Contents

[HexControl v3 2](#_Toc112079783)

[Overview 2](#_Toc112079784)

[Examples 2](#_Toc112079785)

[Hexapod Object 2](#_Toc112079786)

[Consumer Object 3](#_Toc112079787)

[TimerAsync Object 3](#_Toc112079788)

[Hardware Setup 4](#_Toc112079789)

[API 5](#_Toc112079790)

[UDPHex 5](#_Toc112079791)

[Hexapod 6](#_Toc112079792)

[BuildOptions (Standard Functions) 12](#_Toc112079793)

# HexControl v3

## Overview

[Standard Functions](#_BuildOptions_(Standard_Functions)) – Functions for hexapod operations. Read/Write register data, convert desired arm movement (mms) to a byte array, and converting the hexapod response (for a read/write/move command) from a byte array to a string. Also contains a library of the FN registers.

[UDPHex](#_UDPHex) – Extends the Standard functions class. Creates a UDP Connections and send/receives messages to/from the local and remote computer.

[Hexapod](#_Hexapod) – Creates a UDPHex object. Starts the logging file and connects the UDP. Handles receiving, building the byte array of, and sending the desired commands. Handles resetting the hexapod. Can read and write settings, converts from string to byte array and sends the command, and receives the response and translates from byte array to string. Handles the math for conversion of degrees and frequency to actual arm positions (mms)

Consumer – Class to create a background task that consumes items with an associated order number. This is used to make sure several consumers finish consuming the same item number. Used by the Hexapod Class.

TimerAsync – Async friendly Timer implementation. Provides a mechanism for executing an async method on a thread pool thread at specified intervals. In the Class1.cs file.

## Examples

### Hexapod Object



### Consumer Object



### TimerAsync Object



## Setup

* Software library on [Github](https://github.com/ocular-motor-lab/HexapodControl/tree/main/HexControlv3/HexControl)
* Connect via Ethernet.
* Set local computer IP to “192.168.15.100”

## API

### UDPHex

#### Public Variables

* None

#### Functions

* Connect()
  + Purpose:
    - Establish a UDP connection between the host computer and the remote computer
  + Detail:
    - Creates the UDP connection using the specified port numbers by sending a broadcast signal (IPremote variable ending in “.255”) heard by any computer in the 192.168.15.XX IP range. The remote computer has an IP address of 192.168.15.201, so when the broadcast is sent, the remote computer responds, and the two computers establish a UDP connection.
  + Input:
    - string IPremote
      * Default value of “192.168.15.255”
    - int UDPremote
      * Default value of 7408
    - string IPlocal
      * Default value of “192.168.15.100”
    - int UDPlocal
      * Default value of 8410
  + Output:
    - UDPHex connection
* UdpSendReceive()
  + Purpose:
    - Send a command from the host computer to the remote computer and receive a response from the remote computer
  + Detail:
    - This function is an asynchronous task, which lets it operate outside of the strict linear timing of regular functions. This task sets up a local listener for the response, then sends the byte array to the remote computer, and then awaits the response locally before ending. This ensures that the command is sent and that regardless of the exact timing of the response, a response is received.
  + Input:
    - byte array bytes
  + Output:
    - byte array response
* Disconnect()
  + Purpose:
    - Break the connection between the host computer and the remote computer
  + Detail:
    - This function calls upon the [Close()](https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.udpclient.close?view=net-6.0) and [Dispose()](https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.udpclient.dispose?view=net-6.0) function of a UDP client. This ensures that the connection is stopped, and that the ports are freed for use by other application.
  + Input: None
  + Output: None

