Create custom Game Controller with Accelerometer

Overview:

Engduino has a 3-axis xyz-accelerometer built in. This would allow us to apply trigonometry to calculate the angle between the axes and turn it into many interesting applications. One of them is a game controller based on accelerometer which we are going to demonstrate to you here.

Aim:

This tutorial aims to guide you through on how to create and customise your own version of game controller using the Accelerometer and Button on Engduino.

Objectives:

- Get Accelerometer reading to calculate the tilt angle
- Use external java library in MATLAB to map the input to a keyboard key

Learning Outcomes:

By the end of this tutorial, you should be able to

- Import external java library and use it in MATLAB
- Get data inputs from Accelerometer and Button
- Calculate the angle from xyz-axis of accelerometer
- Map inputs into keyboard keys or mouse on your computer.

Pre-requisite

- Java SE 7 and above installed
- Engduino MATLAB Support toolbox and MATLAB installed
- Engduino configured to make it discoverable in MATLAB

Getting Started

We assume that you already setup the connection for Engduino in MATLAB, if you haven't, please refer to the documentation provided in the support package.

This is a simple yet interesting application as you can physically map any keyboard keys or mouse control to the tilt angle calculated from the accelerometer readings. You can go on with your creativity to customise the keys to be mapped in your program and make a game controller of your own.

For starters, we have included a simple Space Shooter game that only has 5 controls which are up, down, left, right and fire. As such, we are going to guide you on how you can map these controls using Engduino.

Import Libraries

We need the following java library to simulate any keypress in the computer. First, start a new script and type in the following line.

```
import java.awt.Robot;
```

Initialise Variables

We initialise the java object "Robot" to a variable.

```
% Declare the java object
robot = Robot;
% Set reading frequency [Hz] - readings per second.
frequency = 100 ;
% Set the left right steering sensitivity
LRsensitivity = 30;
% Set the up down steering sensitivity
UpDownSensitivity = 10;
```

The **frequency** set how fast the program run in hertz. The higher the frequency the faster the program scans for the changes in sensors and the faster it responds to the tilt. However, setting the frequency too high or too low would result in negative impact on the responsiveness of the program. We set it at 100 as an optimum solution.

The **LRsensitivity** and **UpDownSensitivity** set how sensitive your program should response to a tilt angle in degree. The lower the number, the higher its sensitivity. You may adjust these variables to suits your playing style.

Connect to Engduino

The following lines check if the object 'e' is available in MATLAB workspace. If it does not exist, it calls the function "engduino()" which will connect the Engduino hardware and store it as an object.

```
if (~exist('e', 'var'))
    e = engduino();
end
```

Initialise Game Controller' holding Position

Many of us hold the game controller in a different tilt angle. For example, when you are playing a game on your mobile phone, some people would like to hold the phone in a more upright positon whereas some prefer to hold it more flat. As such, we need to offset this initial tilt angle to the angle calculated from the accelerometer readings.

```
% initialise starting accelerometer position
newReading = e.getAccelerometer();
gx = newReading(1);
gy = newReading(2);
gz = newReading(3);
% set the initial tilt position of the accelerometer
thetaUD_init = atand(gx/gz);
```

This code will take the initial reading from the accelerometer. The getAccelerometer() function will return the accelerometer xyz-axis reading in a 1x3 matrix. We then calculate the initial up/down tilt angle and store it in a variable which we will use it later for offset.

Main Program Loop

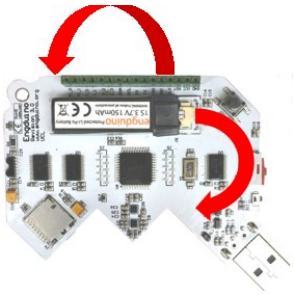
After all the initialisation, we will now create the main loop in our program to keep it running, reading in accelerometer data, calculate the tilt angle and simulate the keypress. For simplicity, we will create an infinite while loop as we want our game controller to keep working until we press ctrl+c in MATLAB to terminate the program.

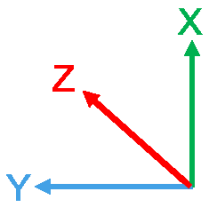
```
while (1)
```

This code will keep the code in the body of the while loop running infinitely as its condition statement is fixed. First, we need to keep getting inputs from the accelerometer.

```
% Read acceleration vector from Engduino's accelerometer sensor.
newReading = e.getAccelerometer();
gx = newReading(1);
gy = newReading(2);
gz = newReading(3);
```

Next, we will calculate the UP/DOWN, LEFT/RIGHT tilt angle. If we lay the Engduino flat with LED facing down, then the z-axis will point downwards.





$$Angle = tan^{-1}\frac{Y}{X}$$

```
% calculate the angle of the resultant acceleration Left Right
thetaLR = atand(gy/gx);

% calculate the angle of the resultant acceleration Up Down
thetaUD = atand(gx/gz);
```

Then we apply our initial tilt angle offset to the UP/DOWN axis.

```
% offset the up/down tilt axis
upDownAxis = thetaUD - thetaUD_init;
```

