InvenSense

SmartMotion Server Technical Specifications

1. PRODUCT OBJECTIVES

The MoveaTV platform, with its SmartMotion Server, provides advanced motion sensing and motion control platform services to the Interactive TV ecosystem.

The SmartMotion services completely abstract the complexity involved in dealing with sensors. Interactive TV providers can focus on content and add rich motion aware context for an improved entertainment user experience without leaving the couch.

This document is intended to facilitate the integration of the SmartMotion Server by describing every feature available. It will also explain how to interact with the SmartMotion Server, from the input frame to the JSON API to configure it.

2. ACRONYMS

You'll find below a list of acronyms you may encounter in this document.

API	Application Programming Interface
JSON	JavaScript Object Notation
STB	Set Top Box
RCU	Remote Control Unit
SMS	SmartMotion Server

3. PRODUCT ARCHITECTURE

INTRODUCTION

The SmartMotion Server is a program running as a background process. It can be seen as 4 different packages:

- The ***Core*** is the infrastructure element of the SmartMotion Server framework. This package is responsible for managing every other package.

- ***Devices*** allow you to send raw data to the SmartMotion Server. Several method are available, including sending the data through a UDP channel or reading them on an HID device or even using a custom file to inject the data to the SMS.

- ***Features*** are the main components where the data are processed from raw value to motion feature. For example, the *Pointing* feature will transform the raw data into a 2D displacement. Each feature can be activated / deactivated individually. Each feature can be associated with only one device.

- ***Actions*** are the outputs of the SmartMotion Server. You can, for example, emulate a standard mouse with the action *PointingToCursor* attached to the feature *Pointing*. Each action has to be attached to at least 1 feature. Note that some actions are compatible with only certain features (for example, the action *PointingToCursor* will be compatible only with the *Pointing* or *AbsolutePointing* features). Other actions are compatible with multiple features (the action *Log* is compatible with every feature).

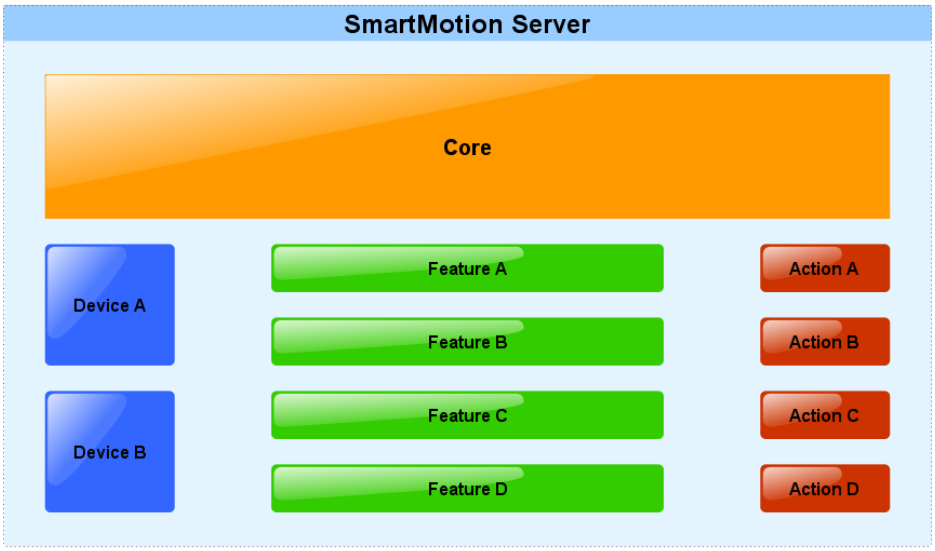


***Figure 1**: Generic architecture of the SmartMotion Server*

EXAMPLES

For example, below is a possible configuration of the SmartMotion Server for *Point & Click* interactions:

- A **Remote** *Device* is associated with 2 motion *Features*: **Pointing** and **Click**
- The **Pointing** *Feature* is associated with a **MouseCursor** *Action* to emulate cursor movement
- The **Click** *Feature* is associated with a **MouseClick** *Action*, to emulate click events, and a **Logger** *Action* to print received click events as console outputs
- A **ConfigServer** *Feature* allows 3rd parties applications to configure the SMS through a network connection (UDP or TCP)

