ISA code review

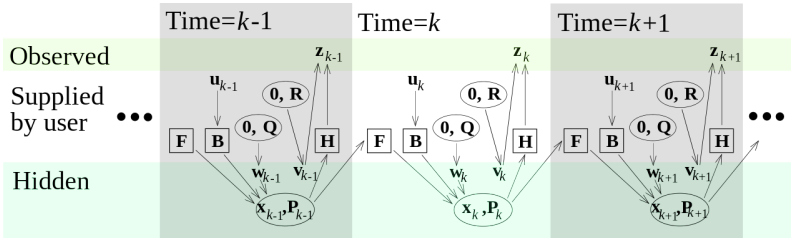
Naar aanleiding van het tussentijdse gesprek voor het ISA project, ben ik mij meer gaan verdiepen in de code die nodig is om de sensor werkend te krijgen. De code voor mijn ISA project, waarvoor ik een 9-axis sensor gebruik, heb ik niet zelf geschreven, omdat het dusdanig gecompliceerd is dat ik dan ofwel te weinig tijd heb in het algemeen of dat er geen tijd is voor de andere delen van mijn ISA (3d-modelleren en Unity). De code die ik gebruikt heb, maakt echter enkel gebruik van 6-axissen, namelijk de accelerometer en de gyroscope. Dit betekent dat de hoeken berekend worden op basis van de eerste positie van de sensor bij het maken van de connectie of het opstarten van een game. Met de magnetometer (zie het als een compas) kan er op basis van magnetische velden berekend worden wat de hoek van de meter is ten opzicht van de aarde. Dit geeft echter een extra complexiteit waar ik niet de tijd voor heb om het verder uit te pluizen.
https://en.wikipedia.org/wiki/Magnetometer

Om wel te leren en te begrijpen wat de gedachten achter de code is die ik voor het project overgenomen heb, doe ik bij deze een code review.

```
/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").
 Contact information
 -------------------
 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
 */
```

Include libraries and define variables and adresses

Om een connectie te maken en de data op te halen van de sensor is de Wire library nodig. Hierbij wordt het Kalman filter gebruikt om de accelerometer en de gyroscope te calculeren tot 1 stabiele hoekberekening. Dit is complexe wiskunde waar ik na veel onderzoek niet veel wijzer van zal worden. Het komt neer op een algoritme dat de waardes pakt van de sensoren en daar de ruis en onnauwkeurigheden omzet naar een schatting van wat ze zouden moeten zijn (wat dus ook beter blijkt te zijn).



De code bevat ook calculaties door enkel gebruik te maken van gyroscope en door middel van het complementary filter. Het complementary filter is ook een algoritme zoals het Kalman filter, maar iets minder precies en wiskundig minder betrouwbaar. Hij is wel een stuk sneller, makkelijker te gebruiken en toepasbaar en wordt door vele hobbyisten wel betrouwbaar genoeg bevonden. Hiervoor wordt dan ook geen speciale library gebruikt in de code.
https://en.wikipedia.org/wiki/Kalman_filter
https://www.pieter-jan.com/node/11

Verder wordt er een buffer, address en timeout ingesteld. Dit heeft te maken met een i2c-bus. Zo'n bus zijn 2 draden, SDA (om data te synchroniseren) en SCL (de data 'line'). Deze hebben allebei een address die in de variabel IMUAddress gezet wordt.
https://robot-electronics.co.uk/i2c-tutorial

Een buffer is een i2c-tool die (onder andere) gebruikt wordt om de rise time te verlagen. Rise time is de tijd die een signaal nodig heeft om van een opgegeven lage waarde naar een opgegeven hoge waarde te gaan (dus kan gezien worden als een vertraging).
https://www.allaboutcircuits.com/technical-articles/i2c-bus-when-to-use-an-i2c-buffer/

De timeout is de tijd tussen iedere keer dat er een check naar een error wordt uitgevoerd.

```
#include <Wire.h>
#include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter

#define RESTRICT_PITCH // Comment out to restrict roll to ±90deg instead

Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;

/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;

double gyroXangle, gyroYangle; // Angle calculate using the gyro only
double compAngleX, compAngleY; // Calculated angle using a complementary filter
double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data


const uint8_t IMUAddress = 0x68; // AD0 is logic low on the PCB
```

```
const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C communication
```

