University of California, Davis

Department of Land, Air and Water

Field Data-logger 1.0

Author:
Hengjiu Kang

July 19, 2015

Field Data-Loger 1.0

Hengjiu Kang[*]

Department of Electrical and Computer Engineering, University of California, Davis

July 19, 2015

[*]Electronic address: `hjkang@ucdavis.edu`

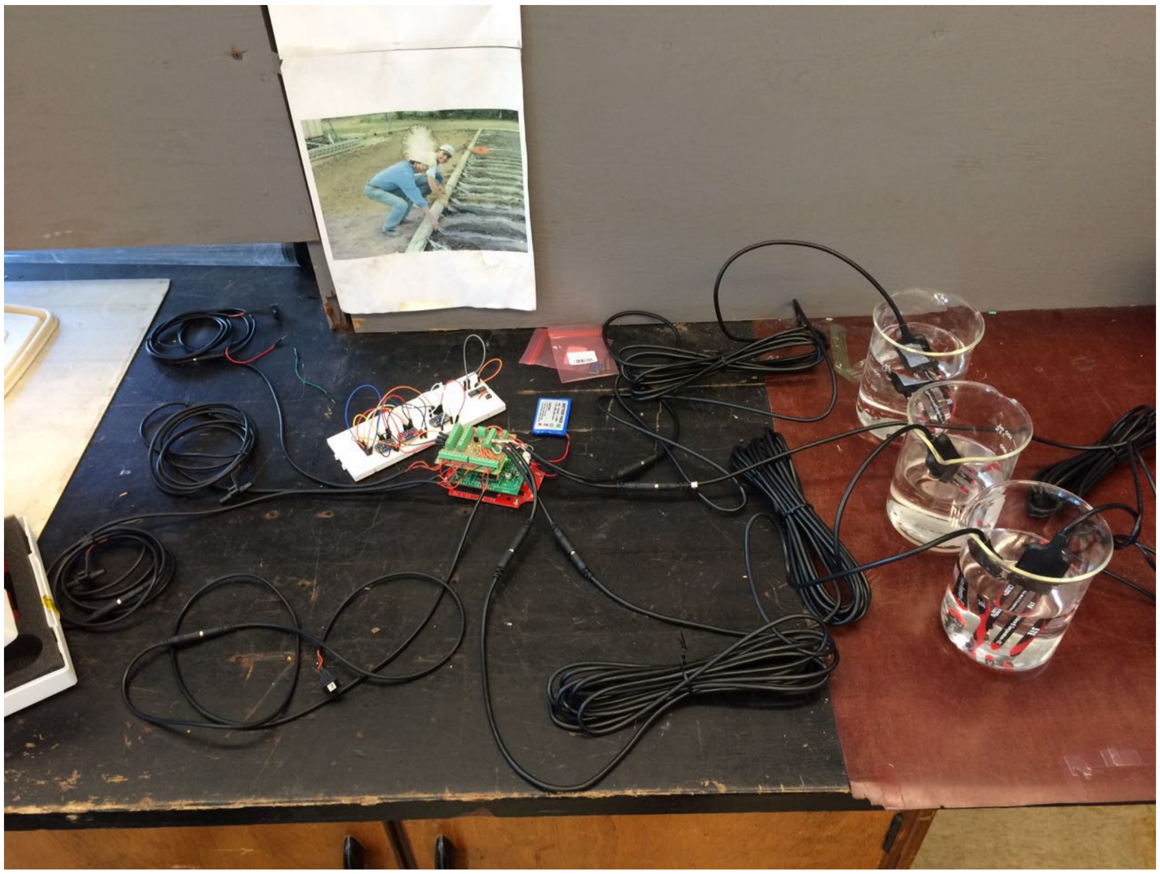

Figure 1: *Overall view of field Data-logger*

Contents

1 Overview **5**

2 Features **5**

3 Hardware Design **6**
3.1 Electrical Systems 6
3.1.1 Controllers 6

4 Software Design **7**
4.1 Overall strucutre 7

5 Results **7**
5.1 Design achivement 7

6 Future Work **7**

7 Acknowledgements and References **8**

Appendices **9**

A Mechanical Design **9**
A.1 Mechanical Design 9

B PCB Design **9**
B.1 Shield 9
B.2 Ankle Band 9

C BOM **9**

D Source code **9**
D.1 Applications 9
D.1.1 Main on Beaglebone Black 9
D.1.2 IMU on Arduino 12
D.1.3 IMU on Beaglebone Black 19
D.2 Libries 21
D.2.1 Customized Libs 21
D.2.2 IR Camera Driver 23
D.2.3 PID control 29
D.3 Device tree 29

E User Manual **31**

List of Figures

1 *Overall view of field Data-logger* 2
2 Seeeduino Stalker v2.3 6
3 Unpoured Band PCB Layout 9

1 Overview

This Field Data-logger is a new hardware based on Atmega328 chip which fuses multiple type of sensors from different brands. It can collect data in programmable period of time, and thanks to low power design, this Field Data-logger can last for very long time when solar panel is installed.

2 Features

- 8 16-bit single ended or 4 16-bit differential ADC inputs
- 4 I^2C connection ports
- 4 Read only sensors inputs
- Programmable wake up RTC
- Solar panel
- On-board SD card
- XBee network (Will be available in the next version)

