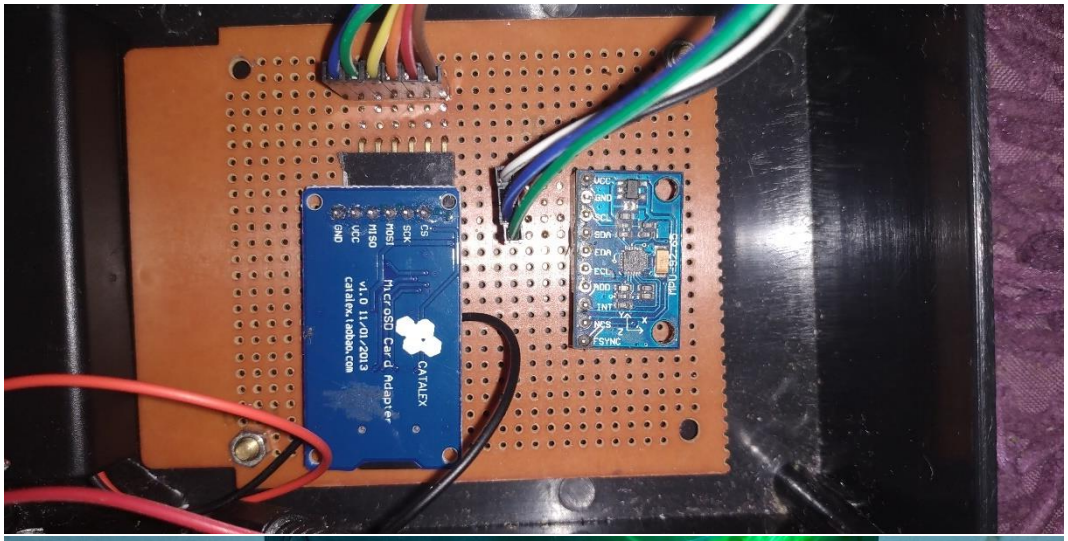
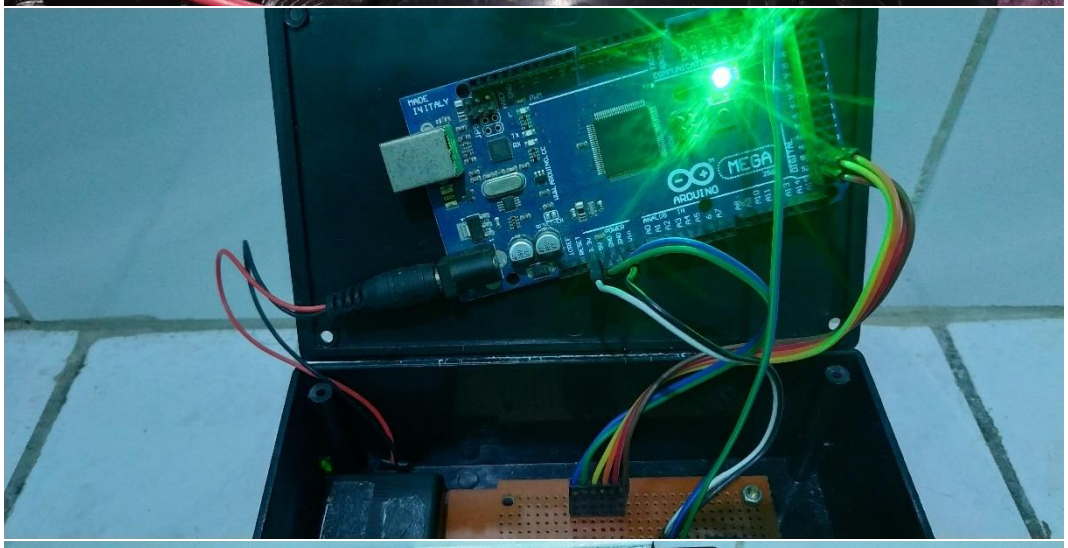
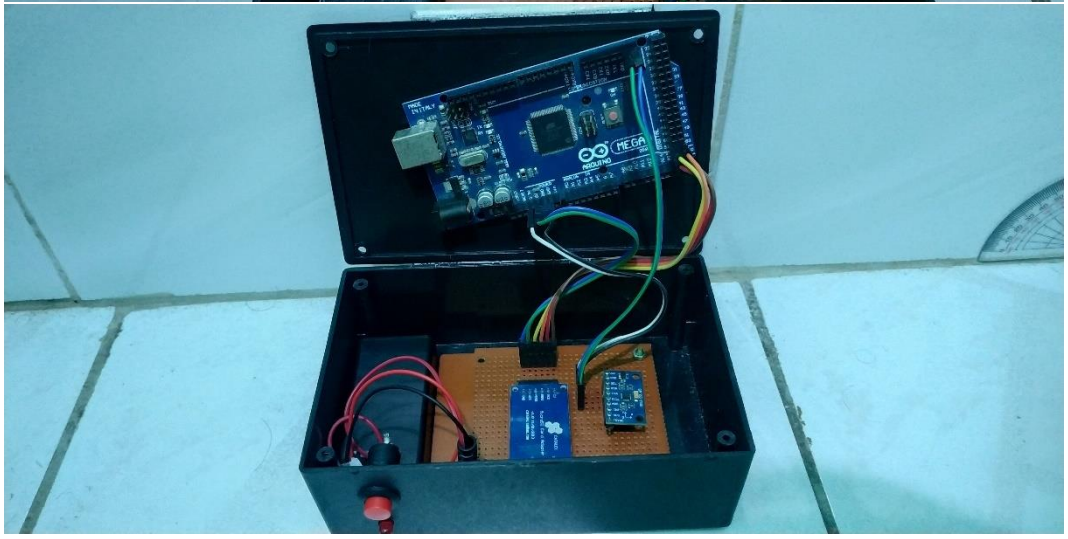
LAMPIRAN

LAMPIRAN A Foto Alat







LAMPIRAN B Perbandingan PSoC dan Arduino

	Arduino Mega 2560	**PSoC CY8CKIT-044**
Microcontroller	Atmega2560	Arm Cortex M0
Operating Voltage	5V	3,3V & 5V
Digital I/O Pins	54	55
Analog Input Pins	16	6
I2C	✓	✓
SPI	✓	✓
UART	✓	✓
Flash Memory	256 KB (8 KB for bootloader)	128 KB
SRAM	8 Kb	16 KB
FRAM	-	1 Mb
EEPROM	4 Kb	-
ROM	-	8 KB
Clock Speed	16 MHz	32 KHz

LAMPIRAN C *Listing Program* Arduino

Sumber listing program arduino :

[1] Remington, S.J. *Jremington/MPU-9250-AHRS*. 2020. Tersedia dari: https://github.com/jremington/MPU-9250-AHRS. [URL dikunjungi pada 13 Januari 2022]

[2] Winer, K. *kriswiner/MPU9250*. 2017 Tersedia dari: https://github.com/kriswiner/MPU9250. [URL dikunjungi pada 04 Oktober 2021]

[3] Tai, H. *hideaiktai/MPU9250*. 2020. Tersedia dari: https://github.com/hideakitai/MPU9250. [URL dikunjungi pada 17 Desember 2022]

```
#include "Wire.h"
#include "/libs/I2Cdev.cpp"
#include " libs/MPU9250.cpp"
#include "SD.h"
#include "SPI.h"

MPU9250 mpu;
I2Cdev   I2C_M;
File sdCard;

int CS_pin = 53;

// offsets and correction matrix for accel and mag
float A_B[3]
{ 1529.76,  467.31,  -25.25};

float A_Ainv[3][3]
{ {  0.59100, -0.04303,  0.02990},
  { -0.04303,  0.63281, -0.00879},
  {  0.02990, -0.00879,  0.62263}
};

// mag offsets and correction matrix
float M_B[3]
{   -14.41,   21.43,   13.82};

float M_Ainv[3][3]
{ {  1.84445, -0.01730,  0.00046},
  { -0.01730,  1.90095, -0.01225},
  {  0.00046, -0.01225,  1.78335}