### Hexapod

#### Public Variables

* IPremote
  + Type: string
  + Purpose: Set the remote IP address
  + Default: “192.168.15.255”
* UDPremote
  + Type: int
  + Purpose: Set the remote port number
  + Default: 7408
* IPlocal
  + Type: string
  + Purpose: Set the local IP address
  + Default: “192.168.15.100”
* UDPlocal
  + Type: int
  + Purpose: Set the local port number
  + Default: 8410
* DataFolder
  + Type: string
  + Purpose: Set the folder that will hold log and data files
  + Example: "C:\\Users\\USERNAME\\Desktop";
* Connected
  + Type: Boolean
  + Purpose: Check whether a connection currently exists or not

#### Functions

* aysncInitialize()
  + Purpose:
    - Creates a .csv file to log commands and responses
      * Default filename:
        + “Hexapod\_yy\_M\_dd\_HH\_mm\_sent.csv”
    - Set-up real-time data collection
    - Establish a UDP connection between the host computer and the remote computer
  + Detail:
    - Uses [StreamWriter](https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-6.0) to write lines to the designated .csv file
    - Creates a #####Consumer<string> recording task to allow for real-time logging of data
    - From the created hexapod object, uses the [UDPHex.Connect()](#UDPHex_Connect) function
  + Input: None
  + Output: None
* asyncMove()
  + Purpose:
    - Send a position command from the local computer to the remote computer for hexapod movement
  + Detail:
    - Takes the input variables and turns them into a byte array using #####BuildOptions.MoveBuildCommand()
    - Turns the byte array into a string for logging using #####BuildOptions.InterpretMoveCommandResponse()
    - Sends the byte array using [UDPHex.UdpSendReceive()](#UDPSendRecieve)
    - Once a response is sent from the remote (Hexapod) computer #####BuildOptions.InterpretMoveCommandResponse() is called to convert the byte array into a string for logging
    - Make a use the function #####Consumer.TryAdd() to log the sent message and the response message in string format
  + Input:
    - double x
      * Desired position of the x hexapod arm in millimeters
    - double y
      * Desired position of the y hexapod arm in millimeters
    - double z
      * Desired position of the z hexapod arm in millimeters
    - double u
      * Desired position of the u hexapod arm in millimeters
    - double v
      * Desired position of the v hexapod arm in millimeters
    - double w
      * Desired position of the w hexapod arm in millimeters
    - int time
      * Current time, used for log-keeping
  + Output: None
* asyncReset()
  + Purpose:
    - Sets all hexapod arm positions to 0
  + Detail:
    - Same process as [Hexapod.asyncMove()](#AsyncMove), but with the arm length values all set to 0
  + Input: None
  + Output: None
* Initialize()
  + Purpose:
    - Wrapper for [Hexapod.asyncInitialize()](#AsyncInitialize)
    - Ease of use
* Move()
  + Purpose:
    - Wrapper for [Hexapod.asyncMove()](#AsyncMove)
    - Ease of use
* Reset()
  + Purpose:
    - Wrapper for [Hexapod.asyncReset()](#AsyncReset)
    - Ease of use
* Stop()
  + Purpose:
    - Stops logging data
  + Detail:
    - Calls #####Consumer.Stop()
* ReadSetting()
  + Purpose:
    - Check the status/value of registers on the hexapod
  + Details:
    - Uses #####BuildOptions.RegistryReadCommand() to convert the input parameters into a byte array
    - Then uses [UDPHex.UdpSendRecieve()](#UDPSendRecieve) to send the command and receive a response as a byte array
    - Uses #####BuildOptions.InterpretReadRegistryCommandResponse() to translate the response byte array into a human-readable register status
    - Records the register status in the .csv
  + Input:
    - string RegType
      * Select whether you want to read a “CN”, “DN”, or “FN” register
    - int StartAddress
      * List the memory address of the register
    - int NumberParameters
      * How many bytes of data to read
  + Output:
    - string response
* ChangeSetting()
  + Purpose:
    - Alter the value of registers on the hexapod
  + Details:
    - Uses #####BuildOptions.RegistryWriteCommand() to convert the input parameters into a byte array
    - Then uses [UDPHex.UdpSendRecieve()](#UDPSendRecieve) to send the command and receive a response as a byte array
    - Uses #####BuildOptions.InterpretReadRegistryCommandResponse() to translate the response byte array into a human-readable register status
    - Records the register status in the .csv
  + Input:
    - string RegType
      * Select whether you want to read a “CN”, “DN”, or “FN” register
    - int StartAddress
      * List the memory address of the register
    - int NumberParameters
      * How many bytes of data to read
    - int ParamCounter
      * #####
    - int array ParamValues
      * An array of the new values to set the registers
  + Output:
    - string response
* RecordToFile()
  + Purpose:
    - Write a string to a csv file
  + Details:
    - Adds the Time to the string input
    - Sends to the logging file
  + Input:
    - string logline
  + Output:
    - bool true
* CalculateArmLength()
  + Purpose:
    - Take the desired height and degree rotations of the hexapod and converts them to displacement of motor arms
  + Details:
    - Contains the mechanical constraints of the hexapod
    - Uses a transformation matrix to output motor arm displacement
  + Input:
    - double x
      * forward/back position
    - double y
      * left/right position
    - double z
      * up/down position
    - double phi
      * pitch
    - double theta
      * yaw
    - double psi
      * roll
    - double RefX
      * rotation reference x (forward/back)
    - double RefY
      * rotation reference y (left/right)
    - double RefZ
      * rotation reference z (up/down)
  + Output:
    - double array ArmExtensionMMS