3 Hardware Design

3.1 Electrical Systems

3.1.1 Controllers

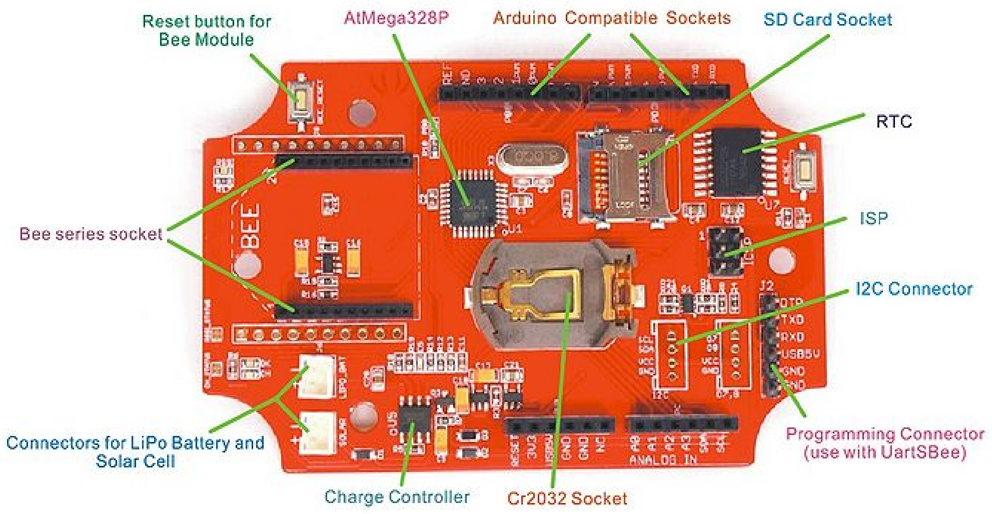
Reset button for Bee Module
AtMega328P
Arduino Compatible Sockets
SD Card Socket
RTC
ISP
Bee series socket
I2C Connector
Connectors for LiPo Battery and Solar Cell
Charge Controller
Cr2032 Socket
Programming Connector (use with UartSBee)

Figure 2: Seeeduino Stalker v2.3

4 Software Design

4.1 Overall strucutre

All the components in this robot are written in C++ 11 with STL and Linux components for speed consideration. The operations

5 Results

5.1 Design achivement

6 Future Work

7 Acknowledgements and References

References

[1] Derek Molloy, *Exploring Beaglebone*, Wiley, Hoboken, 1nd edition, 2014.

[2] Sigurd Skogestad, *Probaly the best simple PID tuning rules in the world*, Department of Chemical Engineering, Norwegian University of Science and Technology, 2001.

[3] Greg Welch, Grary Bishop, *An Introduction to the Kalman Filter*, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC.

[4] Justin Cooper, *Introduction to the BeagleBone Black Device Tree*, Adafruit, https://learn.adafruit.com/introduction-to-the-beaglebone-black-device-tree/overview, 2014.

[5] Kristian Lauszus, *KalmanFilter*, TKJ Electronics 2012, https://github.com/TKJElectronics/KalmanFilter, 2012.

[6] Yiğit Yüce, *Black Lib*, http://blacklib.yigityuce.com/.

[7] Eric Jacob, *Wii Remote IR Camera Hack with Arduino Interface*, http://www.instructables.com/id/Wii-Remote-IR-Camera-Hack/, 2010.

Acknowledgements

Special thanks to Bay Area Circuit

Appendices

A Mechanical Design

A.1 Mechanical Design

B PCB Design

B.1 Shield

B.2 Ankle Band

Figure 3: Unpoured Band PCB Layout

C BOM

Due to limited space, please check individual BOM file.

D Source code

D.1 Applications

D.1.1 Main on Beaglebone Black

```
#include <iostream>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctime>
#include "../naughtyException/naughtyException.h"
#include "../BlackLib/BlackLib.h"
#include "../BlackLib/BlackGPIO.h"
#include "../BlackLib/BlackPWM.h"
#include "../BlackStepper/DualStepperMotor.h"
#include "../PVision/IRRim.h"
#include "../PID/PID.h"

using namespace BlackLib;
using namespace std;

#define BIAS_COEFF 110

DualStepperMotor* motorPair;
IRRim* rim;
```

```
timer_t stepperIntID;
timer_t IRRimIntID;


void clean_up();
void handleStepper();
void handleIRRim();

void sig_handler(int signo)
{
    if (signo == SIGINT) {
        cout << "\nMAIN::Received SIGINT" << endl;
        clean_up();
        exit(0);
    }
}

// static void timerHandler( int sig, siginfo_t *si, void *uc ) {
//     timer_t *tidp;

//     tidp = (timer_t*)si->si_value.sival_ptr;

//     if ( *tidp == stepperIntID )
//         handleStepper();
//     else if ( *tidp == IRRimIntID )
//         handleIRRim();
// }

// static int makeTimer( timer_t *timerID, int expireMS, int intervalMS )
// {
//     struct sigevent te;
//     struct itimerspec its;
//     struct sigaction sa;
//     int sigNo = SIGRTMIN;

//     /* Set up signal handler. */
//     sa.sa_flags = SA_SIGINFO;
//     sa.sa_sigaction = timerHandler;
//     sigemptyset(&sa.sa_mask);
//     if (sigaction(sigNo, &sa, NULL) == -1) {
//         perror("sigaction");
//     }

//     /* Set and enable alarm */
//     te.sigev_notify = SIGEV_SIGNAL;
//     te.sigev_signo = sigNo;
//     te.sigev_value.sival_ptr = timerID;
//     timer_create(CLOCK_REALTIME, &te, timerID);

//     its.it_interval.tv_sec = 0;
//     its.it_interval.tv_nsec = intervalMS * 1000000;
//     its.it_value.tv_sec = 0;
//     its.it_value.tv_nsec = expireMS * 1000000;
//     timer_settime(*timerID, 0, &its, NULL);

//     return 1;
// }

string ZeroPadNumber(int num)
{
    stringstream ss;

    // the number is converted to string with the help of stringstream
    ss << num;
    string ret;
    ss >> ret;
    // Append zero chars
    int str_length = ret.length();
    for (int i = 0; i < 9 - str_length; i++)
        ret = "0" + ret;
    return ret;
}

int main (int argc, char* argv[]) {
    // Register sigint
    if (signal(SIGINT, sig_handler) == SIG_ERR) {
        cerr << "MAIN::Cannot register SIGINT handler" << endl;
    }


    // int retry_count = 0;
    try {
        motorPair = new DualStepperMotor(GPIO_15, EHRPWM0B, GPIO_27, EHRPWM2A, SPI0_0
            , GPIO_117, 4.5f, 0.0f, 10.0f);
        // motorPair = new DualStepperMotor(GPIO_15, EHRPWM0B, GPIO_27, EHRPWM2A,
            SPI0_0, GPIO_117, 0.0f, 0.0f, 0.0f);
        cout << "MAIN::Setup concluded" << endl;

        uint64_t freq = 100;
        int bias = 0;


        try {
            rim = new IRRim(4, EHRPWM1B, GPIO_48, AIN0);
        } catch (naughty_exception ex) {
            if (ex == naughty_exception_PVisionInitFail) {
                cerr << "MAIN::One or more IR sensors are malfunctioning, exiting" <<
                     endl;
                clean_up();
                exit(1);
            }
        }

        motorPair->setAcceleration(700);
        // cout << "MAIN::running at freq " << freq <<endl;
        motorPair->moveForward(0);
        motorPair->setBias(bias);

        // PID pid_turn(0.5, 1.0f / 40, 1.0f / 80, 3, 0);

        // int lost_count = 0;
    } catch (...) {
        clean_up();
        throw;
    }

    // makeTimer(&stepperIntID, 3, 20); //20ms
    // makeTimer(&IRRimIntID, 5, 5); //10ms
    for (;;) {
        // cout << "running " << endl;
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        // rim->run();
        IR_target newTarget = rim->run();
        // cout << newTarget << endl;
        if (newTarget.target_located) {
            // cout << "ir: " << newTarget.angle << endl;
            int angle = newTarget.angle + 100;
            if (angle < 0) {
                angle += 360;
            } else if (angle > 360) {
                angle -= 360;
            }
            if (angle < 15 || angle > 345) {
                motorPair->moveForward(3000);
            } else if (angle <= 180) {
                motorPair->moveLeft(angle);
            } else {
                // motorPair->moveForward(0);
                motorPair->moveRight(angle - 180);
            }
            // cout << "raw: " << angle << endl;
        } else {
            motorPair->moveForward(0);
        }
        motorPair->run();
         usleep(10);
    }
        return 0;
}

// void handleStepper() {
//     motorPair->run();
// }

// void handleIRRim() {
//     IR_target newTarget = rim->run();
//     if (newTarget.target_located) {
//         cout << newTarget.angle << endl;
//     }
// }


void clean_up() {
    if (motorPair) {
        delete motorPair;
    }
    if (rim) {
        delete rim;
    }
}
```