};

float G_off[3] = { -335.7, -86.1, 275.8}; //raw offsets, determined for
gyro at rest

float pi = 3.14159f;
float GyroMeasError = pi * (40.0f / 180.0f);   // gyroscope measurement
error in rads/s (start at 40 deg/s)
```

```
float GyroMeasDrift = pi * (0.0f  / 180.0f);   // gyroscope measurement
drift in rad/s/s (start at 0.0 deg/s/s)
float beta = sqrt(3.0f / 4.0f) * GyroMeasError;   // compute beta
float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;   // compute zeta, the
other free parameter in the Madgwick scheme usually set to a small or
zero value

//raw data and scaled as vector
int16_t ax, ay, az;
int16_t gx, gy, gz;
int16_t mx, my, mz;
float Axyz[3];
float Gxyz[3];
float Mxyz[3];

float Araw[3];
float Graw[3];
float Mraw[3];

float Acal[3];
float Gcal[3];
float Mcal[3];
#define gscale (250./32768.0)*(PI/180.0)

// Vector to hold quaternion
static float q[4] = {1.0, 0.0, 0.0, 0.0};
float yaw, pitch, roll; //Euler angle output
float lin_acc[6] {0.f, 0.f, 0.f, 0.f, 0.f, 0.f};  // linear acceleration
(acceleration with gravity component subtracted)

unsigned long now = 0, last = 0; //micros() timers
float deltat = 0;  //loop time in seconds
unsigned long now_ms, last_ms = 0; //millis() timers
unsigned long print_ms = 20; //print every "print_ms" milliseconds

void setup() {
  // put your setup code here, to run once:
  Wire.begin();
  Serial.begin(115200);
  while (!Serial); //wait for connection

  // initialize device
  mpu.initialize();
  // verify connection
  Serial.println(mpu.testConnection() ? "MPU9250 OK" : "MPU9250 ??");
  last = micros();