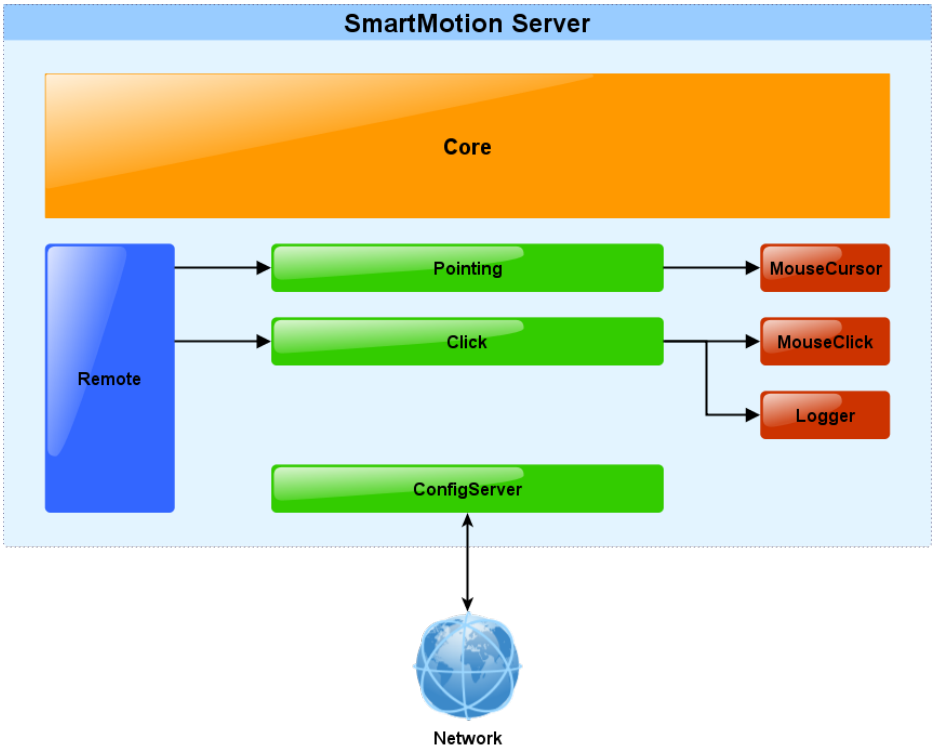


***Figure 2**: Example of configuration for a Point & Click interaction*

A more complex example could be the configuration below for gesture based interactions:

- A **SmartPhone** *Device* is associated with 2 motion *Features*: **Gesture** and **DynamicRotation**
- A **UserIntentionAnticipation** *Feature* allows smart cohabitation between **Gesture** and **DynamicRotation** *Features*
- The **Gesture** *Feature* is associated with a **GestureToKey** *Action* to emulate pre-defined keyboard key press, depending on recognized gesture
- The **DynamicRotation** *Feature* is associated with a **RepeatKey** *Action* to emulate pre-defined keyboard keys press, depending on detected axis and associated angle value
- **Gesture** and **DynamicRotation** *Features* are both associated with a **ForwardEvent** *Action* to send received events to remote application(s)
- A **ContextChanger** *Feature* allows local Middleware applications to dynamically switch active motion context

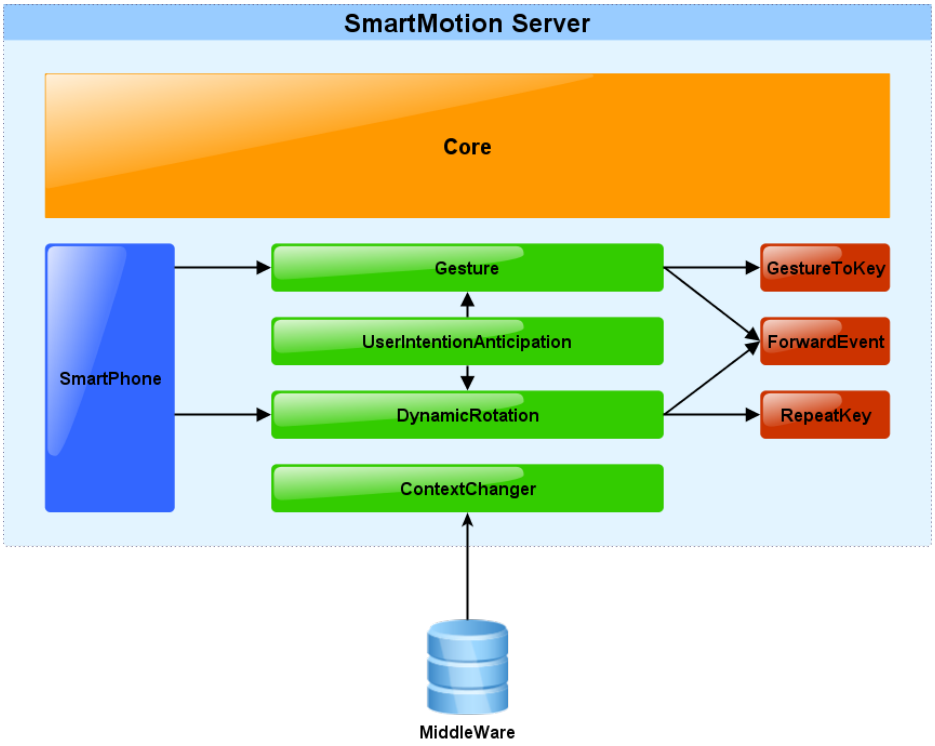


***Figure 3**: Example of configuration for a gesture based interaction*

AVAILABLE DEVICES

DEVICE NAME	DESCRIPTION
Custom USB HID Class	A custom HID report will be sent via USB to a custom HID file and will be then read by the SmartMotion Server. *Requires: **RawHID** kernel module*
Char-device	Usually, if the radio chip is integrated in a STB, a driver running on the kernel space will be responsible for reading the sensor's data and forwarding them to the SMS. In this case, For efficiency and stability purpose, we recommend the following actions items: - **Use a static dedicated character device file** for driver to forward sensor data from Kernel to User space. Created at startup, such file will avoid naming conflicts with other devices and SMS potential connection issues. For example, /dev/mva_sensor. - **Add a specific delimiter to each sensor frame** to improve SMS integrity checking. Delimiters will dramatically improve error handling of inconsistent sensor frames (bad format, length, data). For example, 2 bytes at frame beginning with 0xAAAA hexadecimal value.
UDP Binary	The sensor data frame will be sent through UDP to a specified port where the SMS will be listening. Please, refer to our JSON API Documentation for more information.

