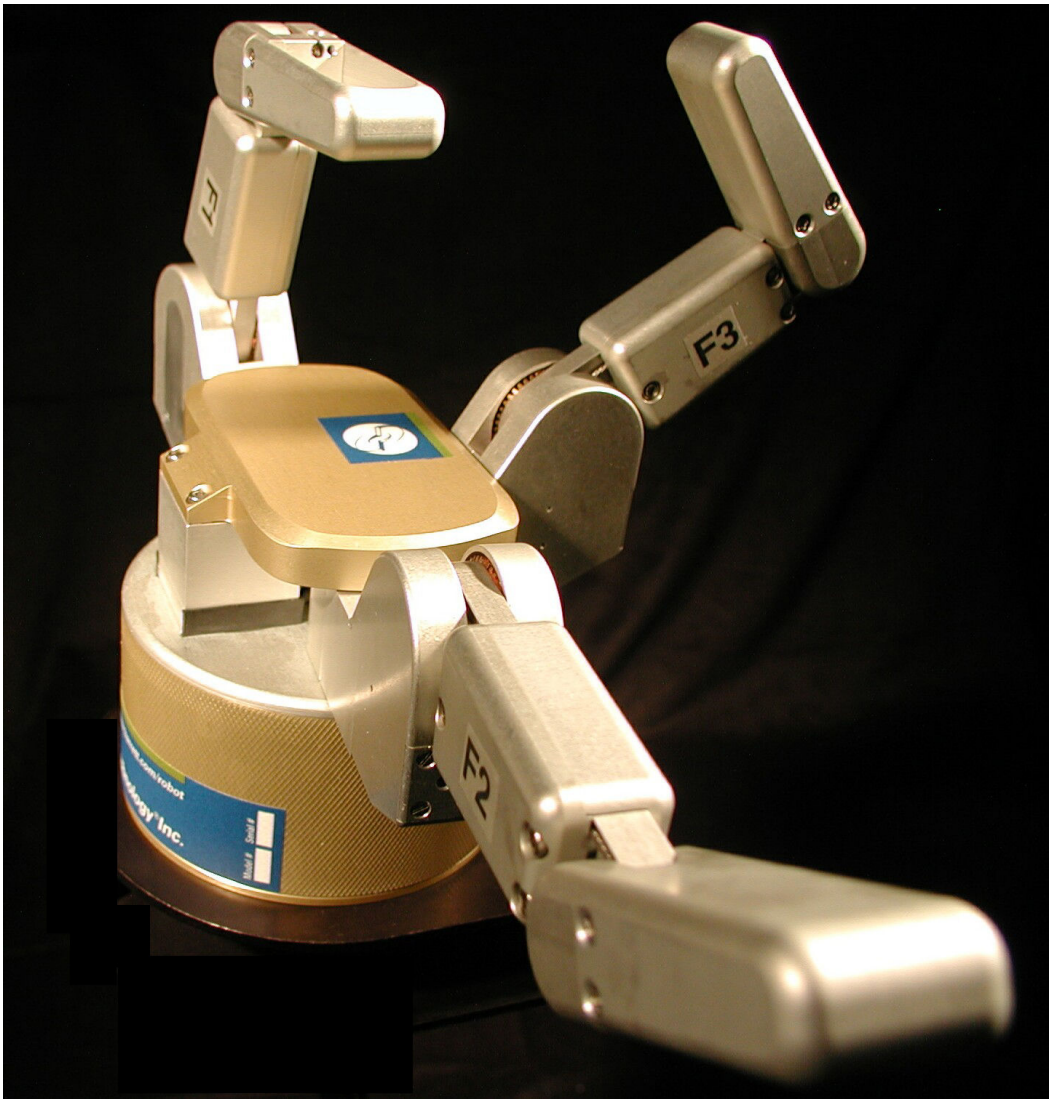
BarrettHand™

BH8-Series User Manual
Firmware Version 4.4.x



Barrett Technology® Inc.

Updated on October 27, 2010

Table of Contents

List of Figures

List of Tables

List of Equations

1 System Description

1.1 Standard BH8-SERIES System Components

1.1.1 System Features

Thank you for choosing the BarrettHand™ Grasper™. The BarrettHand™ is designed to overcome the inflexibility of conventional grippers with microprocessor-enabled dexterity while maintaining durability, compactness, and ease of use. The BarrettHand™ is a multi-fingered Grasper™ with the dexterity to secure target objects of different sizes, shapes, and orientations. Rather than rely on pinching gripper friction or permanent gripper-jaw shape customization the BarrettHand™ gently envelops the object, securely locking its joints until commanded to release.

System integration with any robotic arm is fast and simple. Even with its low, 1.2-kg, weight and compact form, it is totally self-contained. Communication with the BarrettHand™ offers a couple of connectivity options. The BH8-280 hand has high-speed CAN connectivity with Barrett's universal Puck controllers embedded in the hand. The BH8-262 model and earlier hands use industry-standard serial communications, which has been the common denominator of communications, for guaranteed universal compatibility. The BarrettHand™ features a common cross-platform compatible API and provides powerful on-board motion control through Barrett's open Grasper Control Language (GCL) facilitating supervisory control.

The compactness and low weight of the BarrettHand™ assures that the enhanced dexterity does not compromise arm payload. Its low mass and short base-to-grasp-center distance minimize joint loading on the host robot and reduce extraneous arm movements during object reorientation. The custom control-electronics package is contained entirely within the palm shell, reducing electrical wiring to a single cable carrying all communications and motor power.

We hope that you enjoy the versatility and functionality of the BarrettHand™. Please don't hesitate to give feedback and to ask for advice when needed. US+617-252-9000, <service@barrett.com>, or <http://www.barrett.com/robot/>.

1.1.2 Documentation

The baseline turnkey BarrettHand™ now includes the C++ Function Library and comes with two (2) manuals:

1. BH8-SERIES User Manual for setup and basic operation (this manual)
2. BarrettHand Control GUI Manual

This manual is up-to-date as of version 4.4.1 of the BH8-262 BarrettHand™ firmware and provides documentation, which may still be under development for the new BH8-280 hand.

The User Manual (this manual) covers:

- System components and options
- System setup and operation
- Maintenance
- Troubleshooting
- Theory of operation
- Technical specifications
- Frequently asked questions

The second manual is the BarrettHand Control GUI Manual for the cross-platform BHControl application that is built in Code::Blocks. It enables immediate functionality in intuitive and increasingly sophisticated supervisory, RealTime, and visual control tabs. Refer to this manual for detailed instructions.

All manuals will only be available in electronic form. They can be found after installing the latest BarrettHand software included on the USB flash drive and in the customer-only section of the Barrett web site.

Barrett Support Website Link: http://www.barrett.com/robot/support.htm

1.1.3 BarrettHand™

The BarrettHand™, shown in Figure 1, has three fingers labeled F1, F2 and F3. Two of the fingers, F1 & F2, rotate synchronously and symmetrically about the base joint in a spreading action. The "spread" motion around the palm allows "on-the-fly" grasp reconfiguration to adapt to varying target object sizes, shapes, and orientations.

Aside from the spread motion, each of the three fingers on the BarrettHand™ feature two joints driven by a single DC brushless servo motor. The joints of each finger are coupled through Barrett's patented TorqueSwitch™, which automatically switches motor torque to the appropriate finger joint when closing on a target object. Using the fingers together allows the BarrettHand™ to "grasp" a wide variety of objects securely. The TorqueSwitch™ combined with the spread function, makes object grasping nearly target-independent.

The BarrettHand™, shown in Figure 1, is equipped with a threaded base for compact and secure mounting. The threaded base is fully compatible with the BarrettArm. And, with the arm adapter, it can be mounted on virtually any robot with a standard ISO tool plate, for easy installation and maintenance.

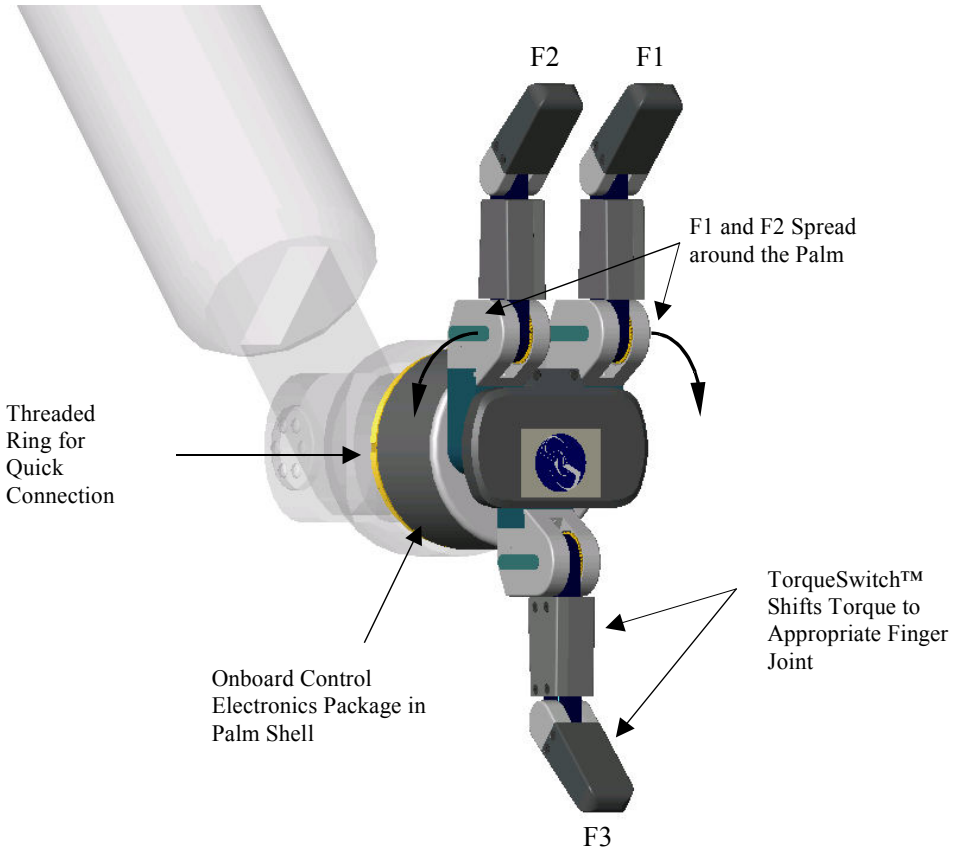


Figure 1 - BarrettHand™

1.1.4 Power Supply

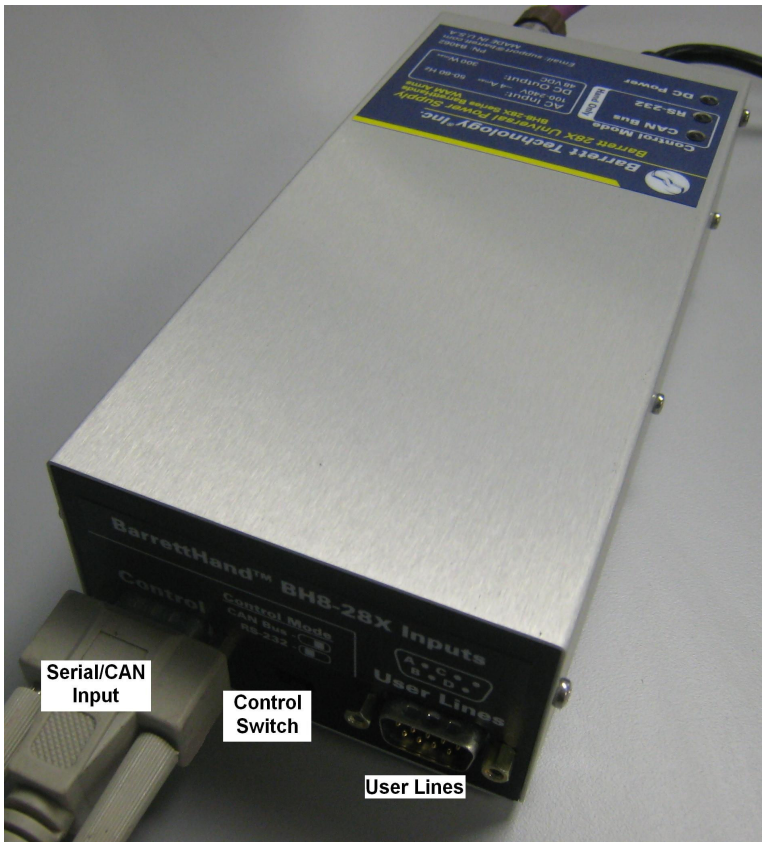


Figure 2 - BarrettHand™ BH8-28x 48-Volt Universal Power Supply

BH8-28x Universal 48-V Power Supply

The BH8-28x power supply shown in Figure 2 is smaller than earlier ones and is designed to run a hand from a host computer over serial or CAN. The user should connect all wires, which connect to the power supply before turning on power. A control switch sets RS-232 or CAN mode and accepts control input from the same DB-9 connector. The unit is powered through an AC line cord and provides all the necessary power to the hand. An industrial grade power supply called the Synqor AcuQor provides a semi-regulated 48 V to power the hand.

CAUTION: Do not try to power previous hand models with this power supply because earlier models were limited to 24 V. Never provide RS-232 signals while in CAN mode because it can cause damage to the electronics.

Earlier BH8-SERIES 24-V Power Supply

The power supply for the BH8-SERIES prior to the 280 hand, shown in Figure 4, can be plugged into any regular AC power source. It provides DC motor bus power (24V), electronic component logic power (5V), and supports RS-232 communications between the host computer (via the serial cable) and the control electronics in the BarrettHand™ palm shell via the Hand cable. This Power Supply switches automatically to local voltage standards (95-130 & 190-260 VAC at 50-60Hz) around the globe and contains built-in surge protection.

1.1.5 Electrical Cables

All necessary electrical cables are included in the basic BH8-SERIES System. The required electrical connections to the different power supplies are shown in Figure 2 and Figure 5. An AC Line Cord connects the Power Supply to a wall source. A DB-9 Extension Cable provides the communications connection from a host computer via RS-232 serial or CAN. A Hand Cable for the particular Power Supply and BarrettHand™ supply communications, logic power, and motor power. This cable is durable and flexible, allowing the BarrettHand™ to be used on any robot with minimal effect on robot performance. Use the included set of adhesive guide clips for cable management. Since the control electronics reside inside the BarrettHand™ itself, no other electrical cabling is required.

The typical connection to BH8-280 systems use a Peak USB to CAN adapter that connects to a DB-9 extension cable and the BH8-28x power supply, which is shown in Figure 3.

Figure 3 - USB to CAN Adapter for BH8-280 BarrettHand™

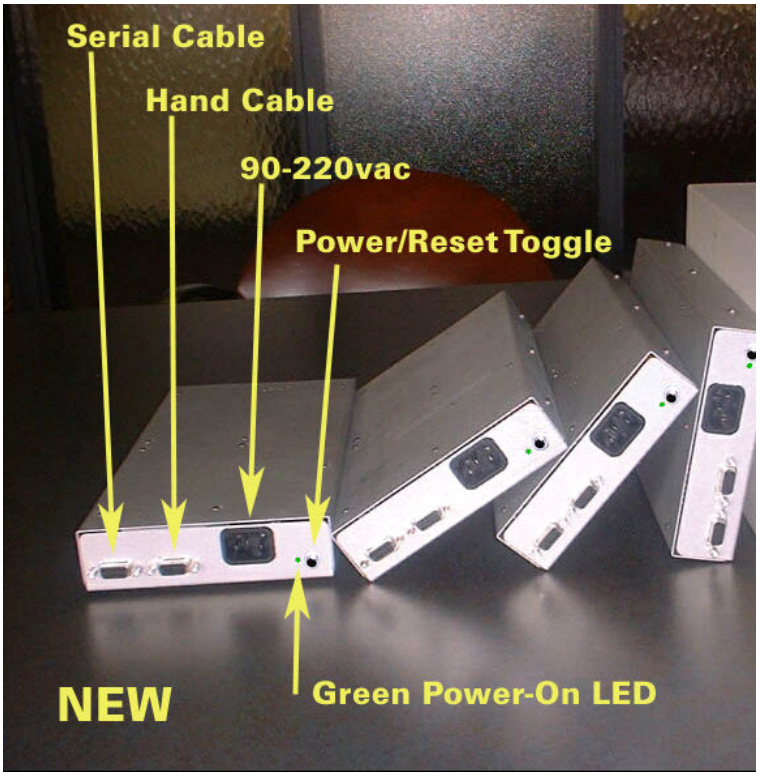


Figure 4 - Earlier BarrettHand™ BH8-SERIES 24-Volt Power Supply

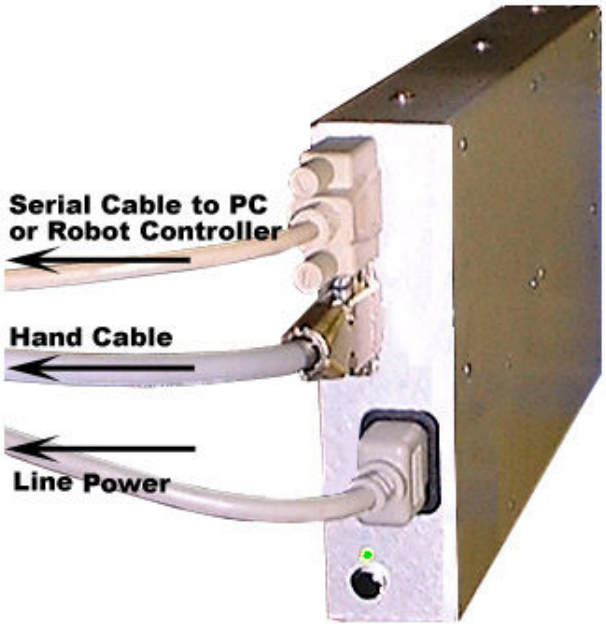


Figure 5 - Cable Connections to the Earlier BH8-SERIES Power Supply

1.1.6 Lab Bench Stand

The bench mount stand for the BarrettHand™, shown in Figure 6, is ideal for off-arm development. The durable Lexan® stand comes complete with cable management clips and mounting features to hold your BarrettHand™ unit securely on any flat surface. Non-slip rubber feet keep the stand from sliding during testing and programming. A threaded locking ring for base mounting will secure the hand to the stand.

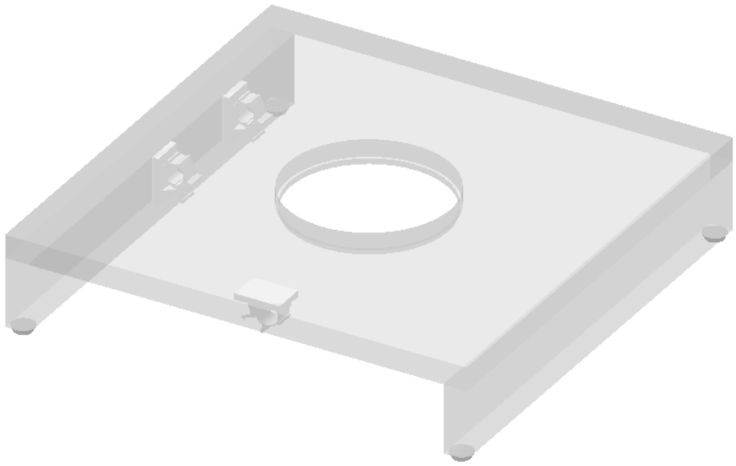


Figure 6 - Lab Bench Stand

1.1.7 Control Software and Firmware

The BH8-SERIES System control software consists of:

1. BarrettHand Control GUI application and API,
2. Firmware (latest *.s19 file for the BH8-262 and *.tek file for the BH8-280 hand), and
3. Example and demo programs.

Included with the software in electronic form are:

1. BarrettHand Control GUI Manual,
2. BH8-SERIES User Manual (this manual), and
3. API documentation in HTML format.

BarrettHand Control GUI

The BarrettHand Control GUI is a cross-platform compatible Windows/Linux application that allows control of the BarrettHand™ quickly and easily. The BarrettHand Control GUI can be used to demonstrate functionality, test Supervisory and RealTime control sequences, and how to save those sequences as ASCII text or even as cross-platform compatible C++ code along with a Makefile (literally with the click of a "Generate C++ Code" button). See the BarrettHand Control GUI Manual for more information on the using this application and the requirements.

C++ Function Library

The BarrettHand™ C++ Function Library is an API for programming the BarrettHand™ using the C++ language on IBM-compatible PC's without having to manage various communication and timing issues. The library contains a hand control class that has easy-to-use functions that permit the use of Supervisory and RealTime commands in software developed by the user. All of the functions are available when the library and its dependencies are linked to the program. Dependencies are usually installed by default so the user can focus on development. The C++ API

includes HTML generated documentation that describes all of the classes, variables, and methods that users should use in detail and gives examples.

The API is written in C++ and compiled for 32-bit versions of Ubuntu 9.10 and Windows XP. It is a typical C++ library, providing a class from which you instantiate one BHand object and use it for all communications. The library uses a multithreaded mechanism for sending commands, which allows both synchronous and asynchronous access to the low-level thread and ensures that all communications are executed with high priority. The low-level thread manages all input and output buffers and makes controlling the BarrettHand™ easy.

Firmware
The BarrettHand™ has firmware that resides on the control electronics inside the palm. The firmware is a compiler generated text file that may be uploaded to the hand through the boot loader and the configuration tab in the GUI. The firmware receives commands, controls the motors, sets and retrieves properties, and reads or writes to the EEPROM. See Sections 4 through 6 for more information on firmware commands and properties.

1.1.8 Maintenance Kit

Included in each BarrettHand™ package is a maintenance kit. Use the maintenance kit in accordance with the instructions in Section 7. The maintenance kit now includes the following:

- Mobil 1® synthetic grease lubricant in syringe
- Hex Wrench Kit (1.0 mm, 1.27 mm, 2.0 mm, etc.)
- 2 mm Hex Driver
- Torque wrench
- Loctite 222
- Tweezers
- Pull tool
- Finger Angle Fixture
- Phillips Head Screwdriver
- Flathead Screwdriver
- 2 Screws with ORings

1.2 System Options

1.2.1 Arm adapter

Barrett Technology provides an arm adapter (Figure 7) matching the make and model of any robot specified by the customer. This lightweight arm adapter is made to work with the end-effector bolt pattern on your robot, allowing quick, easy mounting and wire management for a BH8-SERIES System. The arm adapter is bolted to the end of the robot arm and the BarrettHand™ is secured to the arm adapter with its standard threaded locking ring. The arm adapter is also equipped with an anti-rotation feature to prevent rotation during operation.