After that, we will work out the condition to map the tilt angle to a keyboard key on the computer and simulate the keypress. We have assigned our java object to the variable “robot”, now we can just use it to call its function to simulate the keypress. First, we map the LEFT/RIGHT axis to our keyboard.

```
if(thetaLR<-LRsensitivity&&thetaUD<0)
    % Move left
    robot.keyPress(java.awt.event.KeyEvent.VK_LEFT);
elseif(thetaLR>LRsensitivity&&thetaUD<0)
    % Move  right
    robot.keyPress(java.awt.event.KeyEvent.VK_RIGHT);
elseif(thetaLR<-LRsensitivity&&thetaUD>=0)
    % inverse the control when up/down tilt angle >=0
    % Move right
    robot.keyPress(java.awt.event.KeyEvent.VK_RIGHT);
elseif(thetaLR>LRsensitivity&&thetaUD>=0)
    % Move left
    robot.keyPress(java.awt.event.KeyEvent.VK_LEFT);
else
    % release the key
    robot.keyRelease(java.awt.event.KeyEvent.VK_LEFT);
    robot.keyRelease(java.awt.event.KeyEvent.VK_RIGHT);
end
```

The last condition of the if-else statement is to simulate the key release when the tilt angle does not fall into any of the condition above.

Similarly, we map the UP/DOWN axis to the keys on the keyboard

```
if(upDownAxis<-UpDownSensitivity)
    % Move down
    robot.keyPress(java.awt.event.KeyEvent.VK_DOWN);
elseif (upDownAxis>UpDownSensitivity)
    % Move up
    robot.keyPress(java.awt.event.KeyEvent.VK_UP);
else
    robot.keyRelease(java.awt.event.KeyEvent.VK_UP);
    robot.keyRelease(java.awt.event.KeyEvent.VK_DOWN);
end
```

Next, we will map the button on the Engduino to the SPACE key.

```
% Map the button on Engduino to a Key
if (e.getButton())
% key to fire
    robot.keyPress(java.awt.event.KeyEvent.VK_SPACE);
elseif (not(e.getButton()))
    robot.keyRelease(java.awt.event.KeyEvent.VK_SPACE );
end
```

Optionally, we can print out the calculated tilt angle from accelerometer on the screen for the purpose of fine tuning the game controller.

```
% display the tilt angle calculated
title(['LeftRight tilt angle: ' num2str(thetaLR, '%.0f'), ...
    ' UpDown tilt angle: ' num2str(thetaUD, '%.0f')]);
```

We now set a delay in the loop to set the frequency of the running program.

```
pause(1/frequency);
```

That is all we need in the main while loop.

```
end
```

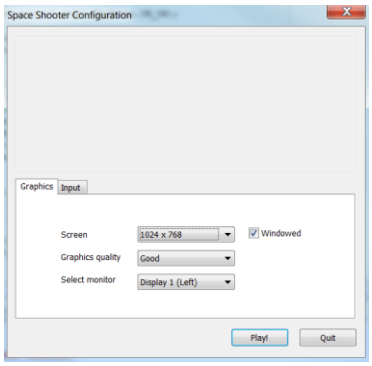
Warning! We have not set the parameter in our program on when do we want our game controller to start or stop working. So, the moment you start running the program, it is going to map the tilt angle to a keyboard key. Do not be afraid if you experience some weird behaviour from your computer, it is simply just receiving a keypress caused by our program.

As an extension of this tutorial, you may modify the program to implement a start/stop condition for the while loop.

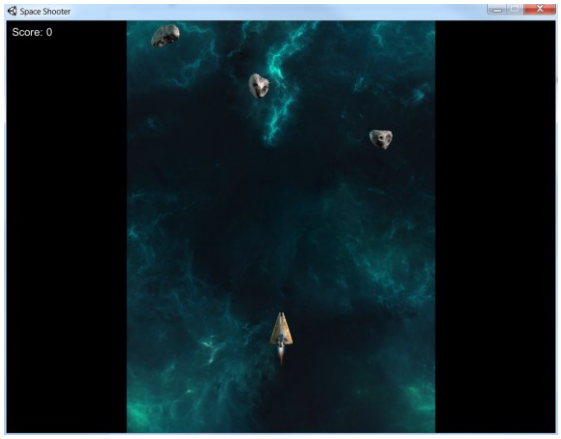
You may now test out the program that you have just created. Launch the included game "SpaceShooterKeyboard.exe" which is designed for the purpose to demonstrate this tutorial. Connect your Engduino and run the code.

Hold the Engduino with the LEDs facing down as shown in the image below.

When you launch the game, tick the windowed mode under the configuration so that the game will not go into full screen.



Start playing the space shooter game. You will not lose your life in this game. This will allow you to have infinite amount of time to test out your controller.



Extension of the project.

You have just completed the tutorial and made Engduino a game controller. There are few things you can do to further improve the program.

- Modify the program to start/stop the game controller on a condition
- Customise the game controller to make use of other sensors on the Engduino as inputs
- If you have a Bluetooth module on the Engduino, you can turn it into a Bluetooth game controller by just changing one line of code.