AVAILABLE FEATURES

FEATURE NAME	DESCRIPTION
FeatureAcceleration	Compute and notify device acceleration.
FeatureAngularVelocity	Compute and notify device angular velocity.
FeatureCalibration	Monitor and propagate calibration values changes as events. Updated calibration values are stored into file each time the feature is disabled / the context change, and restored on feature enabling.
FeatureClick	Notify click buttons state changes (pressed/released). Working along with a Pointing feature, it improves click and double-click behavior.
FeatureCursorFreeze	Allow smart cohabitation between Pointing feature and Gesture / DynamicRotation features by automatically freezing Mouse cursor while, and after, performing a gesture.
FeatureDataTrigger	Notify on sensor data reception, or data loss.
FeatureDynamicRotation	Detect triggered "single axis rotation" movements along yaw, pitch or roll device axes, then compute relative angle from trigger press position to current one. Axis detection is exclusive: as soon as a "single axis rotation" is identified, associated axis is locked until trigger release.
FeatureGestureOffline	Process triggered gesture recognition and notify recognized gestures as events.
FeatureJoystickButtons	Notify Joystick specific buttons state changes (pressed/released). Supported buttons are: A, B, C, X, Y, Z, TL, TR, TL2, TR2, SELECT, START, MODE, THUMBL, THUMBR.
FeatureLibSensors	*For Android only*. This feature will send raw data and orientation values to a custom LibSensors on an Android OS.
FeatureMagneticField	Compute and notify device magnetic field.
FeatureOrientation	Compute and notify relative device attitude.
FeatureOrientationToJoystick	Compute and notify Joystick axes movements [X,Y], based on device Orientation.
FeaturePointing	Compute and notify Mouse cursor relative movements [dX,dY].
FeaturePointingAbsolute	Compute and notify Mouse cursor absolute position [X,Y].
FeatureSensorData	Propagate sensor data as events, to be used by any Actions.
FeatureServerTCP	Listen to a TCP/IP channel all commands using JSON-RPC syntax.
FeatureServerUDP	Listen to an UDP/IP channel all commands with JSON-RPC syntax.
FeatureTrigger	Notify sensor trigger state changes (pressed/released).
FeatureUIA	Allow smart cohabitation between DynamicRotation and Gesture features.

AVAILABLE ACTIONS

ACTION NAME	DESCRIPTION
ActionButtonsToKey	Emulate key according to Joystick specific buttons.\n Supported buttons are: A, B, C, X, Y, Z, TL, TR, TL2, TR2, SELECT, START, MODE, THUMBL, THUMBR.
ActionForwardToLibSensors	*For Android only.* Forward events from FeatureLibSensors to the actual LibSensors
ActionForwardToUDP	Forward any kind of event using UDP protocol to the specified IP and port. A delay can be set to limit the number of sent events.
ActionGestureToClick	Emulate click on Gesture event
ActionGestureToKey	Emulate key press/release on Gesture event
ActionGestureToSwipe	Emulate a swipe (programmable mouse translation) on Gesture event
ActionGestureToWheel	Emulate wheel on Gesture event
ActionJoystickAxes	Emulate a virtual joystick with two axis X and Y
ActionJoystickButtons	Emulate Joystick specific buttons. Supported buttons are: A, B, C, X, Y, Z, TL, TR, TL2, TR2, SELECT, START, MODE, THUMBL, THUMBR.
ActionJoystickToKey	Emulate key presses according to received Joystick events.
ActionLog	Log received events to default logger (according to configuration of SMS, default is standard console output)
ActionPointingToCursor	Emulate Mouse cursor moves according to received pointing events.
ActionRepeatKey	Emulate repeated key presses according to received DynamicRotation events.
ActionRepeatWheel	Emulate repeated wheel according to received DynamicRotation events.
ActionRotationToMultitouch	Emulate multitouch.
ActionRotationToVolume	Emulate absolute volume setting using key press.
ActionTriggerToClick	Emulate Mouse click events (left/middle/right button pressed/released) according to received Trigger events.

4. FUNCTIONALITIES

This section will detail all the functionality available through the SmartMotion Server.

2D POINT & CLICK

Required features	**FeaturePointing** or **FeaturePointingAbsolute**
Suggested Actions	**ActionPointingToCursor**

On screen 2D pointing purpose is to move a cursor on a display (point and click interfaces) by tracking the angular displacement of the wrist and translate it into a linear displacement of the cursor on display.

2 pointing modes are available in the SmartMotion Server: **Relative** Pointing and **Absolute** Pointing.

With the relative pointing, the cursor will be constrained inside the screen. When you'll reach a border, the cursor will continue to move and a misalignment will be created between your hand position and the cursor position. This behavior is equivalent to a standard mouse.

In the other hand, with absolute pointing, the cursor will continue to move outside of the screen. No misalignment should appear and the cursor will stay aligned with the remote in accordance to the reference position. This behavior is equivalent to a touchpad or a graphic tablet.

Movea recommends using a trigger button to control when the 2D pointing is activated. Such approach serves both a better interaction and optimized power consumption model.

