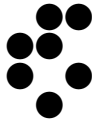


Institut “Jožef Stefan” Ljubljana, Slovenija



DP11039

BarrettHand Grasper Control

Matjaž Ogrinc
Andrej Gams
Aleš Ude

Ljubljana, December 2011

Contents

1	Introduction	3
1.1	Installation	4
1.2	The repository	4
2	Supervisory Control	4
2.1	Supervisory Server	5
2.2	Supervisory Matlab Script	6
2.3	Supervisory Windows Client	6
3	Real-Time Control	7
3.1	Real-Time Server	7
3.2	Client in Simulink	8
3.3	Control Modes	8
4	Graphical Server Launcher	10
5	C++ templates	11

1 Introduction

The robotic gripper 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.

The BarrettHand, shown in Figure 1, has three fingers labeled F1, F2 and F3. Two of the fingers, F1 and 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 features two joints driven by a single DC brushless servo motor. The joints of each finger are coupled through Barretts 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 [1].

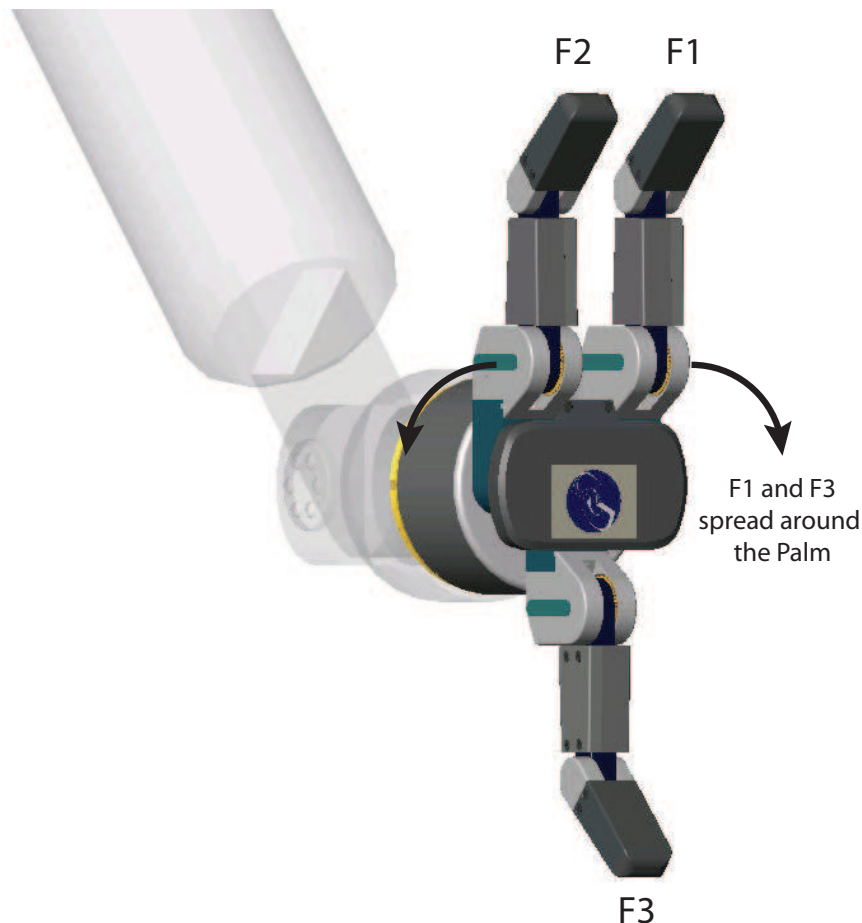


Figure 1: BarretHand

The newer model BH280 is more advanced than BH262 in many aspects and offers some new features. It is equipped with pressure profile sensors (PPS), mounted on each

finger and the palm. Each of these surfaces consists of 24 individual sensors. The resolution of sensors' output is 10 bits. The BH280 also introduces a better position measurement resolution. The encoders used in the BH280 model provide about 10-times better resolution than those used in BH262. Another major improvement is the communication protocol. The BH280 can be connected to the host machine using the faster CAN connection.