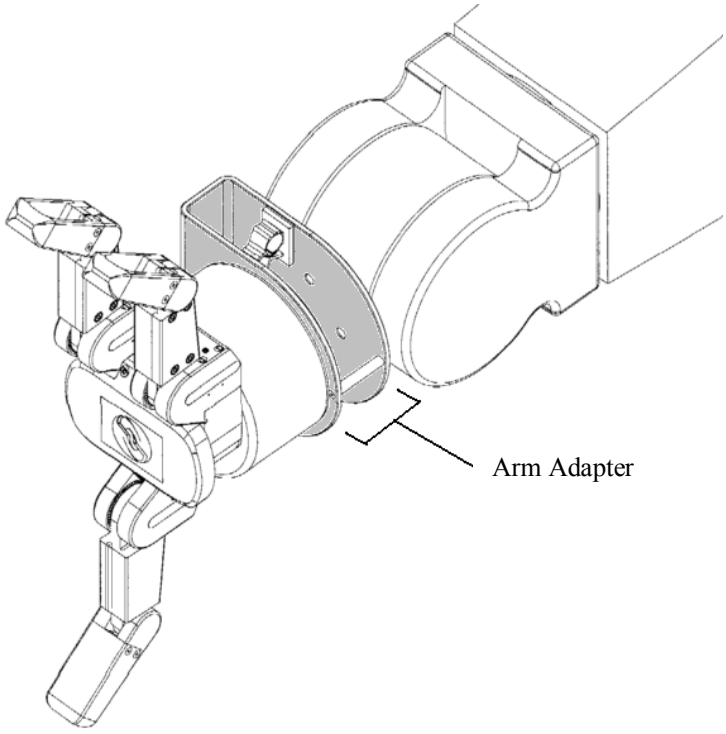


Figure 7 - Arm adapter

1.2.2 Strain gage Joint-Torque Sensors

Barrett Technology offers a factory-installed torque sensor for the BH8-SERIES System. This sensor measures the torque externally applied about the distal joint over a range of +/- 1 N-m (+/-2 kg at finger tip is about 20 N @ 5 cm = +/- 1 N-m). This option uses strain gages to measures the differential tension in the "tendon" running through each finger to the second joint. The information is processed in additional on-board circuitry when this option is installed; it is accessed by requesting the present strain gage property. The strain gage property represents the amount strain on the strain gage sensors. The strain gage values can be calibrated by the customer to relate strain to joint torque. See Section 9.4 for more detailed information on how the sensor works.

1.2.3 Control Software/Firmware Upgrades

Barrett Technology makes software and firmware upgrades periodically. Upgrades are available for purchase or free of charge for customers of Barrett's subscription service. Refer to Barrett's enclosed Warranty and Subscription Service Policy for more information.

2 Safety and Cautions

PLEASE READ THIS SECTION IN ITS ENTIRETY BEFORE USING YOUR BARRETTHAND™.

Following these safety instructions will help prevent user injury and equipment damage.

- As with any piece of robotic equipment, it is ultimately up to you to be aware of your surroundings during robot operation. The workspace of the system comprising the BarrettHand™ and robot arm should be clearly marked to prevent persons or objects from inadvertently entering the equipment's reach. Before attaching the BarrettHand™, test host robot trajectories to confirm that it will not inadvertently collide with other objects in the workspace.

- NEVER connect or disconnect any electrical cables while the Power Supply is turned on. Failure to follow this instruction could impart irreparable damage to the onboard electronics or put you at risk of electrical shock.

- Always plug the Power Supply into a properly grounded wall source. Failure to do so could damage the BarrettHand™ electronics and put you at risk of electrical shock.

- Do not place any part of your body or delicate objects within the grasp of the BarrettHand™ without first verifying control of the unit and confirming appropriate force levels.

- Do not allow the BarrettHand™ to be exposed to liquids that may cause electrical short-circuit and put you at the risk of electrical shock.

- Keep dirt away from the exposed gear and cable drives located at the joints.

- Do not exceed the load limit of the fingers, 2 kg per finger. Consider all loading situations including accelerated loads, cantilever loads from long objects, robot collisions, active loads, etc.

- On models before the BH8-260: a portion of the onboard control electronics is exposed through the base of the BarrettHand™. Before installing pre-BH8-260 models to a robot arm, take necessary precautions to protect the electronics from impact, contaminants and static discharge. Do not rest the BarrettHand™ unit directly on its base. Use the included lab bench stand during standalone operation.

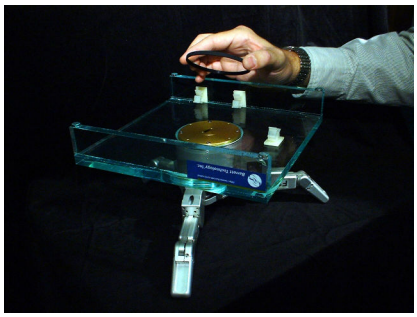
- Remove/replace the fingers only as instructed in Section 7.3.

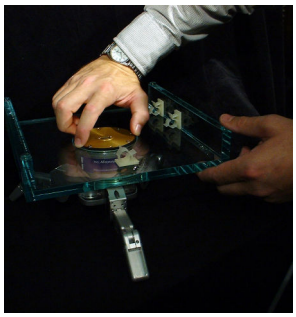
- Monitor the operating temperature of the BarrettHand™ so that it does not exceed 65C. Under normal conditions, the Hand operates between 40 and 60C. The BarrettHand™ was designed with non-backdrivable finger joints to take advantage of the motors' peak operating performance in short bursts. The spread, however, is backdrivable to aid in target-independent grasping (see Section) and requires constant motor current to actively hold position. Idling the spread motor, when possible, will help keep the temperature lower.

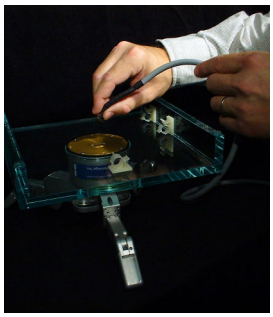
3 System Setup

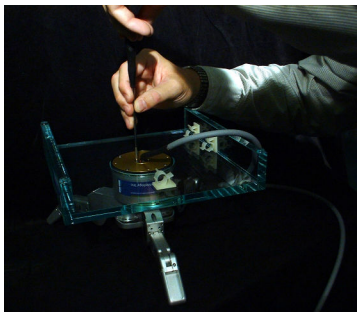
3.1 Mounting Method 1: Lab Bench Stand

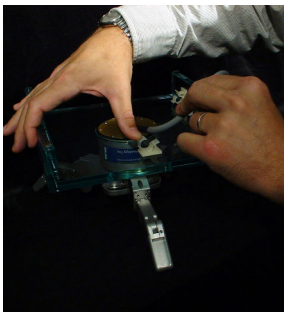
The Lexan Bench Stand you received with the BarrettHand™ has been provided for convenience in programming the BarrettHand™ when a host robot arm is not available. Use the wire guide clips to provide strain relief to the Hand cable. Figure 8 illustrates how to mount the Hand in the Lexan Bench Stand.


a) place base on Hand


b) attach threaded ring


c) attach Hand cable


d) screw in 2 retainer screws


e) open cable clips

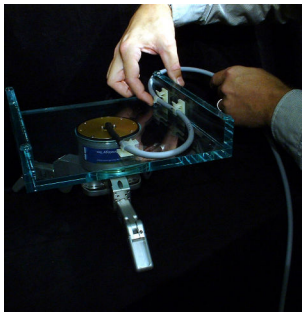

f) secure all 3 clips

Figure 8 - Lab Bench Stand with Wire Strain Relief

Use the 2-mm hex wrench to open fingers 1, 2, & 3. Then, spread fingers to roughly 120 degrees and rest the unit on its fingertips. Place the stand, feet up, onto the hand. Note the alignment of the BarrettHand™ relative to the wire strain relief clips to ease connection of the BarrettHand™ Cable.

Make sure the Power Supply is turned OFF, and then route the BarrettHand™ Cable through all three cable clips on the lab bench stand and plug it into the BarrettHand™. Tighten the cable clips to hold the cable in place.

3.2 Mounting Method 2: On Robot Arm

3.2.1 Robot-Arm Adapter

Like the Lexan Bench Stand, the Robot Arm Adapter is made to secure the BarrettHand™ in place and to provide strain relief to the Hand cable as shown in Figure 9. The Arm Adapter is fabricated for the tool-plate of your specific robot arm and is designed for low-profile, rigidity, and low weight.

To mount your BarrettHand™ on a robot, bolt the arm adapter onto the tool-plate bolt circle, located at the end tip of the robot arm. Next, insert the threaded base of the BarrettHand™ through the hole in the arm adapter shown in Figure 9 aligning the indexing tab on the arm adapter to the mating alignment slot on the BarrettHand™.
Secure the BarrettHand™ by threading the locking ring (included with your system) onto the base of the BarrettHand™. Note that, depending on the details of your robot arm, you may need to loosen the locking ring when installing the Hand cable the next Section.

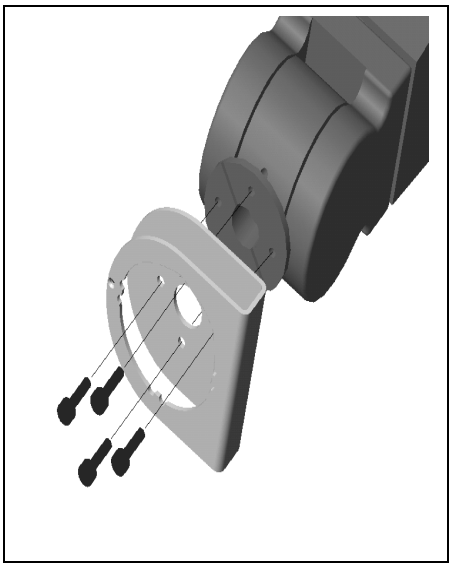


Figure 9 - Installing an Arm Adapter.

3.2.2 Installing the Hand Cable on a Robot Arm

All of the power and communications for the BarrettHand™ have been consolidated into a single high-durability robot cable which has a 15-pin connector at the Power Supply end and a tiny 10-pin connector at the Hand end. To accommodate the complex motions of a robot arm, the BarrettHand™ Hand Cable is extremely flexible and has been designed for compatibility with both internal and external mounting schemes. When a robot arm does provide an internal channel, the cross-section of the channel is tightly constrained. Therefore the Hand cable has been made with a particularly tiny connector at one end to ease internal installation. The base of the Hand Adaptor includes an opening to accommodate direct access from an internal cable to the back of the BarrettHand™.

For external installation, plan to route the Hand Cable close to the center of each joint. Each segment will need enough slack to accommodate the most extreme motions but not so much that the cable might become snagged. Mount the cable clips to flat, dry, and clean link surfaces at strategic points along the robot arm. Clean cable clip attachment areas with alcohol before attaching via the self-adhesive backings. Place the BarrettHand™ Hand Cable loosely through the cable retaining clips on the robot and the Arm Adapter. Move the robot arm through all of its motion extremes to verify that the cable slack is adequate in each segment and that it will not snag. Once verified, tighten the cable clips to secure the cable in place.

3.3 Electrical Connections

- Place the Power Supply on a flat, secure surface anywhere between the base of the robot and the host PC (or robot controller).
- With your PC off, attach the DB-9 extension cable from your 9-pin COM Port or Peak USB to CAN adapter to the Power Supply. Barrett Technology supplies a 3-meter standard straight-through DB-9 cable, but you may purchase a longer cable if desired.
- Attach the Power Supply Line Cord into any convenient outlet and verify that it is switched **OFF**.
- Attach the one end of the Hand Cable to the Power Supply and the other end into the Hand. Tighten the strain-relief screws using the Phillips screwdriver provided in the toolkit. **CAUTION:** Be sure you are not supplying 48 V to a 24 V hand.
- Check the switches on the bottom of a BH8-280 hand under the access panel. Make sure the CAN termination is ON (SW1, #4), SW3 is set to CAN (the side with the dot), and J35/J36 is correct for which port you are using (jumper on J36 for bottom connector).

Figure 10 - BH8-280 Quick-Start Guide

3.4 Host Computer

The BH8-SERIES control software was written for computers running Windows XP or Ubuntu 9.10. The BarrettHand Control GUI requires a good amount of CPU power and memory for running this wxWidgets based application. Barrett Technology requires the minimum specs for using the corresponding operating system. A processor with a CPU clock speed of 1 GHz or greater, 1 GB of RAM, at least a Gigabyte of free disk space, and a modern graphics card is recommended. Communications requires either one available USB port for the Peak USB to CAN adapter for the BH8-280 hand or a 9-pin serial port for BH8-262 hand.

3.5 Installing BH8-SERIES Control Software

The BH8-SERIES Control Software consists of the BarrettHand Control GUI, firmware, example, and demo programs. The BarrettHand Control GUI is a graphical user interface that allows you to control the BarrettHand™ quickly and easily. The BarrettHand Control GUI can be used to test Supervisory and RealTime control sequences, determine communication loop rates, demonstrate functionality, help you learn how to independently write C++ code, and automatically generate C++ code based on tested algorithms[1]. Run the *bhand*.exe* installer on Windows or *bhand*.deb* on Ubuntu to install all necessary files that are included on a flash drive or obtained online for using the BarrettHand Control GUI, the most recent version of firmware, online manuals, and example programs.

3.6 Power-Up Sequence

Once the previous steps are complete, your BH8-SERIES System is ready for use. Power up the system according to the instructions below:

1. Verify the DB-9 extension cable is plugged into the desired communications port (or Peak USB to CAN adapter) and into the 9-pin connector on the back of the Power Supply.
2. Verify the Hand Cable is plugged into the the back of the Power Supply and into the bottom of the BarrettHand™.
3. Verify the AC line Line Cord cord is plugged into a valid power source and into the power outlet on the Power Supply.
4. Turn on the host computer.
5. Turn on the Power Supply. The main power switch is located on the back panel.
6. The BarrettHand™ is now ready for operation.

[1] The code generated by the BarrettHand Control GUI requires the C++ Function Library and dependencies installed to compile.

3.7 Upload Firmware

The BarrettHand™ firmware resides on board the electronics located inside the hand. BH8-262 hand firmware is stored in RAM that receives its power from the Power Supply when the system is turned on and from an embedded super capacitor when powered down. This super capacitor is designed to maintain the firmware in RAM for two days. However, especially in humid weather, the memory will begin to degrade after two days and eventually will be cleared. Since degraded memory may result in erratic control, please reload firmware if the hand has been turned off for several days. When the firmware has been cleared or degraded, it will need to be reinstalled. BH8-280 hands have firmware stored in flash and uploading firmware should only need to be done if upgrading to a newer firmware version. The latest version of firmware should be uploaded to the hand whenever new software is installed on the host computer to ensure proper operation.

The download process takes only a few minutes, as follows:

1. Verify the BarrettHand™ is plugged into the Power Supply.
2. Verify the host computer is plugged into the Power Supply.
3. Verify the Power Supply is attached to a power source.
4. Run the BarrettHand Control GUI from Windows or Ubuntu.
5. Select the Hand model number and either CAN or Serial depending on your connection.
6. Open the appropriate file according to Table 1.
7. Press the *Upload* firmware button in the Upload Firmware box.
8. Follow the on-screen instructions:
    - BH8-262 users: Cycle power: Switch the Power Supply off, wait 5 seconds, and then turn it back on. The upload begins automatically.
    - BH8-280 users: Have the Hand powered for 5 seconds and then click OK to begin uploading firmware.
9. After uploading the file, the BarrettHand™ is ready for operation.
10. Initialize the software by pressing the *Initialize Library* button. Check that new firmware is version is reported correctly in the GUI.

Table 1 - Firmware File List

File Name	Description
“bhand4.4.1.S19”	Latest BH8-262 Firmware
“puck2r153.TEK”	Latest BH8-280 Puck 2 Firmware

4 Control Modes – Supervisory and RealTime

The BarrettHand™ can be used in either of two (2) modes:

1. high-level Supervisory mode or
2. low-level RealTime mode.

Most users of the BarrettHand™ can rely exclusively on Supervisory mode since it handles virtually every function of the BarrettHand™. Supervisory mode leverages the control capabilities of the BH8-262 on-board Motorola microprocessor or the BH8-280 Pucks in the hand. BH8-280 hands will directly run motion received motion commands, whereas a BH8-262 hand will need to apply control signals across the four (4) HCTL-1100 motion-control microprocessors. Supervisory mode allows you to command individual or multiple motors to close, open, and move to specific positions; it also provides for setting the various configuration properties and reporting positions and torques (torque measurement requires optional strain-gage sensors).

At the simplest level, the BH8-262 Supervisory mode allows you to type and receive ASCII text characters on a terminal (using any type of computer hardware or operating system, such as UNIX, Macintosh, PalmPilot, and proprietary robot controllers, etc.). To automate grasping applications, you can write programs, scripts, or macros that send and receive these text characters through the serial port (e.g. the optional BarrettHand™ C++ Function Library).

All Supervisory commands that are found on the 262 hand are also available on the 280 hand. BH8-280 Supervisory commands are handled separately by each Puck that communicates on the same CAN bus in the Hand. It is recommended to use the BarrettHand API for handling the underlying low-level CAN communication methods. The 280 hand is property based and CAN messages are sent to set Hand properties and Get messages may be sent to request property values. Properties are able to be read at anytime so properties from sensors (e.g. positions or strain) may be read simultaneously, through a registered callback function that is called periodically, when running motion commands that are under the programmer's control.

RealTime mode extends the capability of Supervisory mode. Sometimes users may wish to bypass the Supervisory functions and apply control directly to the motion-control microprocessors. RealTime mode enables users to close control loops in real time from their host PC or robot controller. This benefits users interested in real-time control via a data-glove, PhanTom, visual-servoing applications, real-time force control (via optional strain-gage sensors), etc. Users can switch between Supervisory and RealTime modes on-the-fly as desired, allowing for mixed mode operation.

If you use the BarrettHand Control GUI that runs on a PC, you will likely wish to experiment with the "Visual" control window to familiarize yourself with the BarrettHand™. The Visual window relies exclusively on RealTime control mode since it must follow your RealTime mouse-cursor movements.

5 Supervisory Control Mode

An application designed to run and create Supervisory programs, that all users should become familiar with, is the BarrettHand Control GUI. Sequences of most commands may be formed though intuitive input controls, may be added to an execution buffer, and run with the click of a button. Standalone cross-platform compatible C++ programs can be generated with a Makefile that run sequences of Supervisory commands. The electronics and the means of communication in the BH8-SERIES BarrettHands are different as described in Section 9.1. The BH8-280 hand is the newest hand in the BH8-series and has Section 5.6 dedicated documenting how Supervisory mode is used. The following sections pertain to earlier models of the hand and explain the low-level serial protocol.

5.1 Overview

An overview of serial protocol for the BH8-262 hand and earlier hands is developed in this first section that forms the structure for all Supervisory mode commands. The hand will send company and product information about the hand on power up followed by a command prompt when the hand is ready. Supervisory commands are executed sequentially, desired commands are initiated by the host, and the hand responds to all commands appropriately with any error message followed by a prompt for the next command.

5.1.1 Commands

When the BarrettHand™ firmware is ready to process a command, it prints a prompt of "=> " to the host computer. A command can then be entered as a single line, terminated by a carriage return character (0x0d). Once the firmware receives the carriage return, it processes the line, executes the command, prints any error result, and then prints a new prompt. Once a command has been started, no configuration changes can be made until the command has completed or has been aborted.

Many of the commands take one or more parameters; space characters should separate these from the command and each other.

5.1.2 Motor commands

Motor commands act on one or more of the four hand motors. By default, all four motors are affected. To select fewer than four motors, a motor prefix must be placed before the command (with no space between the prefix and the command). A motor prefix consists of one or more of the following characters:

Table 2 - Motor Prefixes

Value	Motor
1	Finger F1
2	Finger F2
3	Finger F3
4	Spread
G	Finger F1, Finger F2, Finger F3
S	Spread
<No Motor Specified>	Finger F1, Finger F2, Finger F3, Spread (see the Firmware EN property in Section 5.3.4)

Examples:

"12<command>" executes the given command on fingers 1 and 2
"3S<command>" executes the given command on finger 3 and the spread
"<command>" executes the given command on all fingers and the spread

5.1.3 Status Codes

When a Supervisory mode command encounters an error or unexpected result, the command is terminated, a status code is printed, and then a new prompt is printed to the host PC. The status code is in the format "ERR <value>", where <value> is the sum of the status codes encountered. Note that the status codes are powers of 2 so that the sum may be decomposed into the individual status codes.

Table 3 - Hand Status Codes

Hand Status Code	*Description*
1	No motor board found
2	No motor found
4	Motor not initialized
8	(not used)
16	Couldn't reach position
32	Unknown command
64	Unknown parameter name
128	Invalid value
256	Tried to write a read only property
512	(not used)
1024	Too many arguments for this command
2048	Invalid RealTime control block header
4096	Command can't have motor prefix
8192	Overtemperature fault tripped
16384	Cntl-C abort command received

5.2 Command List

Supervisory mode commands are organized into the following five (5) categories:

- Movement
- Motor properties
- Global properties
- Administrative
- Advanced

5.2.1 Movement Commands

Movement commands are motor commands: they immediately affect one or more of the motors. Each can take motor prefixes.

Command: **C**
Name: Close
Purpose: Commands the selected motor(s) to move fingers in close direction with a velocity ramp-down at target limit. If selected, each finger moves towards the palm, and the spread motor moves so that fingers F1 and F2 are adjacent to finger F3.
Arguments: (none)
Example: SC
Notes: The C command is similar to a MOVE command, with the value of the Close Target (CT) motor property as the destination. However, unlike the Move command, the C command will not return an ERR32 if it does not reach CT within the maximum position error (MPE) property.

Command: **HI**
Name: Hand Initialize
Purpose: Initializes the selected motor controller(s), preparing them for use by other movement commands.
Arguments: (none)
Example: HI
Notes: HI must be run before any other movement command. Generally it is run without a motor prefix, initializing all four motors; although, if desired, a subset of the motors can be specified. After an HI, all motors are in their home position; at 0 encoder counts.

Command: **HOME**
Name: Home
Purpose: Moves the selected motor(s) to position 0.
Arguments: (none)
Example: SGHOME
Notes:

Command: **IO**
Name: Incremental Open
Purpose: Opens the selected motor(s) the given number of counts. If no argument, then the amount to open is specified by value(s) the DS property.
Arguments: Distance to move (0 to 20000) (optional)
Example: 12IO 5000
Notes:

Command: **IC**
Name: Incremental Close
Purpose: Closes the selected motor(s) the given number of counts. If no argument, then the amount to close is specified by the value(s) DS property.
Arguments: Distance to move (0 to 20000) (optional)
Example: GIC 5000
Notes:

Command: **LOOP**
Name: Loop
Purpose: Enters RealTime mode.
Arguments: (none)
Example: LOOP
Notes: See Section 6 for more information on RealTime mode.

Command: **M**
Name: Move
Purpose: Moves the selected motor(s) to the given position. If no argument specified, then the motor(s) will move to the position(s) given by DP motor property.
Arguments: Position (0 to 20000) (optional)
Example: 13M 1000
Notes:

Command: **O**
Name: Open
Purpose: Commands the selected motor(s) to move fingers in open direction with a velocity ramp-down at target limits. If selected, F1, F2, and F3 open away from the palm, and the spread motor moves so that fingers F1 and F2 are opposite finger F3.
Arguments: (none)
Example: GO
Notes: The O command is similar to a MOVE command, with the value of the OT motor property as the destination. However, unlike the Move command, the O command will not return an ERR32 if it does not reach OT within the maximum position error (MPE) property. This command will reset the breakaway detected flag, BD.

Command: **T**
Name: Terminate power
Purpose: Turns the selected motor(s)'s power off.
Arguments: (none)
Example: ST
Notes: Although T will return an "ERR 1" if any of the selected motor(s) isn't initialized; it will still turn the selected and initialized motor(s)'s power off.

