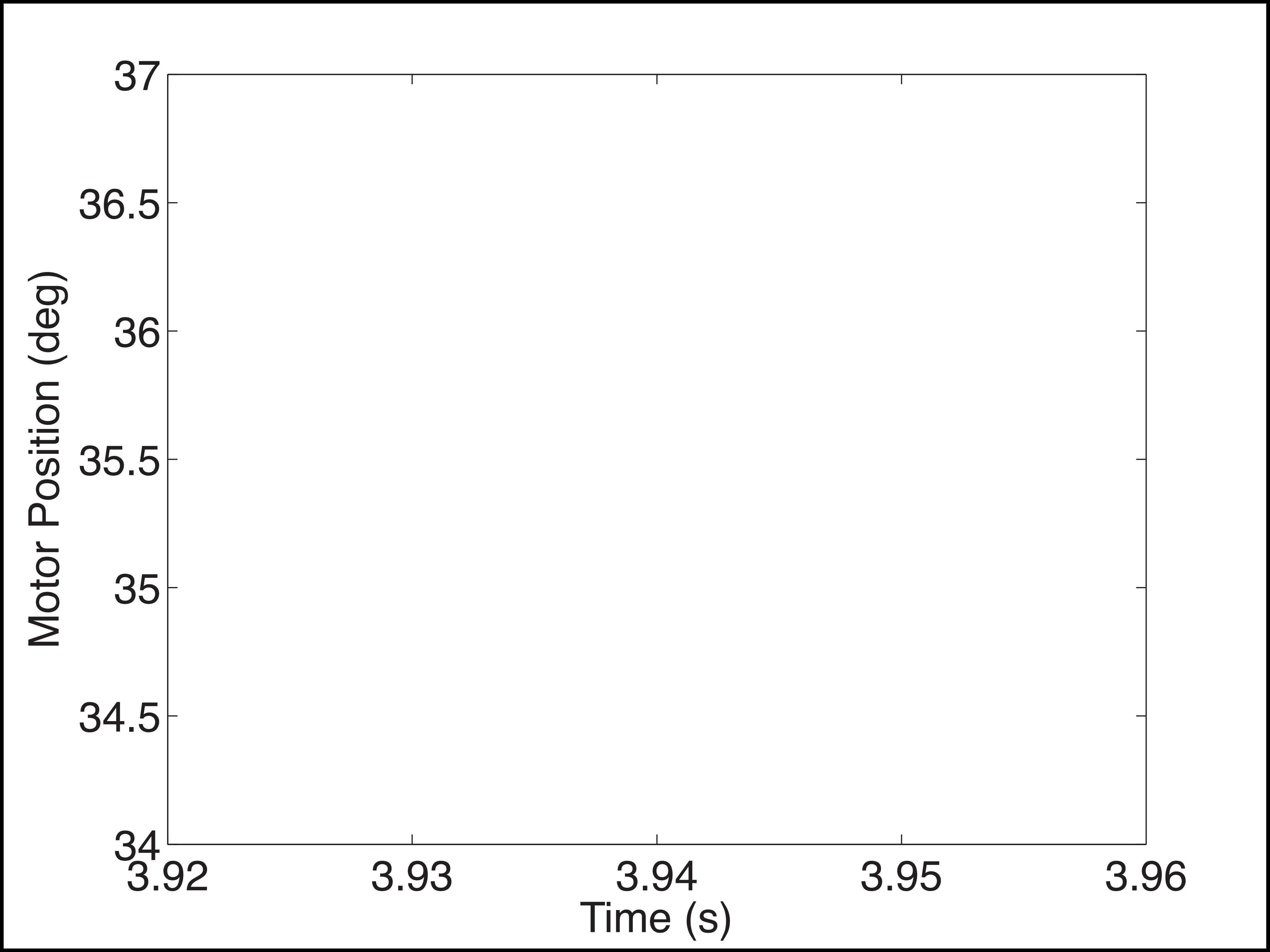


Motor Position (deg)
Time (s)
0
10
20
30
40
50
60
70
80
90
0
2
4
6
8

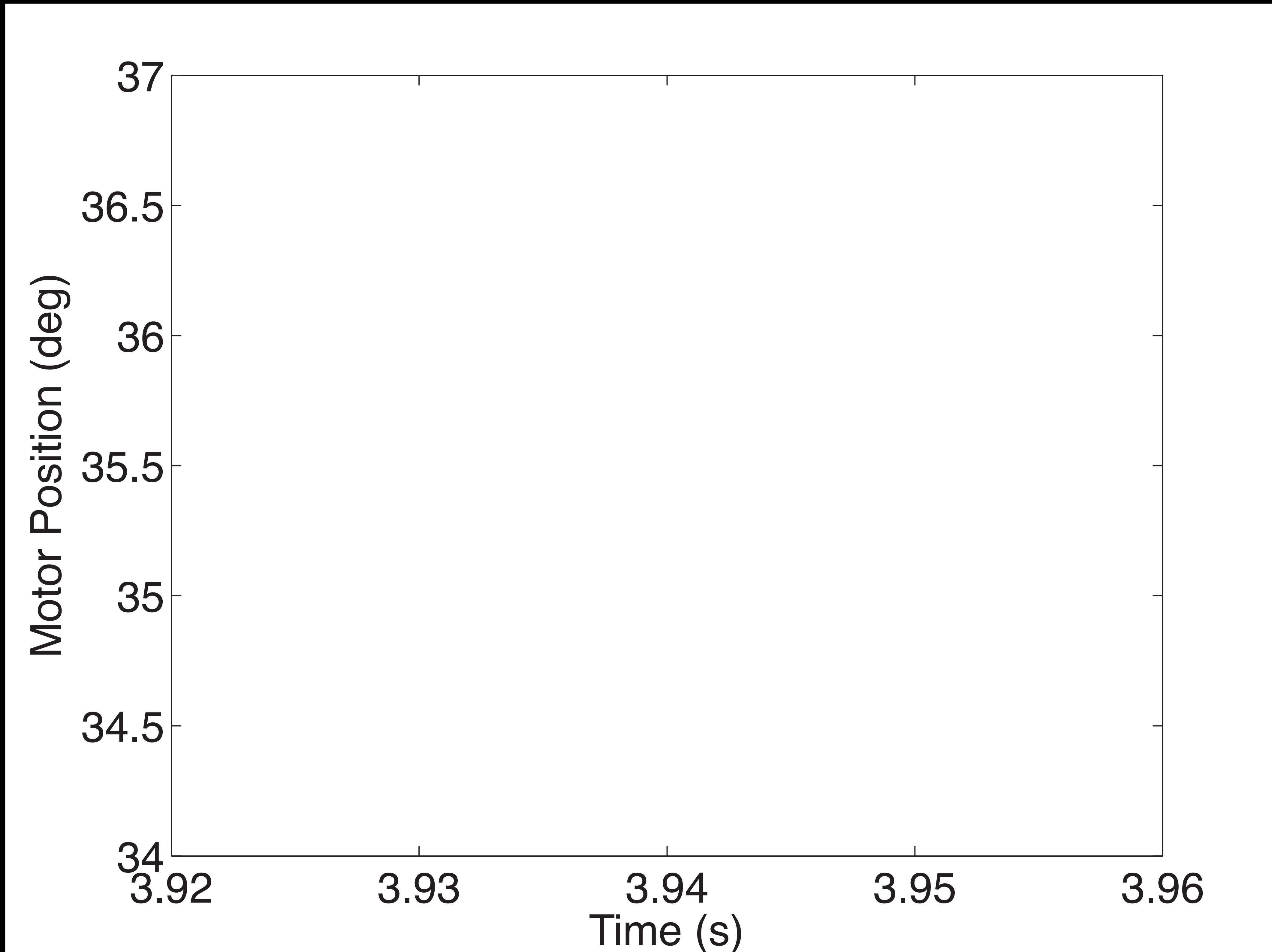


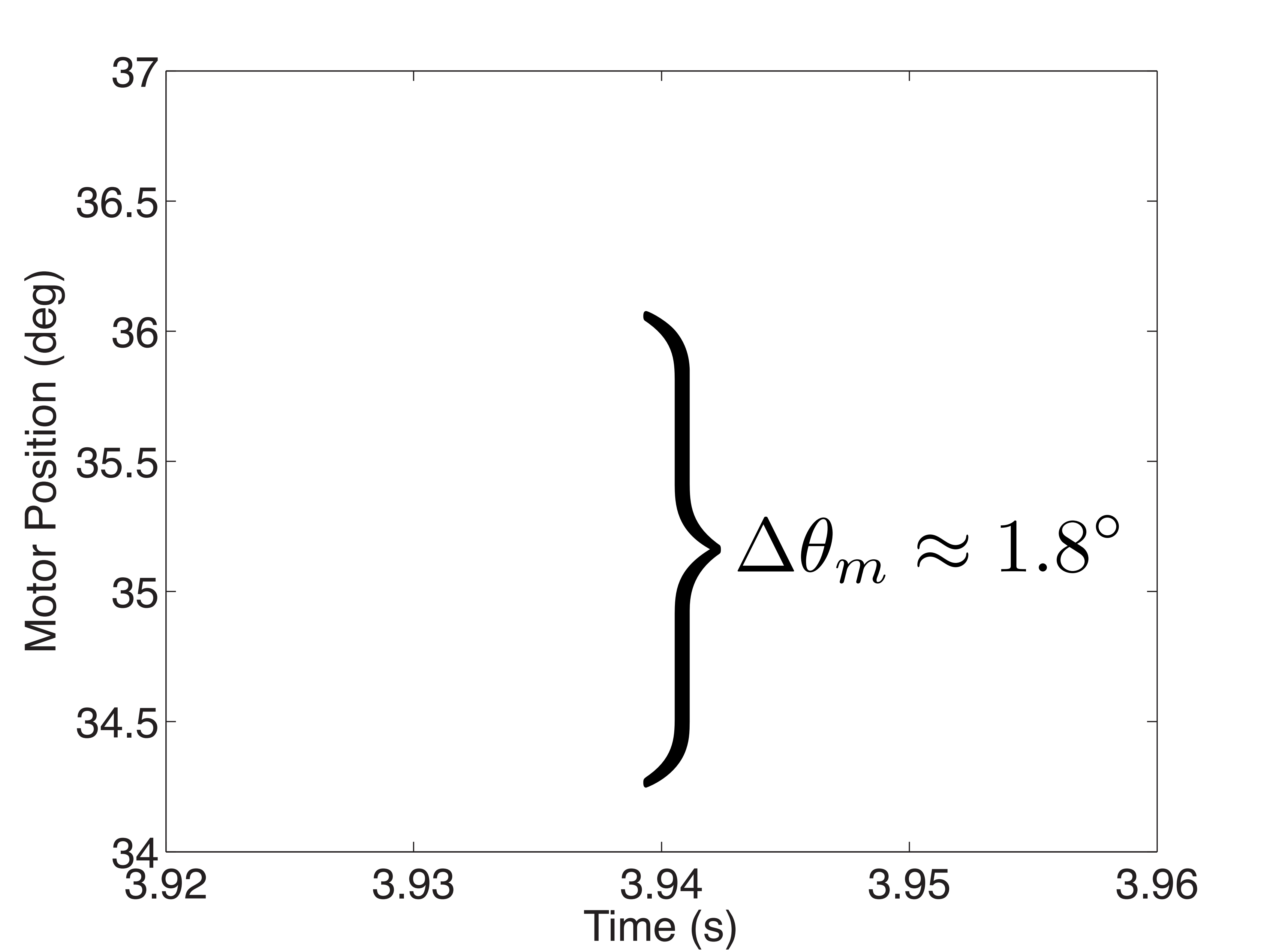
40
38
36
34
32
30