  //---SD Card SetUp---//
/*
  Serial.print("Initializing SD card...");
  if(!SD.begin(CS_pin)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
  sdCard = SD.open("DATA.txt", FILE_WRITE);
  if (sdCard) {
```

```
    Serial.println("File opened ok");
    // print the headings for our data
    sdCard.println("Ax, Ay, Az, Gx, Gy, Gz, Mx, My, Mz, q0, q1, q2,
q3"); //Ax, Ay, Az, Gx, Gy, Gz, Mx, My, Mz, LinAx, LinAy, LinAz, q0, q1,
q2, q3
  }
  sdCard.close();*/
}

void loop() {
  // put your main code here, to run repeatedly:
  get_MPU_scaled();
  //get_MPU_cal();
  now = micros();
  deltat = (now - last) * 1.0e-6; //seconds since last update
  last = now;

  /* Sensors x (y)-axis of the accelerometer/gyro is aligned with the y
(x)-axis of the magnetometer;
   *  the magnetometer z-axis (+ down) is misaligned with z-axis (+ up)
of accelerometer and gyro!
   * We have to make some allowance for this orientation mismatch in
feeding the output to the quaternion filter.
   * For the MPU9250+MS5637 Mini breakout the +x accel/gyro is North,
then -y accel/gyro is East. So if we want te quaternions properly
aligned
   * we need to feed into the Madgwick function Ax, -Ay, -Az, Gx, -Gy, -
Gz, My, -Mx, and Mz. But because gravity is by convention
   * positive down, we need to invert the accel data, so we pass -Ax,
Ay, Az, Gx, -Gy, -Gz, My, -Mx, and Mz into the Madgwick
   * function to get North along the accel +x-axis, East along the accel
-y-axis, and Down along the accel -z-axis.
   * This orientation choice can be modified to allow any convenient
(non-NED) orientation convention.