Command: **TC**
Name: Torque-Controlled Close
Purpose: Commands velocity of selected motor(s) in the direction that closes the finger(s) with control of motor torque at stall.
Arguments: (none)
Example: STC
Notes:

Command: **TO**
Name: Torque-Controlled Open
Purpose: Commands velocity of selected motor(s) in the direction that opens the finger(s) with control of motor torque at stall.
Arguments: (none)
Example: STO
Notes:

5.2.2 Motor Property Commands

Motor property commands act on the configuration properties for one or more of the motors. All except FLIST can take motor prefixes. See section 5.3 for a complete list of motor properties.

Command: **FSET**
Name: Finger Set
Purpose: Sets the to the given value(s) for one or more motor properties
Arguments: <propertyName> <propertyValue>
Example: SFSET DS 100 DP 1500
Notes: You can set more than one property by listing more than one propertyName/propertyValue pair.

Command: **FGET**
Name: Finger Get
Purpose: Gets and prints the property value(s) for the selected motor(s). Each property has its value(s) printed on one line, with one value for each selected motor separated by spaces.
Arguments: <propertyName>
Example: SFGET DS DP
Notes: More than one property name may be added.

Command: **FLOAD**
Name: Finger Load
Purpose: Loads the selected motor(s)'s properties from non-volatile storage. This is done whenever the firmware starts up.
Arguments: (none)
Example: 3FLOAD
Notes: The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost.

Command: **FSAVE**
Name: Finger Save
Purpose: Saves the selected motor(s)'s properties to non-volatile storage.
Arguments: (none)
Example: 123FSAVE
Notes: The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost. However, this command should not be performed more than 5,000 times or the hand electronics may need repair.

Command: **FDEF**
Name: Finger Default
Purpose: Sets the selected motor(s)'s properties back to their factory default values.
Arguments: (none)
Example: SFDEF
Notes: Does not save the changed values to non-volatile storage.

Command: **FLIST**
Name: Finger List
Purpose: Lists all of the standard motor properties and their descriptions.
Arguments: (none)
Example: FLIST
Notes: Does not take a motor prefix.

Command: **FLISTV**
Name: Finger List Value
Purpose: Lists the motor-property values for the selected motor(s). Each property has its value(s) printed on one line, with one value for each selected motor separated by spaces.
Arguments: (none)
Example: 3FLISTV
Notes:

5.2.3 Global Property Commands

Global property commands configure the hand as a whole, without referencing a particular finger or motor. Except for not taking a motor prefix they are identical to the set of motor property commands. See section 5.4 for a complete list of global properties.

Command: **PSET**
Name: Property Set
Purpose: Sets one or more properties to the given value(s)
Arguments: <propertyName> <propertyValue>
Example: PSET OTEMP 60
Notes: More than one property may be set be listing more than one propertyName/propertyValue pair.

Command: **PGET**
Name: Property Get
Purpose: Gets and prints one or more given property value(s). Each property has its value(s) printed on one line.
Arguments: <propertyName>
Example: PGET TEMP UPSECS
Notes: Get more than one property value by listing more than one property name.

Command: **PLOAD**
Name: Property Load
Purpose: Loads the global properties from non-volatile storage. This is done whenever the firmware starts up.
Arguments: (none)
Example: PLOAD
Notes: The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost.

Command: **PSAVE**
Name: Property Save
Purpose: Saves the global properties to non-volatile storage.
Arguments: (none)
Example: PSAVE
Notes: The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost. However, this command should not be performed more than 5,000 times or the Hand electronics may need repair.

Command: **PDEF**
Name: Property Default
Purpose: Sets the writable global properties to their default values.
Arguments: (none)
Example: PDEF
Notes: Does not save the changed values to non-volatile storage.

Command: **PLIST**
Name: Property List
Purpose: Lists all of the standard global properties and their descriptions.
Arguments: (none)
Example: PLIST
Notes:

Command: **PLISTV**
Name: Property List Value
Purpose: Lists all of the standard global properties' values. Each property has its value(s) printed on one line.
Arguments: (none)
Example: PLISTV
Notes:

5.2.4 Administrative Commands

Administrative commands implement various housekeeping functions.

Command:	**?**
Name:	Help
Purpose:	Lists all of the standard commands. If immediately followed by a command name, then it lists the command name and its description.
Arguments:	<commandName>
Example:	?HI
Notes:	There must be no space between "?" and any command name.

Command:	**RESET**
Name:	Reset
Purpose:	Resets the hand software. Equivalent to doing a power cycle.
Arguments:	(none)
Example:	RESET
Notes:	

Command:	**ERR**
Name:	Error
Purpose:	If given an argument, lists the errors represented by that argument. If not given an argument, lists all possible error values and descriptions.
Arguments:	<errorNum>
Example:	ERR 3
Notes:	

Command:	**VERS**
Name:	Version
Purpose:	Prints the firmware version.
Arguments:	(none)
Example:	VERS
Notes:	

5.2.5 Advanced Commands

Users do not generally need these commands and should avoid using them. They are not listed by the "?" command; they are only listed by the "A?" command.

Command: **A?**
Name: Help All
Purpose: Lists all of the standard and advanced commands.
Arguments: (none)
Example: A?
Notes: Use the ? command to get a description of an advanced command.

Command: **FLISTA**
Name: Finger List All
Purpose: Lists all of the standard and advanced finger property names.
Arguments: (none)
Example: FLISTA
Notes:

Command: **FLISTAV**
Name: Finger List All Value
Purpose: Lists all of the standard and advanced motor property values for the selected motor(s). Each property has its value(s) printed on one line, with one value for each selected motor separated by spaces.
Arguments: (none)
Example: 3FLISTAV
Notes: Can take a motor prefix.

Command: **PLISTA**
Name: Property List All
Purpose: Lists all of the standard and advanced global property names.
Arguments: (none)
Example: PLISTA
Notes:

Command: **PLISTAV**
Name: Property List All Value
Purpose: Lists all of the standard and advanced global property values. Each property has its value printed on one line.
Arguments: (none)
Example: PLISTAV
Notes:

5.3 Motor Properties

Motor properties are used to determine or retrieve motor-related settings.

5.3.1 Movement

Movement properties affect how a given motor moves.

Property: **BDAT**
Name: Breakaway Detection Acceleration Threshold
Purpose: Used to adjust the breakaway detection accuracy
Values: 0 to 20000
Default: Finger: 1500
Spread: N/A
Notes: Units of BDAT are: Encoder counts/(Ticklet^2). There are 307 ticklets per millisecond.

Property: **BS**
Name: Breakaway Stop
Purpose: Used to stop finger as soon as breakaway has been detected.
Values: 0 or 1
Default: Finger: 0
Spread: N/A
Notes: When set to 1, the finger motor will stop when breakaway is detected.

Property: **DP**
Name: Default Position
Purpose: Destination of M command if no argument specified
Values: 0 to 65,535
Default: Finger: 8500
Spread: 1575
Notes: While DP can be set as high as 65,535, its true range of useful values is bounded by the joint limits of the axes (e.g. approximately 0 to 18,000 for fingers and approximately 0 to 3150 for spread).

Property: **DS**
Name: Default Step
Purpose: Size of IC or IO command movement if no argument specified
Values: 0 to 65,535
Default: Finger: 1700
Spread: 315
Notes: While DS can be set as high as 65,535, its true range of useful values is bounded by the joint limits of the axes (e.g. approximately 0 to 18,000 for fingers and approximately 0 to 3150 for spread).

Property: **HSG**
Name: Highest Strain Gauge Value
Purpose: In O, C, IO, IC, M, TO, TC, and HOME commands, a motor's motion is terminated if its strain gauge value exceeds HSG.
Values: 0 to 256
Default: 256
Notes: This command is an alias for MSG. When writing new code, it is recommended to use this command instead of MSG. A value of 255 or 256 disables the strain gage checking during motion commands.

Property: **LSG**
Name: Lowest Strain Gauge Value
Purpose: In O, C, IO, IC, M, TO, TC, and HOME commands, a motor's motion is terminated if its strain gauge value falls below LSG.
Values: 0 to 256
Default: 256
Notes: A value of 255 or 256 disables the strain gage checking during motion commands.

Property: **MOV**
Name: Maximum Open Velocity
Purpose: Controls the maximum velocity while opening a motor.
Values: 16 to 4080
Default: Finger: 100
Spread: 60
Notes:

Property: **MCV**
Name: Maximum Close Velocity
Purpose: Controls the maximum velocity while closing a motor.
Values: 16 to 4080
Default: Finger: 100
Spread: 60
Notes:

Property: **MSG**
Name: Maximum Strain Gauge
Purpose: In O, C, IO, IC, M, TO, TC, and HOME commands, a motor's motion is terminated if its strain gauge value exceeds MSG.
Values: 0 to 256
Default: 256
Notes: HSG is the preferred alias to MSG. When possible, please use HSG in place of MSG. A value of 255 or 256 disables the strain gage checking during motion commands.

5.3.2 Status

Motor status properties are read-only and give useful motor information and feedback.

Property: **BD**
Name: Breakaway-Detected Flag
Purpose: To determine if breakaway has occurred
Values: 0 or 1
Default: N/A
Notes: Flag is set when a breakaway is detected. Flag is cleared after an O, TO, or HI command.

Property: **BP**
Name: Breakaway Position
Purpose: Stores position of last breakaway
Values: 0 or 20000
Default: N/A
Notes: Stored position of location of last breakaway. Position is cleared after an O, TO, or HI.

Property: **OD**
Name: Odometer
Purpose: The total number of counts traveled by the selected motor, divided by 1000.
Values: 0 to 4 billion
Default: N/A
Notes: This value is never reset; it is maintained through power failures and firmware downloads.

Property:	**P**
Name:	Position
Purpose:	The present position of the motor.
Values:	Finger: 0 to approximately 17,800
Spread:	0 to approximately 3150
Default:	N/A
Notes:	The range of the position property is dependent on the values of the OT (open target) and the CT (close target) properties. If these properties are set beyond the joint stops then the joint stops themselves will dictate the range of position values, in which case the ranges may differ slightly from finger to finger.

Property:	**S**
Name:	Status
Purpose:	The present status of the motor. 0 if ready to be used, or a status code otherwise.
Values:	0, 1, 2, 4,…8192, 16384 and sums of these
Default:	N/A
Notes:	See status codes in section 5.1.3.

Property:	**SG**
Name:	Strain Gauge
Purpose:	The present strain gage value for the motor.
Values:	0 to 255
Default:	N/A
Notes:	Returns 255 if there is no strain gauge.

5.3.3 RealTime

RealTime properties affect exchange of control and feedback data for motors in RealTime mode. See Section 6 for more information on RealTime mode.

Property:	**LCV**
Name:	Loop Control Velocity
Purpose:	If non-zero, then a velocity byte will be sent in the control block for the motor.
Values:	0, 1
Default:	1
Notes:	

Property:	**LCVC**
Name:	Loop Control Velocity Coefficient
Purpose:	When the firmware receives a velocity byte in a control block, it multiplies the value by the value of LCVC before passing it to the affected motor.
Values:	0 to 255
Default:	1
Notes:	

Property:	**LCPG**
Name:	Loop Control Proportional Gain
Purpose:	If non-zero, then a Proportional Gain byte will be sent in the control block for the motor.
Values:	0, 1
Default:	1
Notes:	This controls the constant that is multiplied by the velocity error in order to produce the motor torque.

Property: **LCT**
Name: Loop Control Torque
Purpose: If non-zero, then a signed 2-byte torque will be sent in the control block for the motor.
Values: 0, 1
Default: 0
Notes: Motor torque is set using position mode by setting the commanded position reference equal to the actual position plus the desired torque value.

Property: **LFAIN**
Name: Loop Feedback Analog Input
Purpose: If non-zero, then a 1-byte unsigned analog value will be sent in the control block for the motor.
Values: 0, 1
Default: 0
Notes:

Property: **LFBP**
Name: Loop Feedback Breakaway Position
Purpose: If non-zero, then the firmware sends two unsigned bytes giving the most recently recorded breakaway position for the motor.
Values: 0, 1
Default: 0
Notes:

Property: **LFV**
Name: Loop Feedback Velocity
Purpose: If non-zero, then the firmware sends a signed byte giving the present velocity for the motor divided by the LFVC property.
Values: 0, 1
Default: 1
Notes:

Property: **LFVC**
Name: Loop Feedback Velocity Coefficient
Purpose: Before sending a Loop Feedback Velocity byte, the firmware divides the velocity by this property's value.
Values: 0 to 255
Default: 1
Notes:

Property: **LFS**
Name: Loop Feedback Strain
Purpose: If non-zero, then the firmware sends an unsigned byte giving the strain gauge value for the motor.
Values: 0, 1
Default: 1
Notes:

Property: **LFAP**
Name: Loop Feedback Absolute Position
Purpose: If non-zero, then the firmware sends an unsigned two-byte value giving the present position of the motor.
Values: 0, 1
Default: 1
Notes:

Property: **LFDP**
Name: Loop Feedback Delta Position
Purpose: If non-zero, then the firmware sends a signed byte giving the change in position since the last datum, divided by the value of the LFDPC property.
Values: 0, 1
Default: 1
Notes: A conflict occurs when the change in position is too great to transmit in a single signed byte, even after scaling. See section 6.3.3 for more information.

Property: **LFDPC**
Name: Loop Feedback Delta Position Coefficient
Purpose: Used to scale a delta position value.
Values: 0 to 255
Default: 1
Notes: Delta position is the change in position from the last reported position and is limited to one signed byte. The Present position is read and compared to the last reported position. The difference is divided by the RealTime variable LFDPC, clipped to a single signed byte, and then sent to the host. The value sent to the host should be multiplied by LFDPC and then added to the last reported position

5.3.4 Advanced

Users do not generally need these properties and should avoid using them. They are not listed by the FLIST or FLISTV commands; they are only listed by the FLISTA and FLISTAV commands.

Property: **ACCEL**
Name: Acceleration
Purpose: Maximum acceleration and deceleration when moving from one position to another.
Values: 0 to 65,535
Default: Fingers: 4
Spread: 2
Notes: While the ACCEL property has a rather large range of values that it can accept, the motor can only follow a small subset of those values. The range of the subset is based on the motor control properties (e.g. SAMPLE, FPG, FDZ, FIP). See http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf.

Property: **CT**
Name: Close Target
Purpose: This is the position gone to by a C (“Close”) command.
Values: 0 to 65,535
Default: Finger: 17,000
Spread: 3150
Notes: While CT can be set as high as 65,535, its true range of useful values is bounded by the joint limits of the axes (e.g. approximately 0 to 18,000 for fingers and approximately 0 to 3150 for spread).

Property: **EN**
Name: Enabled
Purpose: If non-zero, then a motion command with no motor prefix will act on this motor.
Values: 0, 1
Default: 1
Notes:

Property: **FDZ**
Name: Filter Derivative Zero
Purpose: Used to calculate the desired motor torque.
Values: 0 to 255
Default: 0
Notes: FPG sets B in the HCTL-1100 controller, which is applied as specified in Equation 1 on page 23 of: http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf. Setting this property to a non-zero value increases susceptibility to random high-frequency noise.

Property: **FIP**
Name: Filter Integral Pole
Purpose: Used to calculate the desired motor torque.
Values: 0 to 255
Default: 0
Notes: FIP sets A in the HCTL-1100 controller, which is applied as specified in Equation 1 on page 23 of: http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf. Setting this property to a non-zero value can drive the system unstable.

Property: **FPG**
Name: Filter Proportional Gain
Purpose: Used to calculate the desired motor torque.
Values: 0 to 255
Default: 10
Notes: FPG sets K in the HCTL-1100 controller, which is applied as specified in Equation 1 on page 23 of: http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf.

Property: **HOLD**
Name: Hold
Purpose: If non-zero, then the motor is left energized after each motion command in order to hold the position constant.
Values: 0, 1
Default: Finger: 0
Spread: 1
Notes: Since the fingers are not back-drivable, this is generally set to 1 only for the spread motor.

Property: **IHIT**
Name: Initialization hit count
Purpose: If non-zero, then while initializing the motor impacts the hard stop the given number of times.
Values: 0 to 65,535
Default: Finger: 2
Spread: 0
Notes: Barrett recommends not setting IHIT to a value greater than 5. IHIT is used to get a consistent origin for the finger motors, and thus a consistent breakaway force.

Property: **IOFF**
Name: Initialization Offset
Purpose: This is the distance this motor's origin is shifted away from the full open position.
Values: 0 to 65,535
Default: Finger: 50
Spread: 0
Notes: Adjusting this value on a finger motor also affects the force required to cause breakaway of the TorqueSwitch™ clutch. Larger values result in less compression of the Belleville washer, and so result in lower breakaway forces; smaller values similarly result in higher breakaway forces. Although IOFF can be set as high as 65,535, its true range of useful values is from 0 to approximately 500.

Property: **IVEL**
Name: Initialization Velocity
Purpose: This value replaces MOV ("Motor Open Velocity") during initialization; this allows a consistent initialization velocity even if MOV is adjusted.
Values: 16 to 4080
Default: Finger: 300
Spread: 150
Notes: Barrett Technology does not recommend increasing IVEL beyond its default value. Permanent deformation of Bellville washers could result. This would prevent the finger breakaway mechanism from engaging.

Property: **MPE**
Name: Maximum Position Error
Purpose: After moving to a desired position, if the position error is less than MPE then the move is considered a success.
Values: 0 to 65,535
Default: 50
Notes: While MPE can be set as high as 65,535, its true range of useful values is bounded by the joint limits of the axes (e.g. approximately 0 to 18,000 for fingers and approximately 0 to 3150 for spread).

Property: **OT**
Name: Open Target
Purpose: This is the position gone to by an O ("Open") command.
Values: 0 to 65,535
Default: 0
Notes: While OT can be set as high as 65,535, its true range of useful values is bounded by the joint limits of the axes (e.g. approximately 0 to 17,800 for fingers and approximately 0 to 3150 for spread).

Property: **SAMPLE**
Name: Sample Time
Purpose: Controls the sample frequency of the motor controller chip.
Values: 15 to 255
Default: 31
Notes: Written into the HCTL-1100 Sample Timer register according to See http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf. (The HCTL-1100 clock speed used in the BH8-series Hand is 2.00 MHz.) Barrett Technology does not recommend changing this property from default.

Property: **SGFLIP**
Name: Strain Gage Flip
Purpose: If non-zero, then all strain gage values for this motor are subtracted from 255 before being sent to the host.
Values: 0, 1
Notes: Used to invert strain gauge readings. This is a legacy command and may not be supported in future releases. It is strongly recommended that you avoid using SGFLIP since it may be dropped in future revisions of the firmware.

Property: **TSTOP**
Name: Time to Stop
Purpose: Time in milliseconds before motor is considered stopped.
Values: 0 to 65,535
Default: 30
Notes: WARNING: Please use caution when adjusting this property. Setting TSTOP higher than its default can result in the motors heating up very quickly under moderate to heavy usage.

5.4 Global Properties

Global properties are used to configure or observe the hand as a whole.

5.4.1 Configuration

Global configuration properties affect the hand as a whole.

Property: **BAUD**
Name: Baud rate
Purpose: Controls the serial port baud rate. Value is baud rate divided by 100.
Values: 6, 12, 24, 48, 96, 192, 384.
Default: 96
Notes:

Property: **LFT**
Name: Loop Feedback Temperature
Purpose: If non-zero, then when in RealTime mode the firmware sends a signed two-byte datum of temperature in each feedback block.
Values: 0, 1
Default: 0
Notes:

Property: **OTEMP**
Name: OverTemperature
Purpose: If non-zero, then if the temperature exceeds this value then any motor command fails with an overtemperature error.
Values: 0 to 1250
Default: 0
Notes: Value is temperature in tenths of a degrees C.

5.4.2 Status

Global status properties are read-only and give information about the state of the hand.

Property: **TEMP**
Name: Temperature
Purpose: The present temperature on the CPU board in tenths of a degree C.
Values: -550 to 1250
Default: N/A
Notes:

Property: **PTEMP**
Name: Peak Temperature
Purpose: The maximum temperature ever experienced by this hand
Values: 0 to 1250
Default: N/A
Notes: This value is never reset; it is maintained through power failures and firmware downloads.

Property: **UPSECS**
Name: Uptime Seconds
Purpose: The total power-up time for this hand.
Values: 0 to 4 billion
Default: N/A
Notes: This value is never reset; it is maintained through power failures and firmware downloads. This property can accommodate 136 years of power-up time before rolling over.

Property: **SN**
Name: Serial Number
Purpose: The serial number of the hand.
Values: N/A
Default: N/A
Notes: This value is never reset; it is maintained through power failures and firmware downloads. Hands upgraded to firmware version 4.2 in the field will have a serial number of 0.

5.4.3 Advanced

Property: **LFDPD**
Name: Loop Feedback Delta Position Discard
Purpose: If non-zero, then in RealTime mode any position change that cannot be sent in a delta position datum is discarded. If zero, then any unsent position change is accumulated for transmission in the next cycle.
Values: 0, 1
Default: 0
Notes:

5.5 Termination Conditions for Movement Commands

There are eight commands in Supervisory mode that control finger motion:

- Position commands, absolute: M and HOME
- Position commands, relative: IO, IC.
- Velocity commands with ramp-down at target limits: O, C
- Velocity commands with control of motor torque at stall: TO, TC

In all cases, the command terminates and returns when, for every motor specified in the command, one of the following termination conditions applies:

Case 1: Motor stalls because obstacle(s) stop the motion. The obstacles include foreign objects as well as joint stops and other fingers.

- When a motor stalls, the controller will continue driving it for TSTOP milliseconds, after which a termination condition occurs.
- If HOLD is false, the motor is then turned off; if HOLD is true, the motor is servoed to maintain this position.
- For position commands, if the termination position is not within MPE (Maximum Position Error), the status code ERR16 is returned corresponding to "Couldn't reach position."

Case 2: The strain-gage range is exceeded.

- As soon as the sensor value SG rises above HSG (alias MSG) or falls below LSG, a termination condition occurs for that motor.
- If HOLD is false, the motor is then turned off; if HOLD is true, the motor is servoed to maintain the position at which the termination condition occurred.
- For position commands, if the termination position is not within MPE (Maximum Position Error), the status code is ERR16 is returned corresponding to "Couldn't reach position".
- From the PC side of the interface, Case 2 can be distinguished by reading the value of the strain gage, SG, and seeing if it is larger than the specified MSG.

Case 3: Specified goal position is achieved within MPE.

- As soon as the goal position is reached, a termination condition occurs for that motor.
- If HOLD is false, the motor is then turned off; if HOLD is true, the motor is servoed to maintain the position at which the termination condition occurred.
- No status code is returned, and a query to property S returns 0.

Case 4: Control-C character sent

- As soon as the Control C is received, a termination condition occurs for all motors.
- For each motor, if HOLD is false, that motor is then turned off; if HOLD is true, that motor is servoed to maintain the position at which the termination condition occurred.
- The status code returned is 16384.

5.6 BH8-280 Implementation

The BarrettHand API is designed to provide capabilities of what was offered in previous hands and more. Just as properties were used in the 262 hand, the new 280 hand contains Puck 2 motor properties that are written or read through a host PC connected with CAN nodes contained in the hand. The API is meant bring all functionality available to users without the need to know the low-level CAN communication protocol. The same BarrettHand Control GUI connects to all BarrettHands and uses the same Supervisory control interface. The same API methods are called. The GUI provides the purpose/notes of most properties that users will commonly access. See the API documentation for how to use commands in C++ code.

