|  |  |
| --- | --- |
|  | **engduino®**Tutorial  Engduino Support Team - support@engduino.org |

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. However, through this tutorial you will soon realise that when measuring the environment with sensors, they tense to produce a lot of noise. Thus, this application will not provide great accuracy to the measurement. In theory it works, but in practice, we would need additional sensors such as a gyroscope together with the accelerometer to produce better accuracy in the measurement.

# Aim:

This tutorial aims to provide you the step-by-step guide on how to create the application to 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 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

# Getting Started

First, ensure that your Engduino is already configured to work with MATLAB. Connect the Engduino to the computer, create a new script in MATLAB and try to run the code below.

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.

# Initialise Variables

Variables are seen as a 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 should run and sample the sensors value.

% Set reading frequency [Hz] - readings per second.

frequency = 100;

Next, we will need to set a constant multiplier to convert the accelerometer readings to acceleration in . You may adjust this value to give a better result.

% Set multiplier to convert accelerometer data to acceleration m/s^2

multiplier = 26;

The following is a set of variables 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 that was produced, when it gets integrated each time, the result would be significant enough to cause inaccuracy in our measurement. As such, we will apply a high pass filter here to filter the calculation after the integration and also ignore changes in small value.

Now, we will initialise the variables to perform our calculation.

% Set threshold to ignore small noise in acceleration and velocity

acc\_threshold = 0.5

vel\_threshold = 0.3;

% filter

gxFilt = 0;

accFilt =0;

velFilt =0;

% filter control coefficient

alpha = 0.5;

beta = 0.9;

gamma = 0.9;

We need a way to initialise our accelerometer reading so that it can be used as a reference point to measure the changes in the accelerometer reading to give us the acceleration. Before we can initialise this reference point, we want to make sure that you get the Engduino into the position of where you want to start your measurement. As such, we use the push button on the Engduino as an input which when pressed for the first time will start the measurement and pressed the second time to end the measurement.

% 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 is pressed. Now, we can initialise our accelerometer starting values.

%% Initialise accelerometer reading

for i=1:10

newReading = e.getAccelerometer();

gx = newReading(1);

% high pass filter accelerometer output

gxFilt\_filt = gx - ((1-alpha)\*gxFilt + alpha\*gx);

end

% accelerometer data is multiplied to get 1g=10m/s^2

init\_accx = (floor(gxFilt\*100)\*multiplier/100);

# Main Program Loop

We have initialised the variables needed. Now, we will create the main loop for our program to keep running. The program will keep reading the inputs from the accelerometer, convert it to acceleration in , integrate it into velocity, then integrate the second time into distance. Create the main while loop.

while (not(e.getButton()))

Get the accelerometer reading, apply the high pass filter to the accelerometer input, then convert it into acceleration in .

% Read acceleration vector from Engduino's accelerometer sensor.

newReading = e.getAccelerometer();

% high pass filter accelerometer output

gxFilt = gx - ((1-alpha)\*gxFilt + alpha\*gx);

acceleration = (floor(gxFilt\*100)\*multiplier/100 - init\_accx);

After we have calculated the acceleration, we apply the filter again. The if statement ignore the small acceleration value due to noise and make it into 0 to avoid the small noise being summed up.

accFilt = (1-beta)\*accFilt + beta\*acceleration;

% ignore small value acceleration due to noise

if(accFilt>-acc\_threshold&&accFilt<acc\_threshold)

accFilt = 0;

end

We 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, the second and last parameter is the initial and final value for our integration.

x=sym(accFilt);

% Integration from acceleration to velocity to displacement

current\_velocity = previous\_velocity + int(x, previous\_time, current\_time);

% low pass filter to filter out noise from calculated velocity

velFilt = (1-gamma)\*velFilt + gamma\*current\_velocity;

% double integration from accelerometer to displacement

displacement = int(previous\_velocity + int(x, previous\_time, current\_time), previous\_time, current\_time);

total\_displacement = total\_displacement + displacement;

previous\_velocity = velFilt;

previous\_time = current\_time;

We 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. 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. Start walking in constant speed 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 distance travelled. However, if you do not press the push button when you stop, you will notice that the distance keep increasing. This is due to the noise in accelerometer keep adding up the value.

Using accelerometer alone is not a good way to measure distance. We will need other sensors such as a gyro which will measure the inertial of the accelerometer to compensate the noise created by accelerometer to achieve higher accuracy. In the real world, we use an encoder to detect speed and velocity rather than accelerometer.

This tutorial only serves as an exercise to show you that we can apply the physics we learnt and make it into an application.

# Extension of the project

* Try adding a gyro to the Engduino to produce a better result.