The action of pressing a button on the remote often involves rotational and/or linear displacements of the user's wrist. Thus button press impacts the overall pointing. Therefore, Pointing can be coupled with a Click *feature* in order to remove device specific parasitic motion caused by fingers forcing on button during click and double click.

GESTURE RECOGNITION

Required features	**FeatureGestureOffline**
Suggested Actions	**ActionGestureToClick** or **ActionGestureToKey** or **ActionGestureToSwipe** or **ActionGestureToWheel**

Gesture recognition's purpose is the recognition of a gesture amongst a collection of pre-recorded gestures using Movea's pattern recognition algorithm.

A gesture database is constructed by learning the gestures from a set of users.

The SmartMotion Server gesture recognition feature is responsible for monitoring the motion of the device and emits a signal whenever a motion matches one of the gestures of the database that was constructed during the system setup. The user's motion is captured **between the press and release of specified trigger button**. Gesture recognition is performed right after the trigger button is released.

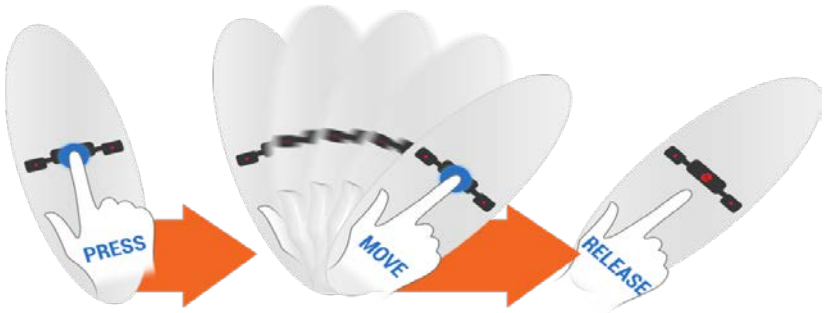
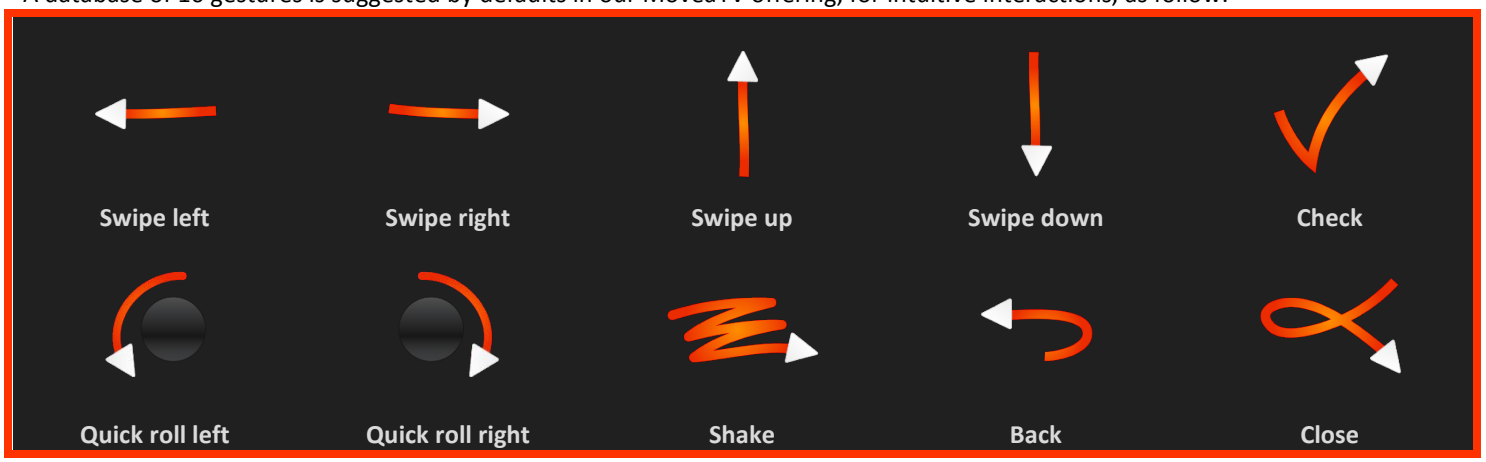


Figure 4: How to do a gesture

A database of 10 gestures is suggested by defaults in our MoveaTV offering, for intuitive interactions, as follow:



Once a gesture has been recognized, the attached action will execute the defined interaction. For example, the 4 swipes (Left, Right, Up and Down) could be bind to keyboard arrows to navigate into the user interface.

DYNAMIC ROTATIONS

Required features	**FeatureDynamicRotation**
Suggested Actions	**ActionRepeatKey** or **ActionRepeatWheel** or **ActionRotationToMultitouch** or **ActionRotationToVolume**

Dynamic rotations purpose is to track rotation angle around particular device axis, in order to associate a graphical user interface driven by continuous data values.

The rotation angle is captured as long as **the user maintains a specified trigger button**.

Typical GUI interactions involve controlling sound volume, image zoom, page scroll, video navigation, etc

USER INTENTION ANTICIPATION

Required features	**FeatureUIA**
Suggested Actions	n/a

User Intent Anticipation purpose is to determine if a user's gesture best applies to a dynamic rotation or a recognized gesture. This allows rich Motion sensitive interactions where all features cohabit well without interfering.