1.1 Installation

Installing a BarrettHand is quite simple. The BH262 model uses serial communication, and the BH280 connects using CAN interface, the supplied USB-CAN adapter can be used. Before use, hand software must be installed. This is simply done using an installer located on the USB stick. It installs all necessary drivers, control software and the API. See the instructions in the manual [1], which can be found on the USB stick. The GUI manual [2] provides information on how to use the very intuitive BarrettHand Control GUI. Short instructions are in the Quickstart guide[4]. API documentation [3] is provided in the HTML format.

All software including source files, the report and USB stick contents are stored here:

```
smb://balsa/Interno/Roboti/BarrettHand/BH_2011
```

Github repository contains everything but the USB stick content.

```
git://github.com/mmmatjaz/BarrettHand.git
```

NOTE: When using the BH262 after it had not been used for more than two days, the firmware must be updated. This can be easily done using BattetttHand Control GUI on either Windows or Linux. Click firmware update button on the first tab in the GUI and follow the instructions. If not sure, see [2].

1.2 The repository

The Supervisory and Realtime servers run on both Windows and Linux. The sources of these and the offline template are cross platform. On linux the applications are built from Terminal using *make* command while in the corresponding directory. To rebuild on Windows, open the *bh.sln* solution file in Visual Studio. The solution also contains projects *bhLauncher* and *bhSclient*, which use .NET framework and are not cross platform. The binaries will appear in *bin-win* and *bin-linux* folders respectively. *port2win* includes definitions that are necessary for compilation of the Linux code on Windows.

2 Supervisory Control

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 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 [1].

2.1 Supervisory Server

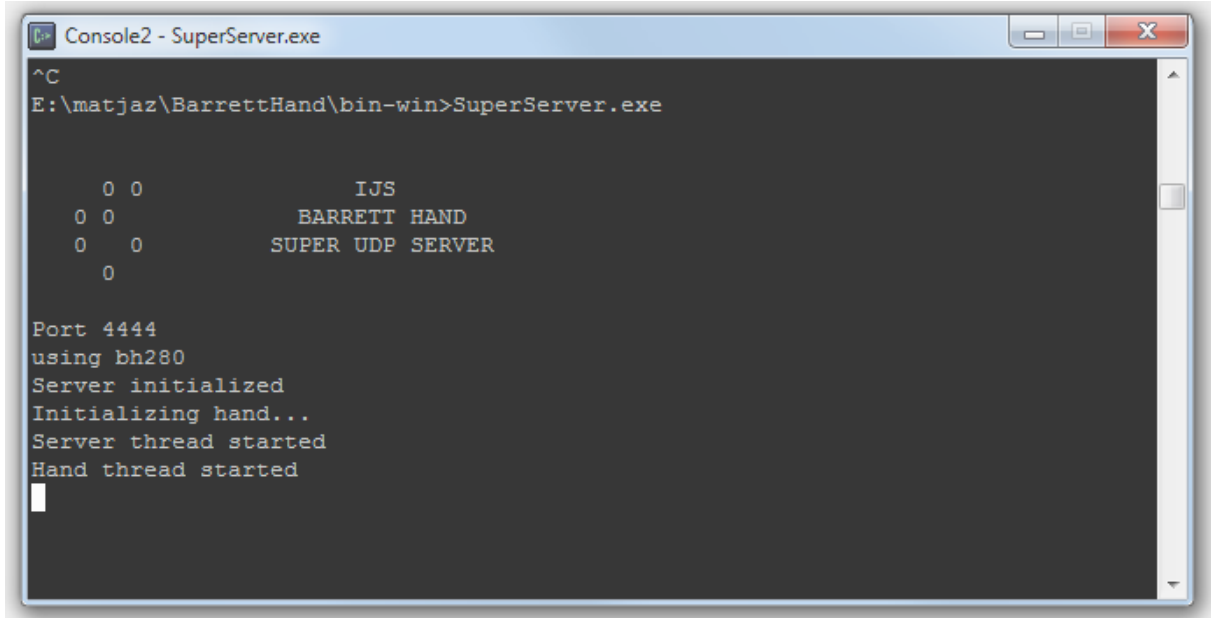


Figure 2: Supervisory server.