D.1.2 IMU on Arduino

```
/* Copyright (C) 2014 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
 */

/*

 * SPI pin numbers:
 * SCK   13  // Serial Clock.
 * MISO  12  // Master In Slave Out.
 * MOSI  11  // Master Out Slave In.
 * SS    10  // Slave Select
 */

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

#define RESTRICT_PITCH // Comment out to restrict roll to +-90deg instead - please
    read: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf
#define LED_PIN 13
 #define KALMAN_DEBUG

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

const uint8_t MPU6050 = 0x68; // If AD0 is logic low on the PCB the address is 0x68,
    otherwise set this to 0x69
const uint8_t HMC5883L = 0x1E; // Address of magnetometer

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

double roll, pitch, yaw; // Roll and pitch are calculated using the accelerometer
    while yaw is calculated using the magnetometer

double gyroXangle, gyroYangle, gyroZangle; // Angle calculate using the gyro only
double compAngleX, compAngleY, compAngleZ; // Calculated angle using a complementary
    filter
double kalAngleX, kalAngleY, kalAngleZ; // Calculated angle using a Kalman filter
volatile float kalmanTmp = 0;
volatile uint8_t kalmanSplit[4];
volatile bool inited = false;

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

#define MAG0MAX 603
#define MAG0MIN -578

#define MAG1MAX 542
#define MAG1MIN -701

#define MAG2MAX 547
#define MAG2MIN -556

float magOffset[3] = { (MAG0MAX + MAG0MIN) / 2, (MAG1MAX + MAG1MIN) / 2, (MAG2MAX +
    MAG2MIN) / 2 };
double magGain[3];

// SPI interrupt routine
```

```
ISR (SPI_STC_vect) {
  if (!inited) {
    SPDR = 0;
  }
  uint8_t cmd = SPDR;
  int32_t* kalmanInt = NULL;
  switch (cmd) {
    case 'x':
      kalmanTmp = (float)kalAngleX;
      kalmanInt = (int32_t*)(&kalmanTmp);
      // Serial.print("Kalman X = ");
      // Serial.println(kalmanTmp);
      for (int i = 0; i < 4; i++, (*kalmanInt) >>= 8) {
          kalmanSplit[i] = (uint8_t) ((*kalmanInt) & 0xFF);
      }
      SPDR = kalmanSplit[0];
      // Serial.print("Byte 1: 0x");
      // Serial.println(kalmanSplit[0], HEX);
    break;
    case 'y':
      kalmanTmp = (float)kalAngleY;
      kalmanInt = (int32_t*)(&kalmanTmp);
      // Serial.print("Kalman Y = ");
      // Serial.println(kalmanTmp);
      for (int i = 0; i < 4; i++, (*kalmanInt) >>= 8) {
          kalmanSplit[i] = (uint8_t) ((*kalmanInt) & 0xFF);
      }
      SPDR = kalmanSplit[0];
      // Serial.print("Byte 1: 0x");
      // Serial.println(kalmanSplit[0], HEX);
    break;
    case 'z':
      kalmanTmp = (float)kalAngleZ;
      kalmanInt = (int32_t*)(&kalmanTmp);
      // Serial.print("Kalman Z = ");
      // Serial.println(kalmanTmp);
      for (int i = 0; i < 4; i++, (*kalmanInt) >>= 8) {
          kalmanSplit[i] = (uint8_t) ((*kalmanInt) & 0xFF);
      }
      SPDR = kalmanSplit[0];
      // Serial.print("Byte 1: 0x");
      // Serial.println(kalmanSplit[0], HEX);
    break;
    case '1':
      SPDR = kalmanSplit[1];
      // Serial.print("Byte 2: 0x");
      // Serial.println(kalmanSplit[1], HEX);
    break;
    case '2':
      SPDR = kalmanSplit[2];
      // Serial.print("Byte 3: 0x");
      // Serial.println(kalmanSplit[2], HEX);
    break;
    case '3':
      SPDR = kalmanSplit[3];
      // Serial.print("Byte 4: 0x");
      // Serial.println(kalmanSplit[3], HEX);
    break;
    default:
      SPDR = 0x00;
    break;
  }
}

void setup() {
  inited = false;
  delay(100); // Wait for sensors to get ready
```

```
//  pinMode(LED_PIN, OUTPUT);
//  digitalWrite(LED_PIN, LOW);
//  delay(20);
//  digitalWrite(LED_PIN, HIGH);
//  delay(20);
//  digitalWrite(LED_PIN, LOW);
//  delay(20);
//  digitalWrite(LED_PIN, HIGH);
//  delay(20);
//  digitalWrite(LED_PIN, LOW);

  //SPI setup
  // have to send on master in, *slave out*
  pinMode(MISO, OUTPUT);
  // turn on SPI in slave mode
  SPCR |= _BV(SPE);
  // turn on interrupts
  SPCR |= _BV(SPIE);

  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(MPU6050, 0x19, i2cData, 4, false)); // Write to all four registers
      at once
  while (i2cWrite(MPU6050, 0x6B, 0x01, true)); // PLL with X axis gyroscope reference
      and disable sleep mode

  while (i2cRead(MPU6050, 0x75, i2cData, 1));
  if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
    Serial.print(F("Error reading sensor"));
    while (1);
  }

  while (i2cWrite(HMC5883L, 0x02, 0x00, true)); // Configure device for continuous
      mode
  calibrateMag();

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

  /* Set Kalman and gyro starting angle */
  updateMPU6050();
  updateHMC5883L();
  updatePitchRoll();
  updateYaw();

  kalmanX.setAngle(roll); // First set roll starting angle
  gyroXangle = roll;
  compAngleX = roll;

  kalmanY.setAngle(pitch); // Then pitch
  gyroYangle = pitch;
  compAngleY = pitch;

  kalmanZ.setAngle(yaw); // And finally yaw
  gyroZangle = yaw;
  compAngleZ = yaw;

  timer = micros(); // Initialize the timer
  inited = true;
}

void loop() {
  /* Update all the IMU values */
  updateMPU6050();
  updateHMC5883L();

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


  /* Roll and pitch estimation */