Read & write

Met deze functies worden een connectie gemaakt met de i2c bus van de sensor. Met de Wire.beginTransmission(IMUAddress) wordt een transmissie met de slave van de sensor gemaakt. Een slave is iets dat reageert op de master. De master is het device die de SCL (mechanisme dat stroom regelt) aandrijft. Samen kunnen ze data ophalen van de sensor. De write functie is ter voorbereiding op de endTransmission.
https://www.arduino.cc/en/Reference/WireBeginTransmission

```
uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
  return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0 on success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool
sendStop) {
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  Wire.write(data, length);
  uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on success
  if (rcode) {
    Serial.print(F("i2cWrite failed: "));
    Serial.println(rcode);
  }
  return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
  uint32_t timeOutTimer;
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  uint8_t rcode = Wire.endTransmission(false); // Don't release the bus
  if (rcode) {
    Serial.print(F("i2cRead failed: "));
    Serial.println(rcode);
    return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
  }
  Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a repeated start and
then release the bus after reading
  for (uint8_t i = 0; i < nbytes; i++) {
    if (Wire.available())
      data[i] = Wire.read();
    else {
      timeOutTimer = micros();
      while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available());
      if (Wire.available())
        data[i] = Wire.read();
      else {
        Serial.println(F("i2cRead timeout"));
```

```
        return 5; // This error value is not already taken by endTransmission
      }
    }
  }
  return 0; // Success
}
```

Setup()

De setup() runt 1x bij het opstarten van de Arduino. Hierbij worden de serial en i2c device gestart met specifieke frequencies. Daarna zie je dat bepaalde items van de lijst i2cData[] gewijzigd worden. Dit zijn instellingen voor de sensor. Daarna wordt met de write functie de instelling naar de sensor gestuurd.

Daarna wordt met een check (ingestelde sample is dan address) gekeken of er een error plaatsvindt bij het lezen van de sensor. Als dat niet het geval is, wordt er gekeken naar de huidige hoeken van de sensor(en, gyroscope en accelerometer). Er wordt dus geen magnetometer gebruikt, dus wordt de hoek bij de setup eenmalig geïnitialiseerd.

Deze output is alleen nog onbruikbaar. De hoeken zijn in andere eenheden dan we normaliter gebruiken, dus worden ze eerst omgerekend naar radians (door middel van atan2 berekeningen) en daarna naar graden.
https://en.wikipedia.org/wiki/Atan2

```
void setup() {
  Serial.begin(115200);
  Wire.begin();
  TWBR = ((F_CPU / 400000L) - 16) / 2; // Set I2C frequency to 400kHz

  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro
filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g
  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once
  while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference and
disable sleep mode

  while (i2cRead(0x75, i2cData, 1));
  if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
    Serial.print(F("Error reading sensor"));
    while (1);
  }

  delay(100); // Wait for sensor to stabilize

  /* Set kalman and gyro starting angle */
  while (i2cRead(0x3B, i2cData, 6));
  accX = (i2cData[0] << 8) | i2cData[1];
```

```
  accY = (i2cData[2] << 8) | i2cData[3];
  accZ = (i2cData[4] << 8) | i2cData[5];

  // Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25
and eq. 26
  // atan2 outputs the value of -π to π (radians) - see
http://en.wikipedia.org/wiki/Atan2
  // It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
  double roll  = atan2(accY, accZ) * RAD_TO_DEG;
  double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
  double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
  double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

  kalmanX.setAngle(roll); // Set starting angle
  kalmanY.setAngle(pitch);
  gyroXangle = roll;
  gyroYangle = pitch;
  compAngleX = roll;
  compAngleY = pitch;

  timer = micros();
}
```

Loop()

De loop functie wordt constant uitgevoerd door de Arduino. In eerste instantie wordt de data van de sensor opgehaald om de waardes te blijven updaten. Deze worden wederom omgerekend tot radians en daarna tot graden.

```
void loop() {
  /* Update all the values */
  while (i2cRead(0x3B, i2cData, 14));
  accX = ((i2cData[0] << 8) | i2cData[1]);
  accY = ((i2cData[2] << 8) | i2cData[3]);
  accZ = ((i2cData[4] << 8) | i2cData[5]);

  gyroX = (i2cData[8] << 8) | i2cData[9];
  gyroY = (i2cData[10] << 8) | i2cData[11];
  gyroZ = (i2cData[12] << 8) | i2cData[13];

  double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
  timer = micros();

  // Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25
and eq. 26
  // atan2 outputs the value of -π to π (radians) - see
http://en.wikipedia.org/wiki/Atan2
  // It is then converted from radians to degrees
```

```
#ifdef RESTRICT_PITCH // Eq. 25 and 26
  double roll  = atan2(accY, accZ) * RAD_TO_DEG;
  double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
  double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
  double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

  double gyroXrate = gyroX / 131.0; // Convert to deg/s
  double gyroYrate = gyroY / 131.0; // Convert to deg/s
```

Filteren

Vanaf nu worden de waardes berekend met gebruik van geen filter, het complementary filter en het Kalman filter. Hier wordt ook rekening gehouden met drifts. Dit gebeurt wanneer de accelerometer of gyroscope op een punt komt dat hij van de waarde 180 graden naar -180 graden gaat. Als dat het geval is, wordt de waarde simpelweg gereset.