Motor Position (deg)
3.8
3.9
4
4.1
4.2
Time (s)


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)



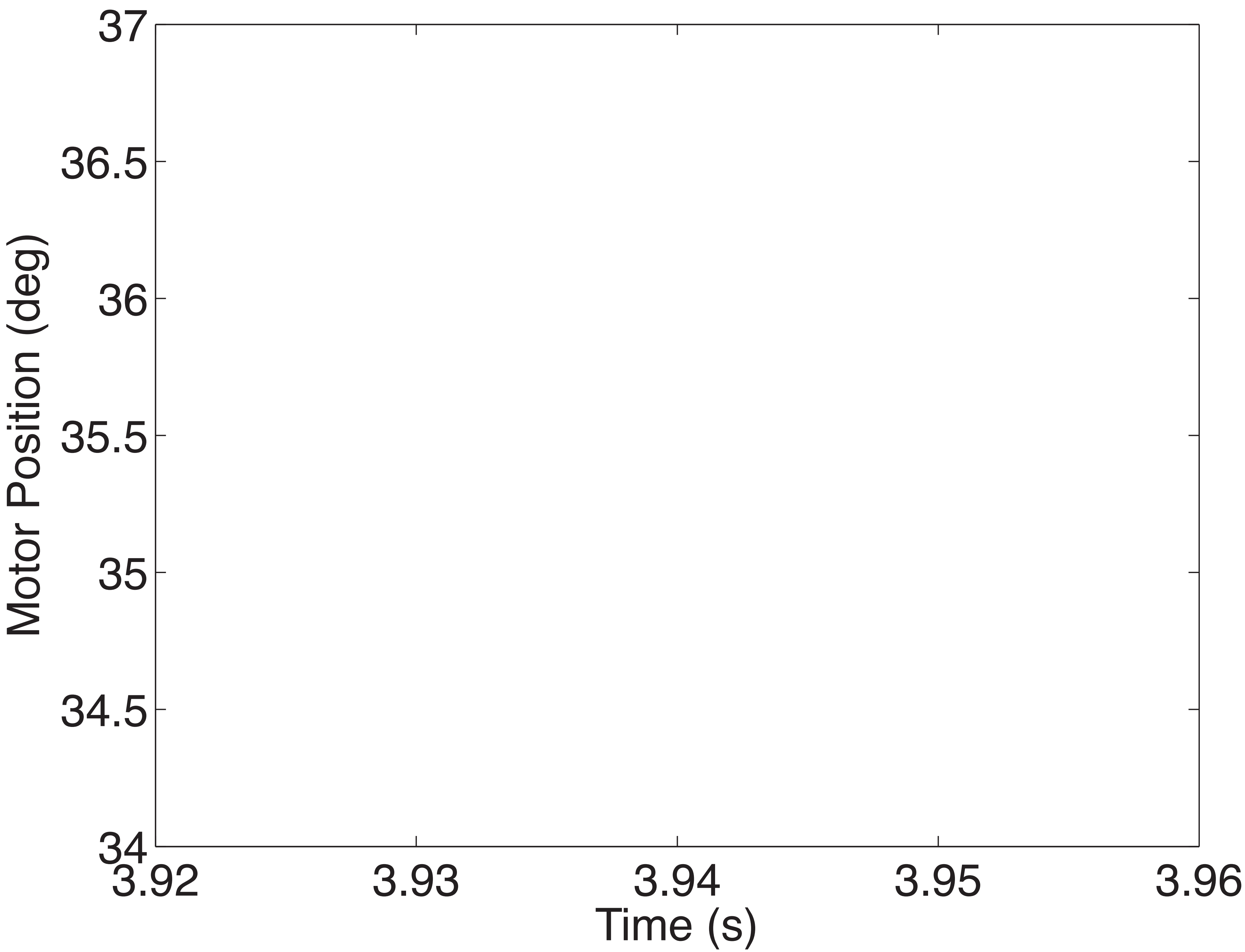
37
36.5
36
35.5
35
34.5
34
Motor Position (deg)
3.92
3.93
3.94
3.95
3.96
Time (s)


