First Person View Gyroscope/Accelerometer Input Controller

Introduction

This asset is meant to be used to directly map the gyroscope's rotational movements to the (main) camera in the scene, thus creating a First Person View when running the game on a handheld device. If no gyroscope is available on the device, the accelerometer will be used to find the device's rotation (note: rotation around the device's y-axis cannot be determined with an accelerometer). In either case, no deeper knowledge of the involved vector calculus and Quaternion Analysis is necessary!

This package can be used in two ways:

- Without the use of any additional coding. Just put the MSP_Input prefab in your scene, and use the inspector to configure which other gameobjects should receive relevant information from the gyroscope, accelerometer, virtual joystick, virtual touchpad and/or virtual buttons. A SendMessage() is being used, and despite the fact that this is slower than directly accessing the scripts using the API, it allows for quick configurations and a flexible workflow.
  The provided demo scene's 1a, 1b en 2a make use of this approach. See the Quickstart section of this manual how to set up your game like this.
- After installation of the package, some additional commands are available through the API. This allows you to read various variables, like the current rotation, or write other variables, like heading offset etc. You can then use these values to further customize your game. This method is used in demo scene 2b and 3. See the Advanced section of this manual which commands are available for you to use in your scripts.

All scripts and demo's where tested on several iOS devices. Some customers have confirmed that the scripts also work very well on their (non-iOS) devices. I have not tested this myself, however. So don't hold me responsible when your character rotates weirdly on your Android or WindowsMobile device ;-)

Tip: To make developing using this asset more convenient, use the *latest* "Unity Remote 4" app (versions prior to version 4 do not support the use of the gyroscope) to test your game directly inside the Unity editor.

For further info, you can contact me at: MouseSoftware@GMail.com

Happy Coding!

Maurits.

Changelog

v5.1
- IMPORTANT: as always, first make a backup of your project, before updating.
- Improved compatibility for older devices that only support accelerometers.
- GetRoll() will now return a clamped result, between -180 and 180 degrees, just like it's GetHeading() and GetPitch() counterparts already did.
- New Api commands GetHeadingUnclamped() and GetRollUnclamped().
- Fixed a bug in autoUpdate, where a clamped roll could suddenly jump from one boundary to another.
- AutoUpdate now supports the option to 'push the viewports edge', when a heading or pitch tries to move beyond it's allowed boundaries.
- GyroAccel, VirtualJoysticks, VirtualTouchpads and VirtualButtons now all support the ability to use the keyboard and mouse to simulate movements of the gyro, joysticks, etc. No more need to always use Unity Remote to test your application ;-)

v5.0
- IMPORTANT: as always, first make a backup of your project, before updating.
- Unity 5.x only
- Heading and pitch can now be forced to a certain value, by using the new API commands SetHeading() and SetPitch(). Finally an easy way to (re)set the rotation of your player character.
- AutoUpdating the orientation of GameObjects now allows you to set boundaries, to keep values within a certain limit. Also, when one chooses not to autoUpdate a certain rotation axis, a fixed value can be chosen (previously, this value would be zero by default).
- Each GameObject in the autoUpdate list can have it's own smoothingTime values.
- OnGUI is only being used while editing the scene. During runtime, all VirtualButtons, VirtualJoysticks and VirtualTouchpads are now directly drawn on screen, to increase drawing speed.
- VirtualButtons, VirtualJoysticks and VirtualTouchpads can now easily be activated or deactivated during runtime, using their new API commands Enable() and Disable().
- VirtualButtons and VirtualJoysticks can now be resized and repositioned during runtime, using their new API commands SetSize() and SetCenter().
- VirtualTouchpads can now be resized and repositioned during runtime, using their new API command SetRect().
- Double tapping VirtualJoysticks and VirtualTouchpads can now be checked by calling their new API commands GetDoubleTap() and GetDoubleTapHold().
- All new demo scenes, based on the sample scenes available from Unity.
- The FPS rigidbody player prefab in the demo scenes has been replaced by a more advanced version. See demo 1a and 1b.
- Various small bug fixes.

v4.6
- IMPORTANT: as always, first make a backup of your project, before updating.
- Configuration of the GyroAccel, VirtualJoystick, VirtualTouchpad and VirtualButtons scripts have greatly improved: There is a specialized custom inspector window for each of them. Customize the gyro's settings and/or create a joystick, touchpad and button with just a view mouse clicks and your done. Al changes are directly visible in the gameview, without running the game.
- Due to the new configuration tools, large parts of the code have been rewritten. The package is backwards compatible, with a few exceptions: The possibility to create a VirtualJoystick, VirtualTouchpad or VirtualButton during runtime has been removed from the API.

- Also the MSP_CharacterMotor.cs script has been removed. It has been replaced by a must more simplistic script, based on RigidBodies. This allows for a better understanding how the input controls can be integrated into your own character movement scripts.
- The package has been tested with and updated for Unity 5.0. The third demo scene (Simple Car Driving Game) still shows some jittering, due to the fact that Unity's 5.x wheelcolliders are not fully compatible with Unity 4.x wheelcolliders. This will be fixed in a future update.
- Please note that this will be the last update for Unity 4.x. All future updates will be Unity 5.x only!

