Using Engduino as a measurement tool to measure distance.

Overview:

In this tutorial, we are going to demonstrate how you can apply the physics you learnt in the class to solve engineering problems and develop interesting application using Engduino. This example will use the accelerometer on Engduino to measure acceleration, then apply double integration to get the distance.

Aim:

This tutorial aims to provide you the step-by-step guide on how to create the application that measure distance using the accelerometer readings from Engduino.

Objectives:

- Get accelerometer reading, apply filter to reduce noise and calculate the acceleration
- Integrate the acceleration into velocity, then to distance

Learning Outcomes:

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

- Get xyz accelerometer reading from Engduino
- Learn how to use high pass and low pass filter to filter out noise from sensor
- Apply integration in MATLAB®

Pre-requisite

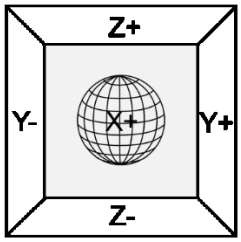
- Engduino MATLAB Support toolbox and MATLAB installed
- Engduino configured to make it discoverable in MATLAB

Tutorial

In physics, you have learnt that integrate acceleration over time will produce velocity, and integrate velocity over time again gives you the displacement. Here, we are going to use the 3-axis accelerometer on the Engduino to get the acceleration and apply this rule to calculate the displacement.

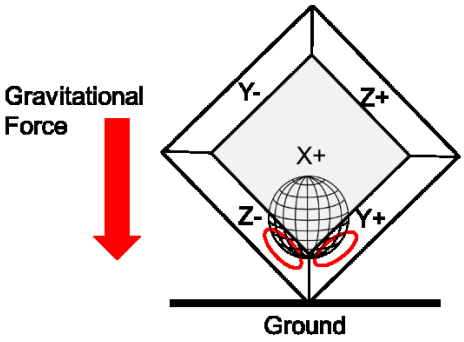
1. How does an accelerometer works?

First, we need to understand how an accelerometer measures the acceleration. Let's visualise the accelerometer as a box that contains a ball in it.

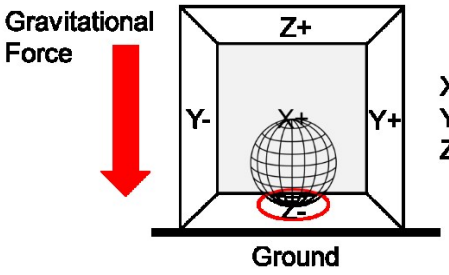


X=0g
Y=0g
Z=0g

If there is no gravitational force or any other fields that might affect the position of the ball, the ball will simply float in the middle of the box. In this case, without touching any sides of the box, the accelerometer will measures 0g on all three axis.

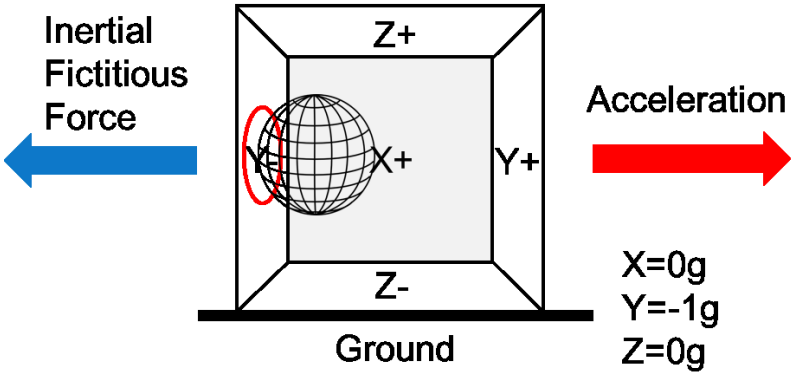


X=0g
Y=0.71g
Z=-0.71g



X=0g
Y=0g
Z=-1g

If we take this model and put it on earth, the gravitational force will act on the ball and cause it to touch the sides of the box. Imagine that the sides of the box are pressure sensitive, the accelerometer will detects this pressure force and gives the reading as shown in the above model. We can compare the readings to get the orientation.



X=0g
Y=-1g
Z=0g

In the model above, we move the accelerometer to the right, the ball will move in the opposite direction of the acceleration due to inertial fictitious force. The accelerometer will gives the reading of this force which is directed at the opposite direction of the acceleration. However, there are many other forces on the earth that can affect the force acting on the ball, such as the electromagnetic force if the ball is made up of steel. We call these other forces as noise which could interfere our readings.

2. Work out the xyz axis of the accelerometer

Now, try out the accelerometer and work out the xyz-axis by rotating the Engduino. Plug in the Engduino to the computer, launch MATLAB and create a new script. Use the code below to connect to Engduino.