  updatePitchRoll();
  double gyroXrate = gyroX / 131.0; // Convert to deg/s
  double gyroYrate = gyroY / 131.0; // Convert to deg/s

#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 restricted 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 restricted accelerometer
        reading
  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a
      Kalman filter
#endif


  /* Yaw estimation */
  updateYaw();
  double gyroZrate = gyroZ / 131.0; // Convert to deg/s
  // This fixes the transition problem when the yaw angle jumps between -180 and 180
      degrees
  if ((yaw < -90 && kalAngleZ > 90) || (yaw > 90 && kalAngleZ < -90)) {
    kalmanZ.setAngle(yaw);
    compAngleZ = yaw;
    kalAngleZ = yaw;
    gyroZangle = yaw;
  } else
    kalAngleZ = kalmanZ.getAngle(yaw, gyroZrate, dt); // Calculate the angle using a
        Kalman filter


  /* Estimate angles using gyro only */
  gyroXangle += gyroXrate * dt; // Calculate gyro angle without any filter
  gyroYangle += gyroYrate * dt;
  gyroZangle += gyroZrate * dt;
  //gyroXangle += kalmanX.getRate() * dt; // Calculate gyro angle using the unbiased
      rate from the Kalman filter
  //gyroYangle += kalmanY.getRate() * dt;
  //gyroZangle += kalmanZ.getRate() * dt;

  /* Estimate angles using complimentary filter */
  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;
  compAngleZ = 0.93 * (compAngleZ + gyroZrate * dt) + 0.07 * yaw;

  // Reset the gyro angles when they has drifted too much
  if (gyroXangle < -180 || gyroXangle > 180)
    gyroXangle = kalAngleX;
  if (gyroYangle < -180 || gyroYangle > 180)
    gyroYangle = kalAngleY;
  if (gyroZangle < -180 || gyroZangle > 180)
    gyroZangle = kalAngleZ;


  /* Print Data */
#ifdef KALMAN_DEBUG
  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("\t");

  Serial.print(yaw); Serial.print("\t");
  Serial.print(gyroZangle); Serial.print("\t");
  Serial.print(compAngleZ); Serial.print("\t");
  Serial.print(kalAngleZ); Serial.print("\t");
#endif
#ifdef KALMAN_DEBUG
  Serial.print(accX / 16384.0); Serial.print("\t"); // Converted into g's
  Serial.print(accY / 16384.0); Serial.print("\t");
  Serial.print(accZ / 16384.0); Serial.print("\t");

  Serial.print(gyroXrate); Serial.print("\t"); // Converted into degress per second
  Serial.print(gyroYrate); Serial.print("\t");
  Serial.print(gyroZrate); Serial.print("\t");

  Serial.print(magX); Serial.print("\t"); // After gain and offset compensation
  Serial.print(magY); Serial.print("\t");
  Serial.print(magZ); Serial.print("\t");
  Serial.print("\t");

  double temperature = (double)tempRaw / 340.0 + 36.53;
  Serial.print(temperature); Serial.print("\t");
  Serial.println();
#endif


  delay(10);
}

void updateMPU6050() {
  while (i2cRead(MPU6050, 0x3B, i2cData, 14)); // Get accelerometer and gyroscope
      values
  accX = ((i2cData[0] << 8) | i2cData[1]);
  accY = -((i2cData[2] << 8) | i2cData[3]);
  accZ = ((i2cData[4] << 8) | i2cData[5]);
  tempRaw = (i2cData[6] << 8) | i2cData[7];
  gyroX = -(i2cData[8] << 8) | i2cData[9];
  gyroY = (i2cData[10] << 8) | i2cData[11];
  gyroZ = -(i2cData[12] << 8) | i2cData[13];
}

void updateHMC5883L() {
  while (i2cRead(HMC5883L, 0x03, i2cData, 6)); // Get magnetometer values
  magX = ((i2cData[0] << 8) | i2cData[1]);
  magZ = ((i2cData[2] << 8) | i2cData[3]);
  magY = ((i2cData[4] << 8) | i2cData[5]);
}

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

void updateYaw() { // See: http://www.freescale.com/files/sensors/doc/app_note/AN4248
    .pdf
  magX *= -1; // Invert axis - this it done here, as it should be done after the
      calibration
  magZ *= -1;

  magX *= magGain[0];
  magY *= magGain[1];
  magZ *= magGain[2];

  magX -= magOffset[0];
  magY -= magOffset[1];
  magZ -= magOffset[2];

  double rollAngle = kalAngleX * DEG_TO_RAD;
  double pitchAngle = kalAngleY * DEG_TO_RAD;

  double Bfy = magZ * sin(rollAngle) - magY * cos(rollAngle);
  double Bfx = magX * cos(pitchAngle) + magY * sin(pitchAngle) * sin(rollAngle) +
      magZ * sin(pitchAngle) * cos(rollAngle);
  yaw = atan2(-Bfy, Bfx) * RAD_TO_DEG;

  yaw *= -1;
}

void calibrateMag() { // Inspired by: https://code.google.com/p/open-headtracker/
  i2cWrite(HMC5883L, 0x00, 0x11, true);
  delay(100); // Wait for sensor to get ready
  updateHMC5883L(); // Read positive bias values

  int16_t magPosOff[3] = { magX, magY, magZ };

  i2cWrite(HMC5883L, 0x00, 0x12, true);
  delay(100); // Wait for sensor to get ready
  updateHMC5883L(); // Read negative bias values

  int16_t magNegOff[3] = { magX, magY, magZ };
  i2cWrite(HMC5883L, 0x00, 0x10, true); // Back to normal

  magGain[0] = -2500 / float(magNegOff[0] - magPosOff[0]);
  magGain[1] = -2500 / float(magNegOff[1] - magPosOff[1]);
  magGain[2] = -2500 / float(magNegOff[2] - magPosOff[2]);

#ifdef KALMAN_DEBUG
  Serial.print("Mag cal: ");
  Serial.print(magNegOff[0] - magPosOff[0]);
  Serial.print(",");
  Serial.print(magNegOff[1] - magPosOff[1]);
  Serial.print(",");
  Serial.println(magNegOff[2] - magPosOff[2]);

  Serial.print("Gain: ");
  Serial.print(magGain[0]);
  Serial.print(",");
  Serial.print(magGain[1]);
  Serial.print(",");
  Serial.println(magGain[2]);
#endif
}
```