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)

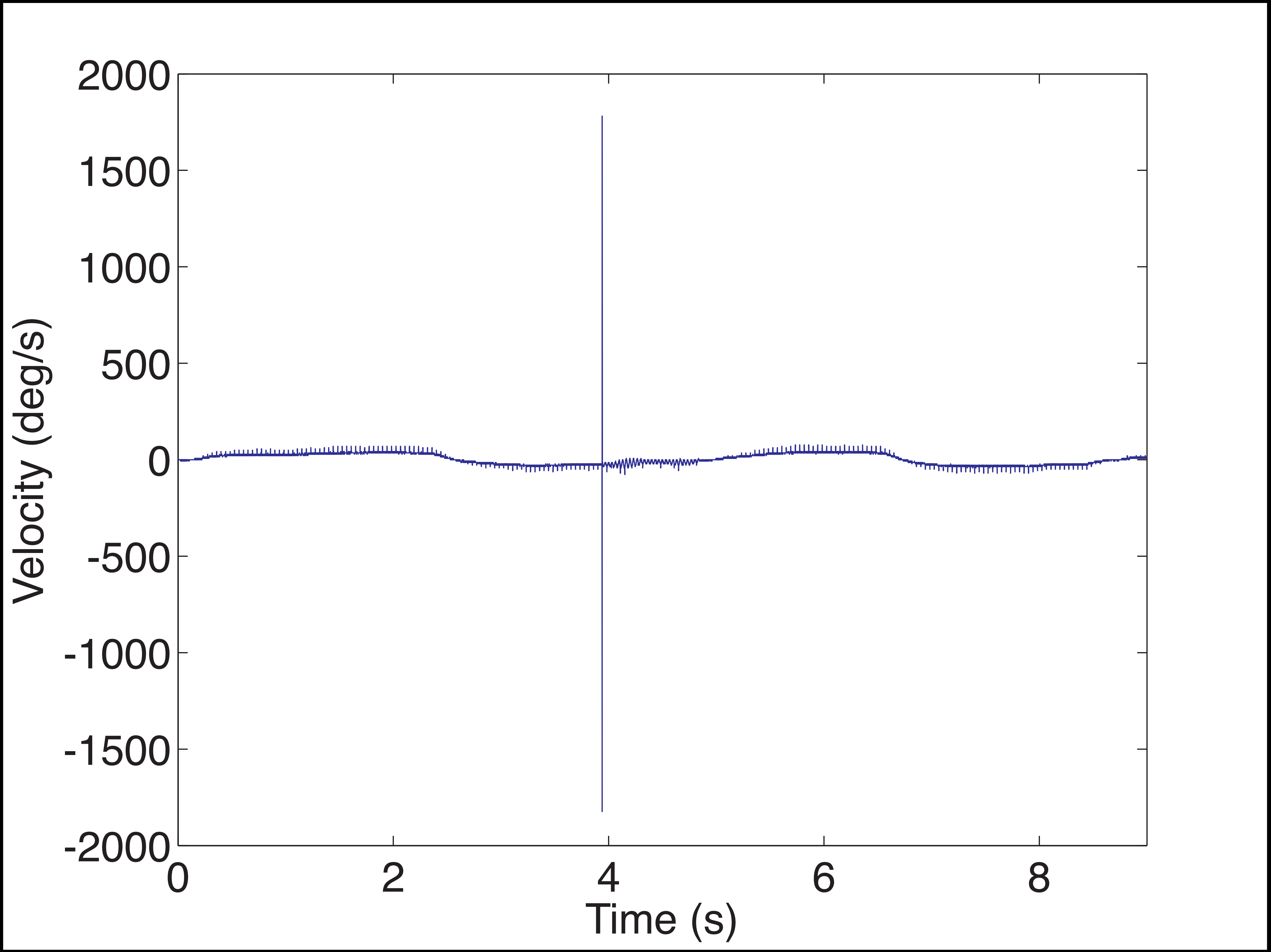


Motor Position (deg)
37
36.5
36
35.5
35
34.5
34
3.92
3.93
3.94
3.95
3.96
Time (s)
$\Delta\theta_m \approx 1.8^\circ$

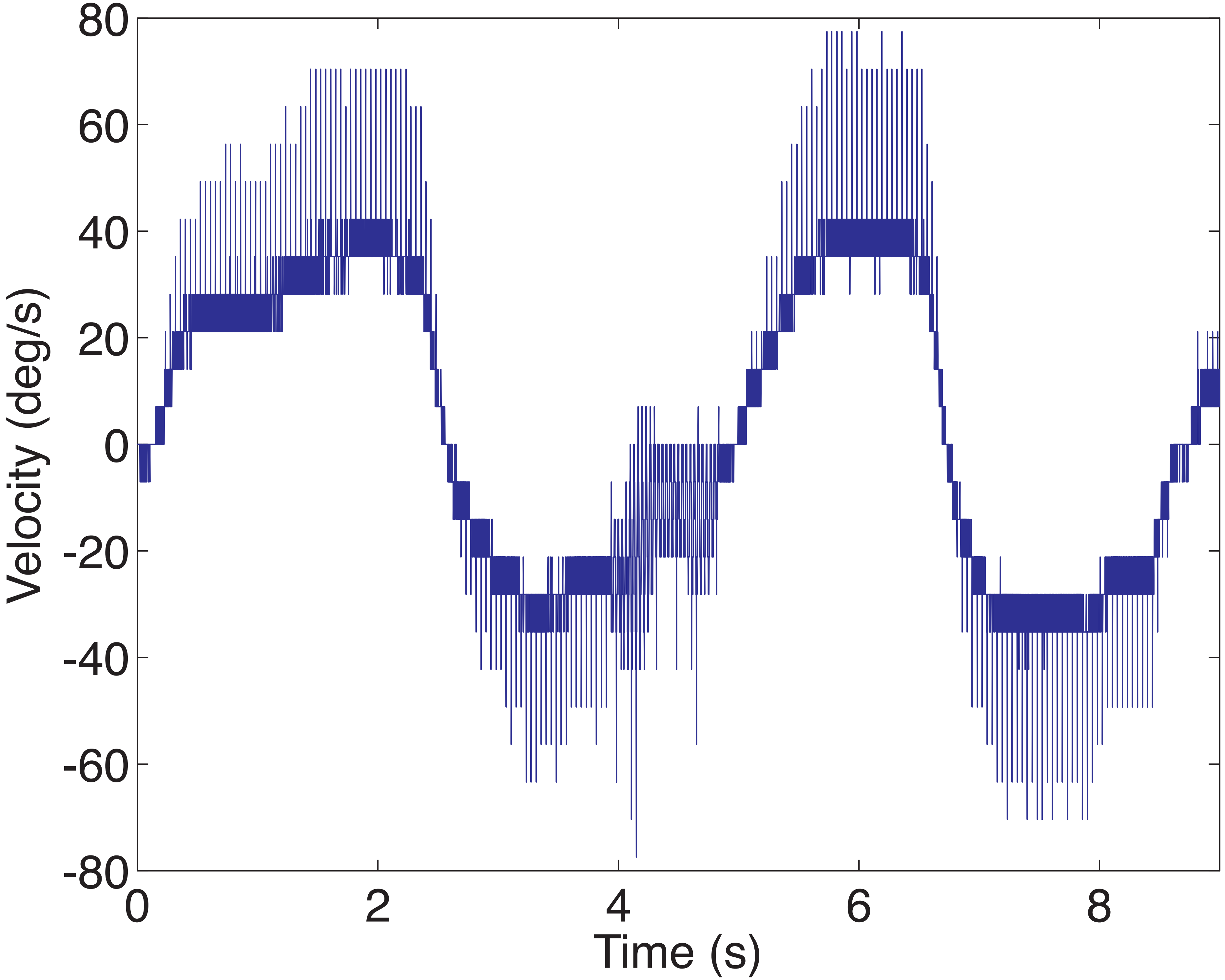
$$\Delta\theta_m = 1.8^\circ \cdot \frac{51200 \text{ counts}}{360^\circ} = 256 \text{ counts}$$