v4.1
- Moved some functionality from FixedUpdate() to Update()
- VirtualButtons can now also be used as a switch
- Fixed a 'missing component'-error in one of the prefabs - fixed swapped naming error of two prefabs ("dual-joystick" <--> "joystick+touchpad") v4.0.5
- Fixed an error in MSP_CharacterMotor:IsTouchingCeiling()
- Fixed an error where a VirtualTouchpad would occasionally not update correctly while using a VirtualJoystick

v4.1
- All new version!!
- Completely rewritten in c#
- Gyro/accel / virtual joysticks / virtual touchpads / virtual buttons are easily configurable in the inspector
- Many new commands, to get/set various settings and variables.
- IMPORTANT: Version 4.x is not compatible with older versions: Please update your existing projects before updating]

v3.1
- For all those people who haven't yet upgraded to Unity 4.x: The asset has been made backwards compatable with Unity Version 3.5.7f6
- Few typo's fixed

v3.0.2
- Now compatible with Unity4.0
- More comment lines added in the scripts
- Readme.txt added with 'how-to-use' instructions

v3.0
- Code has been partly rewritten and optimized

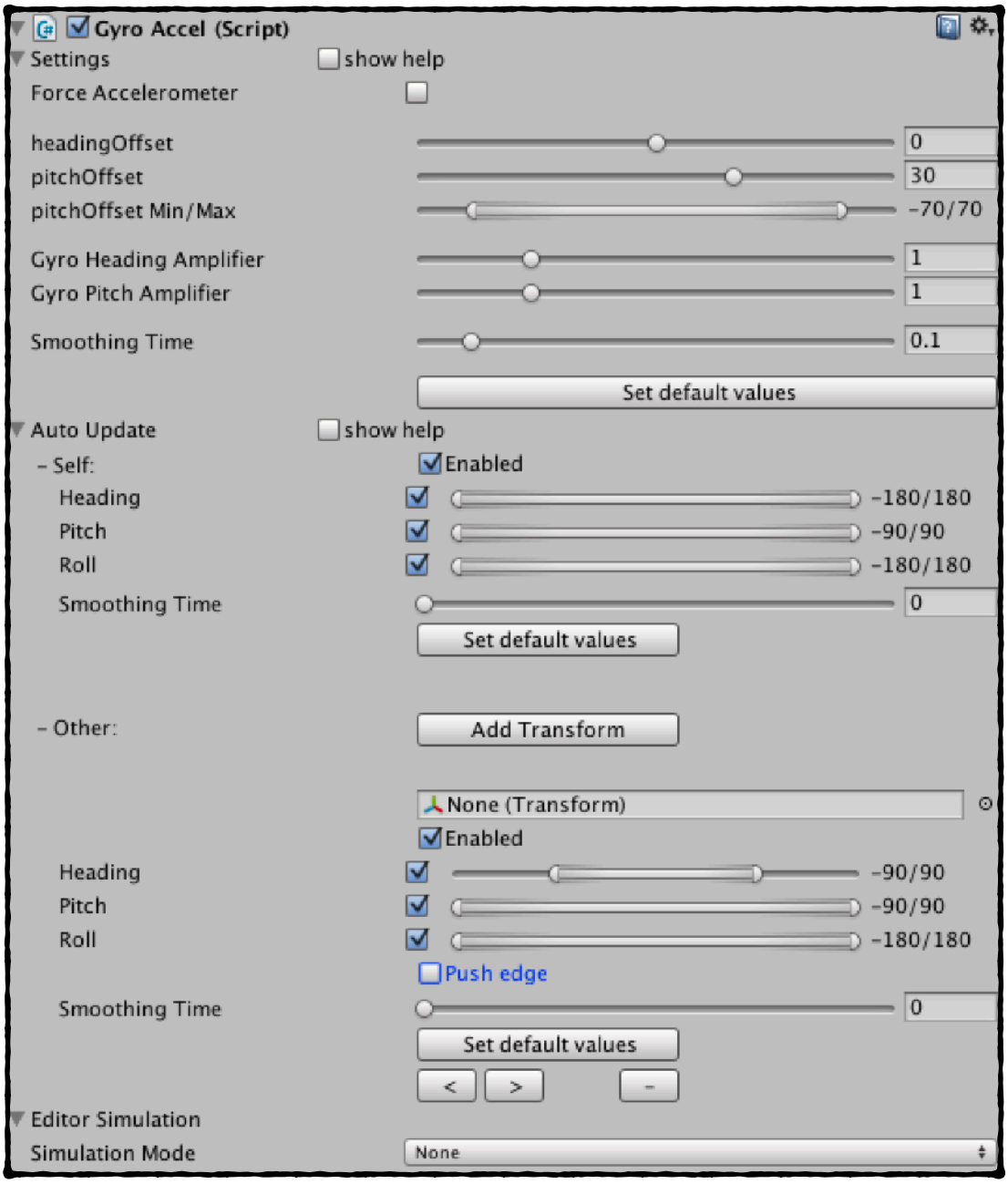
Quickstart

Put the “MSP_Input” prefab in the scene Hierarchy. This prefab contains Several scripts. Each script can be easily configured in Unity’s inspector.

- *GyroAccel.cs*: The main script of this asset, which processes all gyroscope and accelerometer sensor output.
- *VirtualTouchpad.cs*, *VirtualJoystick.cs* and *VirtualButton.cs*: These scripts allows the creation of the Virtual Joysticks, -Touchpads and -Buttons while working in the editor. During gameplay, it processes all user input from these Virtual Joysticks, -Touchpads and -Buttons.
- *GUIDraw.cs*: This script is responsible for drawing all the Virtual Joysticks, -Touchpads and -Buttons on screen.
- *ErrorHandling.cs*: This script handles all error messages.

Important: Make sure there is only *one* instance of each of these scripts active in the scene at all time.