3D ORIENTATIONS

Required features	**FeatureOrientation**
Suggested Actions	n/a

3D orientations purpose is to track the orientation of a particular device in space, relative to an initial reference and allow associating graphical user interfaces to be driven by this information. The output of this feature can be a quaternion, a rotation matrix, or the 3 Euler angles.

Typical GUI interactions involve controlling an object in a 3D scene, or the view of 3D scene itself.

GAMING

Required features	**FeatureOrientation** and **FeatureOrientationToJoystick**
Suggested Actions	**ActionJoystickToKey** or **ActionJoystickAxes**

Motion interactions are not supported by all existing games. However, most of them support gamepad/joystick controller or keyboard events.

In order to support as many games as possible, we developed a feature allowing the SMS to emulate standard joystick/keyboard events based on the orientation of the remote.

Orientation2Joystick *feature* will map the orientation of the remote to an X and Y absolute position (corresponding to a joystick event).

Orientation2Keyboard *feature* will map the orientation of the remote to a repeated key press sequences, the frequency of the repetition depending on the angle of the remote.

MULTI-TOUCH EMULATION

Required features	**FeatureDynamicRotation**
Suggested Actions	**ActionRotationToMultitouch**

Some Android applications will use several fingers to interact with the device, for example pinch to zoom out, two fingers motion to rotate, etc. In order to be compatible with most of the app on the google Play, Movea offers multi-touch emulation with motion.

This action is triggered with Dynamic rotations in some specific applications (Google Maps for example). It will allow you to zoom, rotate or tilt the map.

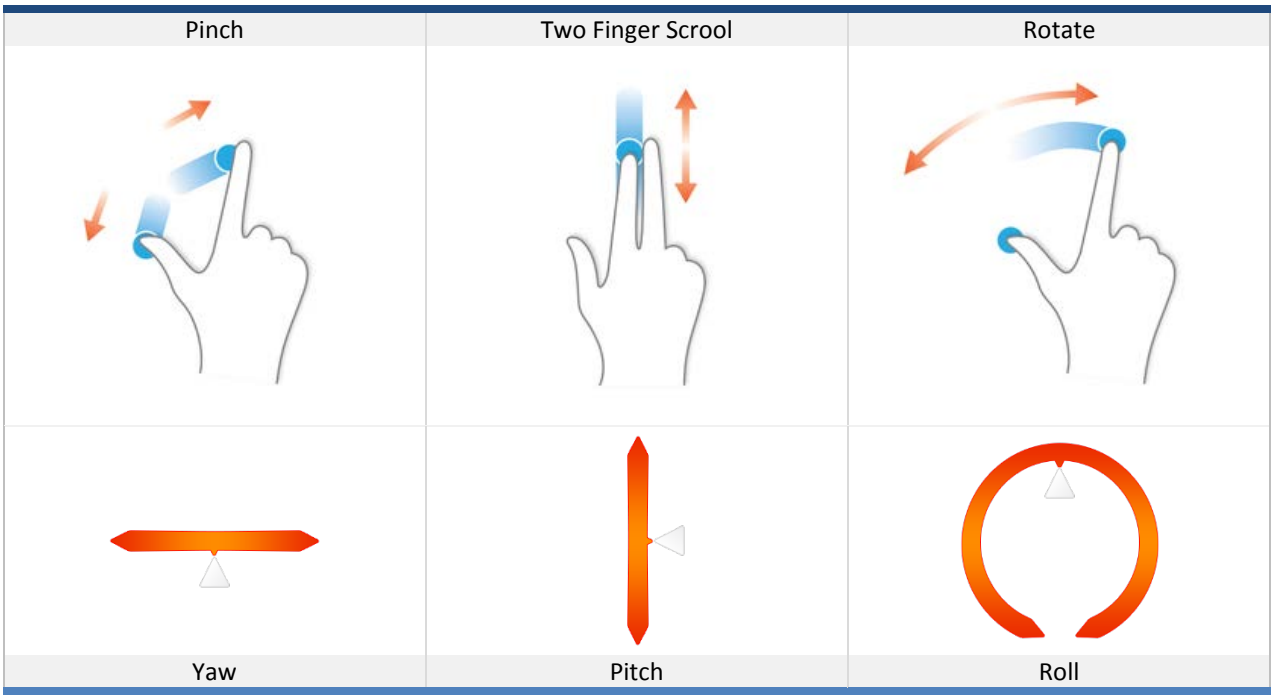


Figure 5: Multi-touch actions and the corresponding binding with our gestures

CALIBRATION

Required features	**FeatureCalibration** (usually protected, so always on)
Suggested Actions	n/a

Most of SmartMotion Server features depend on calibrated sensor signals.

To ensure these features provide the best user experience, Movea recommends to perform a 1st calibration at the time the RCU is 1st paired with the STB. For a 6-axis device, a minimum of 4 seconds of immobility of the RCU is requested to perform this calibration.

To ensure the user experience remains at its best performance level, Movea SmartMotion Server is taking all possible opportunities to perform RCU calibration, and maintain it overtime. Our algorithms are fine tuned to lower CPU usage.

Once performed, the calibration is persistent for the paired RCU and stored on the STB's file system.

5. SMS INTEGRATION INTO A STB

In this section, we will discuss integration related issue and give you some tips to have the best possible user experience.