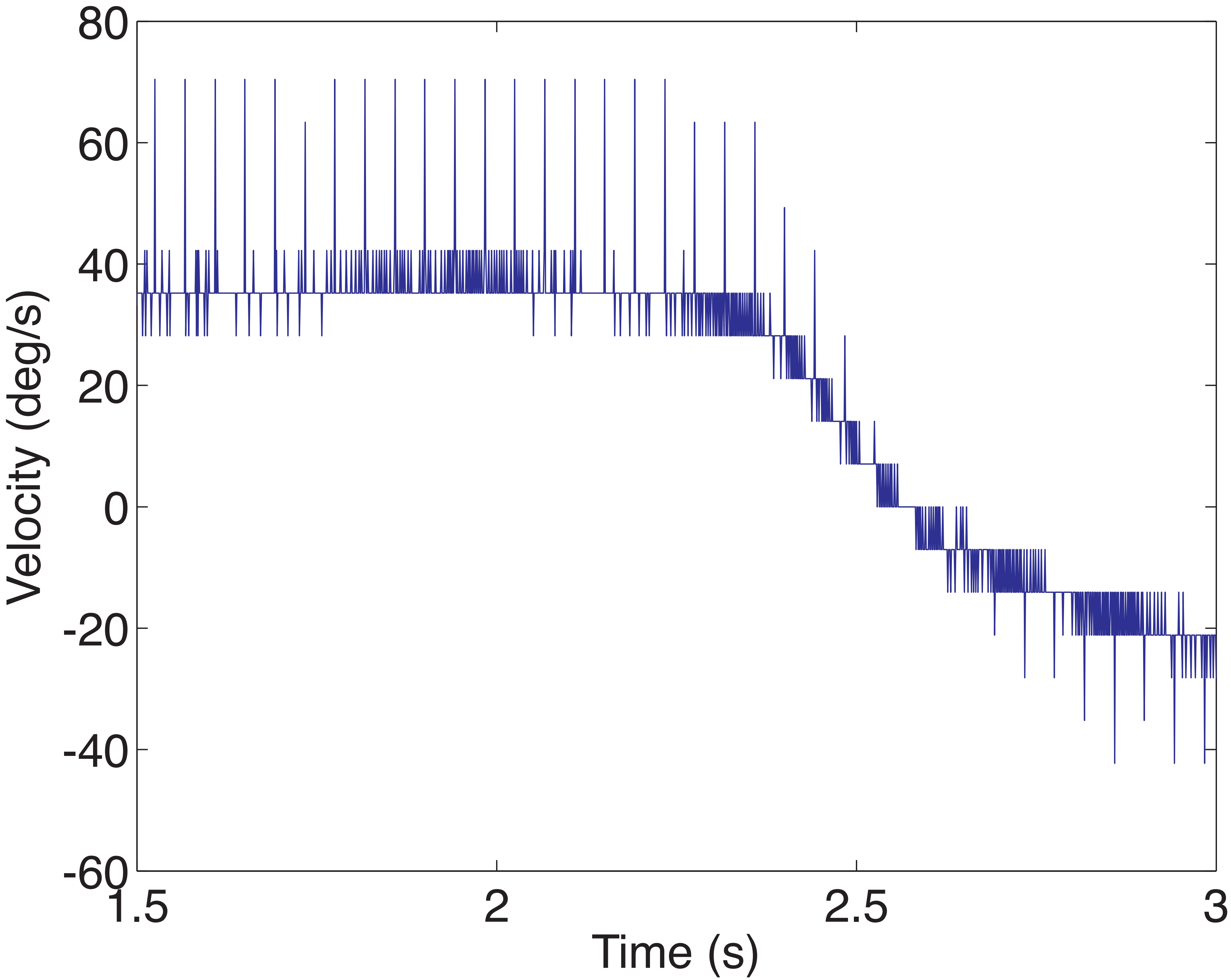


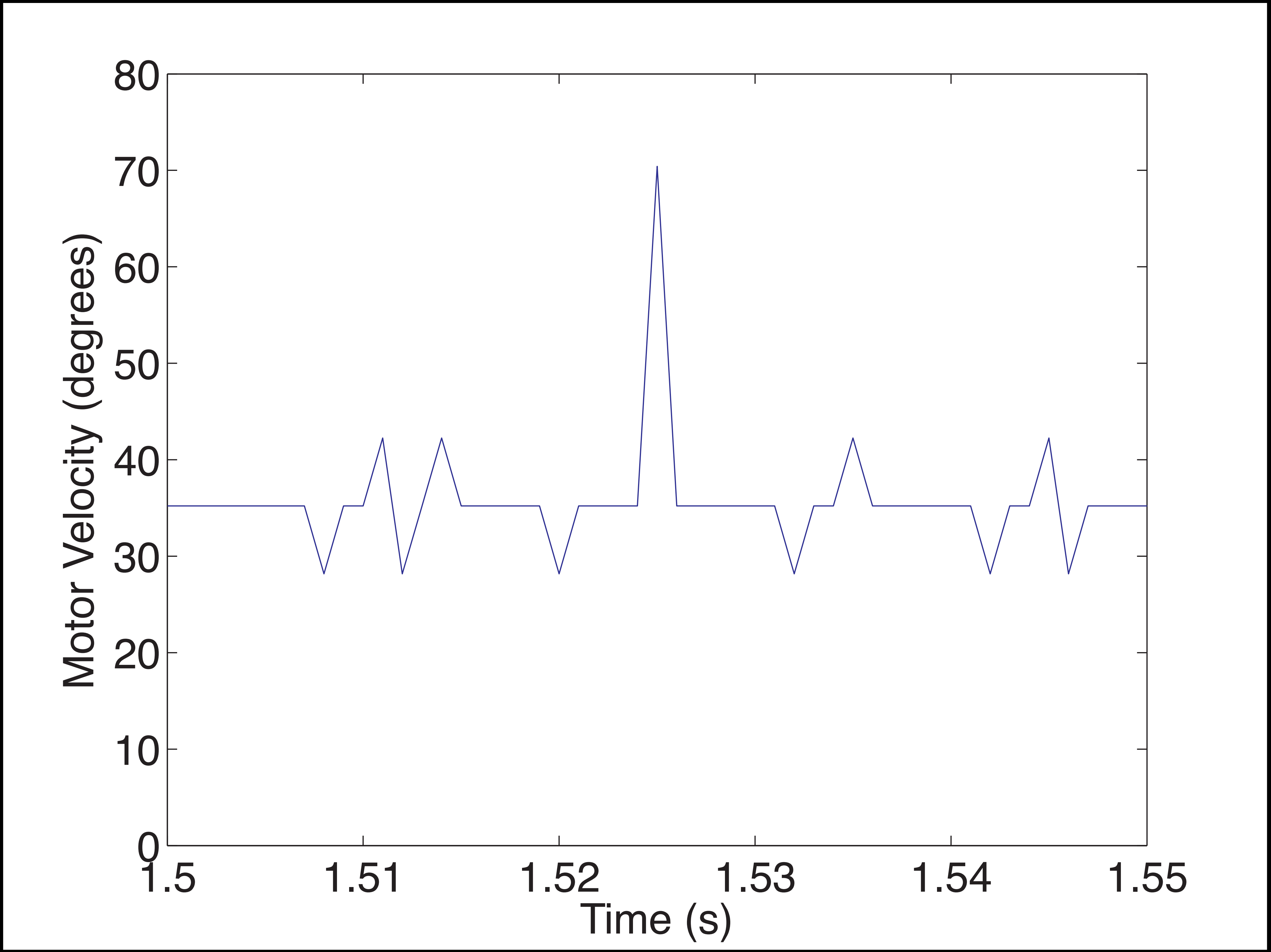
37
36.5
36
35.5
35
34.5
34
Motor Position (deg)
3.92
3.93
3.94
3.95
3.96
Time (s)