GyroAccel.cs



▶*Settings*

show help	Enable this to get quick help about each setting
Force Accelerometer	By default the gyroscope is used. If no gyroscope is available, the accelerometer is used. Set this variable if you want to force the use of the accelerometer, even if there is a gyroscope available.
headingOffset	The heading offset to be used (North = 0, 90 = east, 180 = south, 270 = west, etc.)
pitchOffset	The pitch offset to be used (straight up = 90, level = 0, straight down = -90). Please note that this value can/will be changed during runtime, either by yourself or internally by the script, e.g. when the pitchAmplifier is being used.
pitchOffset Min/Max	The minimum/maximum value of the pitchOffset. For playability issues, don't use values near 90 degrees.
G*yro Heading Amplifier* G*yro Pitch Amplifier*	The heading and pitch multipliers for the gyro. Setting these values >1 will effectively speed up the rotation of your character in the gameworld. Using values < 1 will slow it down.
Smoothing Time	The (general) smoothing time to be used.

▶*Auto Update*

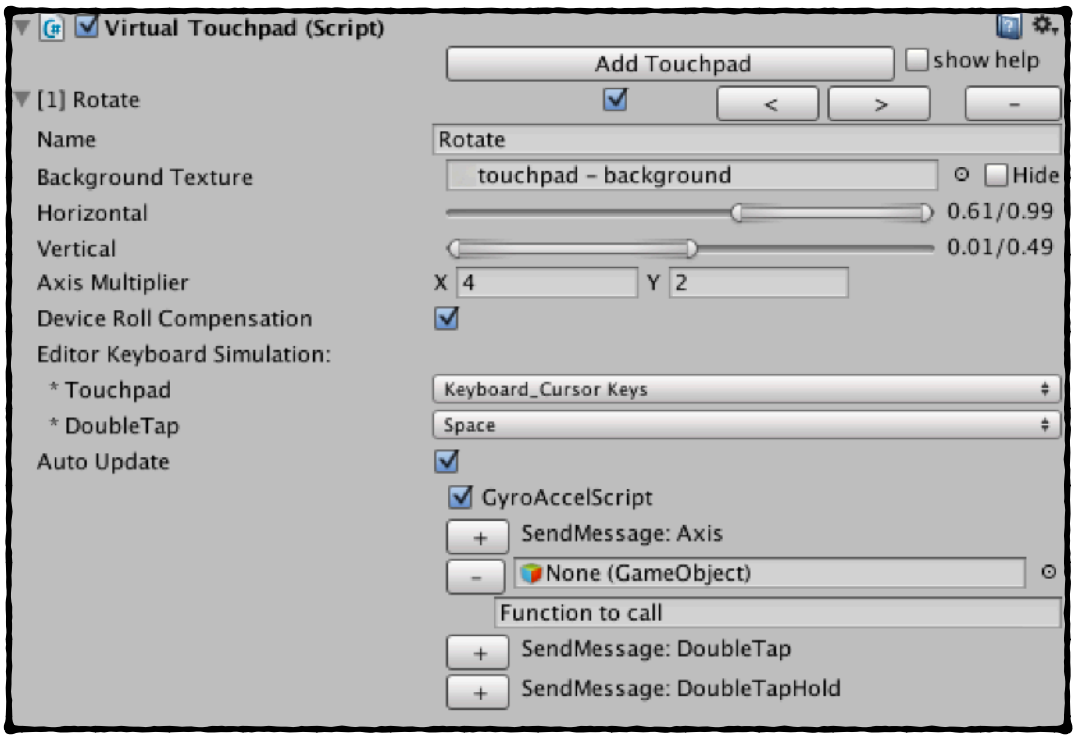
show help	Enable this to get quick help about each setting
Self	When enabled, the script will automatically update the rotation of the gameObject that has this script attached to it. Restrictions can be set for the minimum and maximum values of heading, pitch and roll. If you choose to partially update the rotation (e.g. only the heading, pitch and/or roll will be updated), you can choose a default value for the remaining axis. Also an (additional) smoothing time can be set; the smoothing time in the settings menu will always be applied.
Other	You can add other GameObjects to the AutoUpdateList. All updates will be performed in the order of appearance in this list. E.g.: first 'self' will be updated, then 'other1', 'other2', 'other3', etc. Use the sorting buttons to customize the order in which the rotations are applied. (Usually, the main camera will come last). Restrictions can be set for the minimum and maximum values of heading, pitch and roll, for each gameobject. If you choose to partially update the rotation (e.g. only the heading, pitch and/or roll will be updated), you can choose a default value for the remaining axis. Also an (additional) smoothing time can be set. When a heading or pitch moves beyond it's boundaries, you can choose to 'push the edge' along with it. (This will effectively recalculate all offset variables, and might thereby influence all other game objects being controlled by this asset.)

▶*Editor Simulation*

Simulation Mode	Select Mouse to use the mouse to simulate gyro movements

VirtualTouchpad.cs

Click the “Add Touchpad” button to create a new VirtualTouchpad. All touchpads can be deleted by clicking the “-“ button. The list of VirtualTouchpads can be sorted with the “<“ and “>” buttons.



Name	The name of the touchpad. Make sure you use the exact same name when referring to this touchpad when using it elsewhere (e.g. when using the MSP_Input.VirtualTouchpad.GetAxis() command)
Background Texture	The texture to be used for the touchpad’s background. You can choose to hide this texture during gameplay.