#### Private Functions

Functions for use in matrix operations needed by [Hexapod.CalculateArmLengths()](#HexapodCalculateArmLength)

* scalerOp()
  + Purpose:
    - Multiply/subtract a vector by a scaler
  + Detail:
    - Goes through each value in the vector array and performs the desired operation
  + Input:
    - double array vectory
      * The array to be multiplied/subtracted by the scaler
    - string operation
      * Which operation to choose: “times” or “minus”
    - double value
      * The scaler to be multiplied/subtracted from the vector
  + Output:
    - double array output
* summerSqrt()
  + Purpose:
    - Sums each individual row in a column of a matrix and takes the square root
  + Detail:
    - Creates a double array holding the sums of each row as an element
  + Input:
    - double 2D array matrix
  + Output:
    - double array sumHolder
* minuser()
  + Purpose:
    - Elementwise subtraction of two matrices
  + Detail:
    - Subtract second from first
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array output
* pluser()
  + Purpose:
    - Elementwise addition of two matrices
  + Detail:
    - Add first and second matrix
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array output
* elementTimeser()
  + Purpose:
    - Elementwise multiplication of two matrices
  + Detail:
    - Multiply first and second matrix
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array output
* timeser()
  + Purpose:

Matrix multiplication of two matrices

* + Detail:
    - Multiply first and second matrix
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array c
* RepMat()
  + Purpose:
    - Repeat a vector to create a matrix
  + Detail:
    - Takes a 1D double array and repeats the array to fill a matrix
    - Repeats along the column dimension
  + Input:
    - double array Inputer
      * The array to repeat
    - int dim
      * Dimension to repeat along, currently only 1 is supported
    - int numReps
      * The other dimension size of the output matrix (the first dimension size being the size of Inputer)
  + Output:
    - double 2D array repmat

