Project Midas

# Summary

Project Midas is a system which allows an individual to control a computer using a gesture-recognizing device without the need for the traditional keyboard and mouse peripherals. Currently, Project Midas has been designed to support the Myo armband, which is a wearable technology that interprets both angular data and EMG data to recognize orientation and poses of a user’s arm/hand. Individuals using Midas will be able to perform all operations that could be accomplished with a keyboard and mouse.

Midas is accompanied by a separate application, MidasProfileManager, which is used to modify the specific behavior of Midas at run-time, to provide the end user with more flexibility.

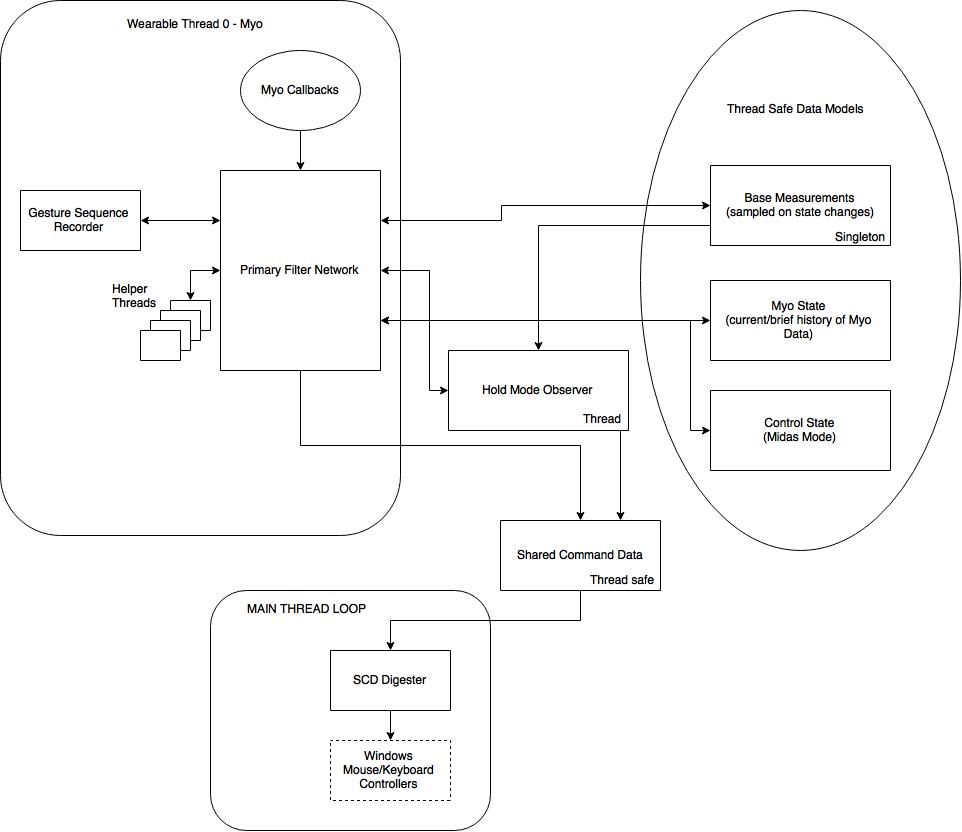
# Architecture

Midas’ architecture is shown in Figure 1. The system primarily functions by receiving callback functions from the Myo armband, translating these into windows commands (via a filter network) and sending them to be ‘digested’ and thereby executed.

To note, the system has been designed in a generic way such that its main thread loop is independent of the type of wearable device used as a data source. As long as data gets placed into the Shared Command Data structure, it will be digested and executed by Midas. Thus far, only the Myo armband has been used to provide data, so the rest of the architecture is designed to interact with the Myo.

In the wearable thread, enumerated 0 since it is the only device in the system, the focus is translation. Data comes in from the Myo and is processed via a network of filters (expanded upon later), which interact with a Gesture Sequence Recorder, many helper threads (for things like timers), a Hold Mode Observer, and multiple thread safe data models. Once processed, it passes data to the Shared Command Data object to be executed.