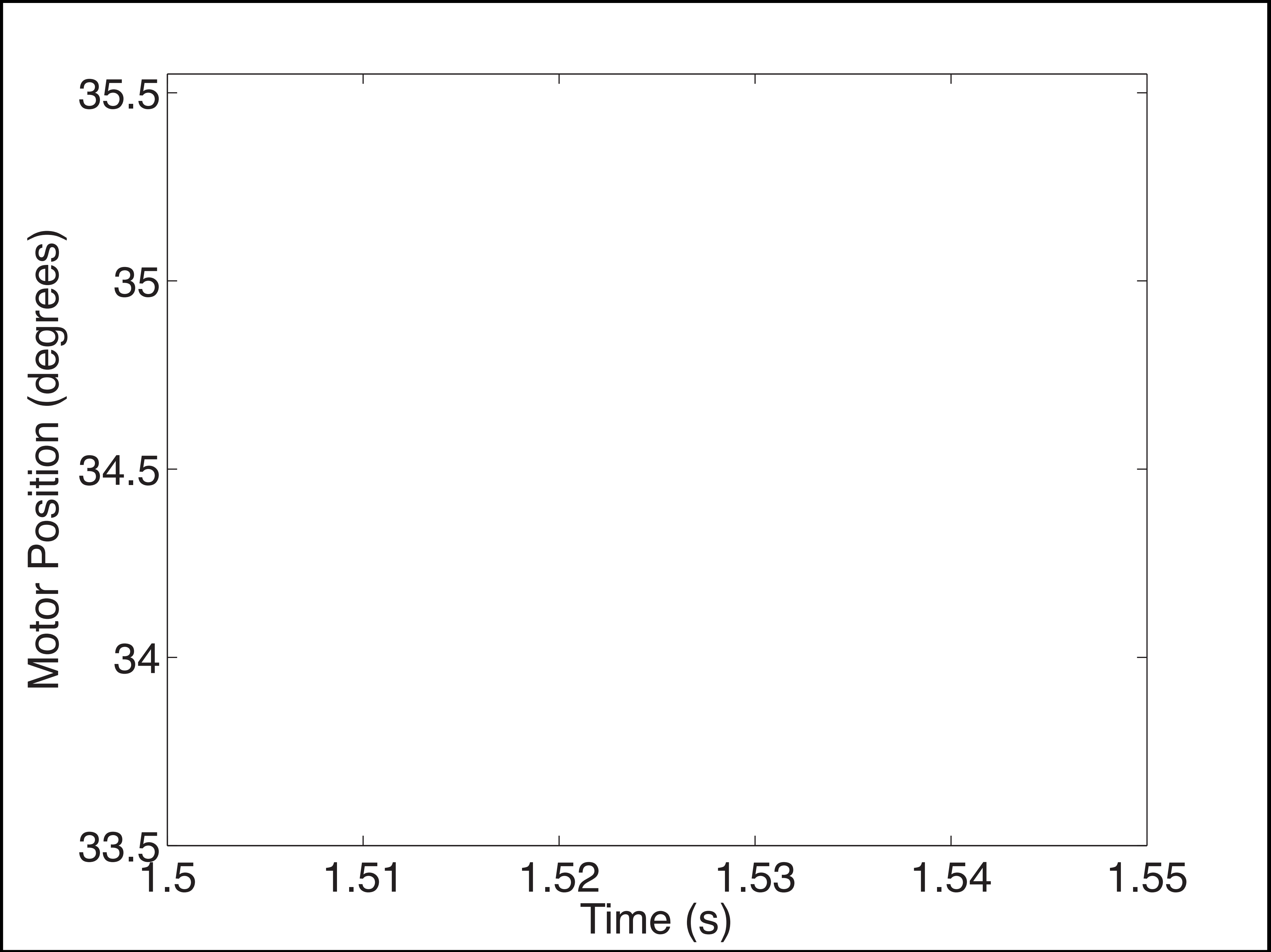
2000
1500
1000
500
0
-500
-1000
-1500
-2000
Velocity (deg/s)
0
2
4
6
8
Time (s)