### BuildOptions (Standard Functions)

#### Public Variables

* leadDistanceUnit
  + Type: const uint
  + Purpose: Number of millimeters per turn of the motor
  + Default: 5 (mm)
* oneTurnPulseNum
  + Type: const uint
  + Purpose: Number of electrical pulses per turn of the motor
  + Default: 10000 (pulses)
* confirmCode
  + Type: const ushort
  + Purpose: confirmation code from the host to the remote computer
  + Default: 0x55aa
* passCode
  + Type: const ushort
  + Purpose: inform the receiving computer to accept the message
  + Default: 0
* whoAccept
  + Type: const ushort
  + Purpose: determine what IP address is able to accept the message
  + Default: 0xFFFF
* whoReply
  + Type: const ushort
  + Purpose: determine what IP address is able to respond to the message
  + Default: 0xFFFF
* functionCode
  + Type: Set what the message does
  + Purpose: For a move command, can set whether in delta time or absolute time, can set to be a read register command or a write register command
  + Default:
    - Absolute time: 0x1401 (Default)
    - Delta time 0x1301
    - Read Register: 0x1101
    - Write Register: 0x1201
* objectChannel
  + Type: ushort
  + Purpose: set what channel to use for communication
  + Default: 1
* absTime
  + Type: const uint
  + Purpose: set the time each command should take (ms)
  + Default: 10 (ms)
* SFXOut
  + Type: uint
  + Purpose: for any special effects in communication (set to default values)
  + Default: 0x1234
* analogOut
  + Type: uint
  + Purpose: used if using an analog output (set to default values)
  + Default: 0x5678abcd
* frameNum
  + Type: int
  + Purpose: set the number of the command
  + Default: 0

#### Functions

* aysncInitialize()
  + Purpose:
    - Creates a .csv file to log commands and responses
      * Default filename:
        + “Hexapod\_yy\_M\_dd\_HH\_mm\_sent.csv”
    - Set-up real-time data collection
    - Establish a UDP connection between the host computer and the remote computer
  + Detail:
    - Uses [StreamWriter](https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-6.0) to write lines to the designated .csv file
    - Creates a #####Consumer<string> recording task to allow for real-time logging of data
    - From the created hexapod object, uses the [UDPHex.Connect()](#UDPHex_Connect) function
  + Input: None
  + Output: None
* asyncMove()
  + Purpose:
    - Send a position command from the local computer to the remote computer for hexapod movement
  + Detail:
    - Takes the input variables and turns them into a byte array using #####BuildOptions.MoveBuildCommand()
    - Turns the byte array into a string for logging using #####BuildOptions.InterpretMoveCommandResponse()
    - Sends the byte array using [UDPHex.UdpSendReceive()](#UDPSendRecieve)
    - Once a response is sent from the remote (Hexapod) computer #####BuildOptions.InterpretMoveCommandResponse() is called to convert the byte array into a string for logging
    - Make a use the function #####Consumer.TryAdd() to log the sent message and the response message in string format
  + Input:
    - double x
      * Desired position of the x hexapod arm in millimeters
    - double y
      * Desired position of the y hexapod arm in millimeters
    - double z
      * Desired position of the z hexapod arm in millimeters
    - double u
      * Desired position of the u hexapod arm in millimeters
    - double v
      * Desired position of the v hexapod arm in millimeters
    - double w
      * Desired position of the w hexapod arm in millimeters
    - int time
      * Current time, used for log-keeping
  + Output: None
* asyncReset()
  + Purpose:
    - Sets all hexapod arm positions to 0
  + Detail:
    - Same process as [Hexapod.asyncMove()](#AsyncMove), but with the arm length values all set to 0
  + Input: None
  + Output: None
* Initialize()
  + Purpose:
    - Wrapper for [Hexapod.asyncInitialize()](#AsyncInitialize)
    - Ease of use
* Move()
  + Purpose:
    - Wrapper for [Hexapod.asyncMove()](#AsyncMove)
    - Ease of use
* Reset()
  + Purpose:
    - Wrapper for [Hexapod.asyncReset()](#AsyncReset)
    - Ease of use
* Stop()
  + Purpose:
    - Stops logging data
  + Detail:
    - Calls #####Consumer.Stop()
* ReadSetting()
  + Purpose:
    - Check the status/value of registers on the hexapod
  + Details:
    - Uses #####BuildOptions.RegistryReadCommand() to convert the input parameters into a byte array
    - Then uses [UDPHex.UdpSendRecieve()](#UDPSendRecieve) to send the command and receive a response as a byte array
    - Uses #####BuildOptions.InterpretReadRegistryCommandResponse() to translate the response byte array into a human-readable register status
    - Records the register status in the .csv
  + Input:
    - string RegType
      * Select whether you want to read a “CN”, “DN”, or “FN” register
    - int StartAddress
      * List the memory address of the register
    - int NumberParameters
      * How many bytes of data to read
  + Output:
    - string response
* ChangeSetting()
  + Purpose:
    - Alter the value of registers on the hexapod
  + Details:
    - Uses #####BuildOptions.RegistryWriteCommand() to convert the input parameters into a byte array
    - Then uses [UDPHex.UdpSendRecieve()](#UDPSendRecieve) to send the command and receive a response as a byte array
    - Uses #####BuildOptions.InterpretReadRegistryCommandResponse() to translate the response byte array into a human-readable register status
    - Records the register status in the .csv
  + Input:
    - string RegType
      * Select whether you want to read a “CN”, “DN”, or “FN” register
    - int StartAddress
      * List the memory address of the register
    - int NumberParameters
      * How many bytes of data to read
    - int ParamCounter
      * #####
    - int array ParamValues
      * An array of the new values to set the registers
  + Output:
    - string response
* RecordToFile()
  + Purpose:
    - Write a string to a csv file
  + Details:
    - Adds the Time to the string input
    - Sends to the logging file
  + Input:
    - string logline
  + Output:
    - bool true
* CalculateArmLength()
  + Purpose:
    - Take the desired height and degree rotations of the hexapod and converts them to displacement of motor arms
  + Details:
    - Contains the mechanical constraints of the hexapod
    - Uses a transformation matrix to output motor arm displacement
  + Input:
    - double x
      * forward/back position
    - double y
      * left/right position
    - double z
      * up/down position
    - double phi
      * pitch
    - double theta
      * yaw
    - double psi
      * roll
    - double RefX
      * rotation reference x (forward/back)
    - double RefY
      * rotation reference y (left/right)
    - double RefZ
      * rotation reference z (up/down)
  + Output:
    - double array ArmExtensionMMS