Note: All settings below are only configurable, when a background texture has been selected

Horizontal / Vertical	Use the sliders to control the position and size of the touchpad on the screen. Given values are in relative screen coordinates: 0 is left / bottom of the screen, while 1 is right / top of the screen.
Axis Multiplier	By default, the touchpad returns a Vector2 with values between -1 and 1. These values can be multiplied with an axisMultiplier. Tip: if you want to invert the touchpad’s movement, use negative values for the axisMultiplier
Device Roll Compensation	Should the output be compensated for the roll of your device. e.g.: when turning your device like a steering wheel, the input direction of the touchpad will follow accordingly.
Editor Keyboard Simulation	Select a key to simulate touch and/or doubletap.

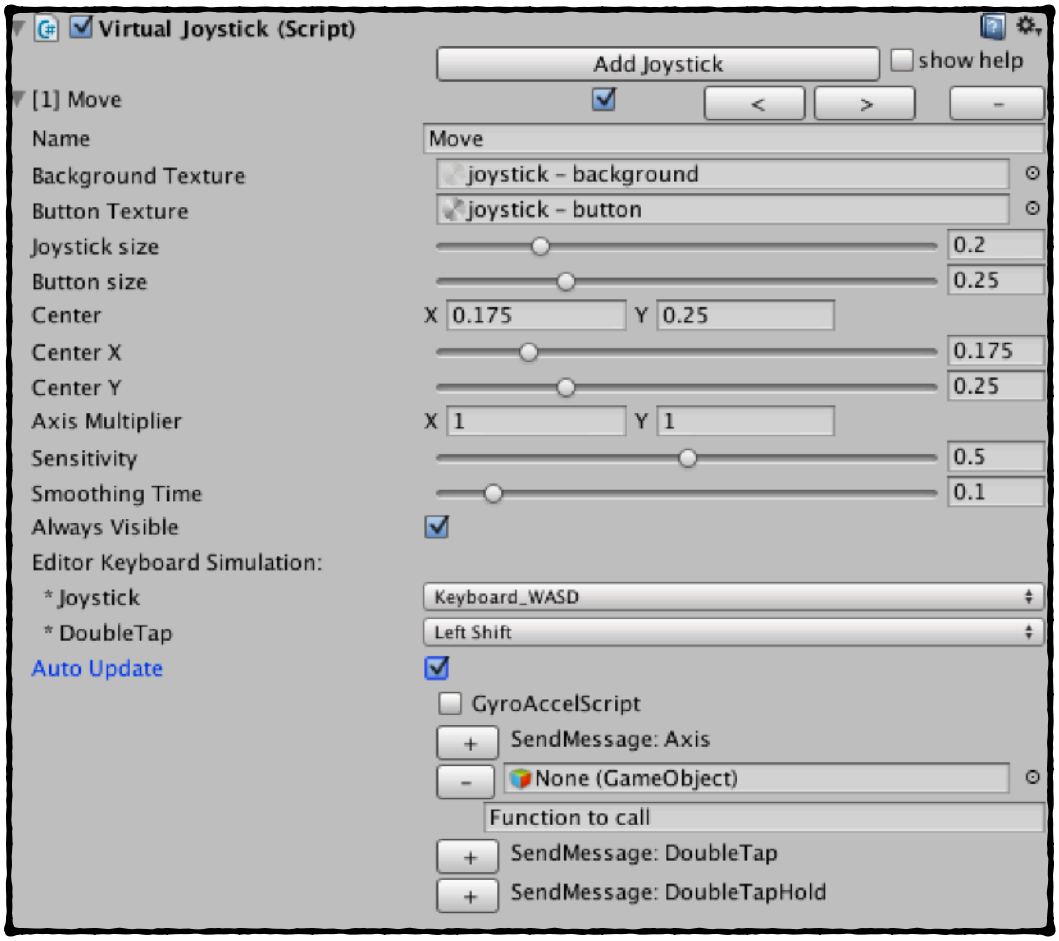
Auto Update

Integration with the GyroAccel script is a build in option, which can be (de)selected.
Other scripts can also be automatically informed of the touchpad's's axis values and/or when the touchpad has been double-tapped. This info is being send using a SendMessage:

- add the GameObject that should receive the message
- pass the name of the function that should be called on this GameObject.
  For sending the axis, this function must be void and excepting a Vector2 as input, e.g.:
  void SomeFunctionName(Vector2 axis)
  For sending the doubleTap or doubleTapHold status, this function must be void and excepting no input, e.g.
  void SomeOtherFunctionName()

VirtualJoystick.cs

Click the “Add Joystick” button to create a new VirtualJoystick. All joysticks can be deleted by clicking the “-“ button. The list of VirtualJoysticks can be sorted with the “<“ and “>” buttons.



Name	The name of the joystick. Make sure you use the exact same name when referring to this joystick when using it elsewhere (e.g. when using the MSP_Input.VirtualJoystick.GetAxis() command)
Background Texture *Button Texture*	The texture to be used for the joystick’s background and button. At least one texture should be provided