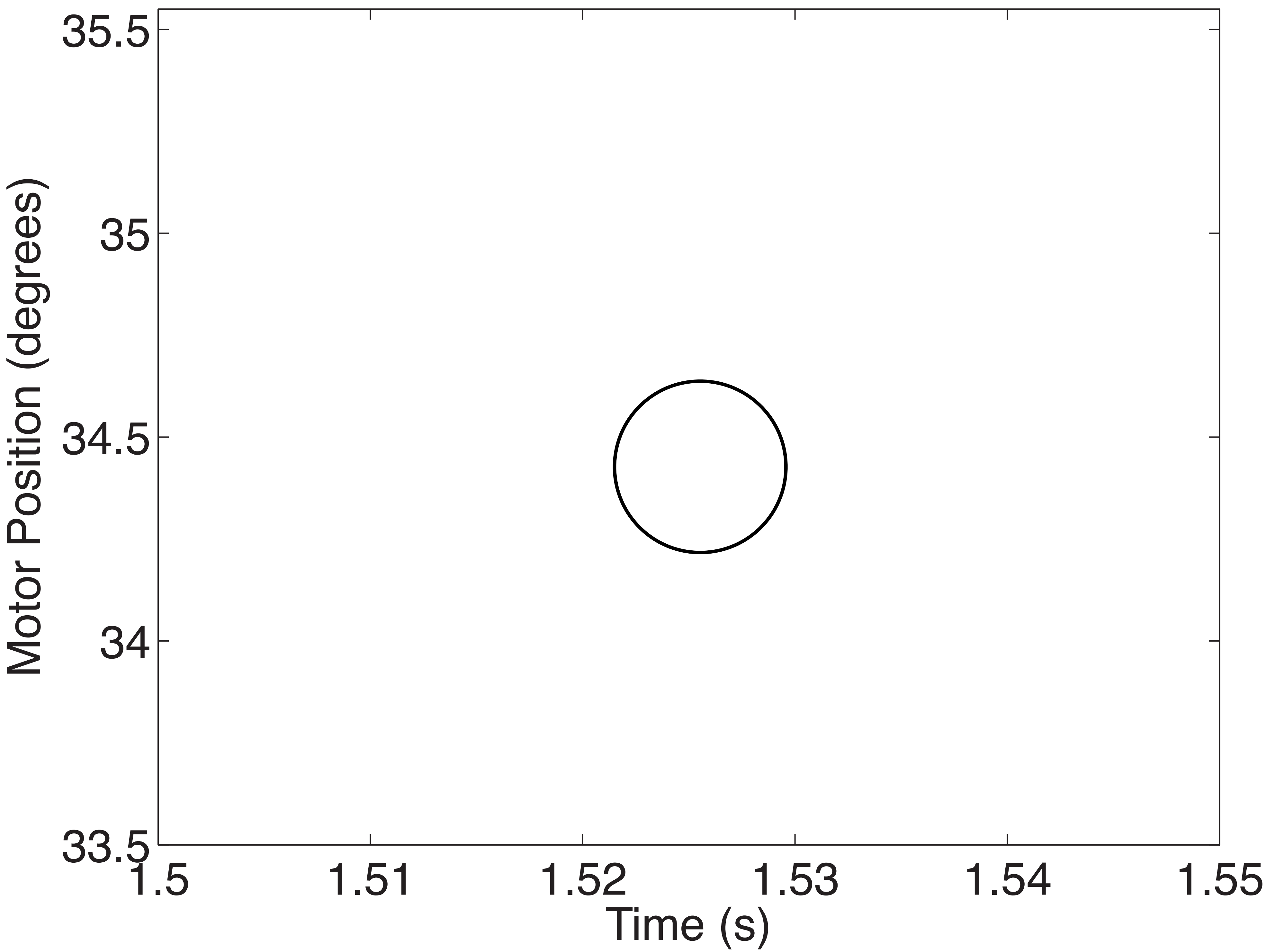

Velocity (deg/s)
80
60
40
20
0
-20
-40
-60
-80
0
2
4
6
8
Time (s)


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


35.5
35
34.5
34
33.5
Motor Position (degrees)
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
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)-----------------------------------

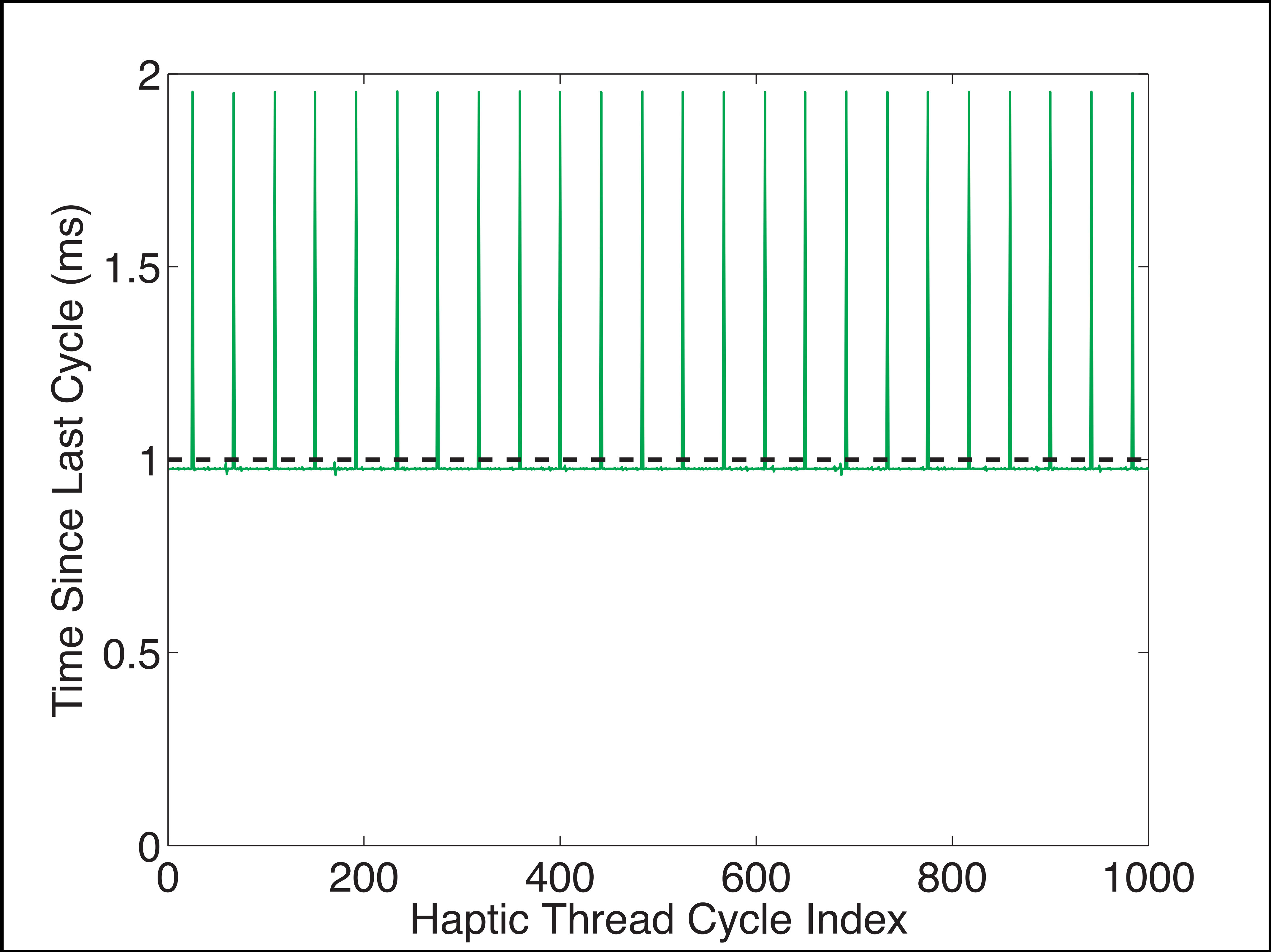
Emacs@tinosa.local

```
        // *** 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;
```