The BHSupervisory server application enables remote control of either BH280, BH262 or both simultaneously. While running on the machine connected to the hand, it applies the commands received from a client machine in a local network. A few parameters must be set to start the server application. These are given as arguments in the order as listed in table 1. If application is launched without any arguments, default settings are used. The following line starts the server on port 5555, and only the bh280 will be controlled.

```
./SuperServer 5555 1 0
```

Parameter	Description	Default
BH280	0-disable 1-enable	1
BH262	0-disable serial port no	1
Port	UDP communication port	4444

Table 1: List of parameters

The client, connected by the UDP protocol, sends commands to be set to the grippers. The list of available commands is installed together with BarrettHand software (browse to *%BHAND%/Manuals*). Some commands return a message, this message is forwarded to the client. The commands must be provided in a string of ASCII characters:

@ "H262_command" @ "BH280_command"

Following a single character '@' is the command to be set to the BH262 model, and the BH280 command follows the second '@' character.

2.2 Supervisory Matlab Script

The following code sends supervisory commands to the grippers from Matlab. The BH262's fingers should form a closed fist (gc=grip close) and BH280's outer fingers should point opposite to the middle finger (so=spread open). The code uses PNET library for sending data over UDP, and is provided with the example. See /SuperClientMatlab in the repository.

```
r=udp('193.2.6.112', 5555, 'LocalPort', 5555 );  
fopen(r);  
fwrite(r, '@gc@go')  
fscanf(r)  
fclose(r);
```