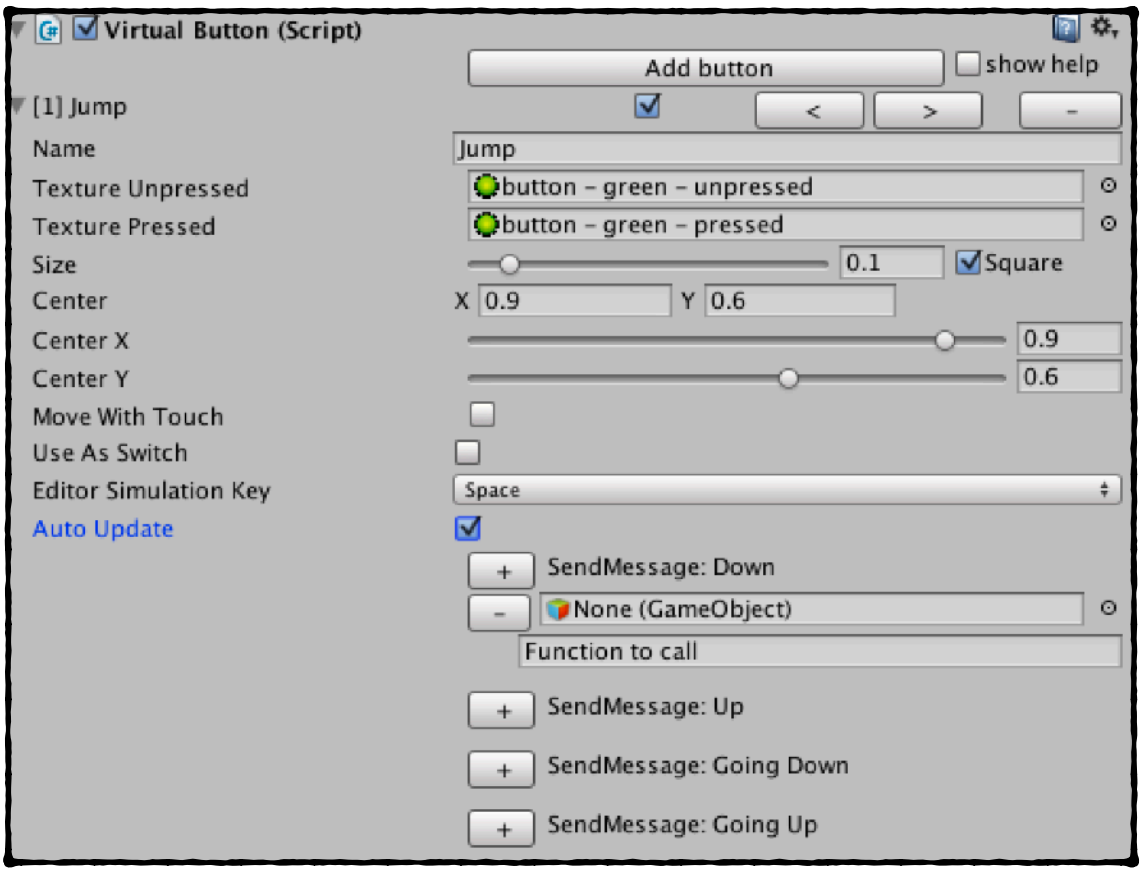
Note: All settings below are only configurable, when a background and/or button texture has been selected

Joystick Size	The size of the joystick, in *relative* screen coordinates. The joystick button cannot be pulled outside it’s background.
Button Size	The size of the joystick, relative to the size of the joystick(background)
Center *CenterX / CenterY*	The center of the joystick, in *relative* screen coordinates. Use the sliders CenterX and CenterY to easily change the position of the joystick on screen.

Axis Multiplier	By default, the joystick returns a Vector2 with values between -1 and 1. These values can be multiplied with an axisMultiplier. Tip: if you want to invert the joystick movement, use negative values for the axisMultiplier
Sensitivity	The sensitivity of the joystick; how much should the joystick respond when you're moving it just a little to the outside? Smaller values will make the joystick less sensitive while the button is near the middle.
Smoothing Time	The (maximum) time in which the joystick moves towards it target position.
Always Visible	Set this to true, if the joystick should be visible on screen all the time. Set this to false, if the joystick should only be visible while in use.
Editor Keyboard Simulation	Select a key to simulate joystick movement and/or doubletap.
Auto Update	Integration with the GyroAccel script is a build in option, which can be (de)selected. Other scripts can also be automatically informed of the touchpad's's axis values and/or when the touchpad has been double-tapped. This info is being send using a SendMessage: - add the GameObject that should receive the message - pass the name of the function that should be called on this GameObject. For sending the axis, this function must be void and excepting a Vector2 as input, e.g.: *void SomeFunctionName(Vector2 axis)* For sending the doubleTap or doubleTapHold status, this function must be void and excepting no input, e.g. *void SomeOtherFunctionName()*

VirtualButton.cs

Click the “Add Button” button to create a new VirtualButton. All buttons can be deleted by clicking the “-“ button. The list of VirtualButtons can be sorted with the “<“ and “>” buttons.



Name	The name of the virtualbutton. Make sure you use the exact same name when referring to this button when using it elsewhere (e.g. when using the MSP_Input.VirtualButton.GetAxis() command)
Texture Unpressed *Texture Pressed*	The texture to be used for the button; *TextureUnpressed* if the button is idle, or *TexturePressed* if the button is being pressed.

Note: All settings below are only configurable, when both TextureUnpressed and TexturePressed have been set.

Size	The size of the button, in *relative* screen coordinates. Select ‘square’ to create a square button. Otherwise, both the vertical and horizontal size can be independently set.
Center	The center of the button, in *relative* screen coordinates. Use the sliders CenterX and CenterY to easily change the position of the button on screen.
Move With Touch	Once pressed, should the button move along with the finger?
Use As Switch	If selected, the button will behave like a switch.
Editor Keyboard Simulation	Select a key to simulate the button.