  */
  //filter.update(gyr.y(),gyr.x(),-gyr.z(),-acc.y(),-acc.x(),acc.z(),-
hag.y(),-hag.x(),hag.z());
  //filter.update(Gxyz[1], Gxyz[0], -Gxyz[2], -Axyz[1], -Axyz[0],
Axyz[2], -Mxyz[1], -Mxyz[0], Mxyz[2]);
  //MadgwickQuaternionUpdate(-ax, ay, az, gx * pi / 180.0f, -gy * pi /
180.0f, -gz * pi / 180.0f,  my,  -mx, mz);
  //MadgwickQuaternionUpdate(Axyz[0], Axyz[1], Axyz[2], Gxyz[0],
Gxyz[1], Gxyz[2], Mxyz[1], Mxyz[0], -Mxyz[2]);
  MadgwickQuaternionUpdate(-Axyz[0], Axyz[1], Axyz[2], Gxyz[0], -
Gxyz[1], -Gxyz[2], Mxyz[1], -Mxyz[0], Mxyz[2]);
  //MadgwickQuaternionUpdate(Gxyz[1], Gxyz[0], -Gxyz[2], -Axyz[1], -
Axyz[0], Axyz[2], -Mxyz[1], -Mxyz[0], Mxyz[2]);

  //linear_acceleration(q[0], q[1], q[2], q[3]);
  /*roll = atan2f(q[0]*q[1] + q[2]*q[3], 0.5f - q[1]*q[1] - q[2]*q[2]);
  pitch = asinf(-2.0f * (q[1]*q[3] - q[0]*q[2]));
  yaw = atan2f(q[1]*q[2] + q[0]*q[3], 0.5f - q[2]*q[2] - q[3]*q[3]);

  // to degrees
  yaw   *= 57.29578f;
  pitch *= 57.29578f;
  roll *= 57.29578f;
```

```
  // corrected for local magnetic declination
  yaw = yaw - 0.29;
  if(yaw <= 0) {
    yaw = 360 + yaw;
  }*/
  /*if(yaw >= 180) {
    yaw = 360 - yaw;
  }
  if(yaw <= -180.0) {
    yaw = 360.0 + yaw;
  }*/

  /*roll  = atan2((q[0] * q[1] + q[2] * q[3]), 0.5 - (q[1] * q[1] + q[2]
* q[2]));
  pitch = asin(2.0 * (q[0] * q[2] - q[1] * q[3]));
  yaw   = atan2((q[1] * q[2] + q[0] * q[3]), 0.5 - ( q[2] * q[2] + q[3]
* q[3]));

  // to degrees
  yaw   *= 180.0 / PI;
  pitch *= 180.0 / PI;
  roll *= 180.0 / PI;

  // http://www.ngdc.noaa.gov/geomag-web/#declination
  //conventional nav, yaw increases CW from North, corrected for local
magnetic declination

  yaw = -yaw + 0.29;
  if (yaw < 0) yaw += 360.0;
  if (yaw >= 360.0) yaw -= 360.0;*/

  now_ms = millis(); //time to print?
  if (now_ms - last_ms >= print_ms) {
    last_ms = now_ms;
    // print angles for serial plotter...
    Serial.print(now_ms*0.001); Serial.print(", ");

    //print_accRawCal(); Serial.println("");
    //print_gyrRawCal(); //Serial.println("");

    //Serial.print(Axyz[2], 2); Serial.println("");
    print_acc(); Serial.print(", ");
    print_gyro(); Serial.print(", ");
    print_mag(); Serial.print(", ");

    print_q(); Serial.println("");
    //Logging();

    //print_linAcc(); Serial.println("");
    //print_linAcc2(); Serial.println("");

    /*Serial.print(yaw, 2);
    Serial.print(", ");
    Serial.print(pitch, 2);
    Serial.print(", ");
    Serial.println(roll, 2);
```

```
  }
}

void get_MPU_scaled(void) {
  float temp[3];
  int i;
  mpu.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);

  Gxyz[0] = ((float) gx - G_off[0]) * gscale; //250 LSB(d/s) default to
radians/s
  Gxyz[1] = ((float) gy - G_off[1]) * gscale;
  Gxyz[2] = ((float) gz - G_off[2]) * gscale;

  Axyz[0] = (float) ax;
  Axyz[1] = (float) ay;
  Axyz[2] = (float) az;
  //apply offsets (bias) and scale factors from Magneto
  for (i = 0; i < 3; i++) temp[i] = (Axyz[i] - A_B[i]);
  Axyz[0] = A_Ainv[0][0] * temp[0] + A_Ainv[0][1] * temp[1] +
A_Ainv[0][2] * temp[2];