The BH8-280 hand implements each of the important commands found in the earlier BH8-Series hands to offer compatibility. These commands are found in the new hand:

- Movement commands: C, HI, HOME, IO, IC, LOOP, M, O, T, TC, TO
- Motor property commands: FSET, FGET, FLOAD, FSAVE, FDEF
- Global property commands: PSET*, PGET*
- Administrative commands: RESET

Note that PSET and PGET are commands for setting and getting global property values. The 280 hand will set all properties to the given value for PSET commands and will return -9 for a PGET command. The API does contain some virtual properties like the global "LFT" loop feedback temperature property. Virtual properties are read and written to variables on the host PC. The following commands have been left out for the 280 hand and probably won't be available because there is no interactive mode via command prompt or global properties.

- Motor property commands: FLIST, FLISTV
- Global property commands: PLOAD*, PSAVE*, PDEF, PLIST, PLISTV
- Global property commands:
- Administrative commands: ?, ERR, VERS
- Advanced commands: a?, FLISTA, FLISTAV, PLISTA, PLISTAV

The hand API provides a special "Command" method for processing a subset of the Grasper Control Language (GCL) character strings containing commands and returns a response to commands in a user supplied receive buffer. The string is parsed into commands that may contain a motor string prefix and possibly some parameters. This allows significant backwards compatibility and allows sending commands in the form of strings and receiving responses from the hand in the form of a string. A list of the presently accepted strings include "RESET" and:

- "HI"
- "C", "O"
- "M <optional position parameter>"
- "HOME"
- "IC <optional step size parameter>", "IO <optional step size parameter>"
- "TC", "TO"
- "FSET <property> <value>"
- "FGET <property>"
- "FLOAD, FSAVE, FDEF"
- "T"
- "LOOP"

The list of properties available on 280 hands is a subset of Puck 2 hand properties and virtual properties such as the ones that contain RealTime variables. Refer to Section 6 for more information on RealTime properties. The API contains documentation for 262 and 280 hands in separate source code files for each hand. The documentation is easily accessible through the supervisory tab of the BarrettHand Control GUI. The programmer no longer needs to flip through this manual to find hand properties and their purpose, descriptions, notes, etc. The property list that each hand contains provides several useful attributes such as being read-only, global, and also a range of values that is used to validate inputs to some commands.

6 RealTime Control

RealTime mode, also known as Loop-Control mode, is the second control method for the BarrettHand™. This control mode allows you to send control data and receive feedback data continuously, without waiting for the motors to stop moving. Any desired control law can be implemented within the host computer by calculating the desired motor control reference, sending the control block with the control reference to the hand, waiting for the requested feedback data, and then repeating this update process. The control bandwidth is a function of the amount of control data sent, the amount of feedback data requested, and how fast the communication is.

Control data from the host computer to the hand is grouped into control blocks; feedback data is grouped into feedback blocks. The structure of the control and feedback blocks is set by various finger and global properties. These are mostly Boolean flags but some are integer coefficients. The structure can only be changed in Supervisory mode; it cannot be changed while in RealTime mode. The BH8-280 hand operates similarly by setting a desired control reference and then sending a batch Get property for each feedback property that is to be received.

6.1 Pucks in Hand API

RealTime mode for the BH8-280 hand is controlled by the host PC through the BarrettHand API so this section focuses on documenting this. The API contains virtual properties that handle what data is sent or requested from the hand. Virtual properties behave just like properties that are read and written on the Pucks and include the Boolean flag properties “LCV”, “LCT”, “LCP”, “LFAP”, “LFV”, and “LFS”. These are set during original RTSetFlags calls or motor Set property calls. These flags control the sending of control references and feedback from the Hand. There can only be a single control reference being sent in RealTime mode. Control references include motor reference velocities, torques, and positions. There can be multiple feedback values requested such as actual positions, velocities, and strain values. Presently, velocity feedback is calculated on the host PC automatically when the “LFV” is set.

When the listed control and feedback flags are set, the RTStart method is called to begin RealTime mode. RTStart accepts a second parameter to determine the desired levels of motor protection. A positive value for the TSTOP property, in units of milliseconds, will terminate motors after TSTOP milliseconds if they don't move. Motors will drop into idle mode if HOLD is false and will servo to the present motor position if HOLD is true. This motor protection scheme is excellent at protecting motors from stalls that would normally harm the motor. The drawback to using it is that it also drops the motor mode to idle so the motor leaves RealTime control up to the user afterwords meaning that they would need to manually set the Puck in the hand back to the desired motor mode. The second motor protection scheme will exponentially filter the square of each motor current and limit the maximum continuous motor current to presumably safe levels. This seems to offer significant advantage in that motors will not give up trying to achieve their desired motor references even with limited torque. The downside to this method is that torque is limited by the host by setting the Puck's MT property and if the program does hang then motor torque may not be limited. The user is advised to be cautious about protecting the hand motors.

RealTime method RTUpdate is called to send control data to the hand and/or receive the desired feedback from the hand. Feedback is received during RTUpdate calls using a batch Get property for higher efficiency. RTUpdate is also overloaded to provide a batch Get motor property method that efficiently sends Get motor property messages to the desired Pucks in the hand and blocks until property values are received. This is provided for reading additional Puck 2 sensors such as inner link joint positions from the “JP” property or 12-bit strain values from the “SG” property. The alternative RTUpdate method may also be used to get finger/spread positions or other Puck properties while running Supervisory commands for 280 hands. This is possible by registering an

appropriate callback function with the BHand setWaitCallbackFunc method. This callback method will be called frequently while Supervisory commands block until movement is done or with the API delay command. This has not been possible with earlier hands.

6.2 Overview of Serial Protocol for Earlier Hands

To enter RealTime mode, the host computer sends the Supervisory mode "LOOP" command, with an optional motor prefix specifying the motors to be controlled. The hand responds with an acknowledgment character ("*"), and then awaits control blocks, with or without control data. When a control block is received, if the control block requests a feedback block, then transmission of the feedback block is started. Once the complete control block is received it is acted upon, and then the hand waits for the next control block. The host should not send a second control block until the first one is acknowledged.

If the Hand software encounters an error, then the next time the Hand would send the "*" acknowledgment character to the host it instead sends "<CRLF>ERR" followed by the error value. It then returns to Supervisory mode.

To terminate RealTime mode, the host should send a single ^C character instead of the header character. This returns the Hand to Supervisory mode.

6.3 Control and Feedback Blocks

Control and feedback blocks consist of a header character, followed if desired by control data. If control data is included then it is sent for each motor selected for the LOOP command, in motor number order, followed by any global datum. For each motor, any of a set of data can be included. Whether or not a specific piece of data should be included is controlled by one of ten flag properties: "LCV", "LCPG", "LCT", "LFAIN", "LFBP", "LFV", "LFS", "LFAP", "LFDP" and "LFT." If a given property is true then its corresponding datum is included in the block; if not, then it is omitted. Four other properties, "LCVC", "LFVC", "LFDPC", and "LFDPD", modify specific data items.

6.3.1 Control Blocks

Control data from the host to the hand is grouped into control blocks. Each control block has a single byte header, optionally followed by a set of control data. The header specifies whether or not control data is to follow, and whether or not a feedback block is to be returned. The header can also terminate RealTime mode.

The possible header byte values are:

'C'	Control data follows; respond with a feedback block
'c'	Control data follows; respond with an acknowledgment character
'A'	No control data follows; respond with a feedback block
'a'	No control data follows; respond with an acknowledgment character
<^C>	Terminate RealTime mode

If the 'C' or 'c' header is used, then the header should be followed by control data. The user needs to choose the control mode themselves by setting LCV and LCPG flags to 1 or 0. If either of these are 1 then LCT must be 0 and if LCT is 1 then LCV and LCPG must be 0. Only one active control mode per motor is allowed. For each motor, the following different data values should be included in order if their corresponding flags are true:

- LCV "Loop Control Velocity" Signed, 1 byte
- LCPG "Loop Control Proportional Gain" Unsigned, 1 byte
- LCT "Loop Control Torque" Signed, 2 bytes

The control data should be sent in a specific order: first all data for motor 1, then all for motor 2, then motor 3, and finally motor 4. Note that if a given motor was not specified in the initiating LOOP command, or if a specific value isn't enabled by the corresponding finger property, then the corresponding datum should not be transmitted.

If the LCV datum is included, then the hand will multiply it by the property LCVC before passing it on to the motor. Note: the hand treats an unscaled LCV datum as 4 bits of integer and 4 bits of fraction; this is different from an unscaled LFV datum, which is all integer.

6.3.2 Feedback Blocks

Data from the hand to the host is grouped into feedback blocks. Each feedback block has a single byte header ("*"), followed (if requested) by a set of feedback data. If the hand has encountered an error, then the header is replaced by "<CRLF>ERR ", followed by the error number; the hand then returns to supervisory command mode.

For each selected motor, six different data values are included in order if their corresponding finger properties are non-zero:

- LFV "Loop Feedback Velocity" Signed, 1 byte
- LFS "Loop Feedback Strain" Unsigned, 1 byte
- LFAP "Loop Feedback Absolute Position" Unsigned, 2 bytes
- LFDP "Loop Feedback Delta Position" Signed, 1 byte
- LFBP "Loop Feedback Breakaway Position" Unsigned, 2 bytes
- LFAIN "Loop Feedback Analog Input" Unsigned, 1 byte

In addition to the motor feedback data, there is a single global feedback datum, which is sent if its corresponding global property is non-zero:

LFT "Loop Feedback Temperature" Signed, 1 byte

The feedback data are sent in a specific order: first all data for motor 1, then all for motor 2, then motor 3, then motor 4, then any global datum. Note that if a given motor was not specified in the initiating LOOP command, or if a specific value isn't enabled by the corresponding finger or global property, then the corresponding datum is not transmitted.

If the LFV datum is included, then the hand will divide it by the property Loop LFVC before sending it to the host. Note: the hand treats an unscaled LFV datum as all integer; this is different from an unscaled LCV datum, which is treated as 4 bits of integer and 4 bits of fraction.

6.3.3 Loop Feedback Delta Position

The LFDP ("Loop Feedback Delta Position") datum is a special case. Each time a motor's position is queried using "FGET P", the reported position is remembered. In loop mode, if the LFDP property is non-zero then the present position is read and compared to the previously reported position. The difference is divided by the LFDPC ("Loop Feedback Delta Position

Coefficient") property, clipped to a single signed byte, and then sent to the host. The host should then multiply the received value by LFDPC and then add it to the reported position.

The problem with using delta position is that the reported position can change at most by +127/-128 in each cycle. If the motor position changes more than this in a single cycle then the reported position will lag behind the actual position.

Example: say LFDPC is 2, the last reported position was 1500, and the position suddenly jumps to 2000. The first feedback block will include the delta position datum 127, which actually means 254; the hand will internally update the reported position to 1754. The next feedback block will include the delta position 123, which actually means 246; the reported position will be updated to 2000. Subsequent feedback blocks will include the delta position value 0 (until the next position change).

If desired, any unreported position change can be discarded by setting the LFDPD ("Loop Feedback Delta Position Discarded") global property to true. With this set, the above example would result in the single value 127 being sent to the host, followed by 0s.

6.4 Property Summary

Table 4 and Table 5 is a summary of the different motor and global properties, which affect RealTime mode. Most of the properties are flags, specifying whether a specific datum is to be present in a control or feedback block. The four remaining properties are coefficients or flags, which affect how the firmware interprets or generates a datum.

Table 4 - RealTime Finger Control Properties for the BH8-262

Property	Name	Type	Function	Size in Block
LCV	Loop Control Velocity	Flag	If True, RealTime control block will contain control velocity	1 signed byte
LCVC	Loop Control Velocity Coefficient	Coefficient (1 to 255)	LCV is multiplied by LCVC to determine control velocity	N/A
LCPG	Loop Control Proportional Gain	Flag	If True, RealTime control block will contain Proportional Gain	1 unsigned byte
LCT	Loop Control Torque	Flag	If True, RealTime control block will contain control torque	2 bytes, signed
LFAIN	Loop Feedback Analog Input	Flag	If True, RealTime feedback block will contain analog input value	1 unsigned byte
LFBP	Loop Feedback Breakaway Position	Flag	If True, RealTime feedback block will contain breakaway position	2 unsigned bytes
LFV	Loop Feedback Velocity	Flag	If True, RealTime feedback block will contain feedback velocity	1 signed byte
LFVC	Loop Feedback Velocity Coefficient	Coefficient (1 to 255)	Actual velocity is divided by LFVC to get LFV	N/A
LFS	Loop Feedback Strain	Flag	If True, RealTime feedback block will contain strain information	1 unsigned byte
LFAP	Loop Feedback Absolute Position	Flag	If True, RealTime feedback block will contain absolute position	2 unsigned bytes
LFDP	Loop Feedback Delta Position	Flag	If True, RealTime feedback block will contain delta position	1 signed byte
LFDPC	Loop Feedback Delta Position Coefficient	Coefficient (1 to 255)	The actual delta position is divided by this to get LFDP	N/A
LFDPD	Loop Feedback Delta Position Discard	Flag	If true, any delta position overflow is discarded	N/A

Table 5 - RealTime Global Control Properties

Property	Name	Type	Function	Size in Block
LFT	Loop Feedback Temp.	Flag	If True, RealTime feedback block will contain temperature	1 unsigned byte

6.5 Example

This application uses fingers 1 and 2, and the spread. The fingers will receive velocity control information and report strain and delta position. The spread will just report delta position. The feedback block will also include the present hand temperature. All relevant coefficients will be set to 1.

To set this, use the following commands:

12FSET LCV 1 LCVC 1 LCPG 0 LCT 0 LFV 0 LFS 1 LFAP 0 LFDP 1 LFDPC 1

4FSET LCV 0 LCT 0 LCPG 0 LFV 0 LFS 0 LFAP 0 LFDP 1 LFDPC 1

PSET LFT 1

124LOOP

The hand will then send a single "*" and wait for control blocks. Each control block will consist of three bytes:

"C" [Control data follows; respond with feedback block]

1 signed byte of velocity for motor 1

1 signed byte of velocity for motor 2

Each feedback block will consist of seven bytes:

"*"

1 unsigned byte of strain for motor 1

1 signed byte of delta position for motor 1

1 unsigned byte of strain for motor 2

1 signed byte of delta position for motor 2

1 signed byte of delta position for motor 4

1 unsigned byte of temperature

Each control block from the host will stimulate a feedback block from the hand. When the host is finished, it will send the single character ^C (0x03); the hand will respond by printing a prompt and waiting for a new command.

7 Maintenance

7.1 Finger Cable Pretension

The third joint in each finger is driven by a brushless servomotor through opposing stainless steel cables that act like tendons transmitting torque from a pulley at the base of the finger out to a pulley at the fingertip joint. If you have purchased the joint-torque sensor option, the difference in tension between the tendon pair is used to determine the torque at the third joint. The fact that we measure the tensions differentially reduces the effect of actual pretension in the cable as long as the cable is not actually loose.

Under normal circumstances, the cables remain pretensioned indefinitely. But under heavy use over thousands of cycles, it can begin to relieve its pretension. You can easily readjust the pretension through Barrett Technology's patented cable tensioning mechanism as follows:

1. Loosen the hex set screw with the right angle hex wrench provided in the maintenance kit. This screw is located adjacent to the termination of the joint 3 cable on each of the fingers as seen in Figure 11.

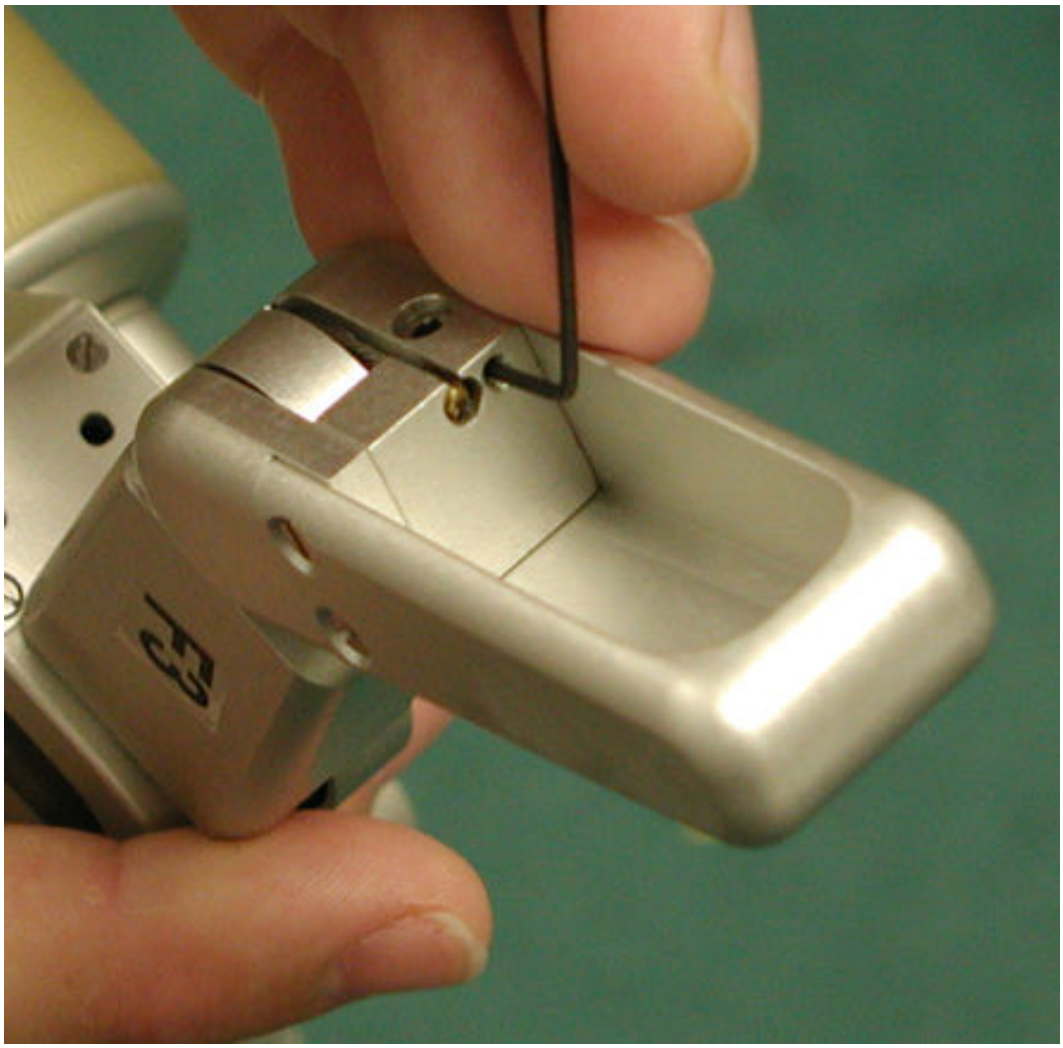


Figure 11 – Hex set screw

2. Apply 15 oz-in of clockwise torque to the tensioner screw located on the back of each Joint 3 housing as seen in Figure 12. A 2-mm hex torque wrench is provided in the maintenance kit for this purpose.

CAUTION:

The tendon is properly tensioned when all loose slack has been removed and you can feel the direct connection of the fingertip to its drive gears. DO NOT OVER-TIGHTEN THE TENDON! The pretensioning mechanism is stronger than the tendon and is capable of snapping it if over-tightened. Excessive pretension will change the frictional properties in the finger drives and may reduce the finger's range of motion.

3. Retighten the hex set screw until it is snug against the tensioner screw.

NOTE:

It is advisable to completely remove the hex set screw to apply Loctite 222 to its threads before retightening it against the tensioner screw. This measure is especially important if the hand is under heavy use.

Figure 12 - Pretensioning the Tendon Cable

7.2 Fastener Check

All screw fasteners in the BarrettHand™ have been installed with a thread locker, which should prevent loosening over the life of the product. However, after prolonged use, Barrett Technology recommends that you conduct a precautionary inspection to ensure all external fasteners are in place and tight. Ideally, this inspection should occur monthly under heavy use conditions.

Should any fasteners have become dislodged during operation, contact Barrett Technology for replacements or replacement specifications. Do not replace fasteners without contacting Barrett Technology as many fasteners have strict length specifications.

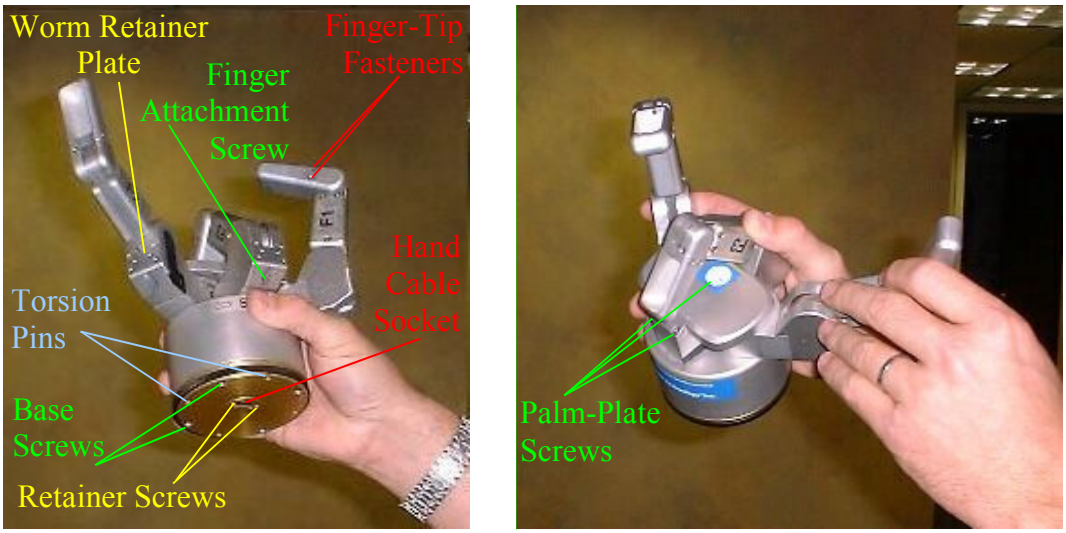


Figure 13 - Important Fastener Locations

7.3 Lubrication

Each BarrettHand™ unit has been lubricated and tested prior to shipping. Periodically, lubrication must be reapplied to areas with high probability of lubricant flow. Use the grease syringe to apply Mobil 1® Synthetic Grease (both included with the maintenance kit) to all exposed gear teeth at the application points according to Figure 14 and the schedule in Table 6.

NOTE for BH8-262 hands:

Issuing the following command to the hand will return the number of encoder counts, divided by 1000, on each motor.

fget od

The number of joint cycles can be determined using the following conversions.