D.1.3 IMU on Beaglebone Black

```
#include "arduinoConnector.h"
#include <iostream>
#include <stdint.h>
#include <unistd.h>
#include <cstdio>

using namespace std;


arduinoConnector::arduinoConnector(spiName port) :
spi(port, 8, SpiDefault, 2400000),
reset_enabled(false)
{
        spi.open(ReadWrite);
        usleep(100);
}

arduinoConnector::arduinoConnector(spiName port, gpioName reset_pin_name) :
spi(port, 8, SpiDefault, 2400000),
reset_enabled(true)
{
        spi.open(ReadWrite);
        usleep(100);
        reset_pin = new BlackGPIO(reset_pin_name, output, SecureMode);
        reset();
}

arduinoConnector::~arduinoConnector() {
        if (reset_enabled) {
                delete reset_pin;
        }
}

float arduinoConnector::angleInfomation(arduinoConnectorKalmanAngle axis) {
        float* rtnValue = new float(0);
        uint8_t parts[4];
        switch (axis) {
                case arduinoConnector_KalmanX:
                        spi.transfer('x', 10) & 0xFF;
                break;
                case arduinoConnector_KalmanY:
                        spi.transfer('y', 10) & 0xFF;
                break;
                case arduinoConnector_KalmanZ:
                        spi.transfer('z', 10) & 0xFF;
                break;
                default:
                return 0.0f;
        }
        parts[0] = spi.transfer('1', 10) & 0xFF;
        // printf("Byte 1: 0x%X\n", parts[0] & 0xFF);
        parts[1] = spi.transfer('2', 10) & 0xFF;
        // printf("Byte 2: 0x%X\n", parts[1] & 0xFF);
        parts[2] = spi.transfer('3', 10) & 0xFF;
        // printf("Byte 3: 0x%X\n", parts[2] & 0xFF);
        parts[3] = spi.transfer(0x00, 10) & 0xFF;
        // printf("Byte 4: 0x%X\n", parts[3] & 0xFF);
        int32_t resultInt = parts[3];
        resultInt <<= 8;
        resultInt |= parts[2];
        resultInt <<= 8;
        resultInt |= parts[1];
        resultInt <<= 8;
        resultInt |= parts[0];
        // printf("int: 0x%X\n", resultInt);
        rtnValue = (float*)(&resultInt);
        return *rtnValue;
}

void arduinoConnector::reset() {
        if (!reset_enabled) {
                return;
        }
        reset_pin->setValue((digitalValue)0);
        usleep(200 * 1000);
        reset_pin->setValue((digitalValue)1);
}
```

```
#ifndef ARDUINO_CONNECTOR_H
#define ARDUINO_CONNECTOR_H

#include "../BlackLib/BlackLib.h"
#include "../BlackLib/BlackSPI.h"
#include "../BlackLib/BlackGPIO.h"

using namespace BlackLib;

typedef enum {
        arduinoConnector_KalmanX,
        arduinoConnector_KalmanY,
        arduinoConnector_KalmanZ
} arduinoConnectorKalmanAngle;

class arduinoConnector {
public:
        arduinoConnector(spiName port);
        arduinoConnector(spiName port, gpioName reset_pin_name);
        ~arduinoConnector();
        float angleInfomation(arduinoConnectorKalmanAngle axis);
        void reset();
private:
        BlackSPI spi;
        BlackGPIO* reset_pin;
        bool reset_enabled;
};

#endif
```