  Axyz[1] = A_Ainv[1][0] * temp[0] + A_Ainv[1][1] * temp[1] +
A_Ainv[1][2] * temp[2];
  Axyz[2] = A_Ainv[2][0] * temp[0] + A_Ainv[2][1] * temp[1] +
A_Ainv[2][2] * temp[2];
  vector_normalize(Axyz);

  Mxyz[0] = (float) mx;
  Mxyz[1] = (float) my;
  Mxyz[2] = (float) mz;
  //apply offsets and scale factors from Magneto
  for (i = 0; i < 3; i++) temp[i] = (Mxyz[i] - M_B[i]);
  Mxyz[0] = M_Ainv[0][0] * temp[0] + M_Ainv[0][1] * temp[1] +
M_Ainv[0][2] * temp[2];
  Mxyz[1] = M_Ainv[1][0] * temp[0] + M_Ainv[1][1] * temp[1] +
M_Ainv[1][2] * temp[2];
  Mxyz[2] = M_Ainv[2][0] * temp[0] + M_Ainv[2][1] * temp[1] +
M_Ainv[2][2] * temp[2];
  vector_normalize(Mxyz);
}

void get_MPU_cal(void) {
  float temp[3];
  int i;
  mpu.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);

  Graw[0] = (float) gx * gscale;
  Graw[1] = (float) gy * gscale;
  Graw[2] = (float) gz * gscale;

  Araw[0] = (float) ax;
  Araw[1] = (float) ay;
  Araw[2] = (float) az;
  //vector_normalize(Araw);
}

float vector_dot(float a[3], float b[3])
{
```

```
  return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
}

void vector_normalize(float a[3])
{
  float mag = sqrt(vector_dot(a, a));
  a[0] /= mag;
  a[1] /= mag;
  a[2] /= mag;
}

/*void AccelEarthFrame() { //(float ax, float ay, float az) {
  //Acceleration earth reference
  //Rotation matrix
  q0 = q[0];
  q1 = q[1];
  q2 = q[2];
  q3 = q[3];

  a11 = (q0 * q0) + (q1 * q1) - (q2 * q2) - (q3 * q3); //a11 = -2.0f *
(q2 * q2 + q3 * q3) + 1.0f;
  a21 = 2.0f * (q1 * q2 + q0 * q3);
  a31 = 2.0f * (q1 * q3 - q0 * q2);

  a12 = 2.0f * (q1 * q2 - q0 * q3);
  a22 = (q0 * q0) - (q1 * q1) + (q2 * q2) - (q3 * q3); //a22 = -2.0f *
(q1 * q1 + q3 * q3) + 1.0f;
  a32 = 2.0f * (q0 * q1 + q2 * q3);

  a13 = 2.0f * (q0 * q2 + q1 * q3);
  a23 = 2.0f * (q2 * q3 - q0 * q1);
  a33 = (q0 * q0) - (q1 * q1) - (q2 * q2) + (q3 * q3); //a11 = -2.0f *
(q1 * q1 + q2 * q2) + 1.0f;

  //Matrix multiplication aR = Rxa
  aRx = (a11 * Axyz[0]) + (a12 * Axyz[1]) + (a13 * Axyz[2]);
  aRy = (a21 * Axyz[0]) + (a22 * Axyz[1]) + (a23 * Axyz[2]);
  aRz = (a31 * Axyz[0]) + (a32 * Axyz[1]) + (a33 * Axyz[2]);
  //aRz = (aRz*0.98)-0.98;

  //Serial.print("lin aRx, lin aRy, lin aRz: ");
  Serial.print(aRx, 2); Serial.print(","); Serial.print(aRy, 2);
Serial.print(","); Serial.println(aRz, 2);
}*/

void linear_acceleration(float qw, float qx, float qy, float qz)
{
  lin_acc[0] = Axyz[0] + (2.0f * (qw * qx + qy * qz));
  lin_acc[1] = Axyz[1] + (2.0f * (qx * qz - qw * qy));
  lin_acc[2] = Axyz[2] - (qw * qw - qx * qx - qy * qy + qz * qz);
  lin_acc[3] = Axyz[0] + (qw * qx + qy * qz);
  lin_acc[4] = Axyz[1] + (qw * qy - qx * qz);
  lin_acc[5] = Axyz[2] - (qx * qx + qy * qy);
}

void print_gyrRawCal() {
```

```
  Serial.print(Graw[0]); Serial.print(", "); Serial.print(Graw[1]);
Serial.print(", "); Serial.print(Graw[2]); //Serial.print(", ");
  //Serial.print(Gcal[0]); Serial.print(", "); Serial.print(Gcal[1]);
Serial.print(", "); Serial.print(Gcal[2]);
}

void print_accRawCal() {
  Serial.print(Araw[0]); Serial.print(", "); Serial.print(Araw[1]);