Fingers 1, 2, 3: (encoder counts) / 17,500
Spread: (encoder counts) / 3,100

Table 6 - Lubrication Schedule

Application Point	Maintenance Cycle
Finger Worm Gears	5000 cycles
Finger Spur Gears	5000 cycles
Finger Motor Spur Gears	5000 cycles
Palm Spur Gears	5000 cycles

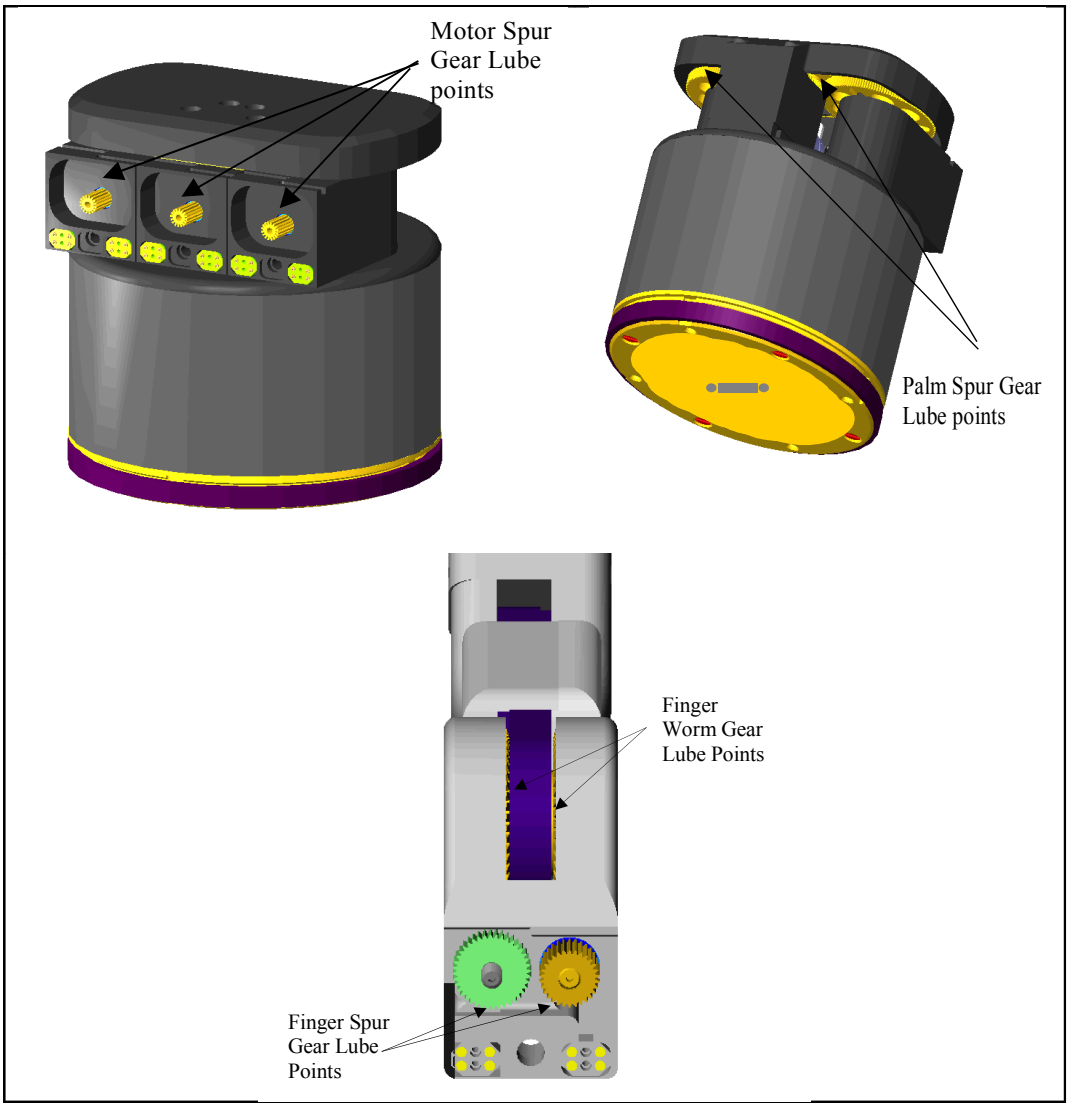


Figure 14 - Lubricant Application Points

Lubricating the finger spur gears requires caution, because you must remove each finger from the palm assembly to access this application point. See the section below on how to detach, lubricate, and reattach the finger.

7.3.1 Finger Replacement

Read all steps below before conducting this maintenance. Pogo pins that connect to sensors in the hand may shear unless the finger is detached and reattached with care. It is best to lubricate only one finger at a time.

Step 1: Turn off power to Hand.

Step 2: Open all fingers on the BarrettHand and open the Spread completely (fingers 1 and 2 opposite finger 3). The fingers can be opened manually using a 2 mm hex wrench in the right-hand hole shown. See Figure 15.

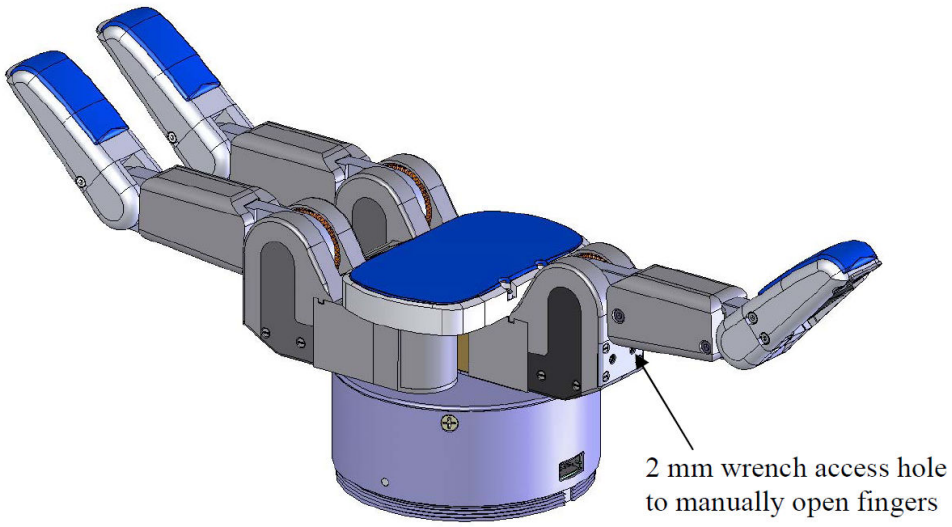


Figure 15: Prepare BarrettHand

Step 3: Locate shoulder screw that connects the finger to the hand. See Figure 16.

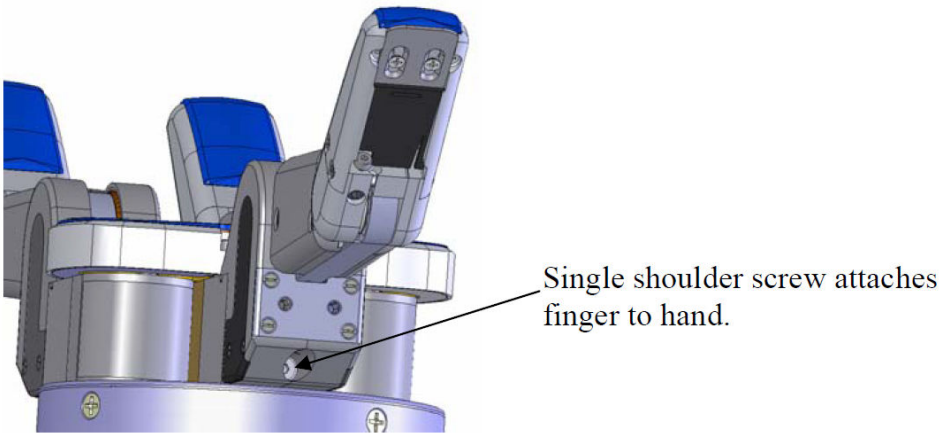


Figure 16: Locate shoulder screw

Step 4: Using a 2 mm hex wrench, unscrew the shoulder screw. Turn the hand sideways and slap the base of the BarrettHand with your palm to get the shoulder screw to fall out.

Step 5: Slide the finger assembly up slightly (Figure 17a), then pivot it out (Figure 17b).

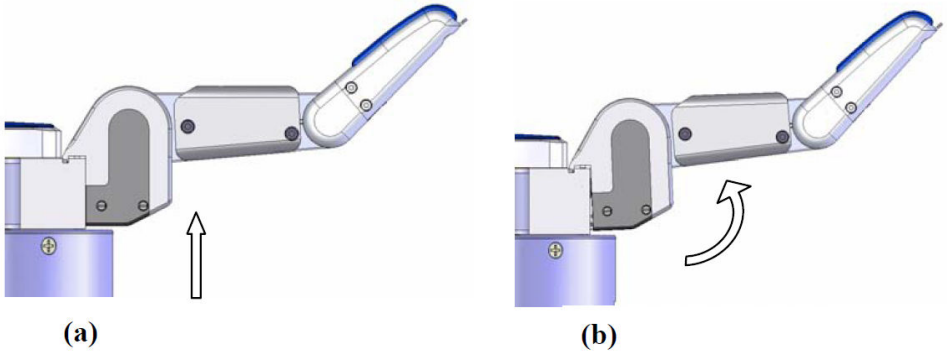


Figure 17: Gently lift and rotate finger off

If the joint-torque sensor option is installed, BE CAREFUL not to damage the gold-plated electrical contact pins when disengaging the teeth. Do not twist or rock the finger when removing or attaching it.

Step 6: Slowly pull finger away from the motor body.

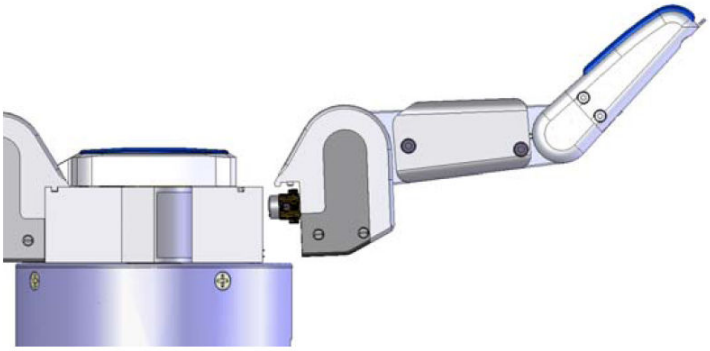


Figure 18: Slowly pull finger away from motor body.

Step 7: Take the replacement finger and verify that the inner link is driven into its stop. Using a 2 mm hex wrench, insert it into the right-hand hole and rotate counter-clockwise until the finger stops moving. See Figure 19.

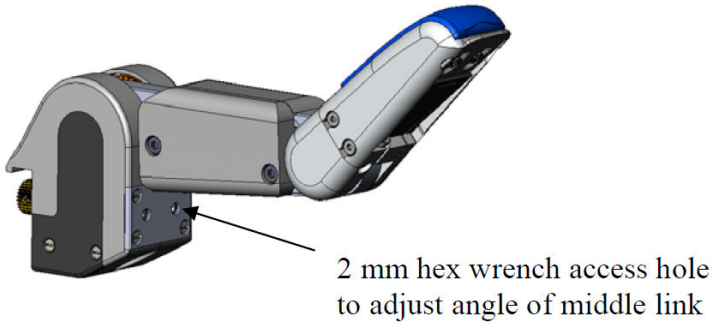
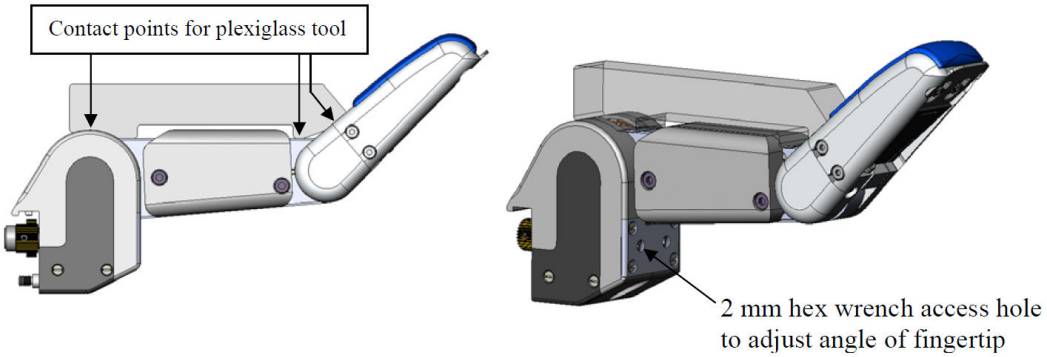


Figure 19: Adjust angle of 2nd link.

Step 8: Check/adjust the angle of the fingertip using the plexiglass finger-angle tool. Place the tool as shown (Figure 20). Using a 2 mm hex wrench, insert it into the left-hand hole and rotate until the finger stops up against the forward edge of the plexiglass tool.



For BH8-262 hands only: Should either link, or the spur gears to which they are attached, move after the finger has been removed, the fingertip position must be reset. Use a 2-mm hex wrench to manually rotate the Joint-3 drive 5 1/2 revolutions from the position where both links are inline and horizontal.

Figure 20: Check/adjust the finger angle

Step 9: If necessary, add Mobil 1® Synthetic grease to the motor body cavity using the syringe provided. Cover all gear teeth with a thick bead of grease. Reverse Steps 5 and 6 to attach the new finger. Be careful not to damage (shear) the electrical “pogo” pins at the base of the motor (Figure 21).

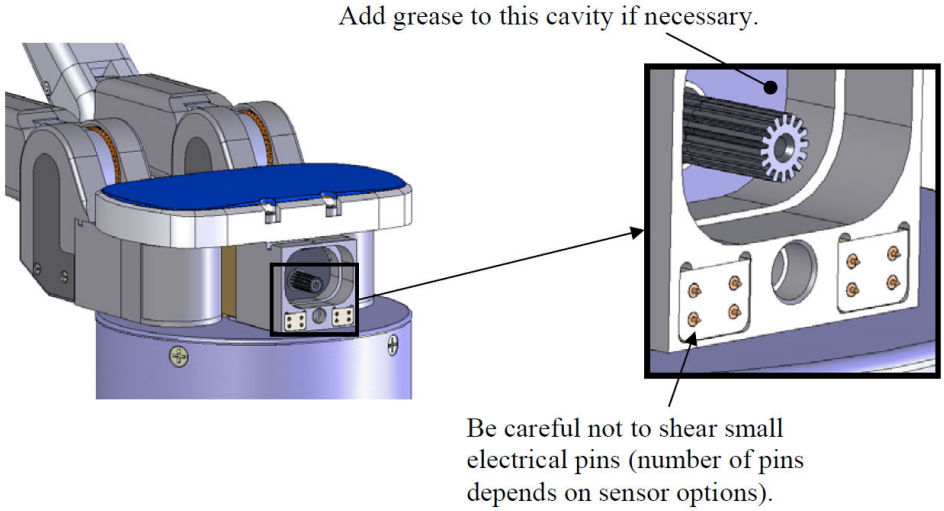


Figure 21: Close-up of electrical "pogo" pins

Step 10: **Once the finger is in place, apply slight pressure down and screw in the shoulder screw.** See Figure 22.

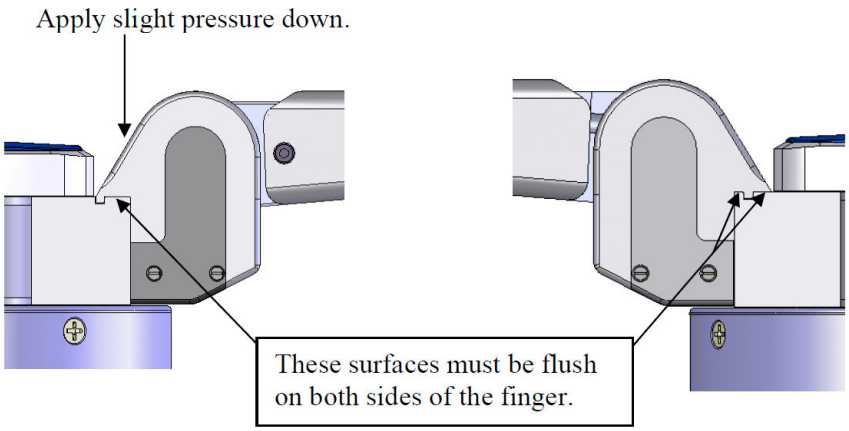


Figure 22: Apply downwards pressure and secure shoulder screw.

Step 11: Verify that the finger operates smoothly. Using a 2mm hex wrench in the right-hand hole (see Figure Error: Reference source not found), drive the finger manually through its range of motion.

Finger replacement is complete.

7.4 Strain Gages

Due to variations in materials, manufacturing and external forces, the strain gage values may change. These changes will affect the zero force reading for each beam differently. To maintain consistent results, the zero force reading needs to remain constant. Each strain gage is equipped with a balancing potentiometer. Adjusting the balancing potentiometer will change the strain gage output for that finger. Adjust the BH8-262 balancing potentiometer until the no-load value is between 100 and 140. Use the following steps to zero the strain gages:

1. Initialize the BarrettHand™.
2. Terminate the spread motor so it can be moved around the palm. (Issue the "T" command)
3. Remove the Shroud Cover screws shown in Figure 23. Some models of the BarrettHand™ will have four Shroud Cover screws. Remove the Shroud from the finger link.

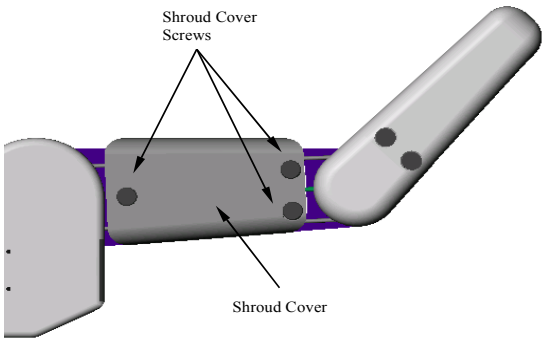


Figure 23 - Shroud Removal

4. BH8-262 users should run the program *Monitor Strain*. This program will continuously sample the strain gage values and print them to the screen. Another option is to run the BarrettHand Control GUI and poll for strain values.
5. Adjust the balancing potentiometer using a small flat head screwdriver until the desired value is reached. The balancing potentiometer requires very small adjustments, due to its sensitivity. Apply as little pressure as possible on the balancing potentiometer during adjustment. See Figure 24.

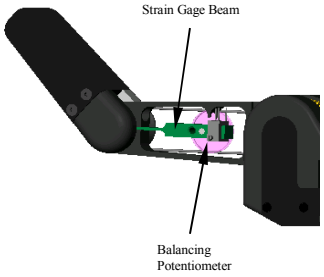


Figure 24 - Balancing Potentiometer

6. After balancing the strain gage, exit the *Monitor Strain* program, put the shroud and shroud cover back on and secure the screws. Be careful not to touch the strain gage or damage any of the electrical wiring when replacing the shroud.

8 Troubleshooting

Most of the symptoms repeated in this section were generated by Barrett's own lab Hands which are assigned to destructive testing over millions of cycles.

Symptom: The Hand behaves erratically. It disobeys some commands while obeying others.

Possible Solution:

1. Reload Firmware for the BH8-262 and earlier hands. If the Hand has been idle for more than a day or two, especially in humid conditions, the SuperCap can lose enough charge that the volatile RAM becomes borderline unstable. There is never a problem if the SuperCap loses *all* of its charge because the Hand simply prompts user to download fresh firmware next time it is powered up. This symptom is most common in Hands that are used infrequently.

Symptom: The host computer will not communicate with the BarrettHand™.

Possible Solution:

1. Verify all connections are secure to the Power Supply, BarrettHand™, and computer.
2. Check that the serial cable is a straight-through extension cable.
3. Verify the Power Supply is turned on.
4. Firmware may no longer be valid. Try downloading the firmware according to Section 3.7.
5. Host computer baud rate and BarrettHand™ baud rate may be set to different rates. Close the BarrettHand Control GUI and reset the BarrettHand™, by cycling power on the BarrettHand Power Supply. Restart the BarrettHand Control GUI and try initializing again.
6. The communications port selected is being used by another program. Close all other programs that use the selected communications port. Reset the BarrettHand™ and restart the BarrettHand Control GUI.
7. If the problem persists, contact Barrett Technology.

Symptom: Initial strain gage values do not fall within specified range.

Possible Solution:

1. The strain gage balancing potentiometer needs to be readjusted. Refer to Section 7.4 for instructions on how to adjust.
2. Verify the cable is riding across both the top and the bottom of the idler pulley, as shown in Figure 25.
3. If the problem persists, contact Barrett Technology.

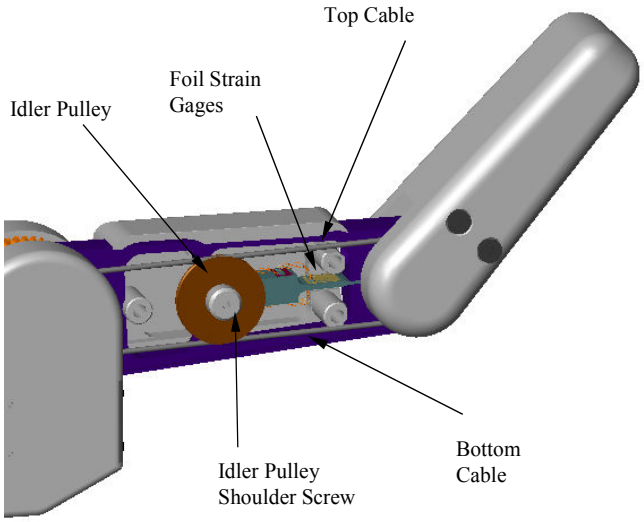


Figure 25 - Cable and Idler Pulley

Symptom: The strain gage values do not follow the expected strain gage curves, shown in Figure 35, while grasping.

Possible Solution:

1. The finger cable pretension is not adjusted properly. Refer to Section 7.1 for instructions on how to adjust.
2. The strain gage balancing potentiometer needs to be readjusted. Refer to Section 7.4 for instructions on how to adjust.
3. Verify the cable is riding properly on the idler pulley, as shown in Figure 25.
4. Verify idler pulley rotates freely on the shoulder screw. The shoulder screw should not be tightened against the idler pulley. If so, loosen shoulder screw, shown in Figure 25, so the idler pulley will move with cable motion.
5. If the problem persists, contact Barrett Technology.

Symptom: Only the fingertip closes when the entire finger should close (Premature Breakaway).

Possible Solution:

1. Verify there is no object blocking the inner link from moving.
2. The finger was not opened completely. Restore that fingers OT, IVEL, IOFF, and IHIT to their default values. Initialize the finger having the problem. The finger should now close properly.
3. If the problem persists, contact Barrett Technology.

Symptom: Finger sticks fully closed.

Possible Solution:

1. Verify there are no objects or other fingers blocking the finger from opening completely.
2. The open velocity is too slow. Try increasing the open velocity to greater than or equal to 40 and opening the finger.
3. Verify that the strain gage value SG is less than HSG (High Strain Gage Limit; alias is MSG).
4. If the strain gages are not installed set HSG to 256 for the BH8-262 and set HSG to 0 for BH8-280 hands.
5. The pretension in the cable is too high. Refer to Section 7.1 to set the finger cable pretension properly.
6. Set the open velocity greater than or equal to 40 and then initialize the finger.
7. Reload firmware.
8. If the problem persists, contact Barrett Technology.

Symptom: Finger sticks open.