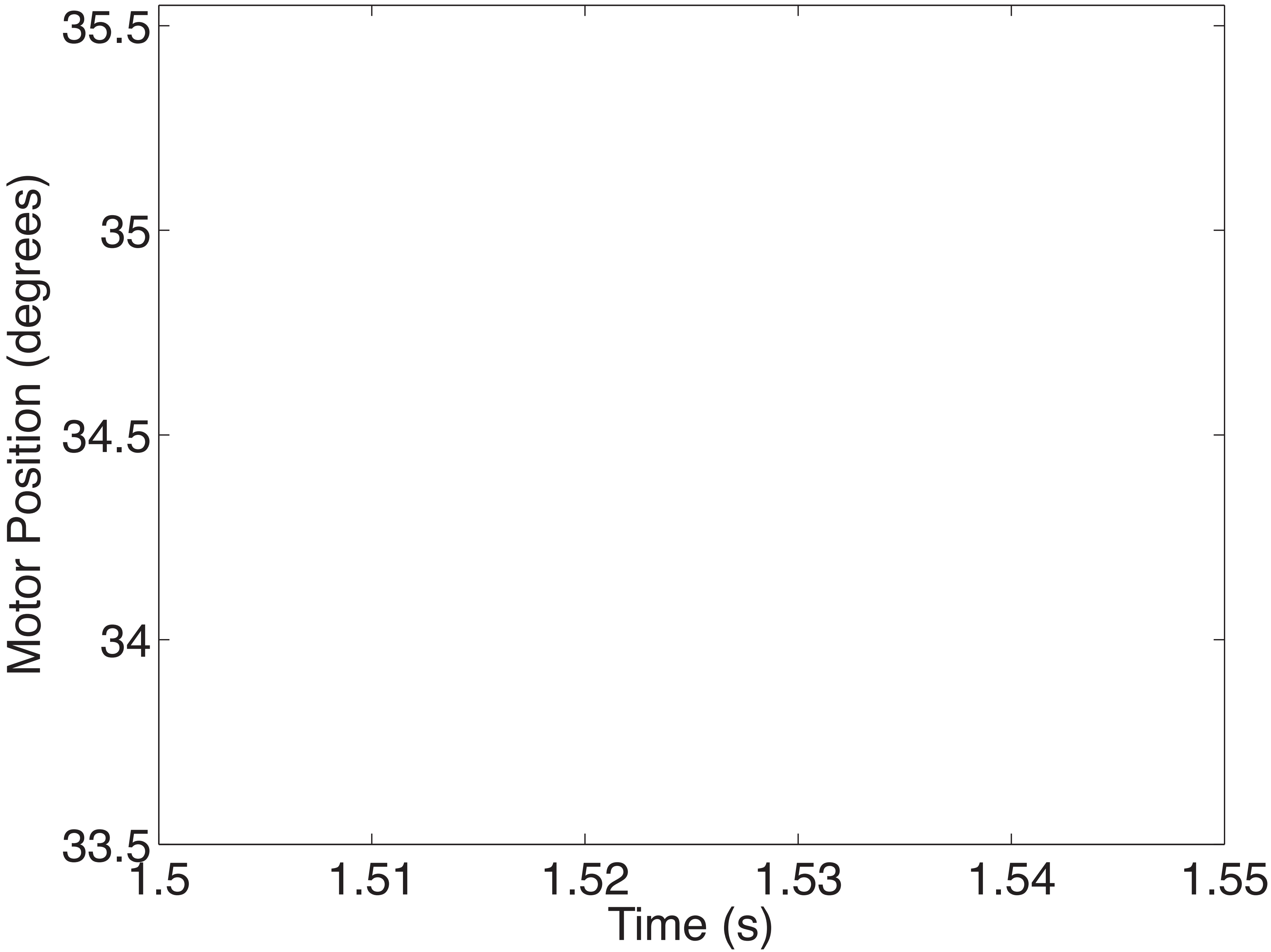
--(DOS)** knob_07_01_05.cpp   69% L1001   (C++ Abbrev)------------------------------

Emacs@tinosa.local

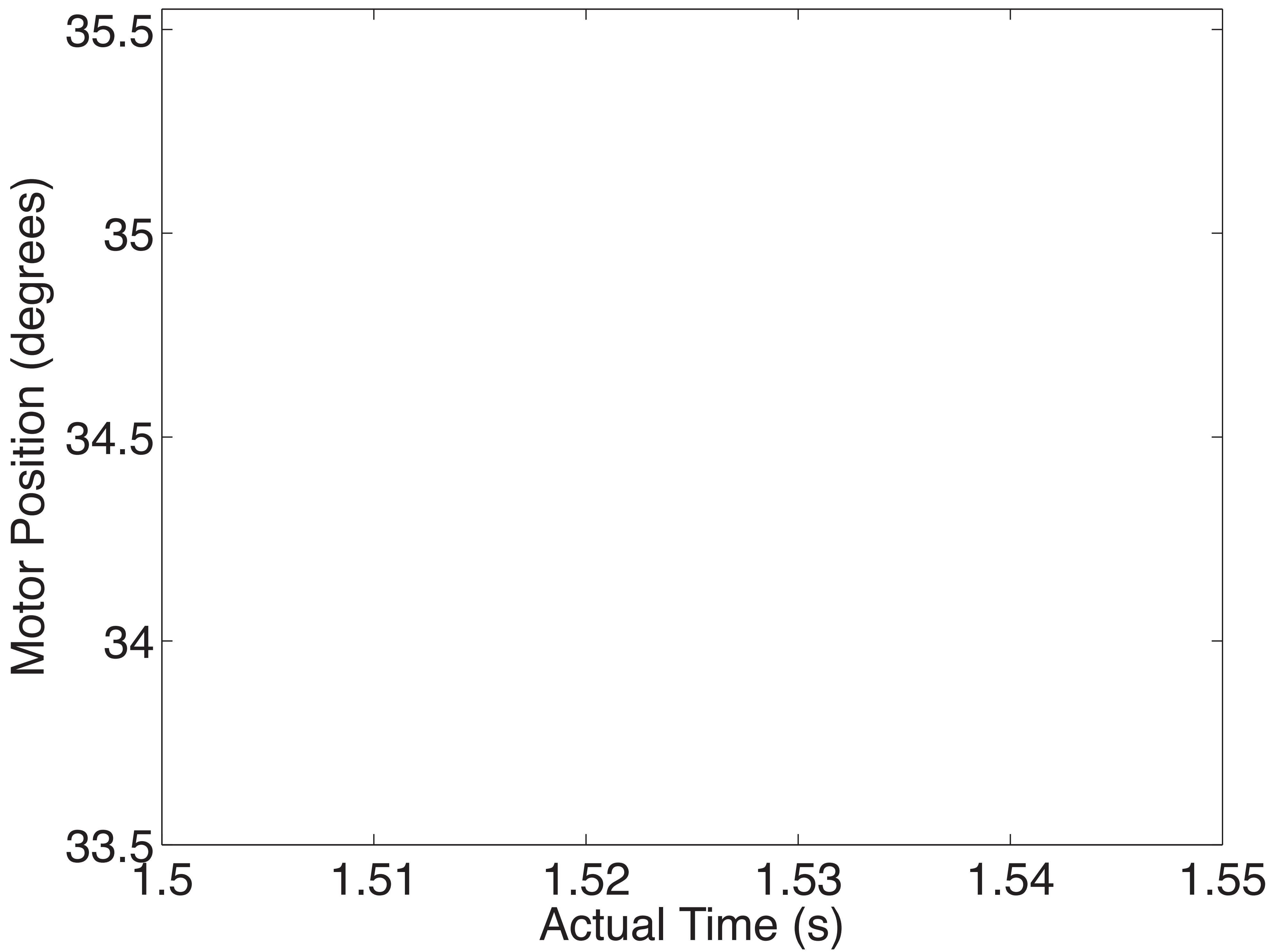
```
otFeedback ? 'D' : 'd', proprioceptiveFeedback ? 'P' : 'p', tactileFeedback ? 'T' : 't', commandPosDeg, co
mmandWidthDeg);
        //      return;
        //}

        // 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)----------------------------------
```