Auto Update

Other scripts can be automatically informed when a button is (going) down or (going) up. This info is being send using a SendMessage:

- add the GameObject that should receive the message
- pass the name of the function that should be called on this GameObject.
  For sending the selected button status, this function must be void and excepting no input, e.g.
  void SomeFunctionName()

Advanced (additional coding with API)

Each script, once active in the scene, will continuously update it's parameters and variables. Each script contains various public static functions, which can be used to read/write these parameters and variables. All available commands of this API are summarized at the end of this document. First some short examples (c#):

Example 1 - Apply the gyro's rotation to a transform

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {

	public Transform someTransform;

	void Update ()
	{
		someTransform.rotation = MSP_Input.GyroAccel.GetRotation();
	}
}
```

Example 2 - Use the mouse to add an extra heading/pitch to the gyroscope's offset

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {

	void Update ()
	{
		MSP_Input.GyroAccel.AddFloatToHeadingOffset(Input.GetAxis("Mouse X"));
		MSP_Input.GyroAccel.AddFloatToPitchOffset(Input.GetAxis("Mouse Y"));
	}
}
```

Example 3 - Start a function when a VirtualButton with name "button1" is pressed

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {

	void Update ()
	{
		if (MSP_Input.VirtualButton.GetButtonDown("button1"))
		{
			DoSomething();
		}
	}

	void DoSomething()
	{
		// do stuff
	}
}
```

Example 4 - Read the axis of a VirtualJoystick with name “moveJoystick” and move a gameobject along it’s (x,z)-axis

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {

	public Transform transformToMove;

	void Update ()
	{
		Vector2 axis = MSP_Input.VirtualJoystick.GetAxis(“moveJoystick”);
		Vector3 move = new Vector3(axis.x,0f,axis.y);
		transformToMove.Translate(move * Time.deltaTime, Space.Self);
    }
}
```

Note: You can also put “using MSP_Input;” at the top of a script. Instead of writing something like “MSP_Input.GyroAccel.GetRotation()”, You can now simply use “GyroAccel.GetRotation()”.

PUBLIC STATIC FUNCTIONS FOR *MSP_Input.GyroAccel*:

Quaternion MSP_Input.GyroAccel.GetRotation()
returns the current rotation (Quaternion)

float MSP_Input.GyroAccel.GetHeading()
returns the current heading, clamped between -180 and 180 degrees (Forward = 0, Right = 90, Back = (-)180 and Left = -90)

right, down counterclockwise

float MSP_Input.GyroAccel.GetHeadingUnclamped()
returns the current unclamped heading

float MSP_Input.GyroAccel.GetPitch()
returns the current pitch, always between -90 (up) and 90 (down) degrees.

float MSP_Input.GyroAccel.GetRoll()
returns the current roll, clamped between -180 and 180 degrees (clockwise/counterclockwise)

float MSP_Input.GyroAccel.GetRollUnclamped()
returns the current unclamped roll

void MSP_Input.GyroAccel.GetHeadingPitchRoll(out float h, out float p, out float r)
gets all the values for heading, pitch and roll

void MSP_Input.GyroAccel.GetDevicePitchAndRollFromGravityVector(out float devicePitch, out float deviceRoll)
reads the device's pitch and roll from the gravity vector and passes them on to the variables *devicePitch* and *deviceRoll.*

void MSP_Input.GyroAccel.SetHeading(float newHeading)
set/force the current heading to the value of *newHeading.*

void MSP_Input.GyroAccel.SetPitch(float newPitch)
set/force the current pitch to the value of *newPitch.* Note: this might not always give the expected result; during the calculation of the pitch, the pitchOffset boundaries put a final restriction on the allowed values for the pitch.

void MSP_Input.GyroAccel.SetSmoothingTime(float smoothTime)
Sets the smoothing time. A (change in rotation) will be applied smoothly during this time.

float MSP_Input.GyroAccel.GetSmoothingTime()
Returns the current smoothing time

void MSP_Input.GyroAccel.AddFloatToHeadingOffset(float extraHeadingOffset)
Adds an extra value of *extraHeadingOffset* to the current heading offset

void MSP_Input.GyroAccel.SetHeadingOffset(float newHeadingOffset)
Sets the headingOffset to it's new value of *newHeadingOffset*

float MSP_Input.GyroAccel.GetHeadingOffset()
Returns the current value of the headingOffset

void MSP_Input.GyroAccel.AddFloatToPitchOffset(float extraPitchOffset)
Adds an extra value of *extraPitchOffset* to the current pitch offset

void MSP_Input.GyroAccel.SetPitchOffset(float newPitchOffset)
Sets the pitchOffset to it's new value of *newPitchOffset*

float MSP_Input.GyroAccel.GetPitchOffset()
returns the current value of the pitchOffset

void MSP_Input.GyroAccel.SetPitchOffsetMinumumMaximum(float newPitchOffsetMinimum, float newPitchOffsetMaximum)
Set the minimum and maximum value of the pitchOffset. For playability issues, don't use values near 90 degrees.

void MSP_Input.GyroAccel.SetGyroHeadingAmplifier(float newValue)

Set the gyroscope's heading amplifier to a value of *newValue.*
newValue < 1 —> decreases a change in the gyroscopes heading
newValue = 1 —> keeps the gyroscope's change of heading unaltered
newValue > 1 —> increases a change in the gyroscopes heading

float MSP_Input.GyroAccel.GetGyroHeadingAmplifier()

returns the current value of the gyroscope's heading amplifier.

void MSP_Input.GyroAccel.SetGyroPitchAmplifier(float newValue)

Set the gyroscope's pitch amplifier to a value of *newValue.*
newValue < 1 —> decreases a change in the gyroscopes pitch
newValue = 1 —> keeps the gyroscope's change of pitch unaltered
newValue > 1 —> increases a change in the gyroscopes pitch

float MSP_Input.GyroAccel.GetGyroPitchAmplifier()

returns the current value of the gyroscope's pitch amplifier.

void MSP_Input.GyroAccel.SetForceAccelerometer(bool newValue)

Set the forceAccelerometer parameter; By default the gyroscope is used. If no gyroscope is available, the accelerometer is used. Set this variable if you want to force the use of the accelerometer, even if there is a gyroscope available.

bool MSP_Input.GyroAccel.GetForceAccelerometer()

returns the current value of the forceAccelerometer parameter.