2.3 Supervisory Windows Client

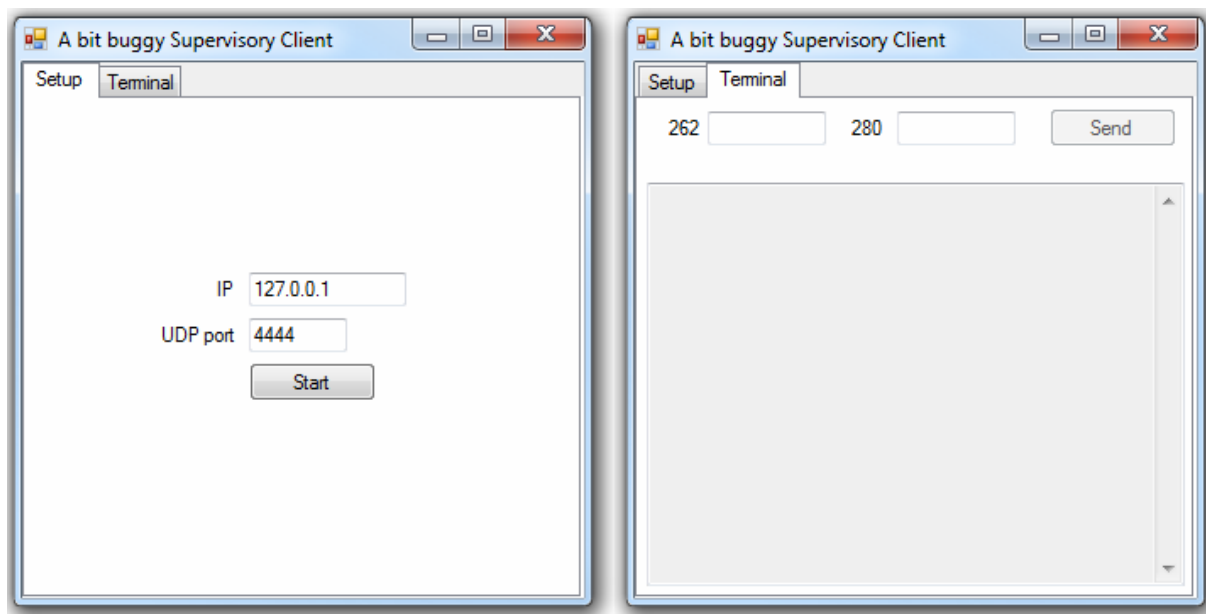


Figure 3: Supervisory Client application.

Figure 3 shows the Setup tab (left) and the Terminal tab (right). After setting the connection parameters, clicking the *Start* button starts the communication. In the Terminal tab the *Send* button is now enabled. The commands for each hand should be input in the textboxes on top. The sent and received history is printed in the Terminal field.

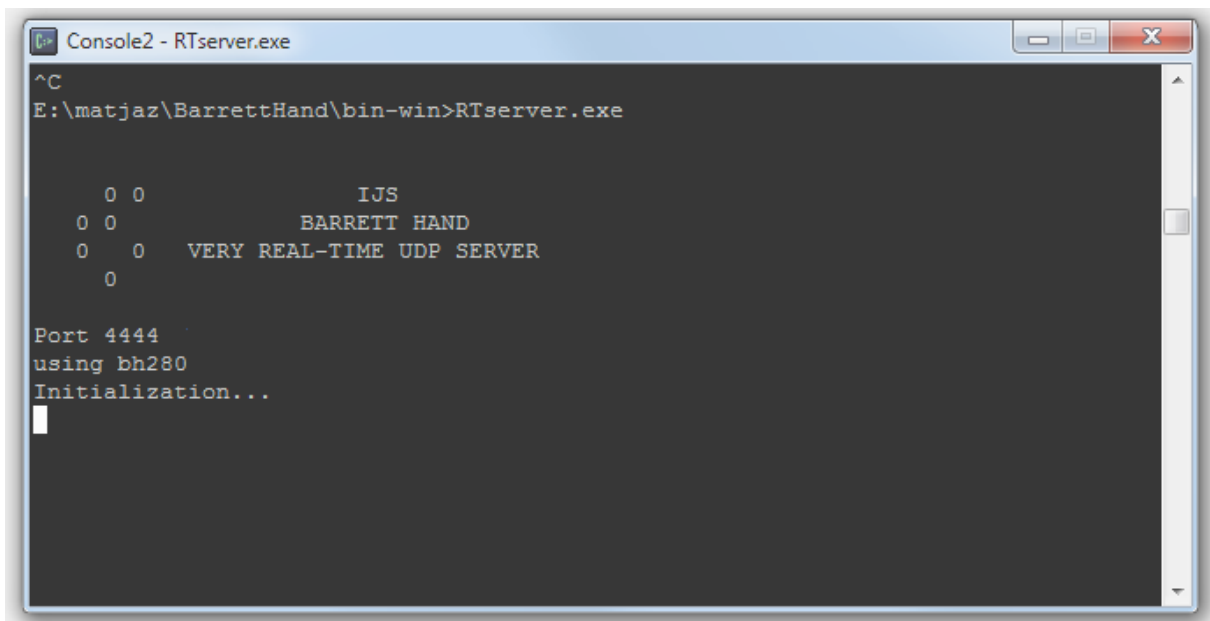
3 Real-Time Control

RealTime mode extends the capability of the 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 [1].

3.1 Real-Time Server

The BHserver application allows real time control of the gripper. Once started, it initializes the gripper. The client connects to the server using UDP protocol, and sends either velocity, position, or desired torque values to the server. The server application applies these values and replies with values read from the gripper (see Figure 5) All read values are of type double (8 bytes), the PPS data is of type int (4 bytes).

