Hindawi
Journal of Sensors
Volume 2018, Article ID 8695397, 14 pages
https://doi.org/10.1155/2018/8695397

Research Article

Design, Development, and Deployment of Real-Time Sensor Fusion (CnW+EKF) for a Linux-Based Embedded System Using Qt-Anywhere

Felipe P. Vista IV and Kil To Chong

Electronic Engineering Department, Chonbuk National University, Jeonju 54896, Republic of Korea

Correspondence should be addressed to Kil To Chong; kitchong@jbnu.ac.kr

Received 20 February 2018; Revised 20 April 2018; Accepted 15 May 2018; Published 24 June 2018

Academic Editor: Hyung-Sup Jung

This paper describes the design, development, and implementation of a real-time sensor fusion system that utilizes the classification and weighing plus extended Kalman filter algorithm to derive heading for navigation using inexpensive sensors. This algorithm was previously tested only through postprocessing using MATLAB and is now reprogrammed using Qt and deployed on a Linux-based embedded board for real-time operation. Various data from inexpensive sensors such as global positioning system devices, an electronic compass, and an inertial measurement unit were utilized to ultimately derive a more reliable and accurate heading value. The algorithm flow can be described with the GPS values first being evaluated and classified which are then fused with the EC heading using classification and weighing, whose result is then passed through an EKF to fuse with the IMU data. Real-time tests and trials were done to prove the operational capability of the developed process. The complete setup and configuration processes of the systems for development and deployment via Qt are also provided for those interested to replicate the process.

1. Introduction

The fusion of cheap sensor devices to generate information that would have performance similar to that of more expensive systems is a continuing and exciting research field. Existing works utilized GPS (global positioning system), EC (electronic compass), IMU (inertial measure unit), or a combination of any of them with varying fusion methods utilized such as numerical discretization [1], Kalman filter or its variations [2, 3], fuzzy logic [4], timing synchronization [5], dead reckoning [6], or ad hoc [7].

Sensor fusion is also utilized in various other sectors such as increasing the reliability of quality assessment and authentication of food and beverages [8] as well as a globally updated map of the plant and the dynamic information about velocities and positions of all automatic guided vehicles (AGVs) [9]. One other factory application that utilized sensor fusion is the monitoring of machining operations dependent on rotary cutters [10].

It is also used for the real-time recognition of human action that relied on diverse modality sensors (inertial and depth of vision) [11] while another work presented a technique for indoor position tracking and localization of pedestrians [12]. There is also the often-researched application of sensor data fusion for the pose estimation of a 3D mobile robot in indoor applications [13]. There is also research that looked on the performance of sensor fusion systems [14]; on the other hand, there was another study that designed and developed an open-source tool that can be used for evaluating data fusion systems which are primarily focused on maritime surveillance systems [15].

In the field of maritime and ships, there are systems designed to assist the ship captain in entering or leaving a harbor [16] including the Advanced Sensor Module of the MUNIN project for autonomous and unmanned shipping [17]. There are also related works on collision avoidance such as those of Flåten and Brekke [18] and Chen et al. [19]. Works similar to ours that used sensor fusion in deriving a

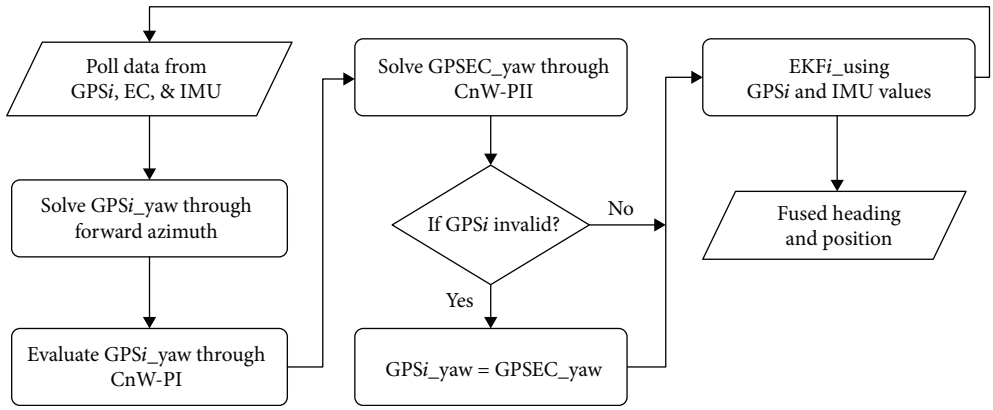


Figure 1: Simplified algorithm.