Quaternion MSP_Input.GyroAccel.GetQuaternionFromHeadingPitchRoll(float inputHeading, float inputPitch, float inputRoll)

returns a Quaternion, by first applying an *inputHeading*, then Applying an (local) *inputPitch* and then applying an (local) *inputRoll.*

void MSP_Input.GyroAccel.EnableAutoUpdate()
void MSP_Input.GyroAccel.EnableAutoUpdate(string name)

Enables the autoUpdate functionality for the GameObject with name *name*. If no name is specified, the AutoUpdate functionality on the GameObject with the script attached will be enabled

void MSP_Input.GyroAccel.DisableAutoUpdate()
void MSP_Input.GyroAccel.DisableAutoUpdate(string name)

Disables the autoUpdate functionality for the GameObject with name *name*. If no name is specified, the AutoUpdate functionality on the GameObject with the script attached will be disabled

PUBLIC STATIC FUNCTIONS FOR *MSP_Input.VirtualJoystick*:

Vector2 MSP_Input.VirtualJoystick.GetAxis(string name)
Returns a Vector2 with the current value of the VirtualJoystick with the name *name.* The axis is defined as a Vector2 with x,y-values between -1 and 1, indicating the direction and magnitude of the joystick's movement relative to it's center. These values are then multiplied with the according axisMultiplier's

void MSP_Input.VirtualJoystick.GetAngleAndMagnitude(string name, out float angle, out float magnitude)
Get the current status of the VirtualJoystick with the name *name:* The joysticks angle (-180 < angle < 180) is returned with the *angle* variable. Its magnitude (0 < magnitude < 1) is returned with the *magnitude* variable.

bool MSP_Input.VirtualJoystick.GetDoubleTap(string name)
Checks if the VirtualJoystick with the name *name* has been double tapped during the passed update-cycle.

bool MSP_Input.VirtualJoystick.GetDoubleTapHold(string name)
Checks if the VirtualJoystick with the name *name* has been double tapped during the passed update-cycle. The returned value remains true, until the VirtualJoystick has been released.

void MSP_Input.VirtualJoystick.Enable(string name)
Enables the VirtualJoystick with name *name*

void MSP_Input.VirtualJoystick.Disable(string name)
Disables the VirtualJoystick with name *name.* Disabled VirtualJoysticks are no longer visible on screen but can easily be (re)activated by using the Enable command

void MSP_Input.VirtualJoystick.SetCenter(string name, Vector2 newCenter)
Set the new position of the VirtualJoystick with name *name* to the value of *newCenter*. Please note that the given value must be in relative screen space (e.g. in the range 0 to 1).

void MSP_Input.VirtualJoystick.SetSize(string name, float newSize))
Set the new size of the VirtualJoystick with name *name* to the value of *newSize*. Please note that the given value must be in relative screen space (e.g. in the range 0 to 1).

PUBLIC STATIC FUNCTIONS FOR *MSP_Input.VirtualTouchpad*:

Vector2 MSP_Input.VirtualTouchpad.GetAxis(string name)
Returns a Vector2 with the current value of the axis of the VirtualTouchpad with the name *name.* The axis is defined as a Vector2 with x,y-values between -1 and 1, indicating the direction and magnitude of the touchpad movement relative to where the player has touched the touchpad first. These values are then multiplied with an axisMultiplier.

bool MSP_Input.VirtualTouchpad.GetDoubleTap(string name)
Checks if the VirtualTouchpad with the name *name* has been double tapped during the passed update-cycle.

bool MSP_Input.VirtualTouchpad.GetDoubleTapHold(string name)
Checks if the VirtualTouchpad with the name *name* has been double tapped during the passed update-cycle. The returned value remains true, until the associated TouchId is no longer within the VirtualTouchpad's area.

void MSP_Input.VirtualTouchpad.Enable(string name)
Enables the VirtualTouchpad with name *name*

void MSP_Input.VirtualTouchpad.Disable(string name)
Disables the VirtualTouchpad with name *name.* Disabled VirtualTouchpads are no longer visible on screen but can easily be (re)activated by using the Enable command

void MSP_Input.VirtualTouchpad.SetRect(string name, float newRect))
Set the new rect of the VirtualTouchpad with name *name* to the value of *newRect*. Please note that the given values of *newRect* must be in relative screen space (e.g. in the range 0 to 1).