Figure 5 also displays the sum of data exchanged between the client and server in bytes (B) when controlling both hands. The sum of the PPS output data is much larger than all the other data read from the gripper's internal controller. When PPS reading is enabled, the delay caused by communication over CAN increases from about 3 ms to 33 ms, which equals the delay when using the serial communication, not reading the PPS sensors. Though the BH280 supports serial communication, transferring the PPS data over it would cause an unacceptable delay.



```
Console2 - RTserver.exe
^C
E:\matjaz\BarrettHand\bin-win>RTserver.exe

    0 0          IJS
    0 0      BARRETT HAND
    0 0  VERY REAL-TIME UDP SERVER
    0

Port 4444
using bh280
Initialization...
█
```

Figure 4: Real-Time server console.

Again, a few parameters must be set to start the server application. These are given as arguments listed in table 2. If application is launched without any arguments, default settings are used. The following command launches the server that will listen at port 5555. It will open the visualization window (argument "3") and initialize the BH262 model at serial port COM1.

```
./RtServer 5555 3 1
```

Parameter	Description	Default
BH280	0-disable 1-enable 2-read PPS 3-show PPS graphics	1
BH262	0-disable serial port no	1
Port	UDP communication port	4444

Table 2: List of parameters

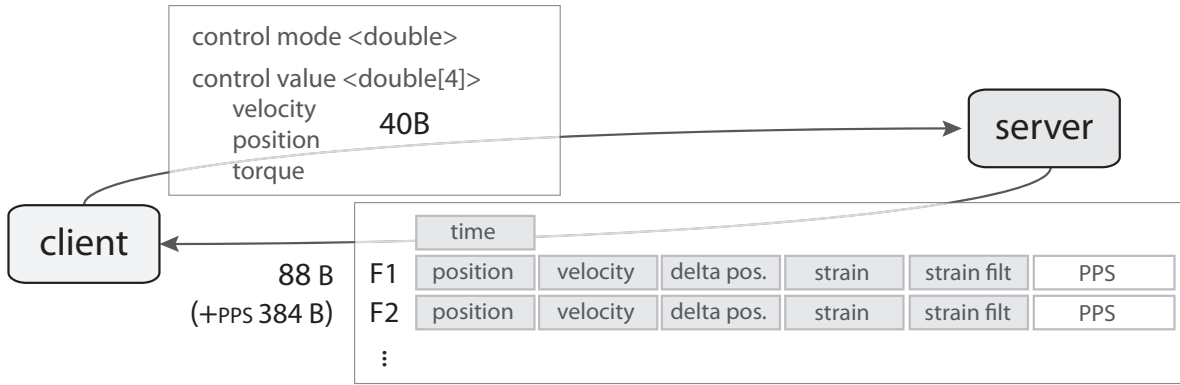


Figure 5: Server-client traffic diagram.

The application consists of multiple threads. One runs the server, commands to each gripper are processed in separate threads. Figure 4 shows the application console window, where messages of each thread are printed. When launched, application opens a window visualizing the sensor data (Figure 6). Pressure sensor output is displayed in shades of blue, higher pressure causes lighter shade. Pressure sensors are only available with the BH280 model. Strain gage sensor output is displayed as orange rectangles, the output level defines the shade of orange color. The outer fingers are rotated according to actual position.

3.2 Client in Simulink

Figure 7 shows the simulink block and its preferences window. In the *Control* tab (left) control modes and connection parameters can be configured. Unused output ports can be disabled in second tab *Outputs*.

3.3 Control Modes

The real-time server provides four modes of control to choose from.