Possible Solution:

1. Verify there are no objects or other fingers blocking the finger from closing.
2. The close velocity is set too low. Try increasing the close velocity to greater than or equal to 40 and closing the finger.
3. Set the close velocity to greater than or equal to 40 and then initialize the finger.
4. Reload firmware.
5. If the problem persists, contact Barrett Technology.

Symptom: Finger moves in opposite direction of commanded motion.

Possible Solution:

1. Reload firmware.
2. There is an encoder feedback problem. Reinitializing the finger should solve the immediate problem. If this recurs, contact Barrett Technology for servicing.

Symptom: The TorqueSwitch™ does not breakaway properly, prohibiting the fingertip from completing a form grasp around an object.

Possible Solution:

1. The close velocity is too slow. Increase the close velocity greater than or equal to 40.
2. Reinitialize the finger; this may reset the TorqueSwitch™.
3. If the BarrettHand™ has been inactive for an extended period or if the finger has been slammed open against its stop with a high velocity setting, the TorqueSwitch™ may need to be manually activated. Insert a 2-mm hex wrench into the left Drive Access hole, as shown in Figure 27. Rotate counterclockwise to open the finger fully. Next, press very hard against the inner link to constrain it from moving while not constraining the outer link, as the person's thumb is doing in Figure 26, while torqueing the hex wrench clockwise. Increase torque until the fingertip breaks free, so that it can be rotated easily while the inner link remains stationary. Remove the 2-mm hex wrench and reinitialize the finger.
4. If the problem persists, contact Barrett Technology.

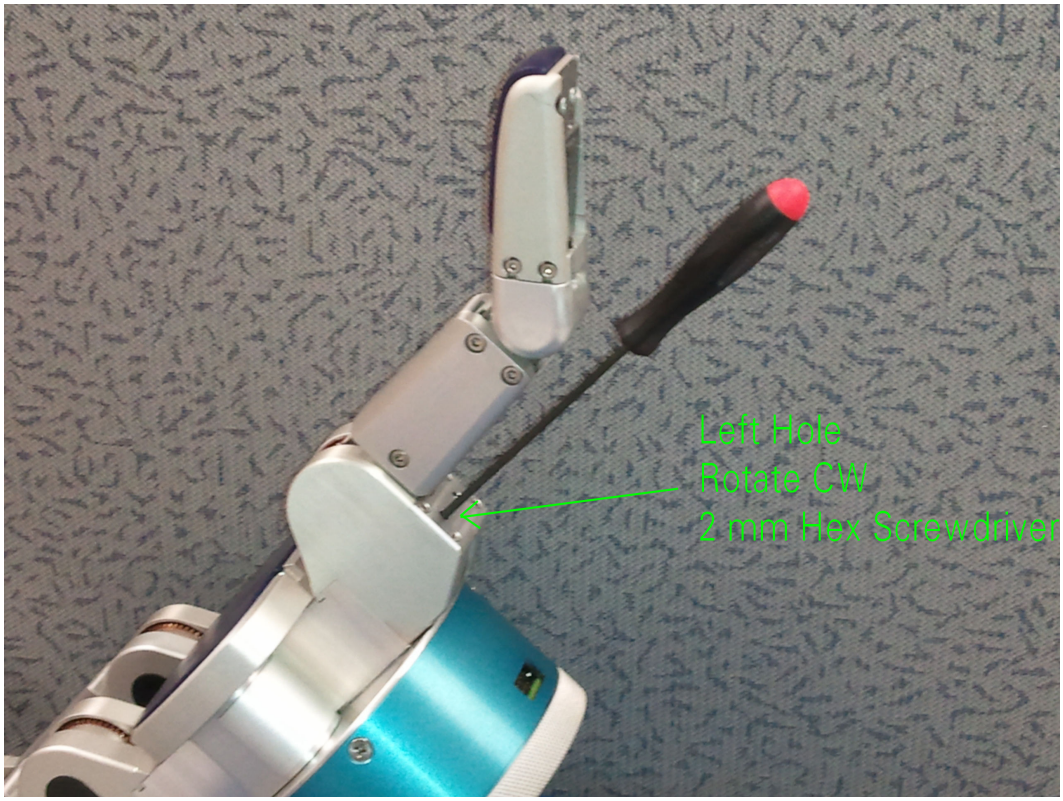


Figure 26 - Manual Torque Switch Activation

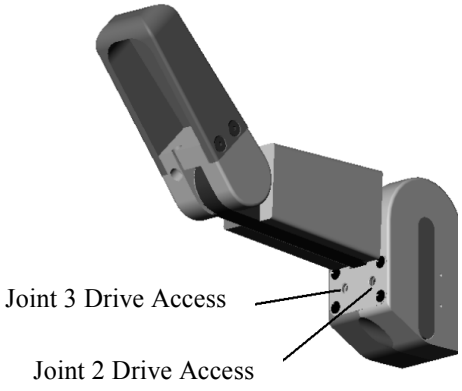


Figure 27 - Manual TorqueSwitch™ Activation Drive Holes

Symptom: Using the GTO or GTC command to open or close the fingers results in the fingers moving at slightly different velocities. This does not happen with GO and GC commands.

Possible Solution:

1. For BH8-262 hands: Verify that the finger velocity and filter properties are the same (MCV, MOV, FPG, FDZ, FIP, SAMPLE, ACCEL).
   For BH8-280 hands: Verify that the MT property is the same.
2. Unlike GO and GC commands in which each finger is position controlled to follow a predefined trapezoidal trajectory that keeps the fingers moving precisely, GTO and GTC commands apply pure proportional velocity control for 262 hands and control motor currents for 280 hands. Each finger has slightly different friction due to manufacturing tolerances, resulting in different actual velocities for the same commanded velocity. Lubricating the high friction fingers will help reduce the friction and increase the velocity. See Section 7.3 for lubrication instructions. Alternatively, the commanded gains or commanded velocities from nominal allows you to compensate.
3. If the problem persists, contact Barrett Technology.

Symptom: Fingers will not close completely.

Possible Solution:

1. Adjust close target CT so that it is either at or just beyond the actual palm surface.
2. Verify that the outer link has not broken away prematurely.
3. Verify proper finger angles when the fingers are removed and replaced during lubrication maintenance. An error here can cause the outer finger link to reach its joint stop prematurely, even without breakaway, before either CT or the inner finger-link joint limit is reached. Verify finger angle is correct by following the directions on disconnecting and reattaching fingers in Section 7.3. Readjust if necessary.
4. Verify there are no objects or other fingers blocking the finger from closing completely.
5. Verify the MSG (Maximum Strain Gage) property is greater than the strain gage value (SG). If the strain gages are not installed, set MSG to 256.
6. If the problem persists, contact Barrett Technology.

Symptom: The spread motion has excessive friction.

Possible Solution:
1. Lubricate the spread motor gears as shown in Section 7.3.
2. If the palm screws have been reinstalled, verify all screws are tightened with the same amount of torque. Excessive torque may cause spread friction.
3. If the problem persists, contact Barrett Technology.

Symptom: The threaded locking ring does not fit on the threaded base of the BarrettHand™.

Possible Solution:
1. The threaded locking ring has been damaged or is warped. Contact Barrett Technology for a replacement part.
2. The threads on the base of the BarrettHand™ have been damaged. Contact Barrett Technology for service.

Symptom: The fingertip flops over backwards after a severe impact against a finger tip.

Possible Solution:
1. The finger cable is broken. Verify this by removing the Shroud Cover, see Figure 23, and inspecting the cable. The cable should be intact and not broken. If the cable is broken, contact Barrett Technology.

Symptom: The fingertip has excessive backlash.

Possible Solution:
1. The pretension in the cable is too low. Refer to Section 7.1 to set the finger cable pretension properly. If the problem persists, contact Barrett Technology.
2. The finger cable is broken. Verify this by removing the Shroud Cover, see Figure 23, and inspecting the cable. The cable should be intact and not broken. If the cable is broken, contact Barrett Technology.

Symptom: The fingertip has driven itself open beyond the normal full-open position and perhaps is hyperextended, but it is not loose.

Possible Solution:
1. This can only happen if the spur-gear teeth are not properly engaged when the finger is reinstalled after being removed for lubrication. Verify that the finger is seated completely and square where it attaches to the palm. Verify also that the finger screw is in place and is not loose. Reset the fingertip angle and reseat the finger carefully and verify that it is seated completely and square. Test.
2. If problem persists, Hand must be serviced.

Symptom: The spread fingers F1 and F2 are at different angles around the palm.

Possible Solution:
1. An internal spread gear is damaged and will need to be replaced. Contact Barrett Technology.

9 Theory of Operation

9.1 Electronic Architectures

BH8-280 Hand

The new BH8-280 BarrettHand uses a total of 4 ultra-miniature, high performance brushless motor controllers contained within the hand. Barrett's PUCk (Powerful Universal Controller) is perfect for power sensitive mobile applications and has integrated power amplifiers and precision current sensing. It can control the torque output of brushless motors with state-of-the-art space vector commutation and very low torque ripple. It has a 32-bit digital signal processor, operates on a 1 MBaud CANbus, and has a built-in strain gage amplifier. The PUCk excels at providing high-level supervisory capabilities for the BarrettHand™ on-board and including RealTime position, velocity, and torque motor control. Like the 262 hand, the PUCk does trapezoidal velocity control with velocity ramp up/down according to acceleration defined with the ACCEL property and maximum velocity is constrained by the MV property. Velocity mode is based on top of a position controller updated at a kilohertz. The velocity reference in units of ticks/ms is added to the present commanded position each servo cycle.

BH8-262 CPU Board

The CPU board handles all set-up, communications and high-level control of the power boards including coordinated motion, force monitoring, and motor speed monitoring. The main processor is a Motorola 68HC811E2FN microcontroller, which contains 256 bytes of RAM and 2Kbytes of EEPROM. The CPU board contains a 128 Kbytes RAM chip external to the microprocessor, which is used to store the BarrettHand™ firmware. The microcontroller operates at 1.25 MHz and communicates via standard RS-232C protocol at a factory-selected baud rate of 9600, no parity bits, eight bits per character and one stop bit. The BarrettHand™ is capable of communicating at rates up to 38.4K baud.

BH8-262 Motor Power Boards

Each power board handles current control of a single DC brushless motor in the BarrettHand™ using a Hewlett-Packard HCTL-1100 motion control chip. The optical incremental encoder signals from each motor, with 360 counts per motor revolution, are amplified and sent to the HCTL-1100 chip. The controller uses the encoder feedback for alignment at startup and for commutation of the motors via 10-20 kHz Pulse Width Modulation (PWM). It can also perform position and velocity control. A functional block diagram of the power board and its relation to other pieces of the control system is shown in Figure 28.

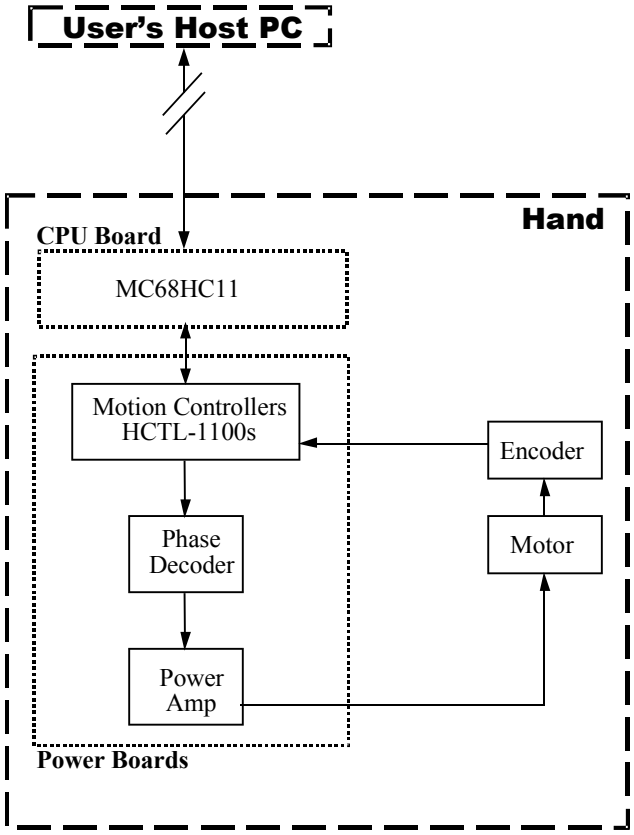


Figure 28 – BarrettHand™ Controller Block Diagram

Brushless Motors

The BarrettHand™ utilizes one of the smallest DC brushless servo motors in the world for their torque range. Because the motors have no brushes, and thus less inherent friction, they achieve a better torque/mass ratio than typical brushed servos. There is also no need to replace worn brushes after the motors have been in service over a period of time. Table 7 shows BarrettHand™ motor properties.

Table 7 - BarrettHand™ Motor Properties

Number of Phases	3
Number of Poles	6
Rotor Magnets	Highest-Grade Samarium-Cobalt Rare-Earth
Commutation	Brushless Electronic PWM
Peak Torque	5 N-cm (8 oz-in)
Motor Constant	0.83 N-$\frac{cm}{\sqrt{W}}$ (1.17 $\frac{oz-in}{\sqrt{W}}$)
Position Feedback	360 counts/rev., incremental optical encoder

9.2 Low-Level Motor Control

The BH8-262 Hand uses the HP-Agilent HCTL-1100 motion controller to drive each of its four (4) motors. The HCTL-1100 manual is located at:
http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf.

The BarrettHand exploits two modes of the HCTL-1100 for controlling individual motors:

- Velocity Control and
- Trapezoidal Profile Control.

9.2.1 Trapezoidal Control

Control of motor position, such as IO, IC and M, uses Trapezoidal-Profile control. This mode drives the motor to the desired position and returns a status code ERR32 (except for O and C commands) if the final position error is greater than MPE. The HCTL1100 applies Trapezoidal control in two steps:

1. Microseconds before the motor-control is executed, the HCTL-1100 constructs a 3-part trajectory that forms a trapezoidal shape when plotted on a motor-velocity-versus-time graph. The first part of the plot is a constant acceleration, plotted as a positive slope, ACCEL. The second part is a constant velocity, MOV or MCV (depending on direction), plotted as a horizontal line. The last part is a constant deceleration, plotted as a negative slope, ACCEL.
2. During motor-control execution, the HCTL-1100 applies a Gain-Pole-Zero style linear control law, driving motor torque to minimize the instantaneous position error. This error is calculated as the desired position minus the measured position and only the gain.

For more information on Trapezoidal, see pages 34-36 of:
http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf.

9.2.2 Velocity Control

The Velocity Control is used during torque-close and torque-open commands (TC and TO). These commands use proportional gain multiplied by the instantaneous velocity error to control the motors. The motor torque responds for each motor controller according to:

$$MCn = (K / 4) * Yn$$

K depends on the Boolean LCPG flag. If LCPG is true than K is equal to the loop control proportional gain, otherwise K is equal to the value of the "FPG" property. Yn is the velocity error equal to control velocity multiplied by the loop control velocity coefficient minus the actual velocity.

For more information, see pages 31 and 32 of:
http://www.hctl-1100.com/HCTL%20docs/HCTL-1100%20Data%20Sheet.pdf.

9.3 Mechanisms

9.3.1 TorqueSwitch™

Barrett Technology's patented TorqueSwitch™ mechanism affords the BarrettHand™ unparalleled weight reduction without sacrificing dexterity or functionality by serving as a "smart" coupling of two finger joints to one motor. The mechanism's operation is similar to that of a simple screw fastener. Theoretically, the torque with which one tightens a uniform screw should be equal to that which is required to subsequently loosen it (neglecting inertia and provided all materials deformations remain elastic). This principle holds true for the TorqueSwitch™ mechanism.

The TorqueSwitch™ consists of a threaded shaft; a pair of Belleville spring washers and a spur gear with a threaded bore, shown in Figure 29.

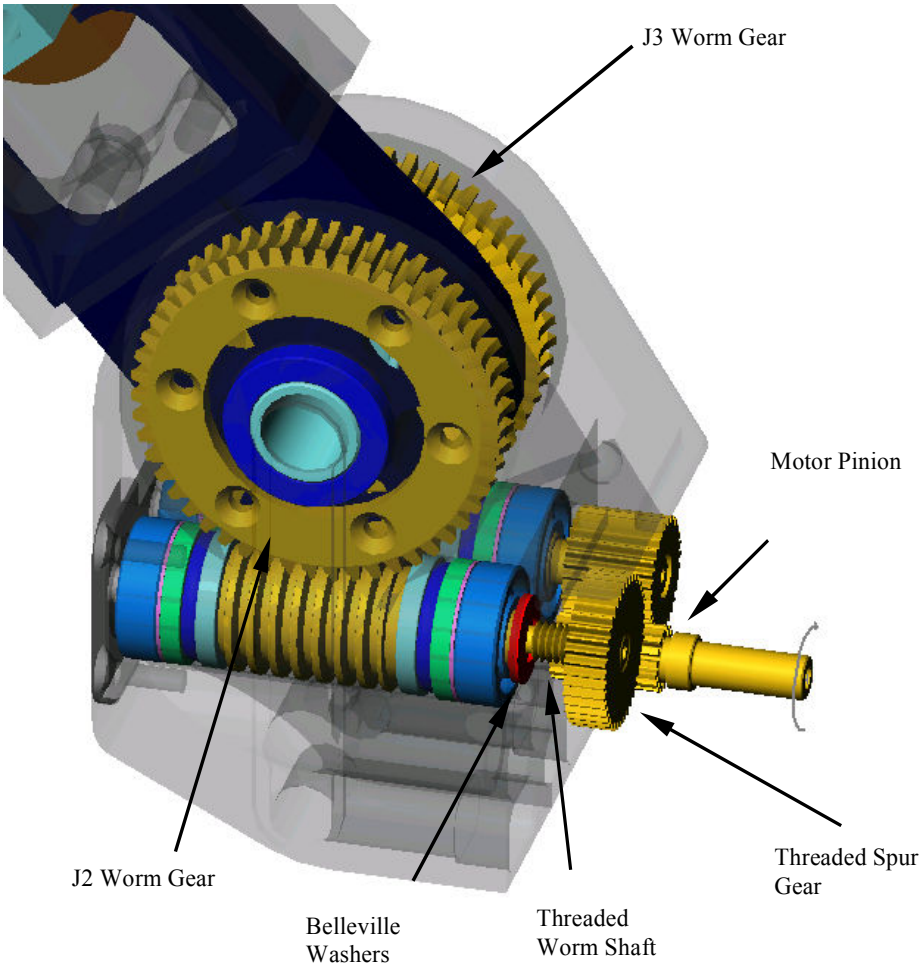


Figure 29 - Barrett's Patented TorqueSwitch™ Mechanism

The following description follows the progression of Figure 30. When the clutch is engaged, both worm gear drives and their corresponding finger links are coupled to the geared servo-motor pinion. In this state, the ratios of motor position to joint position for the 1st and 2nd finger joints are 93.75:1 and 125:1, respectively.

When a finger opens against its motion stop, the threaded spur gear is tightened against the Belleville spring washers with a known motor torque; thereby setting the threshold torque for disengaging the spur gear. If the inner finger link, while closing, contacts a target object of sufficient stiffness to increase the torque in the gear train above the threshold torque, the clutch will disengage from the Belleville spring washers.

When the clutch is disengaged, the threaded spur gear "free-wheels" on the threaded shaft, allowing the motor pinion to turn without inducing motion in the inner link. Instead, only the smaller spur gear, solidly fixed to its shaft, is driven. This fixed spur gear actuates the worm gear drive for the fingertip. Thus, when the clutch is disengaged, the inner finger link remains motionless while the fingertip continues to move allowing the fingers to form-fit around any shape.

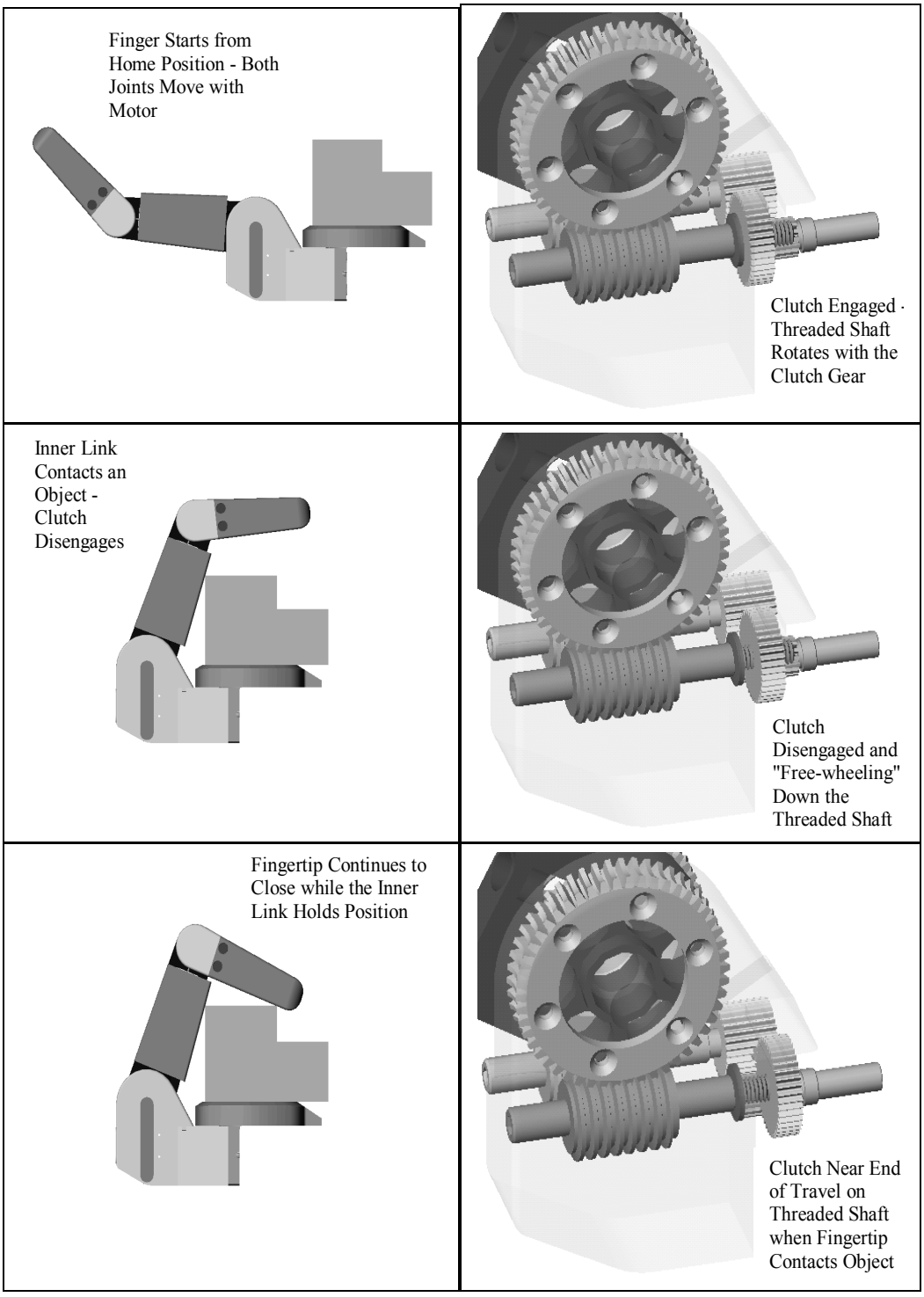


Figure 30 - TorqueSwitch™ Operation

The force required to cause the TorqueSwitch™ to disengage can be set using the properties, IVEL, IOFF, IHIT, and OT. Barrett Technology recommends that users should not change IVEL, IOFF, and IHIT from their default values. The following Breakaway force Curve can be repeated by using OT with the default values.