APPLICATION AWARENESS

SmartMotion Server allows various motion interactions. However, each application will require a specific configuration. Indeed, each application will have its own set of control and the SmartMotion Server will have to adapt its binding for each and any of them.

Application Awareness service supports a per application basis definition of the enabled features and configuration. This supports, for example, associating gestures to generic user input events the applications already know about, thus not requiring any costly adaptation fees to get the application redesigned for standard remote's keys interaction.

VISUAL FEEDBACK

Movea's recommendation is to provide (graphical) user feedback for each gesture in order to make it obvious for the user to realize the gesture has been adequately recognized and can clearly associate the action taken to the motion. Consistent motion to interaction mapping is also a key to a successful user experience.

OFFICIALLY SUPPORTED APPLICATION ON ANDROID TV

The list of applications you can interact with just keeps growing with Movea's SmartMotion Server for Android. The rich feature set and intuitive gesture interactions enable the follow applications on big screen. Please contact Movea if there is an application you would like to see added to this list.

Non exhaustive list of supported application

6. SENSORS INTEGRATION INTO A REMOTE CONTROL UNIT (RCU)

SENSORS SELECTION

Movea recommends including a combination of accelerometers and gyroscopes, typically a 6 axis solution, for the best motion interaction experience. Our SmartMotion Server has been optimized to work with 3 axis accelerometers and 3 axis gyroscopes.

While Movea strives to create sensor agnostic algorithms and not be dependent on specific sensor brand or type, Movea can recommend accelerometers and gyroscopes that have already been successfully integrated into SmartMotion Server.

Accelerometers

For SmartMotion Server algorithms optimal behavior, Movea recommends using 3-axis digital accelerometer.

Resolution	16 bits
Dynamic range	+/- 8g
Output data rate	100Hz

Gyroscopes

For SmartMotion Server algorithms optimal behavior, Movea recommends using 3-axis digital gyroscope.

Resolution	16 bits
Dynamic range	+/- 2000 dps
Output data rate	100Hz

POWER CONSUMPTION

Warning: the following numbers don't take into account the raw data transmission, or the gyroscopes consumption, for example. They simply give an overall overview about typical power consumption data.

*Source: **https://docs.zigbee.org/zigbee-docs/dcn/11-0009.pdf***

The usage model we employed is based upon available customer usage data, with the average consumer watching TV about 8.25 hours per day. The average consumer is assumed to be navigating menus such as the program guide 1.8 times per hour and will watch more half-hour shows but occasionally watching longer programs such as movies or sports throughout their viewing day.

Our usage model also assumes a certain number of button presses for other functions. For example, the model assumes 10 button presses per hour related to TV volume, including mute. Additionally, a remote associated with a set-top box that has a digital video recorder (DVR) function will experience more usage as the user will take advantage of various features such as pause and skip forward.

The power used by the various parts in the remote control will affect battery life. The assumption is that the remote uses AAA alkaline cells wired serially and that the remote ICs can consume 1200 milliamp hour (mA) from the batteries before reaching their lower voltage limit and ceasing to function.

The remote control's processor is assumed to draw 7.5mA when active and 5 microamperes (µA) when idle. When sending ZigBee Remote Control commands, the remote's transceiver is assumed to draw 27mA and 28mA in receive mode when waiting for acknowledgements. This model assumes all ZigBee messages are acknowledged. In idle/sleep mode the transceiver is assumed to draw 2µA. These currents are typical for the parts that are available today for use in remote controls.

For IR transmission, there is a wide range of current draws as various remote manufacturers try to balance IR communication robustness with battery draw. Additionally, the duty cycles for various IR codes and commands vary. We assumed that the IR current draw is 250mA peak with new batteries and with a duty cycle that is 15 percent on, 85 percent off. These values are in the typical range for IR remote controls. Although the average IR current appears to be similar to RF, there is a significant difference in the total time involved. With ZigBee Remote Control, the transceiver will typically be on for ~1millisecond (msec). Allowing for occasional retries of unacknowledged commands, the average power on time for a ZigBee Remote Control transceiver is <<10msec. In the case of a typical IR command, the IR transmission will take >100msec.

	Non-DVR Remote	DVR Remote
IR Only	596	381
ZigBee - STB, IR - TV	904	688
ZigBee Only	1044	766

Table 2 - Calculated Battery Life for a Remote With a Pointing Device in Days

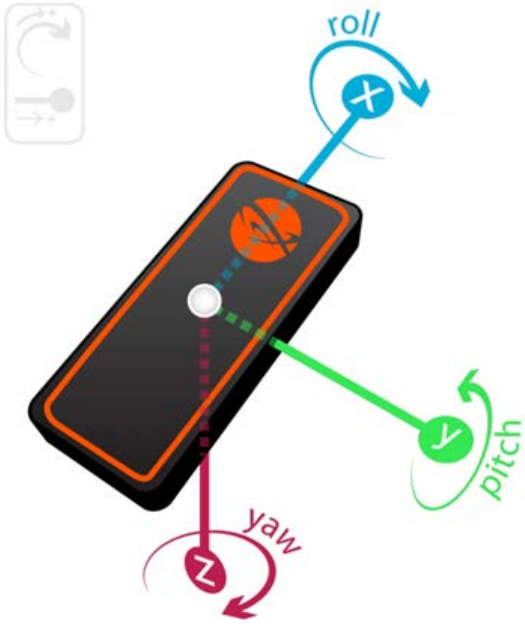
	Non-DVR Remote	DVR Remote
IR Only	200	168
ZigBee - STB, IR - TV	348	310
ZRC Only	367	325

Table 3 - Calculated Battery Life for a Remote Also Used For Internet Activity in Days

COORDINATE SYSTEM

Convention

For your RCU to become a SmartMotion device, we recommend the following sensors axis mapping with regards to the RCU physical configuration.