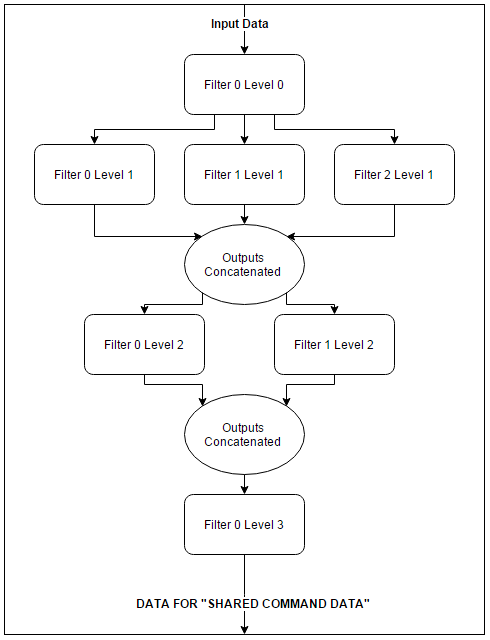
Figure System Architecture



# Filters

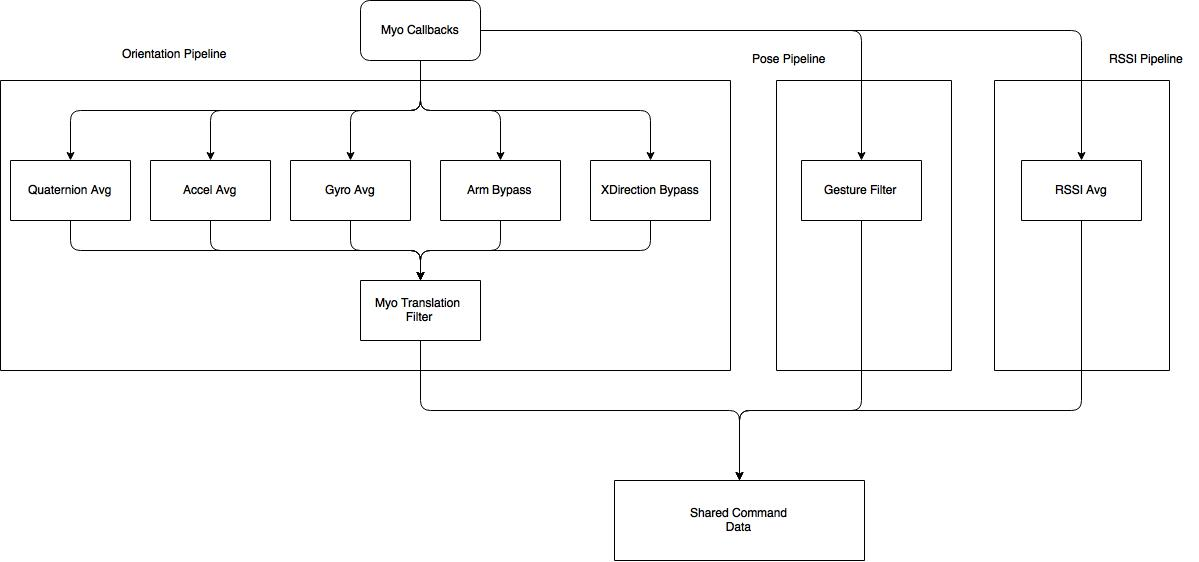
Filters can be anything that manipulates data. They have discreet inputs and discreet outputs, which are programmatically represented as Boost::Any values, so that they can be generic. The programmer must ensure that filters do adequate Boost checks to ensure that they have proper data being fed into them before being processed. Filters get registered into a graph structure that processes data one level at a time and passes all data output from that level into all inputs within the next level. This is shown in Figure 2

Figure Filter Graph Structure



The structure of the primary filter network in Midas is shown within Figure 3.

Figure Primary Filter Network



The filters currently designed in the system are shown in Table 1.