PUBLIC STATIC FUNCTIONS FOR *MSP_Input.VirtualButton*:

bool MSP_Input.VirtualButton.GetButton(string name)
Returns *true* if the VirtualButton with name *name* is being pressed.

bool MSP_Input.VirtualButton.GetButtonDown(string name)
Returns *true* if the VirtualButton with name *name* has just been pressed. Once the button is down, this function will return *false*

bool MSP_Input.VirtualButton.GetButtonUp(string name)
Returns *true* if the VirtualButton with name *name* has been released.

void MSP_Input.VirtualButton.Enable(string name)
Enables the VirtualButton with name *name*

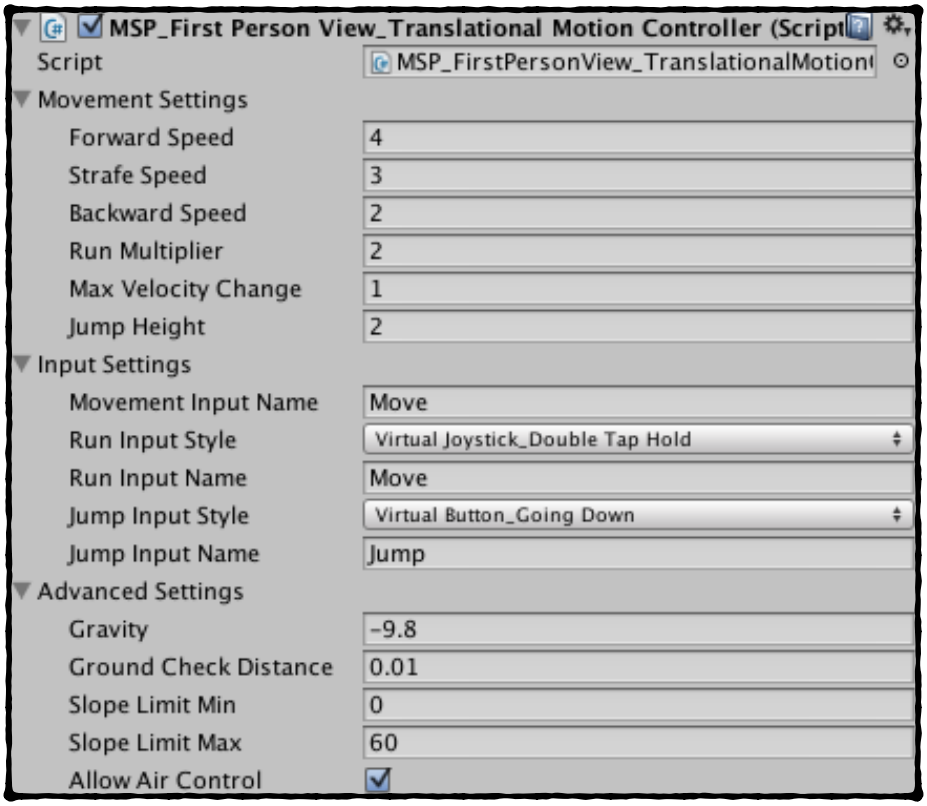
void MSP_Input.VirtualButton.Disable(string name)
Disables the VirtualButton with name *name.* Disabled VirtualButtons are no longer visible on screen but can easily be (re)activated by using the Enable command

void MSP_Input.VirtualButton.SetCenter(string name, Vector2 newCenter)
Set the new position of the VirtualButton with name *name* to the value of *newCenter*. Please note that the given value must be in relative screen space (e.g. in the range 0 to 1).

void MSP_Input.VirtualButton.SetSize(string name, Vector2 newSize))
Set the new size of the VirtualButton with name *name* to the value of *newSize*. Please note that the given value must be in relative screen space (e.g. in the range 0 to 1). For (forced) square buttons, the value of newCenter is ignored.

BONUS: *MSP_FirstPersonView_TranslationalMotionController.cs*

This script is used in demo scene 1a en 1b, and can be found in the GyroAccel_Input_Controller_v5/Demo's/Scripts/demo 1 folder.
It allows quick configuration of the *translational motion* of a simple, rigidbody based First Person View character, e.g. moving forward, left, right, jump. The *rotational motion* of the body and head can be set by directly using the autoUpdate functions of the GyroAccel.cs script.
The script assumes a RigidBody component and Capsule Collider component being used on the same GameObject as this script.



▶Movements Settings

Forward Speed	Maximum speed in forward direction
Strafe Speed	Maximum speed in left/right direction
Backward Speed	Maximum speed in backwards direction
Run Multiplier	While running, the forward/strafing/backward speed will be multiplied with this value.
Max Velocity Change	The maximum allowed change in velocity during one frame
Jump Height	The (maximum) reachable height during a unconstrained jump.

▶Input Settings

Movement Input Name	The name of the VirtualJoystick, controlling the movement of the character.
Run Input Style	Which method will be used to make the player character run: double tapping (and holding) a VirtualJoystick / VirtualTouchpad or tapping (and optionally holding) a VirtualButton?
Run Input Name	The name of the VirtualButton, VirtualJoystick or VirtualTouchpad that is used for a run request
Jump Input Style	Which method will be used to make the player character jump: double tapping (and optionally holding) a VirtualJoystick / VirtualTouchpad or tapping (and optionally holding) a VirtualButton?
Jump Input Name	The name of the VirtualButton, VirtualJoystick or VirtualTouchpad that is used for a jump request

▶Advanced Settings

Gravity	The name of the VirtualJoystick, controlling the movement of the character.
Ground Check Distance	The maximum distance under the capsule collider where the player character is still considered to be ‘on the ground’
Slope Limit Min / Slope Limit Max	The maximum allowed velocity will be reduced when the player character moves on a surface with a slope angle greater than *slopeLimitMin*. If the slope angle exceeds *slopeLimitMax*, movement of the player character will not be possible.
Allow Air Control	Allows the player character to be controlled while in the air