D.2 Libries

D.2.1 Customized Libs

```
#ifndef _BLACK_SERVO_H_
#define _BLACK_SERVO_H_

#include "../BlackLib/BlackLib.h"
#include "../BlackLib/BlackPWM.h"
#include "../BlackLib/BlackADC.h"
#include "../BlackLib/BlackGPIO.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

using namespace BlackLib;

#define BLACK_SERVO_DEFAULT_TOLERANCE 0.03f

class BlackServo {

public:
        BlackServo(pwmName driver, adcName adcPin);
        ~BlackServo();
        void calibrate();
        void move_to(int);
        void move_to(float);
        void set_duty_percent(float);
        float current_position();
        bool target_position_reached();
        void set_tolerance(float);

private:
        BlackPWM _dutycycle;
        BlackADC _feedback;
        float angle;
        bool _calibrated;
        float _adc_pos_low;
        float _adc_pos_high;
        float _adc_span;
        float _tolerance;
};


#endif
```

```
#ifndef BLACK_STEPPER_H
#define BLACK_STEPPER_H

#include "../BlackLib/BlackLib.h"
#include "../BlackLib/BlackPWM.h"
#include "../BlackLib/BlackGPIO.h"
#include <ctime>

#define PERIOD_MICRO_TO_FREQ(period) (1000000/period)
#define FREQ_TO_PERIOD_MICRO(freq) (1000000/freq)

#define PERIOD_MAX 10000
#define PERIOD_MIN 170

#define FREQ_MAX PERIOD_MICRO_TO_FREQ(PERIOD_MIN) // 5882
#define FREQ_MIN PERIOD_MICRO_TO_FREQ(PERIOD_MAX) // 100

#define SPEED_MAX (FREQ_MAX - 100)
#define SPEED_MIN (FREQ_MIN - 100)
```

```
#define BIAS_MAX 0.98

using namespace BlackLib;

class BlackStepper {
private:
        BlackGPIO _direction;
        BlackPWM _frequency;

        timespec _last_timestamp;

        int _target_speed;
        int _correction;
        int _current_speed;
        int _last_accel;
        int _nominal_last_accel;
        int _current_accelration_step;

        // int _turn_freq_bias;

        bool _speedReached;

        void setMovement(bool direction, unsigned int frequency);
        void setGPIOAndPWM(bool direction, unsigned int frequency);
        inline bool isLongEnough();

public:
        BlackStepper(gpioName direction, pwmName frequency);
        ~BlackStepper();
        void adjustSpeed(int speed, int correction);
        void adjustSpeed(int correction);
        void setSpeed(int speed);
        void stop();
        void setAcceleration(unsigned int acceration_step);

        unsigned int getAcceleration();
        float getLinearAcceleration();
        float getLinearSpeed();
        bool targetSpeedReached();

};

#endif
```

```
#ifndef DUAL_STEPPER_MOTOR_H
#define DUAL_STEPPER_MOTOR_H

#include "../BlackLib/BlackLib.h"
#include "../BlackLib/BlackPWM.h"
#include "../BlackLib/BlackGPIO.h"
#include "BlackStepper.h"
#include "../PID/PID.h"
#include "../arduinoConnector/arduinoConnector.h"
using namespace BlackLib;

#define STEPPER_BALANCING_ON

class DualStepperMotor {
private:
        BlackStepper leftStepper;
        BlackStepper rightStepper;
        arduinoConnector kalduino;
        PID pid;
        int turn_bias;
        bool current_direction;
        unsigned current_frequency;
        float roll_adjust;
        float calibratedLevel;

        void adjustBalance(bool direction, unsigned frequency);

public:
        DualStepperMotor(
                gpioName directionLeft,
                pwmName frequencyLeft,
                gpioName directionRight,
                pwmName frequencyRight,
                spiName kalman_spi_name,
                gpioName kalman_reset_pin_name,
                float pGain,
                float iGain,
                float dGain
        );

        void moveForward(unsigned int frequency);
        void moveBackward(unsigned int frequency);
        void moveLeft(int angle);
        void moveRight(int angle);
        void setAcceleration(unsigned int acceration_step);
        void run();
        void setBias(int bias);
        void stop();

        bool targetSpeedReached();

        ~DualStepperMotor();
};

#endif
```

D.2.2 IR Camera Driver

```
#ifndef BLOB_CONTROL_H
#define BLOB_CONTROL_H

#include <stdint.h>
#include "PVision.h"

//This function assume that both data for both cameras are already collected
//Only up to 2 blobs are handled
BlobCluster* normalize(uint8_t result_left, uint8_t result_right, PVision* pv_left,
    PVision* pv_right);
Blob average(uint8_t result, PVision* pv);
float verticalDistance(uint8_t result, PVision* pv);
float running_avg(float distance);
double calculate_target_coordinate(int left_x, int right_x);
#endif
```

```
#ifndef IR_TARGET_H
#define IR_TARGET_H

#include <stdint.h>
#include <iostream>

using namespace std;

typedef enum {
        IR_target_source_camera,
        IR_target_source_proximity

} IR_target_source;