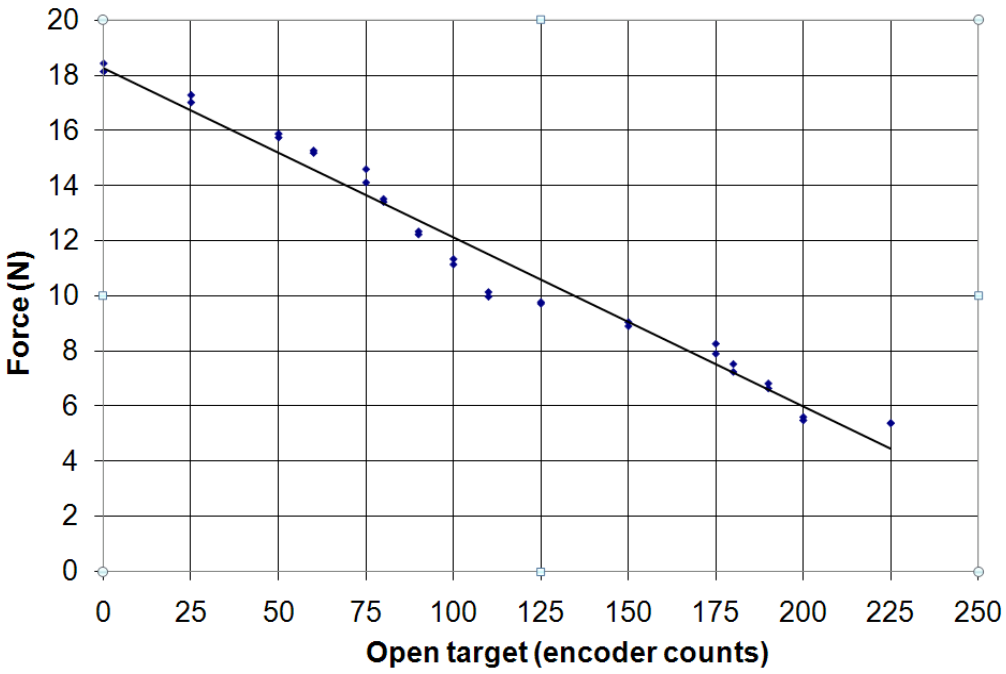


Figure 31 - Breakaway Force Curve

To control how much force is applied to an object being grasped, the command TorqueClose and TorqueOpen must be used. These commands use the Velocity Control Law with the properties MCV and MOV. To determine the amount of desired force at the fingertip use Figure 32 to select proper velocities.

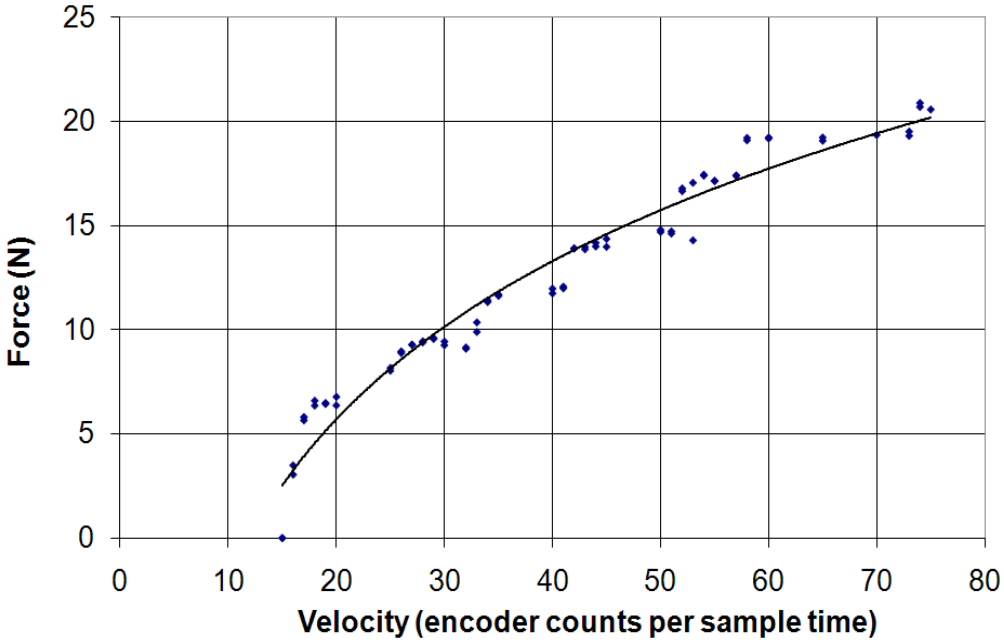


Figure 32 – Stalled FingerTip Force Vs. Commanded Velocity (measured before breakaway).

9.3.2 Spread Motion

The spreading action of fingers F1 and F2 on the BarrettHand™ increases the dexterity of the entire unit with only one additional actuator. Optimal grasp configurations can be achieved "on-the-fly" without costly tool changes associated with traditional grippers. In addition, the backdrivability built into this degree of freedom causes the BarrettHand™'s grasp shape to change in mid-grasp, creating a more stable grasp of oddly shaped target objects.

Should you wish to control the spread position of the fingers, the complete command set available to the fingers is also available for the spread, including commands for fixed-increment motion and move-to-position commands.

The sustainable torque that the spread fingers can exert continuously in a 'pinch' type grasp is shown in Figure 33. These are found by changing the FPG property while keeping all other properties at their defaults. For a given torque setting, larger forces can be achieved by curling F1 and/or F2 closed to the point where the contact point becomes closer to the spread axis.

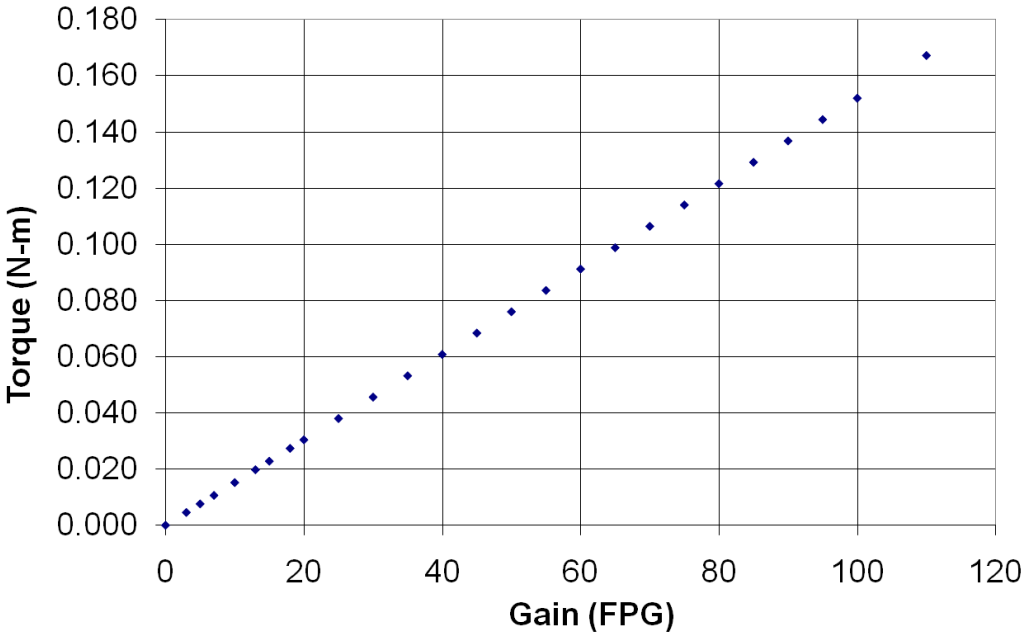


Figure 33 - Pinch Grasp Torque

9.4 Optional Strain Gage Joint-Torque Sensor

The BarrettHand™ provides an optional Joint-Torque sensor for each finger. The Joint-Torque sensor measures the torque about the outer joint on each finger, see Figure 34. The Joint-Torque sensor is comprised of a flexible beam with four foil strain gages applied and wired in a Wheatstone Bridge configuration. When a force is applied to the fingertip, Force A, the torque is measured by the amount of deflection in the beam. The beam deflection is proportional to the difference in cable tension, which translates to a force on the pulley attached to the flexible beam, Force B. The flexing in the beam creates a measurable voltage change in the Wheatstone Bridge. This difference in voltage is conditioned, amplified, converted and available to you in digital form.

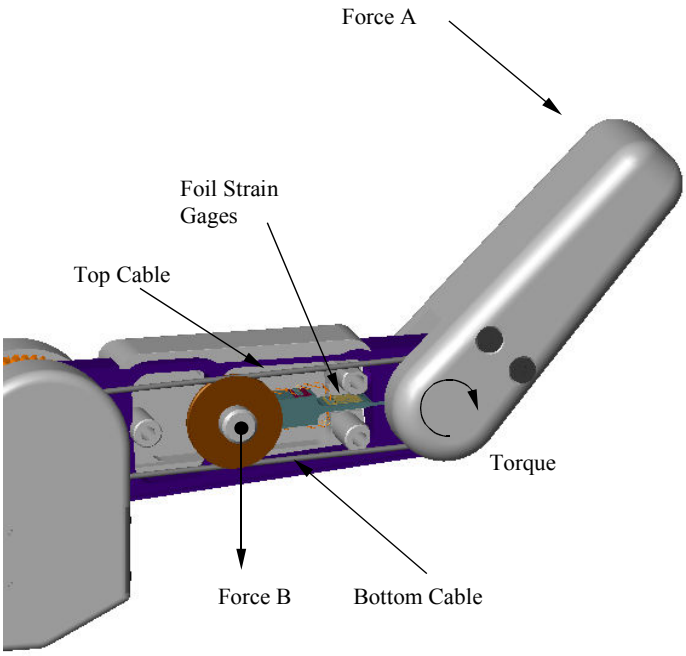


Figure 34 - Strain Gage Joint-Torque Sensor

The gages are adjusted before leaving the factory and should exhibit a no-load SG value between 100 and 140 for 8-bit strain on earlier hands. Newer hands, using Pucks will have 12-bit resolution and the expected no-load SG value should be between 1600 and 2240. If the gage values do not fall within the specified range, see Section 7.4. For improved accuracy, the user can measure the no-load value before taking readings of SG. For example, issue a GO command and then a FGET SG command to open the fingers against their J2 stops. J3 has no open stop, so its torque will measure only second order effects, such as residual friction, gravity, and dynamic inertia effects (on a moving robotic arm).

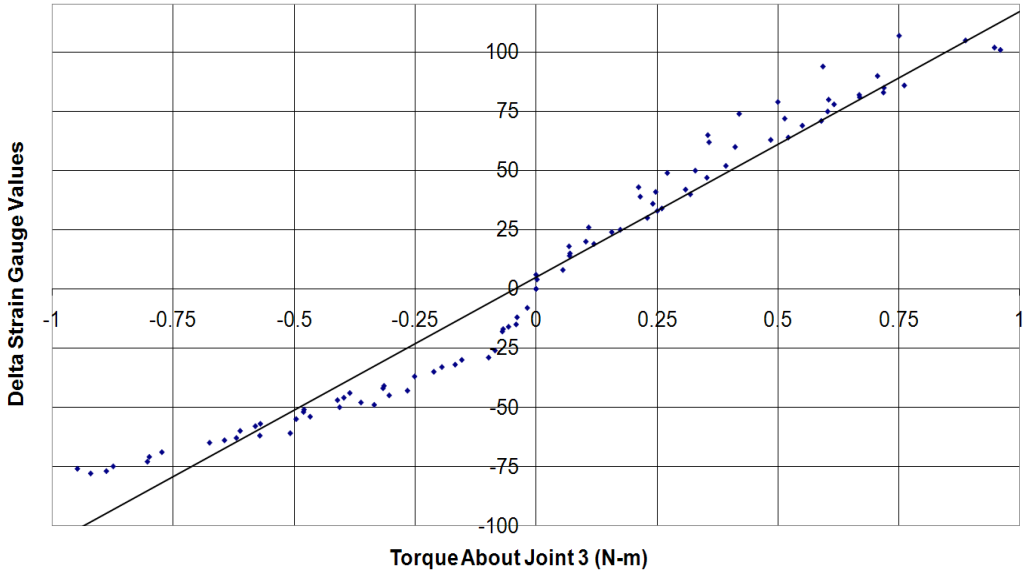


Figure 35 - Strain Gage Torque Curves

Note: In Figure 35, true SG values have been adjusted so that the no-load value corresponds to zero torque. If the torque curve measured does not approximate the torque curve shown in Figure 35, see Section 8. The torque curves for each finger will be different due to the variations in materials.

9.5 Forward Kinematics

The forward kinematics for the BarrettHand™ were determined using the Denavit - Hartenberg notation described in "Introduction to Robotics, Mechanics and Control 2nd Edition", John J. Craig. Each finger is considered its own manipulator and is referenced to a wrist coordinate frame in the center of the palm. Use the forward kinematics calculated in this section to determine fingertip position and orientation with respect to the palm.

Equation 1 is used to determine the transforms between axes *i* and *i-1*.

$$ {}^{i-1}\mathrm{T}_i = \begin{bmatrix} \mathrm{c}\theta_i & -\mathrm{s}\theta_i & 0 & \mathrm{a}_{i-1} \\ \mathrm{s}\theta_i c\alpha_{i-1} & \mathrm{c}\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -\mathrm{s}\alpha_{i-1} d_i \\ \mathrm{s}\theta_i s\alpha_{i-1} & \mathrm{c}\theta_i s\alpha_{i-1} & c\alpha_{i-1} & \mathrm{c}\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Equation 1 - Homogeneous Transform Between Frame {*i-1*} and {*i*}

Where:

a_{i-1} = distance from z_{i-1} to z_i measured along x_{i-1}
α_{i-1} = angle between z_{i-1} to z_i measured about x_{i-1}
d_i = distance from x_{i-1} to x_i measured along z_i
θ_i = angle between x_{i-1} to x_i measured about z_i

$$ c\theta_i = \cos(\theta_i) $$

$$ s\theta_i = \sin(\theta_i) $$

The forward kinematics are determined using the following equation:

$$ {}^{w}T_{T} = {}^{W}T_{1}\,{}^{1}T_{2}\,{}^{2}T_{3}\,{}^{3}T_{T} $$

Equation 2 - Forward Kinematics from Wrist Frame to Fingertip

Table 8 is a list of the parameter values used to compute the forward kinematic transformation matrices for all of the fingers.

Table 8 - D-H Parameter Values for all Fingers

Parameter	Value	Notes
A_w	25 mm	
A_1	50 mm	
A_2	70 mm	
A_3	50 mm	
D_W	84 mm	Palm offset distance depends on model
D_3	9.5 mm	Finger endpoint depth offset
Φ_2	0° to 0.4°	Initialization offset in Radians (IOFF)
Φ_3	42°	Joint 3 offset is actually in Radians

All of the kinematics for the BarrettHand™ are derived from the zero position. The configuration of the fingers and spread in the zero position of the BarrettHand™ is shown in Figure 36 along with the wrist coordinate frame.

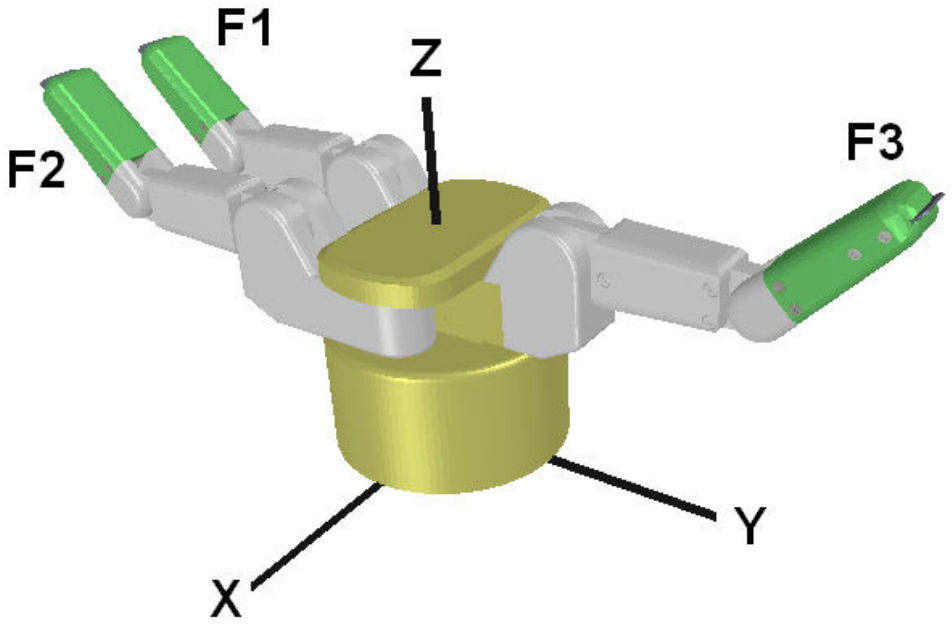


Figure 36 - BarrettHand™ in Zero Position

Finger Kinematics:

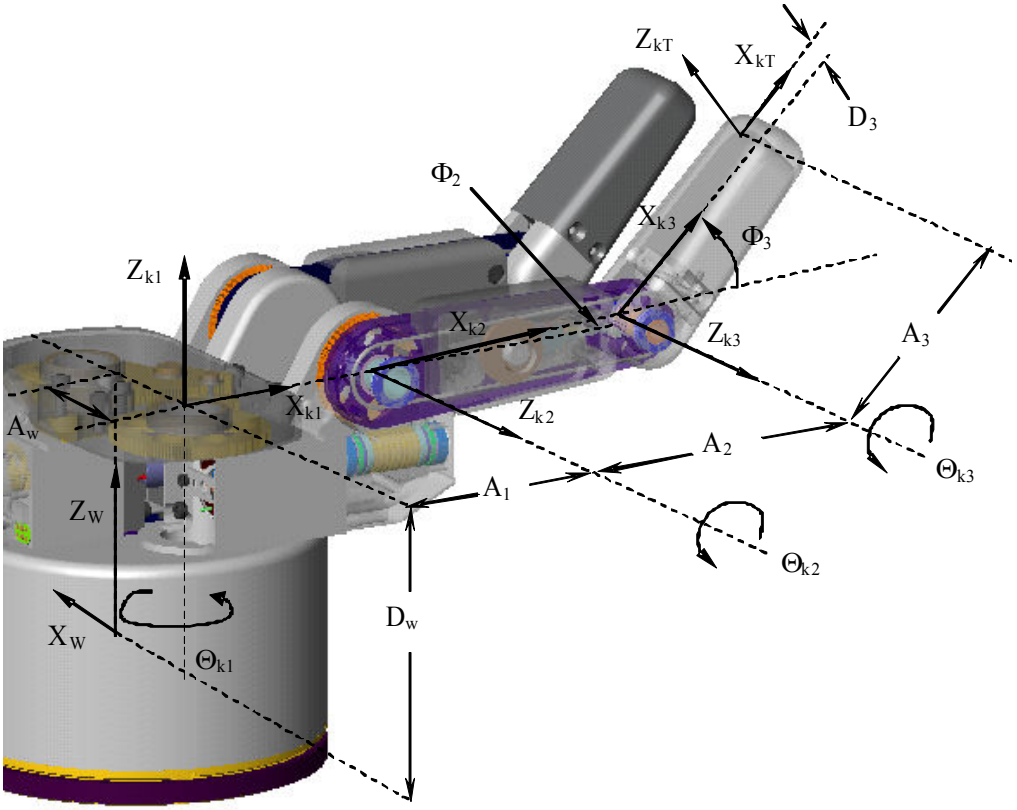


Figure 37 - D-H Frame Assignment for Generalized Finger

Table 9 - D-H Link Parameters for Fingers

Joint	A_{i-1}	α_{i-1}	D_i	θ_i
1	$r * A_w$	0	D_w	$r * \Theta_{k1} - (\pi / 2) * j$
2	A_1	$\pi/2$	0	$\Theta_{k2} + \Phi_2$
3	A_2	0	0	$\Theta_{k3} + \Phi_3$
4	A_3	$-\pi/2$	D_3	0

Where: "k" is defined as the desired finger [1,2,3].
"r" is either [-1,1,0] for [F1,F2,F3] respectively.
"j" is either [1,1,-1] for [F1,F2,F3] respectively.

The transforms from each axis to the next can be determined using the homogeneous transform in Equation 1 and finger link parameters in Table 9. Each of the first three link parameters are fixed and the fourth one is configuration dependent on one of the position variables Θ_{k1}, Θ_{k2}, or Θ_{k3} for the first three joints.

It is useful to check that the multiplication of the four transformation matrices matches for a given finger and at least one hand configuration, such as the zero position. The computed homogeneous transformation matrix from the wrist to tool frame for finger 1 is:

$$ {}^{w}_{T}T = \begin{bmatrix} c_1 c_{ab} & -s_1 & -c_1 s_{ab} & A_3 c_1 c_{ab} - D_3 c_1 s_{ab} + A_2 c_1 c_a + A_1 c_1 + jA_w \\ s_1 c_{ab} & c_1 & -s_1 s_{ab} & A_3 s_1 c_{ab} - D_3 s_1 s_{ab} + A_2 s_1 c_a + A_1 s_1 \\ s_{ab} & 0 & c_{ab} & A_3 s_{ab} + D_3 c_{ab} + s_a A_2 + D_w \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Equation 3 - Forward Kinematics Matrix for Finger F1

Where:

$$ a = \Theta_{k2} + \Phi_2 $$
$$ b = \Theta_{k3} + \Phi_3 $$
$$ c_{ab} = \cos(a + b) $$
$$ s_{ab} = \sin(a + b) $$
$$ c_1 = \cos(\mathrm{r} * \Theta_{k1} - (\pi / 2) * j) $$
$$ s_1 = \sin(\mathrm{r} * \Theta_{k1} - (\pi / 2) * j) $$
$$ k = 1 $$

The hand configuration is determined using position feedback from the encoders. The number of encoder ticks and availability of inner link joint encoders depends on the model number of the hand. The following are the hand positions in units of radians *before* the TorqueSwitch™ is activated:

Θ_{k1} =	(ENC(k) / CNT(k)) * π	for k = 1, 2
	0	for k = 3
Θ_{k2} =	(ENC(k) / CNT(k)) * 140 * (π/180)	for k = 1, 2, 3
Θ_{k3} =	(ENC(k) / CNT(k)) * (140/3) * (π/180)	for k = 1, 2, 3

where,

CNT(k) = total number of quadrature encoder counts of finger k
ENC(k) = present encoder position of finger k

Notice that Θ_{k3} will generally move 1/3 the amount of Θ_{k2} but after the TorqueSwitch™ has been activated the inner link stops moving and all the joint torque is applied to the outer link. Users that have inner link joint position sensors will be able to determine finger joint positions at all times. For earlier hands without inner link position sensors it may be possible to estimate joint positions after detecting breakaway. This section is concerned with equations for forward kinematics and does not attempt keeping track of finger joint positions all the time. Once the outer finger link stops on hands without inner link position sensors after TorqueSwitch™ activation, the joint positions and end tip position cannot be accurately determined until the TorqueSwitch™ mechanism is reset. It may be reset by opening the finger. Refer to Appendix B for information on how to detect TorqueSwitch™ activation.

