ALBA PASTOR Jaime
DENIS MARTIN Hugo

PROJET SENSOR FUSION

MPU6050: Accelerometer / Gyroscope

Reading MPU data

```
int MPU6050_compFilter::read(int start, uint8_t *buffer, int size)
{
  if (!this->is_ready())
    return 1;

  int i;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  Wire.write(start);
  Wire.endTransmission(false); // hold the I2C-bus

  // Third parameter is true: relase I2C-bus after data is read.
  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);

  i = 0;
  while (Wire.available() && i < size)
  {
    buffer[i++] = Wire.read();
  }

  if (i != size)
    return 1;

  this->set_last_read_time(millis());
  return 0; // return : no error
}
```

```
typedef union acc_t_gyro_union
{
  struct
  {
    uint8_t x_acc_h;
    uint8_t x_acc_l;
    uint8_t y_acc_h;
    uint8_t y_acc_l;
    uint8_t z_acc_h;
    uint8_t z_acc_l;
    uint8_t t_h;
    uint8_t t_l;
    uint8_t x_gyro_h;
    uint8_t x_gyro_l;
    uint8_t y_gyro_h;
    uint8_t y_gyro_l;
    uint8_t z_gyro_h;
    uint8_t z_gyro_l;
  } reg;
  struct
  {
    int x_acc;
    int y_acc;
    int z_acc;
    int temperature;
    int x_gyro;
    int y_gyro;
    int z_gyro;
  } value;
};

typedef struct euler_angles
{
  double roll;
  double pitch;
  double yaw;
};
```

Computing Euler Angles from accelerometer

```
void MPU6050_compFilter::compute_acc_angles(euler_angles *angles, acc_t_gyro_union *acc_t_gyro_data)
{
  double acc_x = acc_t_gyro_data->value.x_acc;
  double acc_y = acc_t_gyro_data->value.y_acc;
  double acc_z = acc_t_gyro_data->value.z_acc;

  angles->pitch = asin(acc_x / this->g()) * RADIANS_TO_DEGREES;
  angles->roll = atan(acc_y / acc_z) * RADIANS_TO_DEGREES;
  angles->yaw = 0;
}
```

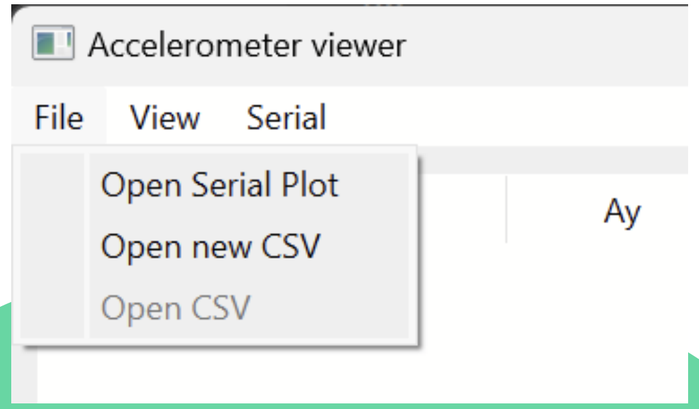
Compute Angle Estimation with complementary filter

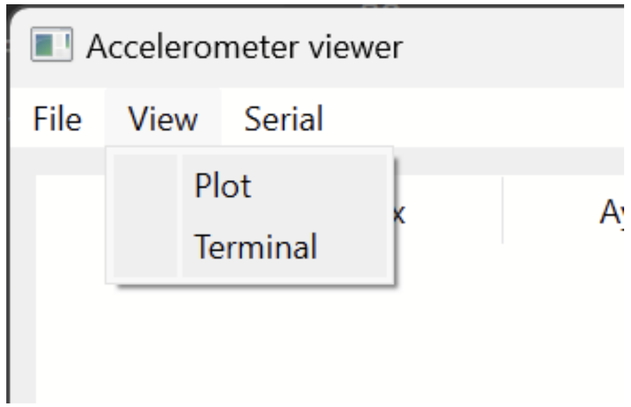
```
void MPU6050_compFilter::compute_angle_estimations()
{
  acc_t_gyro_union acc_t_gyro;
  this->read_acc_gyro_vals((uint8_t *)&acc_t_gyro);
  this->set_g(&acc_t_gyro);

  unsigned long t_now = millis();

  // Accelerometer data
  euler_angles acc_angles;
  this->compute_acc_angles(&acc_angles, &acc_t_gyro);

  // Gyrometer Data
  // Convert gyro values to degrees/sec
  double gyro_x = (acc_t_gyro.value.x_gyro - this->gyro_offset.roll);
  double gyro_y = (acc_t_gyro.value.y_gyro - this->gyro_offset.pitch);
  double gyro_z = (acc_t_gyro.value.z_gyro - this->gyro_offset.yaw);

  // Compute the (filtered) gyro angles
  double dt = (t_now - this->get_last_read_time()) / 1000.0;
  double gyro_angle_x = gyro_x * dt + this->get_last_roll();
  double gyro_angle_y = gyro_y * dt + this->get_last_pitch();
  double gyro_angle_z = gyro_z * dt + this->get_last_yaw();

  // Apply the complementary filter to figure out the change in angle - choice of alpha is
  // estimated now.  Alpha depends on the sampling rate...
  double alpha = 0.6;

  double angle_x = alpha * gyro_angle_x + (1.0 - alpha) * acc_angles.roll;
  double angle_y = alpha * gyro_angle_y + (1.0 - alpha) * acc_angles.pitch;
  double angle_z = (1 + alpha) * gyro_angle_z; // Accelerometer doesn't give z-angle

  this->set_last_angles(angle_x, angle_y, angle_z);
}
```

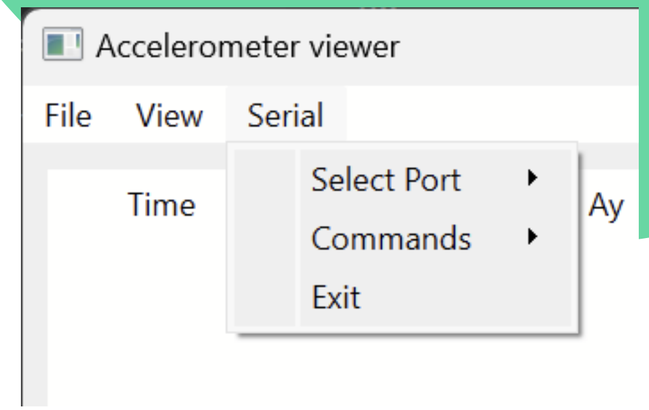
Public functions

```
public:
  MPU6050_compFilter() {}
  ~MPU6050_compFilter(){};

  void begin();
  void calibrate_gyro_offset();
  void compute_angle_estimations();
  euler_angles get_euler_angles(euler_angles *angles) const;
  acc_t_gyro_union get_acc_t_gyro_data(acc_t_gyro_union *acc_t_gyro_data) const;
};
```

Quick actions







Open data from csv