```

```
/**
 * @brief Target information
 *
 * @param target_located         Indicate if this is a package containing valid target
      information, will be false
 *                               if no target is seen within vision
 * @param angle                          angle of target relatvie to the ball, 0-360
 * @param distance                       distance of the target from the ball in
     centimeters, if negative, means a distance
 *                               is not yet available (when following is still
     calculating)
 */
typedef struct {
        bool target_located;
        bool distance_available;
        uint16_t angle; //clockwise, up front is 0 degrees
        float distance_two_cam; // distance in cm
        float distance_one_cam;
        IR_target_source source;
} IR_target;

ostream& operator<<(ostream& os, const IR_target& t);

IR_target IR_target_running_avg(IR_target new_target);

#endif
```

```
#ifndef IRRim_h
#define IRRim_h

#include <iostream>
#include <vector>
#include <ctime>
#include "PVision.h"
#include "PCA9548A.h"
#include "vec.h"
// #include "ringBuf.h"
#include "../BlackLib/BlackCore.h"
#include "../BlackLib/BlackGPIO.h"
#include "../BlackLib/BlackPWM.h"
#include "../BlackServo/BlackServo.h"
#include "../naughtyException/naughtyException.h"

// #include "../ADS1x15/ADS1X15.h"

//TODO: Resolve write fail errors and deallocation delay

using namespace std;
using namespace BlackLib;

enum IRRimState {
        IRRimState_seeking,
        IRRimState_targetFound,
        IRRimState_reversing
};

enum IRSensorPair {
        IRSensorPairFront,
        IRSensorPairBack,
        IRSensorPairInvalid
};

enum IRReadResult {
        IRReadResultLost,
        IRReadResultBlobOnLeft,
        IRReadResultBlobOnRight,
        IRReadResultMiddle
```

```
};

typedef struct {
        float dState; // Last position input
        float iState; // Integrator state
        float iMax, iMin; // Maximum and minimum allowable integrator stat
        float iGain; // integral gain
        float pGain; // proportional gain
        float dGain; // derivative gain

} PID_IRRim;

/**
 * @brief Target information
 *
 * @param target_located         Indicate if this is a package containing valid target
     information, will be false
 *                               if no target is seen within vision
 * @param angle                          angle of target relatvie to the ball, 0-360
 * @param distance                       distance of the target from the ball in
     centimeters, if negative, means a distance
 *                               is not yet available (when following is still
     calculating)
 */
typedef struct {
        bool target_located;
        int angle; //clockwise, up front is 0 degrees
        double distance; // distance in cm
} IR_target;
ostream& operator<<(ostream& os, const IR_target& t);

/**
 * @brief [brief description]
 * @details [long description]
 *
 * @param num_of_sensors [description]
 * @param servoPin [description]
 * @param muxResetPin [description]
 * @param feedbackPin [description]
 * @return [description]
 */
class IRRim {
public:
        /**
         *      Constructor of IRRim
         *      mux is an array of mux GPIO bits from MSB to LSB
         */
        IRRim(uint8_t num_of_sensors, pwmName servoPin, gpioName muxResetPin, adcName
             feedbackPin);

        /**
         *
         */
        ~IRRim();

        /**
         * [reset description]
         */
        void reset();

        /**
         * @brief Update timestamp and do stuff
         * @details [long description]
         */
        IR_target run();

        void force_seek();
        IRReadResult read_IR(IRSensorPair pair);
        IRReadResult read_IR(IRSensorPair pair, Blob* left_avg, Blob* right_avg);

private:
        PCA9548A mux;
        PVision* sensors;
        BlackServo servo;
        uint8_t sensor_count;
        IRRimState seeking_state;
        IRSensorPair current_active_pair;
        timespec target_last_seen;
        int current_iteration;
        int current_lower_bound;
        int current_upper_bound;

        float servo_current_position;
        // bool is_seeking;
        bool seeking_is_upwared;
        IR_target dummy_target;
        IR_target last_target;
        bool should_reverse;

        const vec o_left;
        const vec o_right;
        const vec o_left_m_right;

        // TFRingBuffer<double> distance_list;

        void nextSensor();
        void select(uint8_t num);
        void seek();
        IR_target follow(IRSensorPair);
        void reverse();
        void inspect_sensors();
        double filtered_result(double distance);

        inline void validateBlob(uint8_t index_left, uint8_t index_right);

        // inline timespec time_diff(timespec t1, timespec t2);

        inline double calculate_target_coordinate(int left_x, int right_x);
        inline vec get_directional_vec(int x, int y);
        inline void calculate_intersection_point(vec directional_left, vec
            directional_right, float& z_left, float& z_right);
};

#endif
```

```
#ifndef PCA9548A_H
#define PCA9548A_H

#include <iostream>
#include <stdint.h>
#include "../I2CBase/I2CBase.h"
#include "../BlackLib/BlackLib.h"
#include "../BlackLib/BlackGPIO.h"

using namespace std;
using namespace BlackLib;

class PCA9548A : public I2CBase {
public:
        PCA9548A(gpioName reset_pin);
        ~PCA9548A();
        void selectChannel(uint8_t channel);
        void reset();
private:
        BlackGPIO reset_pin;
};

#endif
```

```
// PVision library for interaction with the Pixart sensor on a WiiMote
// This work was derived from Kako's excellent Japanese website
// http://www.kako.com/neta/2007-001/2007-001.html

// Steve Hobley 2009 - www.stephenhobley.com
//
// Yi Lu 2014, Modified for Beaglebone Black

#ifndef PVision_h
#define PVision_h

#include <iostream>
using namespace std;

// bit flags for blobs
#define BLOB1 0x01
#define BLOB2 0x02
#define BLOB3 0x04
#define BLOB4 0x08


// Returned structure from a call to readSample()
struct Blob
{
        int X;
        int Y;
        int Size;
        uint8_t number;
};

bool Blob_is_valid(Blob& b);
ostream& operator<<(ostream& os, const Blob& b);

typedef struct BlobCluster_t {
        Blob first;
        Blob second;
        Blob third;
        Blob forth;
        int validBlobCount;
} BlobCluster;

class PVision
{

public:
        PVision();
        ~PVision();

        bool isBusReady();
        bool isSensorReady();
        bool init();   // returns true if the connection to the sensor established
            correctly
        uint8_t readBlob();   // updated the blobs, and returns the number of blobs
            detected
        void reset();

        // Make these public
        Blob Blob1;
        Blob Blob2;
        Blob Blob3;
        Blob Blob4;

private:
        int i2cDescriptor;

        // per object data
        // int IRsensorAddress;
        // int IRslaveAddress;
        uint8_t data_buf[16];
        int i;
        int s;

        bool Write_2bytes(uint8_t d1, uint8_t d2);
        uint8_t blobcount; // returns the number of blobs found - reads the sensor

        bool initI2CBus();
        bool busReady;
        bool sensorReady;
};

#endif
```

```
#ifndef PROXIMITY_RING
#define PROXIMITY_RING

#include <stdint.h>
#include <stdbool.h>
#include "../ADS1X15/ADS1X15.h"

typedef struct {
        int degree_low;
        int degree_high;
        bool valid;
} proximity_target;

ostream& operator<<(ostream& os, const proximity_target& t);

class proximityRing {
private:
        Adafruit_ADS1015 lowADC;
        Adafruit_ADS1015 highADC;
        proximity_target current_target;
        uint16_t threadshold[8];

        void invalidateCurrentResult();
        void calibrate();
public:
        proximityRing();
        void pollRing();
        proximity_target checkTarget(uint8_t sensor_index);
        proximity_target currentTarget();
        void testChannel(uint8_t index);
};

#endif
```

```
#ifndef VEC_H
#define VEC_H

#include <iostream>
using namespace std;

class vec {
public:
        float x;
        float y;
        float z;
        vec();
        vec(float, float, float);
        vec operator+(const vec& v) const;
        vec operator-(const vec& v) const;
        vec operator*(const float& f) const;
        static float dot_product(const vec& v1, const vec& v2);
        static vec mid_point(const vec& v1, const vec& v2);
};

ostream& operator<<(ostream& os, const vec& v);

#endif
```