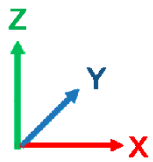
```
if (~exist('e', 'var'))
    e = engduino();
end
```

This code will make the connection to Engduino. Ensure that MATLAB is able to connect to Engduino by checking at the message in the command window before you continue.

Once you have connected the Engduino, stop the script and add the following code below the existing code.

```
while(1)
    newReading = e.getAccelerometer()
    pause(0.5);
end
```

This code will read the accelerometer and returns the xyz axis readings respectively.



Work out the axis, you will get approximately either -1g or 1g on the z-axis if you lay the Engduino flat on a table. You will notice that the readings fluctuate and can be quite noisy. In order to remove some noise, we would have to apply some filtering to the readings. This will be discussed in the next section.

3. The steps to generate a program that measure distance

Let us look at the steps needed to create our program that measures distance using the accelerometer before we dive into the actual code.

In the program, we will get the accelerometer reading at approximately an interval of 0.01s. This interval will be our change in time dt. We will convert the accelerometer reading to acceleration (1g = 9.81m/s^2). Integrate the acceleration over dt to get the change in velocity v(t). Then add this change in velocity to the initial velocity, which is the velocity at the previous time step and integrate it again over dt to get the displacement. We total up the displacement at each time step to get the total displacement. These steps will be put in a loop, "a loop is where the program keep repeating doing the task", until we want to stop our measurement.

In Psuedo code,

```
Initialise the variables

Initialise the accelerometer

While not end of measurement

        Get accelerometer reading

        Convert accelerometer reading to acceleration

        Record the current time

        dt = current time – previous time

        v(t) = Integrate acceleration over dt

        current velocity = previous velocity + v(t)

        displacement = Integrate current velocity over dt
```

```
total up the displacement
```

4. Code the program

Let us start coding the actual program. Create a new script on MALTAB. The first thing we do is initialise the variables needed. Variables are seen as a temporary memory space in the computer to store values that we will be using in our program.

We will set the frequency of our program as 100Hz, this will define how fast the program run at its time interval. T = 1/frequency.

```
% Set reading frequency [Hz] - readings per second.
frequency = 100;
```

Set a constant multiplier to convert the accelerometer readings to acceleration in ms^{-2}. You may adjust this value to give a better result. 1g = 9.81m/s^2.

```
% Set multiplier to convert accelerometer data to acceleration m/s^2
multiplier = 9.81;
```

The following is a set of variables, which are used to store temporary values for filtering the noise from the sensor. Apart from filtering the noise from the sensors, we also apply the filter to our calculation after each integration. The reason is simply because even a small value of noise, when it gets integrated each time, the noise will become greater too.

```
% Set threshold to ignore small noise in acceleration
acc_threshold = 0.2;

% initialise filtered value to 0
xAcc_Filtered = 0;
acceleration_Filtered=0;
velocity_Filtered =0;

% coefficient to apply filtering, adjust these to give better result
alpha = 0.15;
beta = 0.95;
gamma = 0.9;
```

Next, initialise the variables needed for the calculation.

```
% Initialise variables for calculation
current_time = now;
t0 = now;
previous_time =0;
previous_velocity =0;
current_velocity = 0;
total_displacement = 0;
```

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
```

We do not want to start the measurement right away when we run the program. Use the Engduino's push button as start/stop button.

```
% Wait to start calculation
while(not(e.getButton()))
    pause(0.1);
end
```

We create a while-loop that wait for the push button to be pressed. A while loop will keep repeating the code in the body which in this case, it delays the program for 0.1s and does nothing. The reason for the delay is to ensure that the program do not register more than one time when the button was pressed. We will use only the x-axis of the accelerometer to detect the acceleration in the x-direction.

Use the code below to measure the initial value of the accelerometer to act as an offset for the readings

```
%% Initialise accelerometer reading
for i=1:5
    newReading = e.getAccelerometer();
    gx = newReading(1);
    % apply high pass filter to the accelerometer output
    xAcc_Filtered = gx - ((1-alpha)*xAcc_Filtered + alpha*gx);
end

% accelerometer data is multiplied to get 1g=10m/s^2
init_accx = (floor(xAcc_Filtered*100)*multiplier/100);
```

We have initialised the variables needed. Time to write our main program. Create a while-loop to keep the program running until a button is pressed.

```
while (not(e.getButton()))
```

Get the accelerometer reading, apply the high pass filter to the accelerometer input to get a more accurate acceleration reading, then convert it into acceleration in ms^{-2}. Record the current time.

```
% Record the current time
current_time = (now - t0)*10e4;
newReading = e.getAccelerometer();
gx = newReading(1);
% apply high pass filter to the accelerometer output
xAcc_Filtered = gx - ((1-alpha)*xAcc_Filtered + alpha*gx);
% convert to acceleration from accelerometer
acceleration = (floor(xAcc_Filtered*100)*multiplier/100 - init_accx);
```

After we have calculated the acceleration, we apply a low pass filter to remove noise. The if statement set a small cut-off threshold to ignore small noises.