#### Private Functions

Functions for use in matrix operations needed by [Hexapod.CalculateArmLengths()](#HexapodCalculateArmLength)

* scalerOp()
  + Purpose:
    - Multiply/subtract a vector by a scaler
  + Detail:
    - Goes through each value in the vector array and performs the desired operation
  + Input:
    - double array vectory
      * The array to be multiplied/subtracted by the scaler
    - string operation
      * Which operation to choose: “times” or “minus”
    - double value
      * The scaler to be multiplied/subtracted from the vector
  + Output:
    - double array output
* summerSqrt()
  + Purpose:
    - Sums each individual row in a column of a matrix and takes the square root
  + Detail:
    - Creates a double array holding the sums of each row as an element
  + Input:
    - double 2D array matrix
  + Output:
    - double array sumHolder
* minuser()
  + Purpose:
    - Elementwise subtraction of two matrices
  + Detail:
    - Subtract second from first
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array output
* pluser()
  + Purpose:
    - Elementwise addition of two matrices
  + Detail:
    - Add first and second matrix
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array output
* elementTimeser()
  + Purpose:
    - Elementwise multiplication of two matrices
  + Detail:
    - Multiply first and second matrix
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array output
* timeser()
  + Purpose:

Matrix multiplication of two matrices

* + Detail:
    - Multiply first and second matrix
  + Input:
    - double 2D array first
    - double 2D array second
  + Output:
    - double 2D array c
* RepMat()
  + Purpose:
    - Repeat a vector to create a matrix
  + Detail:
    - Takes a 1D double array and repeats the array to fill a matrix
    - Repeats along the column dimension
  + Input:
    - double array Inputer
      * The array to repeat
    - int dim
      * Dimension to repeat along, currently only 1 is supported
    - int numReps
      * The other dimension size of the output matrix (the first dimension size being the size of Inputer)
  + Output:
    - double 2D array repmat