Serial.print(", "); Serial.print(Araw[2]); //Serial.print(", ");
  //Serial.print(Acal[0]); Serial.print(", "); Serial.print(Acal[1]);
Serial.print(", "); Serial.print(Acal[2]);
}

void print_acc() {
  Serial.print(Axyz[0], 2); Serial.print(", "); Serial.print(Axyz[1],
2); Serial.print(", "); Serial.print(Axyz[2], 2);
}

void print_gyro() {
  Serial.print(Gxyz[0], 2); Serial.print(", "); Serial.print(Gxyz[1],
2); Serial.print(", "); Serial.print(Gxyz[2], 2);
}

void print_mag() {
  Serial.print(Mxyz[0], 2); Serial.print(", "); Serial.print(Mxyz[1],
2); Serial.print(", "); Serial.print(Mxyz[2], 2);
}

void print_q() {
  Serial.print(q[0]); Serial.print(", "); Serial.print(q[1]);
Serial.print(", "); Serial.print(q[2]); Serial.print(", ");
Serial.print(q[3]);
  //Serial.print(q0); Serial.print(", "); Serial.print(q1);
Serial.print(", "); Serial.print(q2); Serial.print(", ");
Serial.print(q3);
}

void print_linAcc() {
  Serial.print(lin_acc[0], 2); Serial.print(", ");
Serial.print(lin_acc[1], 2); Serial.print(", ");
Serial.print(lin_acc[2], 2);
}

void print_linAcc2() {
  Serial.print(lin_acc[3], 2); Serial.print(", ");
Serial.print(lin_acc[4], 2); Serial.print(", ");
Serial.print(lin_acc[5], 2);
}

void Logging() {
  sdCard = SD.open("DATA.txt", FILE_WRITE);
  if(sdCard) {
    sdCard.print(now_ms*0.001); sdCard.print(", ");
    sdCard.print(Axyz[0], 2); sdCard.print(", "); sdCard.print(Axyz[1],
2); sdCard.print(", "); sdCard.print(Axyz[2], 2); sdCard.print(", ");
    sdCard.print(Gxyz[0], 2); sdCard.print(", "); sdCard.print(Gxyz[1],
2); sdCard.print(", "); sdCard.print(Gxyz[2], 2); sdCard.print(", ");
```

```
    sdCard.print(Mxyz[0], 2); sdCard.print(", "); sdCard.print(Mxyz[1],
2); sdCard.print(", "); sdCard.print(Mxyz[2], 2); sdCard.print(", ");
    sdCard.print(q[0]); sdCard.print(", "); sdCard.print(q[1]);
sdCard.print(", "); sdCard.print(q[2]); sdCard.print(", ");
sdCard.println(q[3]);
  }
  else {
    Serial.println("can't open sd card");
  }
  sdCard.close();
}
```

LAMPIRAN D *Listing Program* Matlab

Sumber listing program matlab :

[1] Madgwick, S. *Oscillatory-Motion-Tracking-With-x-IMU*. 2017. Tersedia dari: https://github.com/xioTechnologies/Oscillatory-Motion-Tracking-With-x-IMU. [URL dikunjungi pada 17 November 2022]

[2] Karimpour, A. *Wave Upward Zero Crossing Fungction.* 2020 Tersedia dari: https://www.mathworks.com/matlabcentral/fileexchange/59342-wave-upward-zero-crossing-function. [URL dikunjungi pada 26 Januari 2022]


```
%addpath('D:\Uyung\TA\Tsunami\Program\testing\ximu_matlab_library');
% include x-IMU MATLAB library
addpath('D:\Uyung\TA\Tsunami\Program\program_Matlab_Madgwick_FIX\quatern
ion_library');  % include quatenrion library
addpath('D:\Uyung\TA\Tsunami\Program\program_Matlab_Madgwick_FIX\Gyrosco
peIntegration');    % include Gyroscope integration library
addpath('D:\Uyung\TA\Tsunami\Program\program_Matlab_Madgwick_FIX\Acceler
ometerMagnetometer');    % include Accelerometer and Magnetometer
Integration library

close all;                      % close all figures
clear;                          % clear all variables