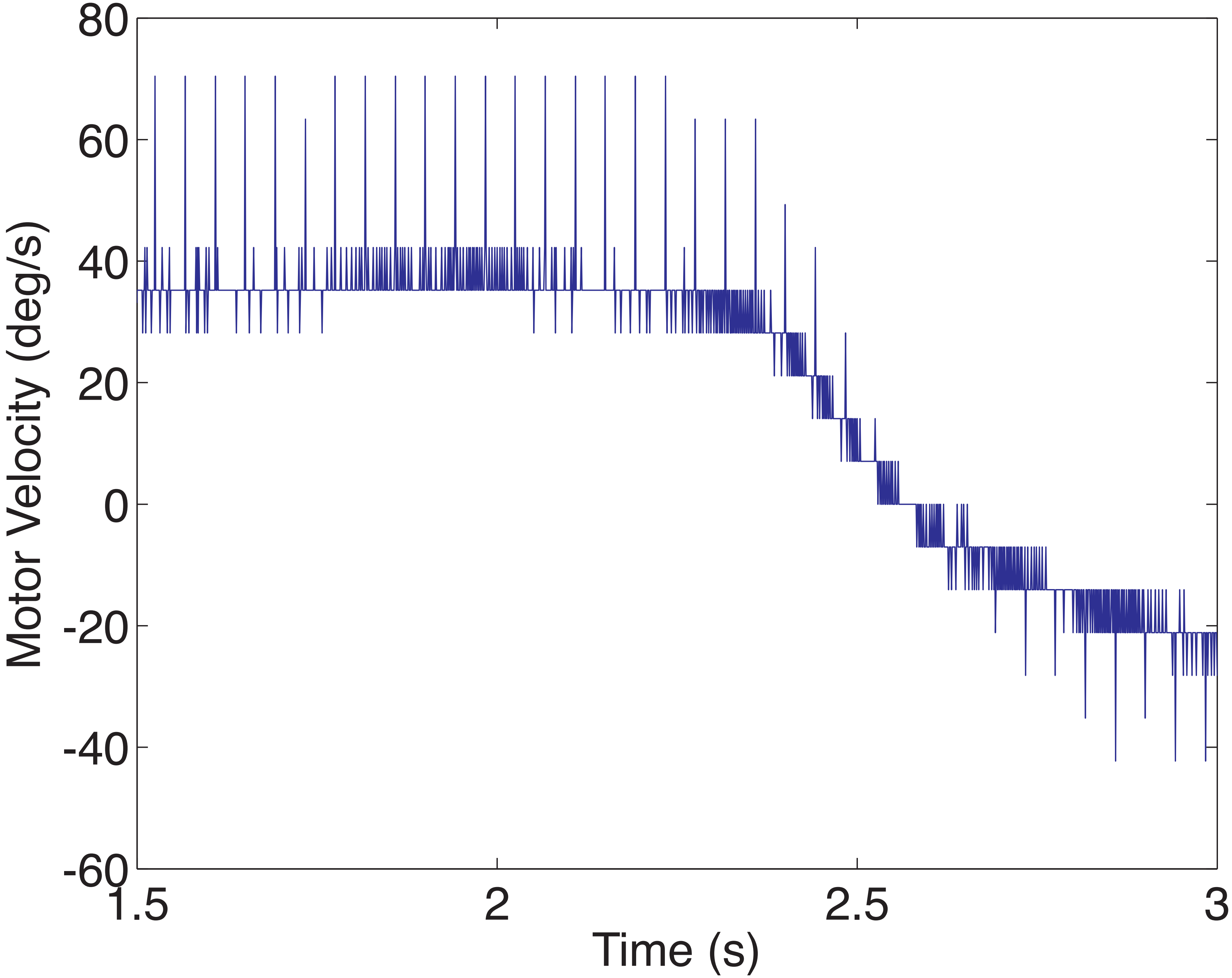


Time Since Last Cycle (ms)
2
1.5
1
0.5
0
0
200
400
600
800
1000
Haptic Thread Cycle Index

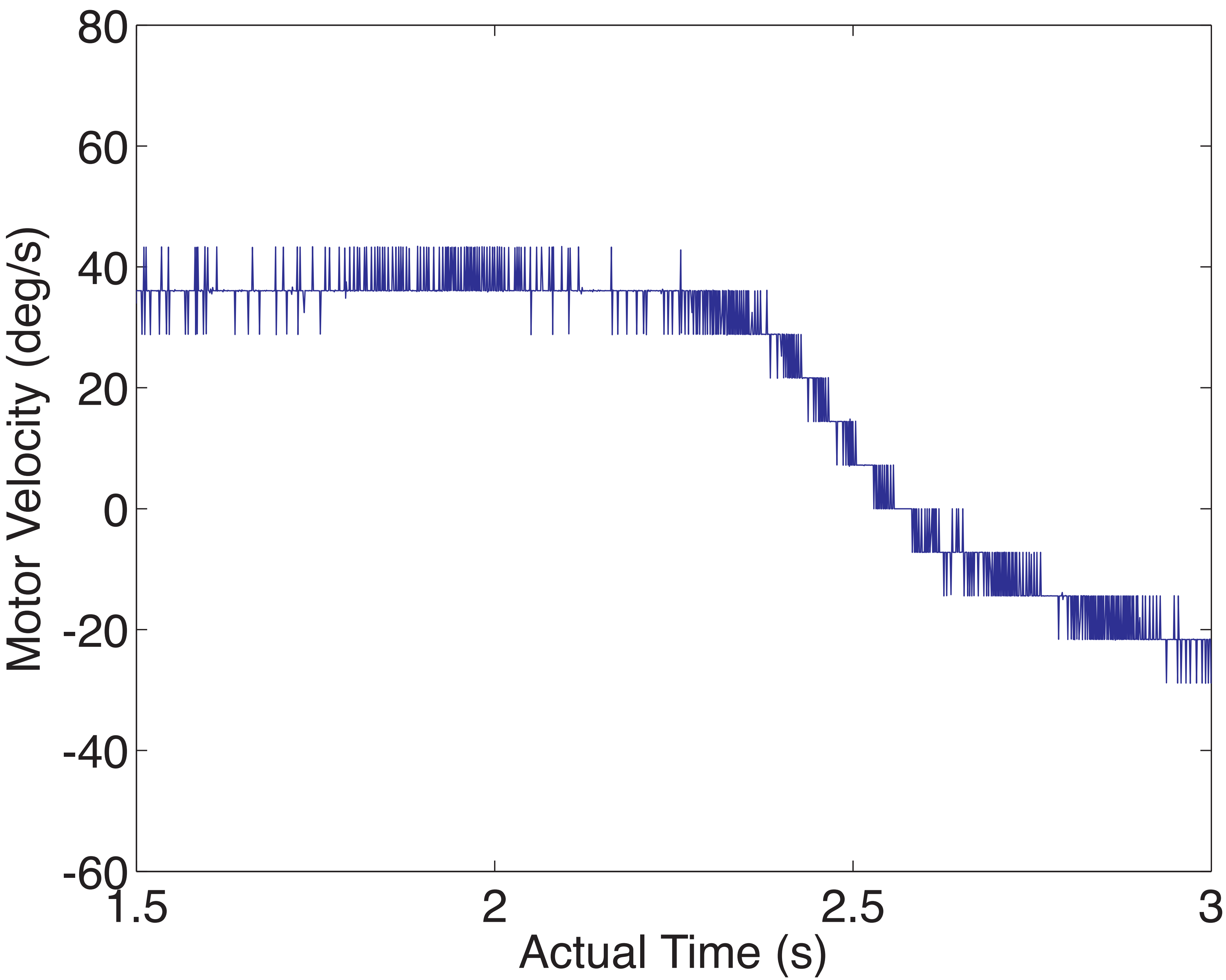


35.5
35
34.5
34
33.5
Motor Position (degrees)
1.5
1.51
1.52
1.53
1.54
1.55
Time (s)


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
1.5
2
2.5
3
Motor Velocity (deg/s)
Time (s)


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

Know your sensors and your signals.

Thank You

Questions?

kuchenbe@seas.upenn.edu
http://haptics.grasp.upenn.edu