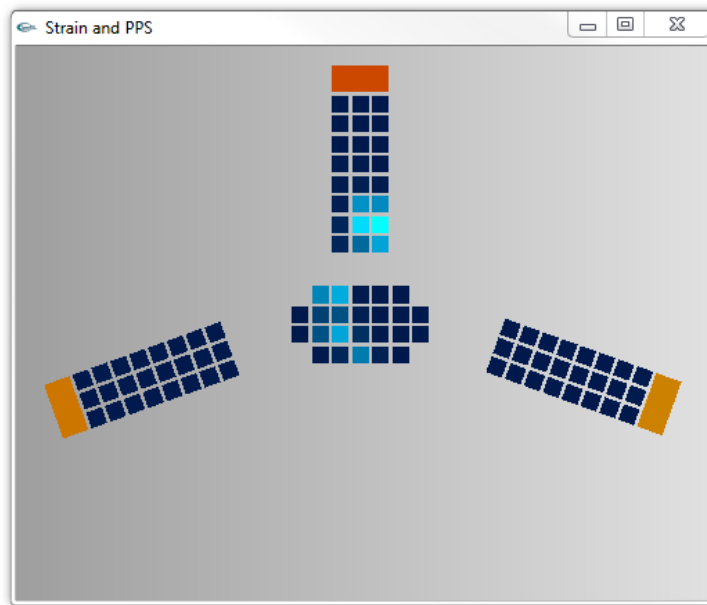


Figure 6: Sensor output visualization.

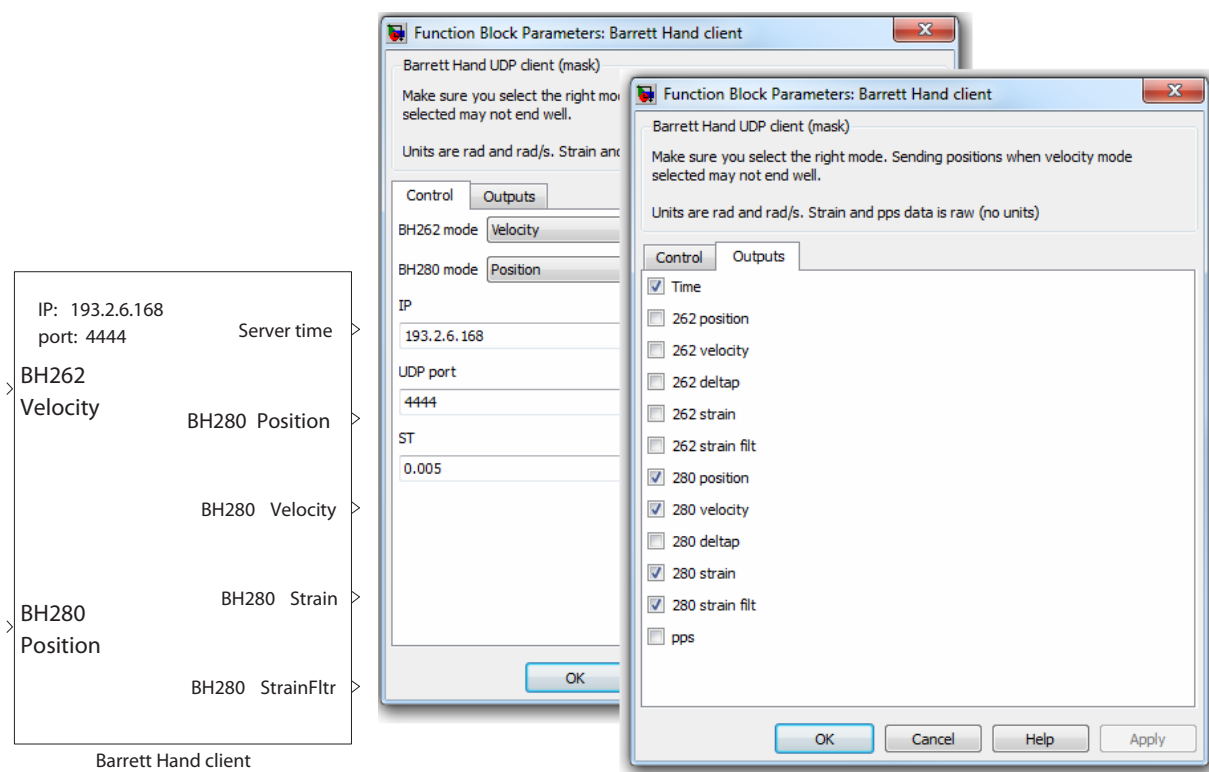


Figure 7: Simulink model.