Table Midas Filters

|  |  |  |  |
| --- | --- | --- | --- |
| **Filter Name** | **Filter Inputs** | **Filter Outputs** | **Purpose** |
| GenericBypassFilter | Any. | Identical to input. | Flow data within a single filter graph without modifying it. |
| GenericAveragingFilter | Float value. | Average of previous *avgCount* floats that were input. | Smooth out data from sensors by only passing a variable average of the data down the pipeline. |
| GestureFilter | Myo Gestures. | Midas specific commands for the SCD, with outlier removal. Ex: mouse click. | Translate Myo gestures into usable SCD, while also ensuring that the user intended to provide the gesture (by ensuring hold time greater than a reasonable threshold). |
| MyoTranslationFilter | Spatial data, represented as floats. Also various Myo Signals (Arm/xDir). | Midas specific commands for the SCD, reliant on spatial sensors from the Myo. Ex: cursor movement data, calculated from arm angles | Translate Myo sensory data into usable SCD, specifically focusing on spatial data translations. |
| EMGImpulseFilter | Myo EMG data. | Boolean stating whether or not an impulse has been detected in the EMG data. | Detect impulses in EMG data as to predict when a gesture is about to be recognized by Myo Connect. This can potentially increase the usability in mouse mode to reduce jarring effects caused by poses. |
| SharedCommandData | SCD specific command data. | None. | Store commands in a queue to be digested later in the data flow path. |

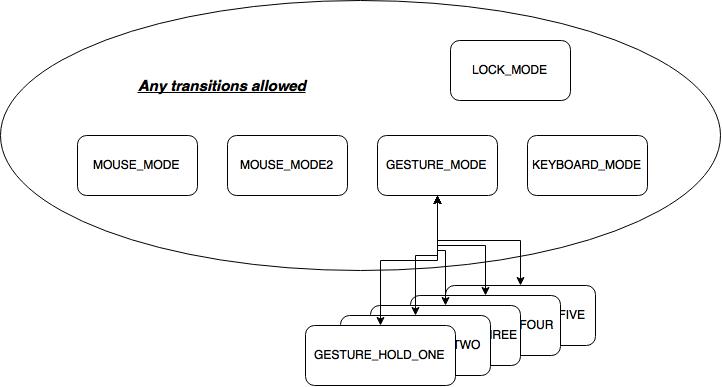
# Main Functionality

Midas functions by modularizing its t­asks. Each device has its own thread, on which it receives data, processes the data, and stores it in a SCD memory location. The SCD is the only component that the main thread of Midas requires.

The main thread acts as a command center. It is constantly polling all available SCD locations, retrieving commands, and then executing them. Through this cycle, it is able to obtain data in a format specific to Midas, and execute all cursor/keyboard commands.

However, commands which can be executed must be limited by some means, or else the user could experience trouble controlling their computer efficiently. Thus, a state machine is being employed to switch between modes that enable/disable certain functionality. Figure 4 shows the available states of Midas.

Figure 4 Midas State Information



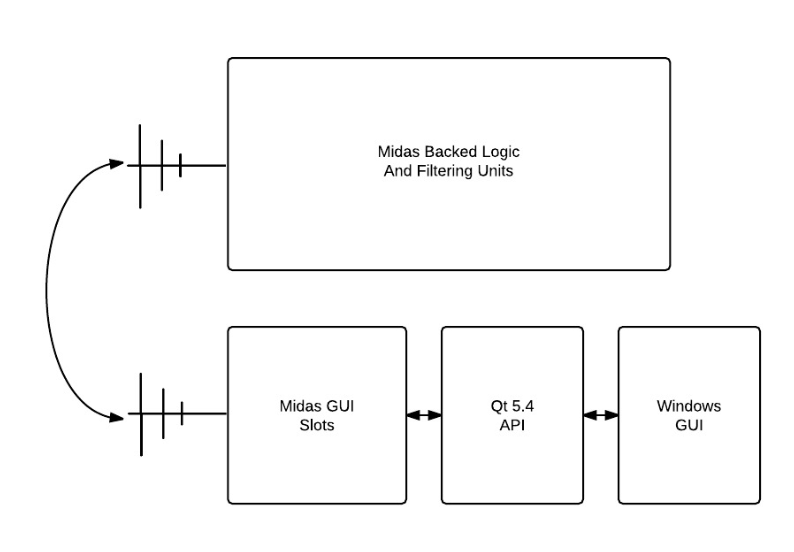
Although the states are pre-defined, Midas allows the user to define which states are accessible and how they are accessed. Gesture hold modes are only intended to be accessed through the gesture mode, however, since they all default to return to the gesture mode when there is a release signal in that state. User definition of state transitions is done through the MidasProfileManager application, which is described later. The states have the following properties:

* Mouse Mode: Move the cursor along with the motion of the users arm, and be able to perform any other desired actions.
* Mouse Mode 2: Move the cursor along with the motion of the users arm, and be able to perform any other desired actions.
* Gesture Mode: Able to perform and desired actions.
* Keyboard Mode: A virtual keyboard appears and the motion of the users arm determines which key is selected. Keyboard GUI actions must be defined by the user.

# Midas GUI

Midas is built with Qt 5.4.0. Qt provides a signal/slot architecture, which Midas makes use of. Simply, Midas emits signals in various locations (filters, etc), which a GUI thread listens for. When the GUI thread receives these signals (in slots), it updates the GUI accordingly. Likewise, there are some buttons in the GUI, which when pressed, emit signals that slots in the Midas backend receive and handle to modify the internal state. This is shown pictorially in Figure 5.

Figure 5 Midas GUI Interaction with Qt



The GUI in Midas is used to provide active-pose, state and available sequence information to the user, as well as to provide a means for input: a settings widget and a profile-selection widget.

# Midas Profile Manager

The MidasProfileManager module handles loading an XML file that contains the profile information and re-packaging the information from the file into a vector of profile structs.

## XML File Format

The profile file is represented by XML using a set of custom elements and attributes. Figure 6 shows an example profile file that demonstrates the structure of the file. A full example can be found in the appendix. This format was chosen as to utilize Boost, a powerful library, for reading and writing as to increase modularity.

Figure Example Profile File



Each profile has ability to use some/all of the following XML elements:

* Sequences
  + Sequence – with relevant state (state that the sequence is applied to) and name
    - Gestures
      * Gesture – with type and value
    - Commands
      * Command – with type
        + Actions

Action name

* Holds
  + Hold – with gesture value
    - Angle – with type
      * anglePositive
        + Command – with type

Actions

Action name

* + - * angleNegative
        + Command – with type

Actions

Action name

* + - * Sensitivity value
    - HoldModeActionType enum value
    - Interval length

The acceptable values, with descriptions, are outlined in the Appendix.

## Midas Back-End Support

The profile file is loaded at the beginning of the program and all of the information for each profile is stored in the MidasProfileManager. Project Midas checks for the selection of a new profile in the Myo-specific device thread. If a new profile is selected, the sequence information must be updated, and the virtual method updateBasedOnProfile is called on all of the registered filters. Currently, only the GestureFilter and MyoTranslationFilter override this method to support loading a profile.