clc;                            % clear the command terminal

%% Import data
[~, ~, raw] = xlsread('D:\DataGelombang2.xlsx','sheet1','A3:N1278');
%Data 1
%[~, ~, raw] = xlsread('D:\DataGelombang2.xlsx','sheet2','A2:N831');
%Data 2
%[~, ~, raw] = xlsread('D:\DataGelombang2.xlsx','sheet3','A3:N831');
%Data 3
%[~, ~, raw] = xlsread('D:\DataGelombang2.xlsx','sheet5','A2:N1023');
%Data 4
%[~, ~, raw] = xlsread('D:\DataGelombang2.xlsx','sheet4','A2:N831');
%Data 5

% Allocate imported array to column variable names
% Create output variable
data = reshape([raw{:}],size(raw));
tim = data(:,1);
acce = data(:,2:4);
gyr = data(:,5:7);
magn = data(:,8:10);
q = data(:,11:14);

% convert degree to radian
% gyr = deg2rad(gyro);

samplePeriod = 1/33;

% Plot
figure('NumberTitle', 'off', 'Name', 'Accelerometer');
```

```
hold on;
plot(acce(:,1), 'r');
plot(acce(:,2), 'g');
plot(acce(:,3), 'b');
xlabel('sample');
ylabel('g');
title('Accelerometer');
legend('X', 'Y', 'Z');
 %
%% Cuplik;
m1 = 500; n1 = 1000; t1 = n1-m1+1;
% m2 = 350; n2 = 700; t2 = n2-m2+1;
% m3 = 200; n3 = 600; t3 = n3-m3+1; %m4 = 780; n4 = 880; t4 = n4-m4+1;
%570:800 #333333333
% m4 = 300; n4 = 800; t4 = n4-m4+1; %150:400, 500:750, 800:1050-800:840
% m5 = 300; n5 = 600; t5 = n5-m5+1;

% Data 1
accX_ = acce(m1:n1,:); %
gyr_ = gyr(m1:n1,:);
mag_ = magn(m1:n1,:);
q_ = q(m1:n1,:);
time_ = tim(1:t1);

% % Data 2
% accX_ = acce(m2:n2,:); %
% gyr_ = gyr(m2:n2,:);
% mag_ = magn(m2:n2,:);
% q_ = q(m2:n2,:);
% time_ = tim(1:t2);

% % Data 3
% accX_ = acce(m3:n3,:); %
% gyr_ = gyr(m3:n3,:);
% mag_ = magn(m3:n3,:);
% q_ = q(m3:n3,:);
% time_ = tim(1:t3);

% % Data 4
% accX_ = acce(m4:n4,:); %
% gyr_ = gyr(m4:n4,:);
% mag_ = magn(m4:n4,:);
% q_ = q(m4:n4,:);
% time_ = tim(1:t4);

% % Data 4
% accX_ = acce(m5:n5,:); %
% gyr_ = gyr(m5:n5,:);
% mag_ = magn(m5:n5,:);
% q_ = q(m5:n5,:);
% time_ = tim(1:t5);


% time = t;
% quat = q;
% accX = acc;
```

```
time = time_; %5,6
quat = q_;
acc = accX_;
gyro = gyr_;
mag = mag_;

% % Plot
% figure('NumberTitle', 'off', 'Name', 'Acceleration Cuplik');
% hold on;
% plot(accX(:,2), 'g');
% xlabel('sample');
% ylabel('m/s2');
% title('acceleration Cuplik');

%% Process data through AHRS algorithm (calcualte orientation)
% See: http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/

R1 = zeros(3,3,length(gyro));     % rotation matrix describing sensor
relative to Earth

AHRS = MadgwickAHRS('SamplePeriod', samplePeriod, 'Beta', 0.1);

quaternion = zeros(length(time),4);
for t = 1:length(time)
    AHRS.Update(gyro(t,:), acc(t,:), mag(t,:)); % gyroscope units must
be radians
    quaternion(t, :) = AHRS.Quaternion;
    R1(:,:,t) = quatern2rotMat(AHRS.Quaternion)';
end

% Quaternion to rotation matrix
R2 = quatern2rotMat(quat);

R = R2;

%% Calculate 'tilt-compensated' accelerometer

tcAcc = zeros(size(acc));  % accelerometer in Earth frame

for i = 1:length(acc)
    tcAcc(i,:) = R(:,:,i) * acc(i,:)';
end

%% Calculate linear acceleration in Earth frame (subtracting gravity)

linAcc = tcAcc - [zeros(length(tcAcc), 1), zeros(length(tcAcc), 1),