Velocity control. When this mode is selected the server expects velocities in radians per second.

Position control. Server receives motor positions in radians.

Torque control. Server expects values of torque. For example, to overcome the static friction of the fingers about 400 units of torque is required (API documentation doesn't specify units of torque).

Other. This mode is reserved for any custom implementation of control on server side. As an example a custom position control is implemented. To implement a custom controller, you must select one of the three control modes in function *int Begin()* in the *switch* statement under *case CUSTOM_CONTROL:*. The *SendToHand()* defines the commands that are executed each iteration. Current and previous measured values are stored in structs *Meas* and *pMeas*, and the commands received from the client can be accessed in *Cons* and *pCons*.

4 Graphical Server Launcher

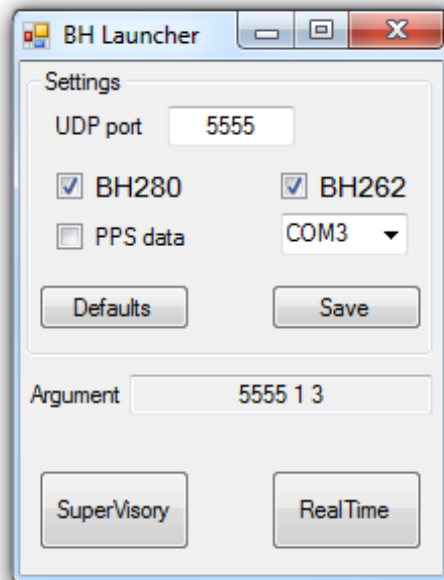


Figure 8: Graphical Server Launcher.

On Windows both real-time and supervisory servers can also be started using the graphical interface, application BHLauncher (see Figure 8). The user can select the necessary parameters, and once set, parameters can be saved. The servers are launched by clicking buttons *Supervisory* or *RealTime*. The bottom textbox shows the assembled string of arguments. When changes are detected, the program offers to save them to an *.xml* file. If the settings file isn't found at program start, default settings are loaded.

By default, the launcher and server applications are in the same folder. If the server applications cannot be located, the BHLauncher displays a dialog to browse for the executable. If you haven't renamed them, you can just delete the config XML file and restart the GUI.

5 C++ templates

The OfflineTemplate project contains the following .cpp files:

- *main.cpp* is the main file, program entry is here. It contains the `main()` function. After it creates instances of all classes, it launches the threads, and waits for a key press to terminate.
- *bh262.cpp* and *bh280.cpp* contain a class for each hand. First, the hand and the software are initialized. The hand now accepts supervisory commands. To control it in real time, the Realtime mode must be started using `RTStart()`. Function `RunRealTime()` is executed each iteration. It sets velocities or positions and reads the data from the hand sensors.
- *glutWindow.cpp* contains the GLUT / freeGLUT functions to display the BH280 sensor data.

The ClientTemplate project is an alternative to using the client in simulink.

- *main.cpp* is the main file, program entry is here. It contains the `main()` function. After it creates an instance of the Client class it generates the position reference and sends it to the client in a single thread.
- *client.cpp* contains a class definition that handles the Ethernet communication.

References

- [1] Barrett Technology Inc, BarrettHand BH8-Series User manual Firmware Version 4.4.x, 2010
- [2] Barrett Technology Inc, Control GUI Manual Version 4.4.3, 2010
- [3] Barrett Technology Inc, Barrett Hand API Documentation Version 4.4.3, 2010
- [4] Barrett Technology Inc, BarrettHand Control SDK Quickstart Guide, 2010