The finger end tip positions are found in the ${}^{W}T_{T}$ matrices in the last column. Discarding the 1 in the last row, you have the Cartesian X end tip coordinate in the first row, the Y coordinate in the 2nd row, and the Z coordinate in the 3rd row.

9.6 Joint Properties

9.6.1 Encoder to Joint Ratios

This section describes all mechanical reductions in the 262 and 280 hands as well as the ratios that go from finger and spread encoder positions to joint positions in units of radians. To find the finger or spread mechanical reduction relative to the motor use the constants in the table below. Table 10 applies to the hands *before* breakaway occurs. Each finger has 3 joints starting with the knuckle joint that swings the spread, the last one moves the outer link, and the one in between moves an amount proportional to the outer link until breakaway occurs. To go from motor encoder position to actual finger position for joint 2 multiply the encoder position by the corresponding joint Radians to ticks ratio. Joint 3 of the finger moves one third this amount. This transformation works *before* breakaway of the TorqueSwitch™ has occurred.

Table 10 - Finger and Spread Joint Ratios

Hand Motor	Encoder (Min Ticks)	Encoder (Max ticks)	Mechanical Reduction (Joint)	Joint Radians/tick Ratio
262 Finger	0	17,500	125 (2), 375 (3)	$(140°)(\pi/180°)/17{,}500$
262 Spread	0	3,150	17.5 (1)	$\pi/3{,}150$
280 Finger	0	$199{,}111.\overline{1}$	125 (2), 375 (3)	$(140°)(\pi/180°)/199{,}111.\overline{1}$
280 Spread	0	35,840	17.5 (1)	$\pi/35{,}840$

The optical encoders for each of the 280 hand motors have 4096 count encoders and 262 hand motors uses a 90 line, or 360 count, encoder. Inner link encoders are the same ones used on the motor for position feedback.

The forward kinematics from section 9.5, that are used to calculate end-tip positions, depend on the configuration of joint positions for each finger (joint 2 and joint 3) and the spread. Calculate positions in radians for each joint including spread, finger joint 2, and finger joint 3. These will be the joint positions *before* breakaway.

Joint 3 position can be represented more precisely if it is calculated relative to the plane of the palm plate that is accurate *before* and *after* breakaway. This position for joint 3 depends only on the model number (262 or 280) of the hand where Φ_3 is a joint 3 offset from joint 2 equal to approximately 42°.

$$\text{262 Joint 3 Position} = (\pi/180°)(\Phi_3 + (4/375)\text{ENC}(k))$$
$$\text{280 Joint 3 Position} = (\pi/180°)(\Phi_3 + (4/375)\text{ENC}(k)(360/4096))$$

Equation 4 - Joint 3 Positions Before and After Breakaway

Note that joint 1 in the knuckle drives the outer link first through a 93.75 reduction and then a 4:1 reduction. The motor position directly determines the outer link angle with the palm plate of the hand as shown in Equation 4. During breakaway joint 2 position needs to be detected in software by using breakaway acceleration threshold for the 262 hand and then this link remains motionless as described in section 9.3. The breakaway position may be used for finding where breakaway occurred. Joint 3 position still only depends on just absolute motor position. On 280 hands, the inner-link joint position sensors may be used together with the outer link position to determine the finger positions at all times.

9.6.2 Joint Motion Limits

The maximum joint motion limits for the BarrettHand™ are calculated based on the zero position seen in Figure 36. Depending on the position of the spread joint, Θ_{11}, and the objects in the grasp, the maximum joint motion limits for the finger links may vary.

The inner link, Θ_{12}, Θ_{22}, Θ_{32}, has a maximum joint motion limit of 140° with no object blocking movement and Θ_{11} in the full close or open position. The outer link, Θ_{13}, Θ_{23}, Θ_{33}, has a maximum joint motion limit of 48° when Θ_{11} is fully open or closed and there is no object in the grasp, as shown in Figure 38. When the spread is in any position other than full open or close, the fingers may not have the full range of motion due to interference with other fingers.

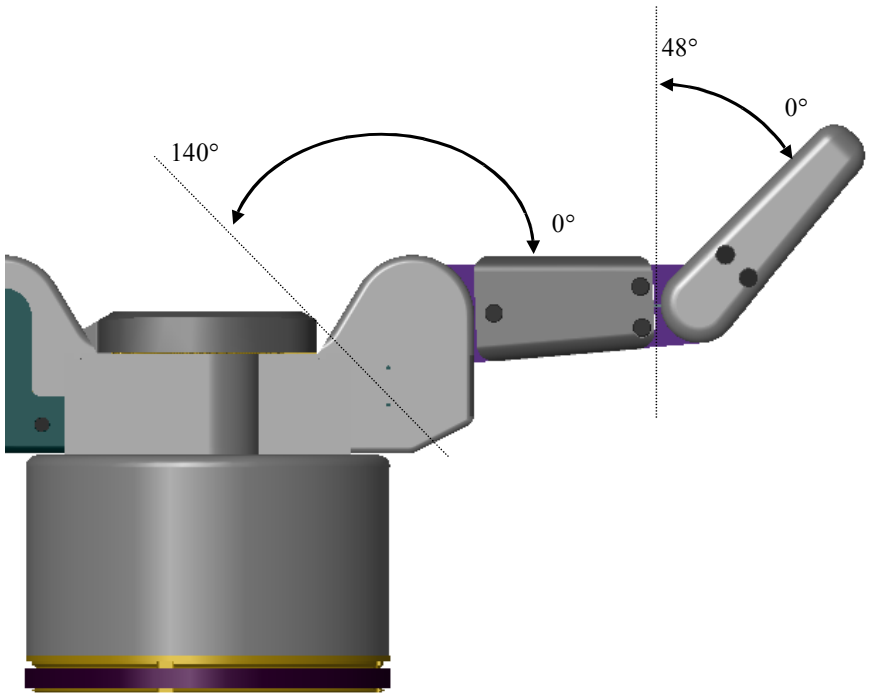


Figure 38 - Finger Joint Motion Limit Range

The spread joint, Θ_{11}, has a maximum joint motion limit of 180° with no object blocking movement and all fingers in the full open position. If the fingers are partially closed or there is an object in the grasp, Θ_{11} may be restricted due to finger interference. See Figure 39.

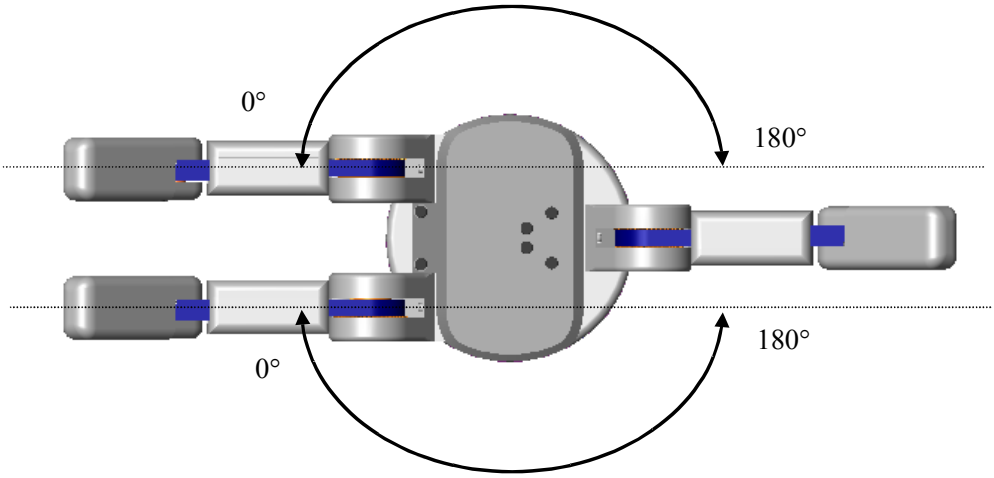


Figure 39 - Spread Joint motion limit Range

Appendix A Technical Specifications

Kinematics	Qty.
Total fingers:	3
Fingers which spread:	2
Joints per finger:	3
Motors per finger:	1
Axes of spread motion:	2
Motors for spread motion:	1
Total axes:	8
Total motors:	4

Range of Motion

Finger base joint:	140°
Fingertip:	48°
Finger spread:	180°

Finger Speed

Finger fully open to fully closed:	1.0 sec
Full 180° finger spread:	0.3 sec

Position Sensing

Type:	optical incremental encoder
Resolution:	0.008° at the finger base joint
	17,500 encoder counts full finger open to full close

Weight

BarrettHand™:	1.18 kg (2.60 lb)
Optional Arm Adapter B0133:	0.2 kg (0.4 lb) additional

Payload
2.0 kg (4.4 lb) per finger at tip

Motor Type
Samarium-Cobalt, brushless, DC, servo motors

Mechanisms
Worm drives integrated with patented cable drive and TorqueSwitch™

Power Requirements
Single phase AC electrical outlet with ground.

Load:	500 W
Phases:	Single
Voltage:	95-130 & 190-260 VAC
Frequency:	50/60 Hz

Power Supply

Location:	dry, stationary surface
Size:	298 x 149 x 42 mm (11.74 x 5.65 x 1.54 in)
Weight:	1224 gm (2.70 lb)

Cables
3-meter continuous-flex cable, 8-mm diameter (BarrettHand™ Cable)
3-meter serial extension cable
AC line cord

Hand Dimensions

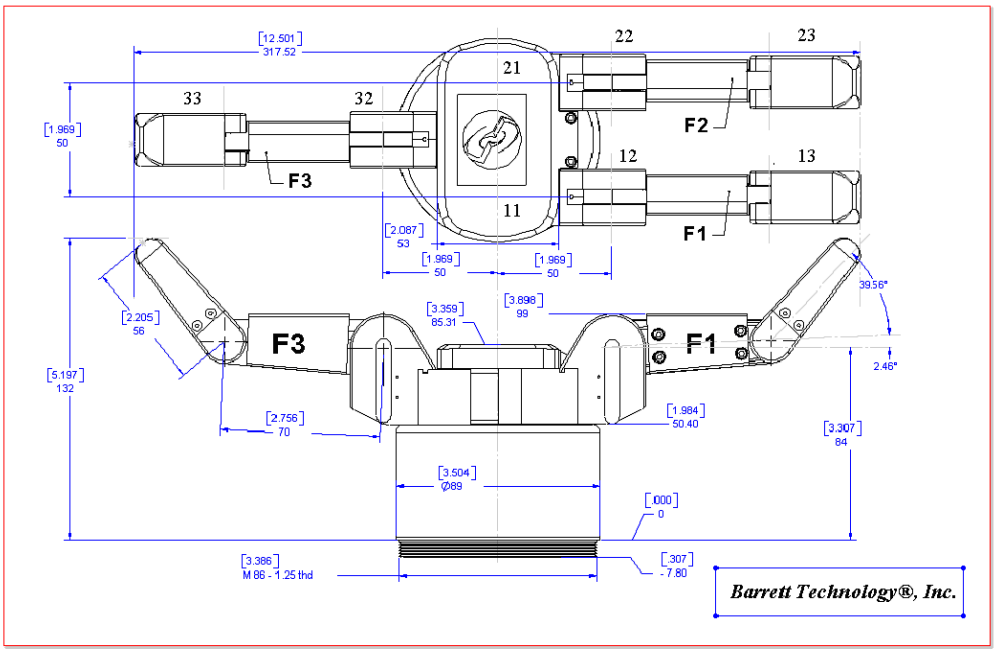


Figure 40 - BarrettHand™ Dimensions

Available Options

B029A	Strain gage Fingertip Torque Sensors for all three fingers
B0111	C++ Function Library
B01C3	Subscription Service

US Patents (patents established and pending in other countries)

5,501,498
5,388,480
4,957,320

Appendix B FAQ

Q1: What type of finger motions are possible with the BarrettHand™ and what servos and mechanisms are used to control them?

A1: The BarrettHand™ has 3 identical curling 2-joint fingers, each with its own built-in, independent, high-performance brushless motor drive. A patented TorqueSwitch™ mechanism then channels torque from the motor to the two joints depending on:

1. TorqueSwitch™ level set for that particular grasp
2. Joint-torque status

In its full-open state, each finger's inner-link surface is almost parallel with the palm plate, while the outer finger link is curled inward by 42 degrees. When a finger is commanded to close under velocity or position control, the links move according to the transforms in Section 9.5, the resulting motion will be curling, to promote form closure before a grasp contact is initiated. If the outer fingertip makes first contact with an object, it develops full force against the object until no motion is detected, at which point the motor current may be removed since the finger joints become mechanically locked. However, if the inner finger link makes first contact, the motor applies the torque to both links until the TorqueSwitch™ level is reached. At this instant, the inner link is locked into position mechanically, and all motor torque is shunted to the outer link, which stops only at motor stall torque. The net result is highly effective grasping.

The patented finger-spread motion has two opposable fingers and one fixed finger. To minimize the number of motors (and thereby the weight, bulk, heat, power, and cost of the BarrettHand™), one motor drives the "spread" action of both fingers synchronously and symmetrically about the palm.

The spread motion adds surprising dexterity. One design feature of the spread motion is that, unlike the curling motions of each finger, the spread is highly backdrivable, so that the spread compliance is controllable. By setting low spread compliance, the BarrettHand™ dynamically finds the lowest energy state as the fingers close, resulting in a firm and reliable grasp.

Q2: Can the outer two finger joint motions be controlled independently?

A2: No, the two joints are controlled by one servo-motor. Although the mechanism behaves in an intelligent manner for grasping, we traded the second motor and motor electronics for the TorqueSwitch™ mechanism to save weight, bulk, heat, power, and cost.

Q3: How does the Torque Switch™ mechanism work and what is the clutch used for?

A3: See Section 9.3 for information on how the TorqueSwitch™ mechanism works.

Q4: What materials are the fingers made of, particularly the fingertips?

A4: The finger tip and covers on the BH8-260 and later are anodized aluminum. Nylon tip and covers are optional. The finger links are made of aluminum and the gearing is made of steel and bronze. The base shell is made of anodized aluminum on the BH8-260 and later; Carbon-Kevlar composite before the BH8-260.

Some new BH8-280 hands have a cast urethane blue material for the finger tip that presently has a durometer of 80 but this may be lowered to 70 for future hands. Finger tips will likely change so please ask about them if you have a need for special finger tips. Barrett is working on black rubber pads for customers without pressure profile sensor tips.

Q5: Is it possible to determine the instant when the TorqueSwitch™ is activated?
A5: The 280 hand has inner-link joint encoders and the outer link is geared to the motor encoder, so you always know the position of the inner and outer links. Knowing the moment of TorqueSwitch™ activation is no longer necessary with the 280 hand.

With 262 hands it is possible to determine the approximate time when the TorqueSwitch™ achieves breakaway by monitoring the breakaway flag, BD. When BD=1, breakaway has occurred at BP encoder counts. Setting BS=1 stops the motor at the instant of breakaway.

Figure 41 plots velocity during TorqueSwitch™ activation, which is identified by a notch in velocity. While the deceleration part of the notch depends on the compliance of the object, the re-acceleration is independent, so the algorithm measures only the re-acceleration. Acceleration is set with parameter BDAT.

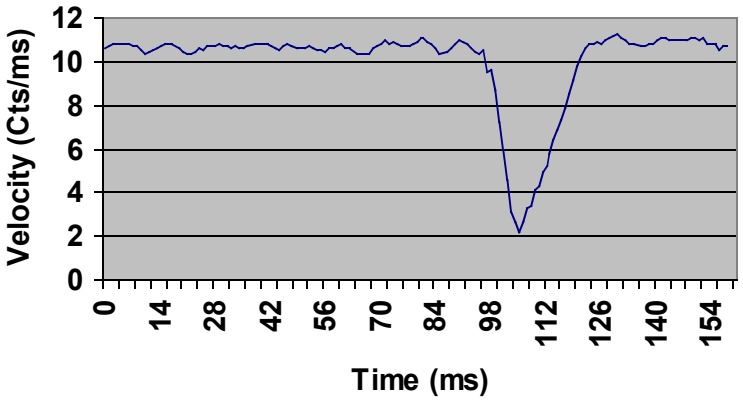


Figure 41 – TorqueSwitch™ Activation Graph

Q6: How do you unwedge jammed fingers from the open or close position?
A6: It is recommended that you have a 2 mm hex wrench and manually turn the Joint 2 drive access to unwedge jammed fingers as shown in Figure Error: Reference source not found.

For more questions, please contact Barrett Technology Customer Service.

Appendix C GLOSSARY

API - An Application Programming Interface (API) is an interface implemented by a software program which enables it to interact with other software. It is implemented to determine application, libraries, and operating systems vocabularies and calling conventions, and is used to access their services. Specifications may include routines, data structures, object classes, and protocols used to communicate between the consumer and the implementer of the API.

Backdrivability - Backdrivability is the measure of how accurately a force or motion that is applied at the output end of a mechanical transmission is reproduced at the input end. In a mechanical robot-like linkage, good backdrivablility means that a person can grab the enditp of the linkage and move it around effortlessly.

BarrettHand™ – The 1.2 kilogram dexterous robotic Grasper™ as described in Section 1.1.2.

BarrettHand™ System - The entire system received from Barrett Technology, Inc. Includes all components as listed in Section 1.1, plus any additional options as described in Section 1.1.2.

Belleville Spring Washer - A conical washer that has geometry specifically formed to produce a desirable spring constant.

Cycle - (Finger) The equivalent to closing and opening the finger completely once, 35000 encoder counts for a 262 hand and 390000 encoder counts for a 280 hand.
(Spread) The equivalent to closing and opening the spread completely, 6200 encoder counts for a 262 hand and 71900 encoder counts for a 280 hand.

Firmware - Software that is embedded in a hardware device that internally controls various electronic devices. Updated firmware that contains additional features and bug fixes is supplied to the end user by the Manufacturer.

Grasp – (n.) The state in which an object has been firmly contained and secured by the BarrettHand™.
(v1.) The method by which the BarrettHand™ closes its fingers around an object in order to secure it.
(v2.) The collective term for fingers one, two and three, as defined in the BarrettHand™ control software.

Graspcenter - The center of an object being grasped by the BarrettHand™.

Hysteresis - The dependence of the state of a system on its previous history.

Idler Pulley - A fixed or adjustable disc support for a drive cable or belt. It is used in the BarrettHand™ for strain gage beam deflection.

Industrial Gripper - These grippers have one degree of motion freedom, and can only execute an open/close motion. Because of the simple kinematics, the gripper fingers often must be specifically designed for the parts that have to be grasped. Product changes require changes to the gripper assortment, increasing the product change-over time. Furthermore, an increase of the number of required grippers increases the number of required gripper changes during assembly.

Kinematics - The science of motion which treats motion without regard to the forces which cause it, specifically all the geometrical and time-based properties of position.

Lexan® - A water clear, high-impact resistant polycarbonate used to make the BarrettHand™ lab bench stand.

Pretension - The process of adding additional tension to a cable during the assembly process.

RealTime Mode - A control mode of the BarrettHand™, which allows you to control the motors in real-time. This mode allows you to monitor position, velocity, and strain gage values during motion and control the motors during motion.

RS-232 - RS-232 defines the specifications for encoding, transmitting, receiving, and decoding "characters". RS-232C is the most recent version of the EIA (Electronics Industry Association) standard for low-speed serial communication. It defines a number of parameters concerning voltage levels, loading characteristics, and timing relationships.

Shoulder Screw - A fastener having a boss with a set diameter, usually for alignment purposes.

Spline wrench - A variation of a hex wrench where the hex cross section has been replaced with a spline pattern

Spread - The patented motion of fingers F1 and F2 about the palm. This motion allows the fingers to be positioned around the palm for the best grasp.

Spur Gears - A gear having straight, parallel teeth that are perpendicular to the gear's face.

Supervisory Mode - A control mode of the BarrettHand™, which allows you to issue high-level commands to control motion and change properties. The BarrettHand™ does not accept a new command until the previous command is finished.

Threaded Locking Ring - The removable circular ring at the base of the BarrettHand™ that is used to mount the hand, such as the Lexan® test stand or the arm adaptor.

TorqueSwitch™ - The patented coupling between the fingers two joints. This coupling allows the use of one motor to control two joints. When the inner link encounters an object with sufficient force, it will stop, while the outer link continues to close around an object. See Section 9.3.1for more information on the theory of operation.

Worm Gear - A long cylindrical gear with skewed teeth, making it capable of driving other gears. The output gear is driven by the worm gear. The worm has a helical tooth similar to a screw thread. The profile of the worm gear looks similar to a conventional gear with the tooth skewed to the worm pitch angle.

Appendix D Pucks In Hand Protocol

The BH8-280 protocol with the Pucks in the Hand is partially listed in this appendix. It will give users an idea of how new hand communication works.

When the pucks first power up, they are in Monitor mode (for downloading firmware). You must issue a "Set STAT(5) = STATUS_READY(2)" to each puck to exit this mode and enter the Main firmware (motor control). Newer firmware versions will automatically enter the Main firmware after a 2 second delay.

The format for a typical CAN message is:

Get Property

- MessageID = CAN ID of the puck you want the property from.
- Payload = 1 byte = the property number you want.
- Example: Data Length Code (DLC) = 1, MsgID = 1, Payload = 48 (Get encoder position P(48) from puck 1)

Set property

- MessageID = CAN ID of the puck you want the property from.
- Payload = 4/6 bytes = 0x80 | property number, 0x00, Low byte, High byte, [Higher byte], [Highest byte]
- Example: DLC = 4, MsgID = 1, Payload = 0x85, 0x00, 0x02, 0x00 (Set STAT(5) of puck ID 1 to STATUS_READY(2)).

Property to issue a command over CAN = CMD(29)

- Commands: HOME = 7, HI = 13, IC = 14, IO = 15, C = 18, M = 19, O = 20, T = 21
- Example: setProperty(bus = 0, canID = 1, property = 29, verify = FALSE, value = 18); // Issue a "Close" command over CAN

Issue high-level command

- MessageID = CAN ID of the puck you want the property from.
- Payload = 4 bytes = 0x80 | CMD(29), 0x00, <command>, 0x00
- Example: DLC = 4, MsgID = 1, Payload = 0x9D, 0x00, 0x13, 0x00 (Set Command(29) to CMD_MOVE(19), Move to Default Position DP(50)).

The pucks have several motor control MODE(8)s:

- MODE_IDLE(0) = Tie motor phases together for resistive braking effect.
- MODE_TORQUE(2) = Apply T(42) input torque
- MODE_PID(3) = Apply whatever torques are necessary to hold the present P(48) position, using KP(79), KD(80) and KI(81)
- MODE_VELOCITY(4) = Update the internal setpoint position by V(44) encoder cts per millisecond (and go there).
- MODE_TRAPEZIOD(5) = Follow a trapezoidal velocity profile between the present P(48) position and the target E(52) endpoint position. If HOLD(94) != 0, then drop to MODE(8) = MODE_PID(3) and hold position when move is complete, else drop to MODE(8) = MODE_TORQUE(2) (free-spinning) when move is complete.

INDEX