On the remote, sensors may be mounted in a way that leads to an output order different from the one specified by the Figures above.

Firmware code will have to re-order these outputs for each senor (accelerometer and gyroscope) so the data in the radio frame is Data_X, Data_Y then Data_Z where X, Y, and Z are the axis described by the Figure above.

Accelerometer

For a given axis, the sensor output must be maximal when this axis is in the same direction as the gravity field, i.e. toward the ground. It must be minimal when its direction is the opposite of the gravity, i.e. toward ceiling.

Gyroscope

For a given axis, the sensor output must be positive when the remote is rotated clockwise around this axis and negative when rotated counterclockwise (assuming that the sensors doesn't have any offsets).

LATENCY CONSIDERATION

This section is an inventory of the information commonly found in the literature pertinent for the various possible Motion entertainment interactions.

AUDIO applications

A latency of *10 milliseconds* or better is the target for audio circuits within professional production structures.

VISUAL applications

In simulators with both visual and motion systems, it is particularly important that the latency of the motion system not be greater than of the visual system, or symptoms of simulator sickness may result. This is because in the real world, motion cues are those of acceleration and are quickly transmitted to the brain, typically in less than *50 milliseconds*; this is followed some milliseconds later by a perception of change in the visual scene. Overall 80ms to 100ms system latencies are often recommended in the literacy.

GAMING applications

A strategy game or a turn-based game with a low pace may have a high threshold or even be mostly unaffected by high delays, whereas a twitch gameplay game such as a first-person shooter with a considerably higher pace may require significantly lower delay to be able to provide satisfying gameplay.

NETWORK Gaming applications

The round-trip network latency between a client game and the host server is referred to as the client's ping time.

For the cloud gaming experience to be acceptable, the round-trip lag of all elements of the cloud gaming system (the thin client, the Internet and/or LAN connection the game server, the game execution on the game server, the video and audio compression and decompression, and the display of the video on a display device) must be low enough that the user perception is that the game is running locally. Because of such tight lag requirements, distance considerations of the speed of light through optical fiber come into play, currently limiting the distance between a user and a cloud gaming game server to approximately 1000 miles, according to OnLive, the only company thus far operating a cloud gaming service.

SMARTMOTION SERVER LATENCY / PROCESSING TIME

2D pointing and Dynamic Rotations

2D pointing and Dynamic Rotations are what we call “brain in the loop” interactions, where latency is key for the functionality to be acceptable. Typically latency needs to be kept within 50ms so people cannot detect there is latency in the system for motion to visual interactions.

Movea has put a lot of effort into optimizing its algorithms so there is no observable latency in the response time from those SmartMotion Server features: 2D pointing and Dynamic Rotations processing time are kept within 2.5ms (measurements taken without CPU loads from other STB components), on an Intel Atom Groveland CPU.

Note: Movea cannot anticipate the latency that may be coming from the RCU, the RF4CE radio or the STB hardware/firmware/middleware and GUI component. Movea recommends the overall component stack latency should be kept within 50ms for 2D pointing and Dynamic Rotations rendering to be acceptable.

Gesture recognitions

Gesture recognition interactions are not per say “brain in the loop” interactions. There is a clear user “release the trigger button” action involved that makes the user accept a bit more time form the system reaction.

Movea has put a lot of effort into optimizing its algorithms so there is no latency annoyance in the response time from our SmartMotion Server features: Gesture recognition processing time are kept within 50ms of the trigger button release (measurements taken without CPU loads from other STB components), on an Intel Atom Groveland CPU.

Note: Movea cannot anticipate the latency that may be coming from the RCU firmware, the RF4CE radio or the STB hardware/firmware/middleware and GUI component. Movea recommends the overall component stack latency should be kept within 150ms for gesture recognition rendering to be acceptable.

SMARTMOTION SERVER FOOTPRINT

This section is set of guidelines to help the overall SmartMotion Server integration goes smoothly yet, realizing a Set Top Box is complex system.

CPU usage

Movea has verified CPU usage from SmartMotion Server is stable (no exaggerated CPU surge) and kept to a reasonable number (5% to 7% typical), maximum 20% peak, when all features are activated, on an Intel Atom Groveland CPU.

Memory usage

Movea has verified Memory usage from SmartMotion Server is stable (no memory leaks) and kept to a reasonable number <25MB, when all features are activated, on an Intel Atom Groveland CPU.

INPUTS/OUTPUT

HDD Access

The SmartMotion Server does require Read/Write access to the gesture databases, the context definition, and action mapping configuration and calibration files.

Device Access

For SmartMotion Server integration to proceed decoupled from the RCU or Front Panel development, Movea can provide, on demand, limited numbers of Reference Remote and associated RF4CE receiver dongle. For this to work properly, the 'Raw HID' module will need to be activated in the Linux Kernel.