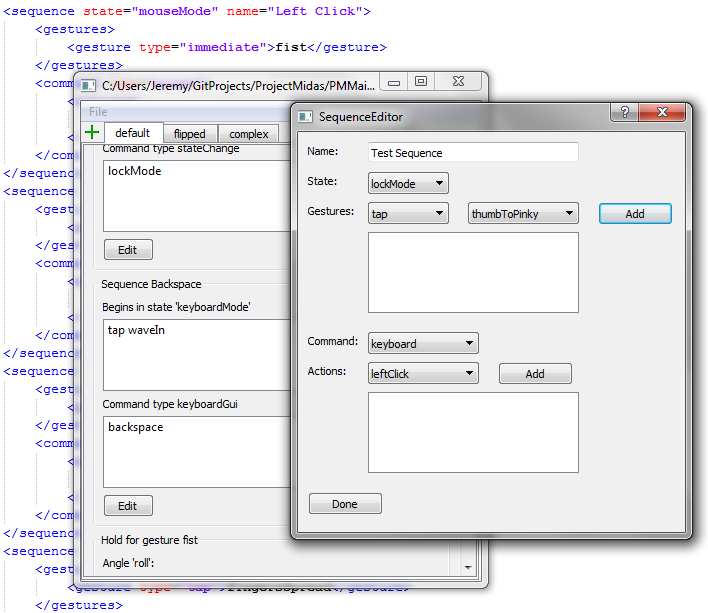
The GestureFilter does so by parsing all of the sequences in the selected profile (it retrieves this information from the MidasProfileManager). With the help of a set of constant maps, which map the strings the profile understands to constants the GestureSequenceRecorder understands, it re-packages the data from the MidasProfileManager into the form needed to register sequences. In this way, each sequence from the profile is registered.

The MyoTranslationFilter works similarly to the GestureFilter in terms of registering profiles, except that it handles hold-gesture information as opposed to sequences. It loops through each hold in the MidasProfileManager and uses constant maps to translate from the profile to the constants needed by Midas.

## Profile Creator

The profile creator GUI shown in Figure 6 is used to create profile files. The files can be saved for later work and loaded for editing. If a user adds another sequence to a profile in the GUI, a sequence editor window is displayed. This window is used for defining a sequence. The hold editor window is similar to the sequence editor window but is used to edit holds. The combination of all sequences and holds added by the user are placed into a profile struct. All of the profiles, which are separate tabs in the GUI, are stored in a vector of profile structs.

Figure Profile Creator



### Constraint Checking

There are some constraints that sequences/holds must pass in order to be added. The most noteworthy constraint is that all sequences must be prefix-free from one another. Ie, if one sequence is A-B-C, no other sequence can start with A-B-C (ex: A-B-C-D).

### Saving and Loading a File

XML profile files can be loaded or saved via this GUI.

# Appendix

## Profiles XML Allowable Values with Descriptions

|  |  |  |
| --- | --- | --- |
| **XML Element** | **Allowable values (with descriptions if non-obvious)** | |
| Sequence States | lockMode  gestureMode  mouseMode  keyboardMode  gestureHoldOne  gestureHoldTwo  gestureHoldThree  gestureHoldFour  gestureHoldFive | |
| Gesture Type | hold: Executes once held for > REQ\_HOLD\_TIME ms.  tap: Executes if release in < REQ\_HOLD\_TIME ms  immediate: Executes immediately once seen | |
| Gesture Value | fingersSpread  fist  rest  doubleTap  waveIn  waveOut | |
| Command Types and Action names | mouse | leftClick  rightClick  middleClick  leftHold  rightHold  middleHold  moveLeft  moveRight  moveUp  moveDown  moveHor  moveVert  scrollLeft  scrollRight  scrollUp  scrollDown  leftRelease  rightRelease  middleRelease  releaseLrmButs |
| keyboard | undo  redo  zoomIn  zoomOut  zoom100  escape  enter  tab  switchWinForward  switchWinReverse  copy  paste  cut  fileMenu  newBrowser  gotoAddrBar  lockDesktop  editMenu  viewMenu  winHome  hideApps  control  volumeUp  volumeDown  backspace  upArrow  downArrow  rightArrow  leftArrow  inputVector  none |
| keyboardGui | swapRingFocus  changeWheels  select  holdSelect  undo  redo  zoomIn  zoomOut  zoom100  escape  enter  tab  switchWinForward  switchWinReverse  copy  paste  cut  fileMenu  newBrowser  gotoAddrBar  lockDesktop  editMenu  viewMenu  winHome  hideApps  control  volumeUp  volumeDown  backspace  upArrow  downArrow  rightArrow  leftArrow  none |
| stateChange | lockMode  mouseMode  keyboardMode  gestureMode  gestureHoldOne  gestureHoldTwo  gestureHoldThree  gestureHoldFour  gestureHoldFive |
| profileChange | moveProfileForward  moveProfileBackward |
| Hold gesture value | fingersSpread  fist  rest  doubleTap  waveIn  waveOut | |
| Hold angle type | roll  pitch  yaw | |
| Hold command details | Same as sequence command | |
| Hold angle sensitivity | float: How many degrees to execute the hold-angle command, but with different functionality based on hold mode action type. | |
| Hold - hold mode action type | absDeltaFinite: However many multiples of “Hold angle sensitivity” is detected in a delta angle is how many times the command will be executed. This results in **repeatable** behavior. Ex, if you move your arm then return to the starting location, the same number of POSITIVE commands will execute as NEGATIVE commands. absDeltaVelocity: However many multiples of “Hold angle sensitivity” is detected in a delta angle dictates the velocity in which the command will be executed. It will be executed angle/sensitivity times per intervalLen milliseconds.  intervalDelta: Much like absDeltaFinite, except it is **not deterministic**. It will only look at how much change has occurred in a given intervalLen milliseconds and determine whether or not to execute a command. NOTE, this allows SLOW movement to not execute the command, whereas fast movement will execute it as if it were in absDeltaFinite mode. | |
| Hold interval length | int: The interval used in the calculations done with “hold mode action type” described above. | |