```
accFilt = (1-beta)*accFilt + beta*acceleration;
% ignore small value acceleration due to noise
if(accFilt>-acc_threshold&&accFilt<acc_threshold)
    accFilt = 0;
end
```

Apply the integration to the acceleration to get the velocity, apply the filter to improve the result, then integrate the velocity into displacement. The "`int(x, previous_time, current_time)`"

is a MATLAB function to perform integration. The first parameter is the equation we want to integrate, 'x' is the acceleration we have just calculated. The second and last parameter is the initial and final value for the integration which is the change in time.

```
x=sym(acceleration_Filtered);

% Calculate velocity
current_velocity = previous_velocity + int(x, previous_time,
current_time);
% low pass filter to filter out noise from calculated velocity
velocity_Filtered = (1-gamma)*velocity_Filtered + gamma*current_velocity;
% Integrate velocity to displacement
displacement = int(current_velocity, previous_time, current_time);
total_displacement = total_displacement + displacement;
previous_velocity = velocity_Filtered;
previous_time = current_time;
```

Add the following line to print the total displacement in the command window.

```
disp(total_displacement);
```

Lastly, set the delay in the loop.

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

That is all we need in the main while loop. We close the while loop with an "end".

```
end
```

This completes our program. You may connect the Engduino to the computer and test run this program that you have just created.

Click run on MATLAB to run the program, hold the Engduino to the start position of where you want to measure, and press the push button to start measure. Move the Engduino in one direction until you reached the place you want to stop measuring and press the push button at the same time. This should give you an estimated total distance moved.

However, if you do not press the push button when you stop, you will notice that the distance keep increasing. The calculations above assume that we are only getting the acceleration in the direction we are moving. But in reality, the accelerometer will take into account of acceleration due to falling, shaking or rolling. These forces leads to additional acceleration and decelerations and the sensor cannot distinguish these forces.

Using accelerometer alone is not a good way to measure distance. We will need other sensors such as a gyroscope in order to adjust these forces to get a more accurate result.

Since accelerometer alone cannot provide accurate data for acceleration, can you think of better ways to measure distance or even height with accelerometer?

Hint: Forget about physics, apply the rules you have learnt in Maths!

5. Visualise the data (Additional)

You can visualise the data and calculation by plotting the graph in MATLAB. The next step will create two graphs, first graph plots the accelerometer data and the second graph plots the velocity. Use

back the existing code. Copy the code below and put it above the first while loop. This will create the graphs.

```
%% For Graph plotting
buffSize = 10;
accelerometer_circBuff = nan;
velocity_circBuff = nan;
time = now;
i=1;

figure;
% graph1
graph(1) = subplot(1,2,1);
plotHandle1 =
plot(graph(1),time,accelerometer_circBuff,'Marker','o','MarkerSize',5,'Li
neWidth',2);
xlabel('Time[s]');
ylabel('Gravitational Force (g)');
title(['Acceleration: ' char(vpa(acceleration_Filtered,3)) 'm/s^2']);
limits = 1.0;
ylim([-limits limits])
axis square;
grid on

% graph2
graph(2) = subplot(1,2,2);
plotHandle2 =
plot(graph(2),time,velocity_circBuff,'Marker','o','MarkerSize',5,'LineWid
th',2);
xlabel('Time[s]');
ylabel('Velocity (m/s)');
limits = 1.0;
ylim([-limits limits]);
title(['Displacement: ' char(vpa(total_displacement,3)) 'm']);
axis square;
grid on
```

Copy the following code into the bottom of main while loop, just above the pause statement.

```
if i < buffSize
    % Add the newest sample into the buffer.
    accelerometer_circBuff(i) = gx;
    velocity_circBuff(i) = velocity_Filtered;
    time(i) = (now-t0)*10e4;
else
    % If we have enough samples then remove oldest sample and add the
    % newest one into the buffer.
    accelerometer_circBuff = [accelerometer_circBuff(2:end), gx];
    velocity_circBuff= [velocity_circBuff(2:end), velocity_Filtered];
    time = [time(2:end), (now - t0)*10e4];
end
% subplot raw X acceleration vector
subplot(graph(1));
limits = 1.0;
xlim([min(time) max(time)+10e-9]);
ylim([-limits limits]);
title(['Acceleration: ' char(vpa(acceleration_Filtered,3)) 'm/s^2']);
set(plotHandle1,'YData',accelerometer_circBuff,'XData',time);

% subplot the velocity
subplot(graph(2));
limits = 1.0;
xlim([min(time) max(time)+10e-9]);
ylim([-limits limits]);
title(['Displacement: ' char(vpa(floor(total_displacement*10)/10,2))
'm']);
set(plotHandle2,'YData',velocity_circBuff,'XData',time);
i = i+1;
```