Adaptors

Once available, the Front Panel driver will be accessed to receive sensor signals from the RCU (Raw Sensor & RCU Ctrl adaptors).

Generic Linux kernel module like 'uinput' will need to be enabled for the SmartMotion standard adaptors to work (mouse, keyboard, remote and raw signal adaptors).

For efficiency and stability purpose, we recommend the following actions items:

- Use a static dedicated character device file for driver to forward sensor data from Kernel to User space.
  Created at startup, such file will avoid naming conflicts with other devices and SMS potential connection issues.
  Ex: *'/dev/mva_sensor'*

- Front panel driver to add a specific delimiter to each sensor frame to improve SMS integrity checking.
  Delimiters will dramatically improve error handling of inconsistent sensor frames (bad format, length, data)
  Ex: 2 Bytes at frame beginning with *'0xAAAA'* hexadecimal value

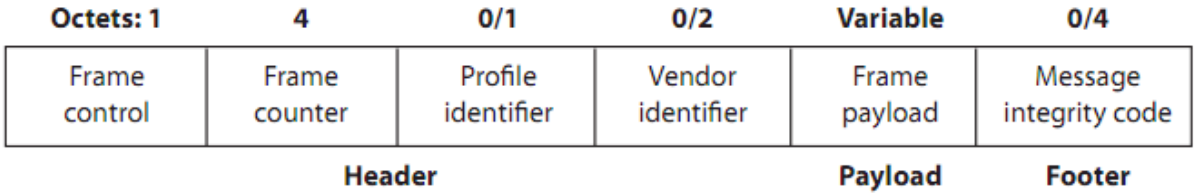
JSON

JSON over UDP/IP or TCP/IP will be required during the debug / integration time for Movea SmartMotion Server testing framework to work (automated test suite and debug tools).

RADIO LINK

Radio packets are sent via RF4CE Zigbee protocol. RF4CE is perfectly suited for a wide range of consumer electronics products, such as TV and STBs.

The general frame format can be found below:



- Frame control: control information for the frame
- Frame counter: incrementing counter to detect duplicates and prevent replay attacks (security)
- Profile identifier: the application frame format being transported
- Vendor identifier: to allow vendor extensions
- Frame payload: contains the application frame
- Message integrity code: to provide authentication (security)

Source: https://docs.zigbee.org/zigbee-docs/dcn/09-5231.PDF

The payload format used is:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ID	Sequence number	Ax LSB	Ax MSB	Ay LSB	Ay MSB	Az LSB	Az MSB	Gx LSB	Gx MSB	Gy LSB	Gy MSB	Gz LSB	Gz MSB	Buttons State

With:

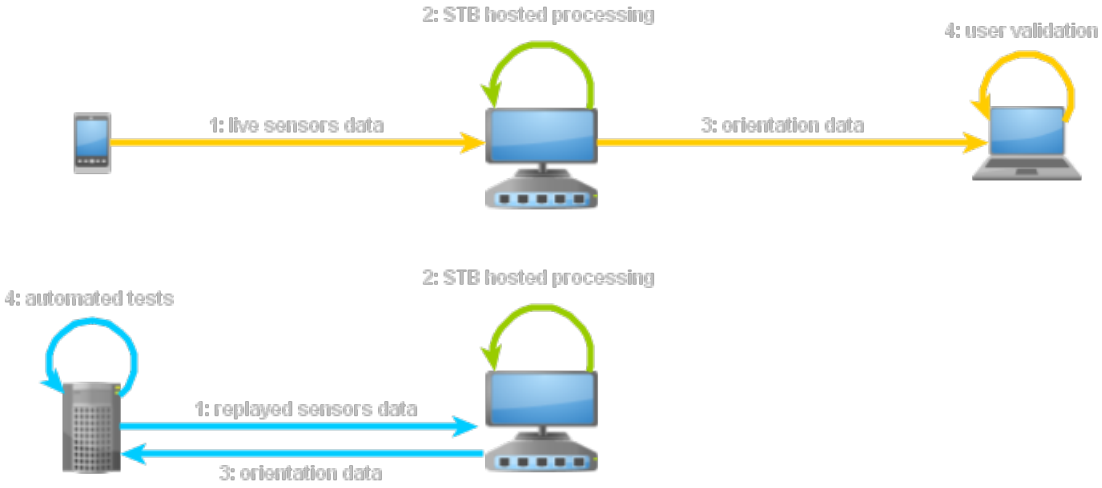
	Buttons State	
	Click state	Trigger state
0x00	released	released
0x01	released	pressed
0x10	pressed	released
0x11	pressed	pressed

VALIDATION

Movea has created a comprehensive set of tests to automatically validate the SmartMotion Server integration on a specific STB. The test suite includes end to end testing of the SmartMotion Server and it various components: framework, features and adaptors.

A subset of the test suite may be adapted to cover specifics of the STB integration. Loads of previously recorded human gestures will be replayed with the new SmartMotion Server instance on the final product CPU ensuring best test coverage.

A number of manual system integration tests will also address quality attributes of each SmartMotion Service. Here again, the JSON API is showing to be handy as the existing set of validations tools available at Movea can be connected to the new SmartMotion Server instance on the final product CPU.



InvenSense