D.2.3 PID control

```
#ifndef PID_H
#define PID_H

class PID {
public:
        PID(float ig, float pg, float dg, float imax, float imin);
        ~PID();
        float kernel(float error, float position);
private:
        float dState; // Last position input
        float iState; // Integrator state
        float iMax, iMin; // Maximum and minimum allowable integrator stat
        float iGain; // integral gain
        float pGain; // proportional gain
        float dGain; // derivative gain
};

#endif
```

D.3 Device tree

```
/*
 * This is a template-generated file from BoneScript
 */

/dts-v1/;
/plugin/;

/{
    compatible = "ti,beaglebone", "ti,beaglebone-black";
    part_number = "BS_PINMODE_P9_25_0x17";

    exclusive-use =
        "P9.25",
        "gpio3_21";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            bs_pinmode_P9_25_0x17: pinmux_bs_pinmode_P9_25_0x17 {
                pinctrl-single,pins = <0x1ac 0x17>;
            };
        };
    };

    fragment@1 {
        target = <&ocp>;
        __overlay__ {
            bs_pinmode_P9_25_0x17_pinmux {
                compatible = "bone-pinmux-helper";
                status = "okay";
                pinctrl-names = "default";
                pinctrl-0 = <&bs_pinmode_P9_25_0x17>;
            };
        };
    };
};
```

```
/*
 * Copyright (C) 2013 CircuitCo
 * Copyright (C) 2013 Texas Instruments
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This is a template-generated file from BoneScript
 */
/dts-v1/;
/plugin/;

/ {
        compatible = "ti,beaglebone", "ti,beaglebone-black";

        /* identification */
        part-number = "BS_PWM_P9_29_0x11";

        /* state the resources this cape uses */
        exclusive-use =
                /* the pin header uses */
                "P9.29",
                /* the hardware IP uses */
                "ehrpwm0B";

        fragment@0 {
                target = <&am33xx_pinmux>;
                __overlay__ {
                        bs_pwm_P9_29_0x11: pinmux_bs_pwm_P9_29_0x11 {
                                pinctrl-single,pins = <0x194 0x11>;
                        };
                };
        };

        fragment@1 {
                target = <&ocp>;
                __overlay__ {
                        bs_pwm_test_P9_29 {
                                compatible      = "pwm_test";
                                pwms            = <&ehrpwm0 1 500000 1>;
                                pwm-names       = "PWM_P9_29";

                                pinctrl-names   = "default";
                                pinctrl-0       = <&bs_pwm_P9_29_0x11>;

                                enabled         = <1>;
                                duty            = <0>;
                                status          = "okay";
                        };
                };
        };
};
```

```
/*
 * Copyright (C) 2013 CircuitCo
 * Copyright (C) 2013 Texas Instruments
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This is a template-generated file from BoneScript
 */
/dts-v1/;
/plugin/;

/ {
        compatible = "ti,beaglebone", "ti,beaglebone-black";

        /* identification */
        part-number = "BS_PWM_P9_31_0x11";

        /* state the resources this cape uses */
        exclusive-use =
                /* the pin header uses */
                "P9.31",
                /* the hardware IP uses */
                "ehrpwm0A";

        fragment@0 {
                target = <&am33xx_pinmux>;
                __overlay__ {
                        bs_pwm_P9_31_0x11: pinmux_bs_pwm_P9_31_0x11 {
                                pinctrl-single,pins = <0x190 0x11>;
                        };
                };
        };

        fragment@1 {
                target = <&ocp>;
                __overlay__ {
                        bs_pwm_test_P9_31 {
                                compatible      = "pwm_test";
                                pwms            = <&ehrpwm0 0 500000 1>;
                                pwm-names       = "PWM_P9_31";

                                pinctrl-names   = "default";
                                pinctrl-0       = <&bs_pwm_P9_31_0x11>;

                                enabled         = <1>;
                                duty            = <0>;
                                status          = "okay";
                        };
                };
        };
};
```

E User Manual