ones(length(tcAcc), 1)];
linAcc = linAcc * 9.81;     % convert from 'g' to m/s/s

% Plot
figure('NumberTitle', 'off', 'Name', 'Linear Acceleration');
hold on;
plot(linAcc(:,1), 'r');
plot(linAcc(:,2), 'g');
plot(linAcc(:,3), 'b');
xlabel('sample');
ylabel('g');
```

```
title('Linear acceleration');
legend('X', 'Y', 'Z');

%% High-pass filter linear Acceleration to remove drift
order = 4;
filtCutOff = 0.1;
[b, a] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
linAccHP = filtfilt(b, a, linAcc);

%% Calculate linear velocity (integrate acceleartion)
linVel = zeros(size(linAccHP));

% for i = 2:length(linAccHP)
%     linVel(i,:) = linVel(i-1,:) + linAccHP(i,:) * samplePeriod;
% end
linVel = cumtrapz(time,linAccHP);

% Plot
figure('NumberTitle', 'off', 'Name', 'Linear Velocity');
hold on;
plot(linVel(:,1), 'r');
plot(linVel(:,2), 'g');
plot(linVel(:,3), 'b');
xlabel('sample');
ylabel('g');
title('Linear velocity');
legend('X', 'Y', 'Z');

%% High-pass filter linear velocity to remove drift
[b, a] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
linVelHP = filtfilt(b, a, linVel);

% Plot
figure('NumberTitle', 'off', 'Name', 'High-pass filtered Linear
Velocity');
hold on;
plot(linVelHP(:,1), 'r');
plot(linVelHP(:,2), 'g');
plot(linVelHP(:,3), 'b');
xlabel('sample');
ylabel('g');
title('High-pass filtered linear velocity');
legend('X', 'Y', 'Z');

%% Calculate linear position (integrate velocity)

linPos = zeros(size(linVelHP));

% for i = 2:length(linVelHP)
%     linPos(i,:) = linPos(i-1,:) + linVelHP(i,:) * samplePeriod;
% end
linPos = cumtrapz(time,linVelHP);
linPoscm = linPos*100;

% Plot
figure('NumberTitle', 'off', 'Name', 'Linear Position');
hold on;
```

```
plot(linPoscm(:,1), 'r');
plot(linPoscm(:,2), 'g');
plot(linPoscm(:,3), 'b');
xlabel('sample');
ylabel('g');
title('Linear position');
legend('X', 'Y', 'Z');

%% High-pass filter linear position to remove drift
[b, a] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
linPosHP = filtfilt(b, a, linPos);
linPosHP = linPosHP*100;

% Plot
figure('NumberTitle', 'off', 'Name', 'High-pass filtered Linear
Position');
hold on;
plot(linPosHP(:,1), 'r');
plot(linPosHP(:,2), 'g');
plot(linPosHP(:,3), 'b');
xlabel('sample');
ylabel('g');
title('High-pass filtered linear position');
legend('X', 'Y', 'Z');

%% Play animation

% SamplePlotFreq = 8;
%
% SixDOFanimation(linPosHP, R, ...
%                 'SamplePlotFreq', SamplePlotFreq, 'Trail', 'Off', ...
%                 'Position', [9 39 1280 720], ...
%                 'AxisLength', 0.1, 'ShowArrowHead', false, ...
%                 'Xlabel', 'X (cm)', 'Ylabel', 'Y (cm)', 'Zlabel', 'Z
(cm)', 'ShowLegend', false, 'Title', 'Unfiltered',...
%                 'CreateAVI', false, 'AVIfileNameEnum', false,
'AVIfps', ((1/samplePeriod) / SamplePlotFreq));
%
%% End of script
waveZ2 = linPosHP(:,3);
waveZ = linPoscm(:,3);
fs = 33;
t = time;

% close all;
%
[H,T,Time,UpCrossIndex,UpCrossTime,UpCrossValue,TroughTime,TroughValue,C
restTime,CrestValue]=WaveUpZeroCrossing(waveZ,'zero',fs,'yes');
```