De berekeningen met het Kalman filter is door de library erg goed leesbaar gemaakt. Je hebt een functie waar je de input geeft (`kalmanX.setAngle(roll)`) en de functie om de output daarvan te krijgen (`kalmanX.getAngle(roll, gyroXrate, dt);`). De berekeningen van het complementary filter lijken ook niet erg onoverzichtelijk. Hiervoor heb je namelijk een enkele formule (`0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll;`).
De berekening zonder filter ziet er alsvolgt uit (`gyroXangle += gyroXrate * dt(= deltatime);`).

```
#ifdef RESTRICT_PITCH
  // This fixes the transition problem when the accelerometer angle jumps between
-180 and 180 degrees
  if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
    kalmanX.setAngle(roll);
    compAngleX = roll;
    kalAngleX = roll;
    gyroXangle = roll;
  } else
    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using
a Kalman filter

  if (abs(kalAngleX) > 90)
    gyroYrate = -gyroYrate; // Invert rate, so it fits the restriced accelerometer
reading
  kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else
  // This fixes the transition problem when the accelerometer angle jumps between
-180 and 180 degrees
  if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
    kalmanY.setAngle(pitch);
    compAngleY = pitch;
    kalAngleY = pitch;
    gyroYangle = pitch;
```

```
  } else
    kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle
using a Kalman filter

  if (abs(kalAngleY) > 90)
    gyroXrate = -gyroXrate; // Invert rate, so it fits the restriced accelerometer
reading
  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a
Kalman filter
#endif

  gyroXangle += gyroXrate * dt; // Calculate gyro angle without any filter
  gyroYangle += gyroYrate * dt;
  //gyroXangle += kalmanX.getRate() * dt; // Calculate gyro angle using the
unbiased rate
  //gyroYangle += kalmanY.getRate() * dt;

  compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll; // Calculate the
angle using a Complimentary filter
  compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;

  // Reset the gyro angle when it has drifted too much
  if (gyroXangle < -180 || gyroXangle > 180)
    gyroXangle = kalAngleX;
  if (gyroYangle < -180 || gyroYangle > 180)
    gyroYangle = kalAngleY;
```

Print hoeken

Nu hebben we alles wat we nodig hebben. De hoeken worden geprint in de serial port en kunnen hieruit door Unity gelezen worden (of door bijvoorbeeld een bluetooth connectie). De variabelen spreken voor zich.

```
  /* Print Data */
#if 0 // Set to 1 to activate
  Serial.print(accX); Serial.print("\t");
  Serial.print(accY); Serial.print("\t");
  Serial.print(accZ); Serial.print("\t");

  Serial.print(gyroX); Serial.print("\t");
  Serial.print(gyroY); Serial.print("\t");
  Serial.print(gyroZ); Serial.print("\t");

  Serial.print("\t");
#endif

  Serial.print(roll); Serial.print("\t");
  Serial.print(gyroXangle); Serial.print("\t");
  Serial.print(compAngleX); Serial.print("\t");
  Serial.print(kalAngleX); Serial.print("\t");
```

```
  Serial.print("\t");

  Serial.print(pitch); Serial.print("\t");
  Serial.print(gyroYangle); Serial.print("\t");
  Serial.print(compAngleY); Serial.print("\t");
  Serial.print(kalAngleY); Serial.print("\t");

  Serial.print("\r\n");
  delay(2);
}
```

Conclusie

Dit is natuurlijk een voorbeeld waarin het meest complexe gedeelte weggelaten is. Toch komt er wat kennis bij kijken over de syntax van de Wire library en de benodigdheden voor een i2c device. Dit laatste is voornamelijk het belangrijkste voor mij, aangezien ik als interaction designer wil kunnen experimenteren en prototypen met meerdere sensoren die ook een slave en master bevatten. Er is echter een kans dat ik dingen verkeerd heb geïnterpreteerd of niet goed begrijp in deze code review. Daarom wil ik tijdens mijn eindgesprek voor dit project de code review bespreken om er meer over te leren.