## Full Example Profiles.xml File

<?xml version="1.0" encoding="utf-8"?>

<profiles>

<profile name="Advanced Profile - Trial 3 - holds">

<sequences>

<sequence state="lockMode" name="Unlock">

<gestures>

<gesture type="tap">doubleTap</gesture>

</gestures>

<commands>

<command type="stateChange">

<actions>

<action>gestureMode</action>

</actions>

</command>

</commands>

</sequence>

<sequence state="gestureMode" name="Rotate Atrium">

<gestures>

<gesture type="immediate">waveIn</gesture>

</gestures>

<commands>

<command type="stateChange">

<actions>

<action>gestureHoldTwo</action>

</actions>

</command>

</commands>

</sequence>

<sequence state="gestureMode" name="Left Click">

<gestures>

<gesture type="tap">fist</gesture>

</gestures>

<commands>

<command type="mouse">

<actions>

<action>leftClick</action>

</actions>

</command>

</commands>

</sequence>

<sequence state="gestureMode" name="Move Cursor">

<gestures>

<gesture type="immediate">fingersSpread</gesture>

</gestures>

<commands>

<command type="stateChange">

<actions>

<action>mouseMode</action>

</actions>

</command>

</commands>

</sequence>

<sequence state="mouseMode" name="Release Cursor">

<gestures>

<gesture type="immediate">rest</gesture>

</gestures>

<commands>

<command type="stateChange">

<actions>

<action>gestureMode</action>

</actions>

</command>

<command type="mouse">

<actions>

<action>releaseLrmButs</action>

</actions>

</command>

</commands>

</sequence>

<sequence state="gestureMode" name="Lock">

<gestures>

<gesture type="tap">doubleTap</gesture>

<gesture type="immediate">doubleTap</gesture>

</gestures>

<commands>

<command type="stateChange">

<actions>

<action>lockMode</action>

</actions>

</command>

</commands>

</sequence>

</sequences>

<holds>

<hold gesture="fingersSpread">

<angle type="pitch">

<anglePositive>

<command type="keyboard">

<actions>

<action>upArrow</action>

</actions>

</command>

</anglePositive>

<angleNegative>

<command type="keyboard">

<actions>

<action>downArrow</action>

</actions>

</command>

</angleNegative>

<sensitivity>0.35</sensitivity>

</angle>

<holdModeActionType>absDeltaFinite</holdModeActionType>

<intervalLength>20</intervalLength>

</hold>

</holds>

</profile>

</profiles>