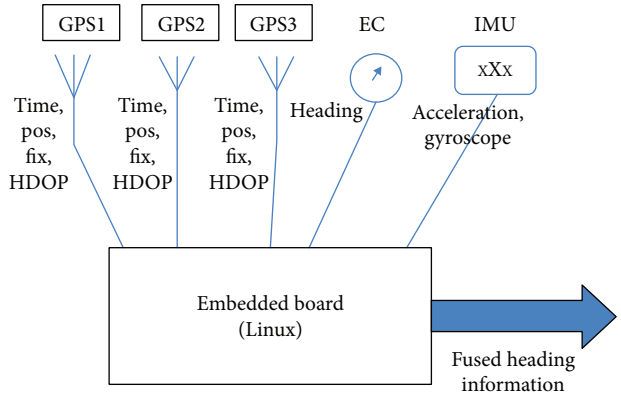


Figure 2: System design.

more accurate heading for navigation were by Hu and Huang [20] and Juang and Lin [21] while there were others that focused on the improved position and attitude aside from the heading such as those by Jaroś et al. [22], Bryne [23], Núñez et al. [24], and Feng-de et al. [25].

Cappello et al. applied sensor fusion to unmanned vehicles such as their work on integrated navigation and guidance systems (NGS) for small-sized UVs using low-cost off-the-shelf sensors [26]. They also worked on NGS for small- and medium-sized Remotely Piloted Aircraft Systems (RPAS) utilizing the GNSS- (global navigation satellite system-) and microelectromechanical system- (MEMS-) based IMU and vision-based navigation (VBN) sensor and the Aircraft Dynamics Model (ADM) that was treated as a virtual sensor [27, 28]. Other research works were implemented in an underwater environment that focused on robot pose estimation [29], towed array shape estimation [30], and passive target tracking [31].

These sensor fusion systems, especially the navigation systems, are usually deployed on embedded systems similar to ones that utilized field-programmable gate arrays (FPGA) with digital signal processors (DSP) [32]. The work presented in this paper details the development and deployment of a real-time sensor fusion system on an embedded board based on a previously proposed system of ours that was just done postprocessing [33]. Previously related works or our works are the real-time fusion of several GPS devices with an electronic compass initially on a notebook through classification and simplified weighing which was then followed by the first phase of real-time sensor fusion of three GPS devices and an electronic compass on an embedded board. There were also preceding studies done on postprocessing sensor fusion such as fuzzy logic and simplified classification and weighing. The current system was programmed using Qt-anywhere on a Linux desktop system and then deployed on an embedded board with the Linux system. This paper is arranged with the theoretical background of the implemented sensor fusion algorithm given in Section 2. The system design and implementation are then detailed in Section 3 while the developed system is presented in Section 4 followed by the concluding remarks.

2. Theoretical Background

The simplified algorithm for the real-time implementation of the previous theoretical work done via postprocessing is given in Figure 1. The currently implemented system on an embedded board utilized forward azimuth (FAz), classification and weighing (CnW), and extended Kalman filter (EKF) in order to derive the fused heading value from multiple inexpensive GPS, an EC, and an IMU.

The overall system design is in Figure 2 showing the inputs from various devices and the corresponding data that are utilized in the process with a 64-bit Ubuntu Linux desktop system acting as the platform for the design and development process of the real-time sensor fusion system. The developed program was then deployed on an embedded board with 32-bit Ubuntu Linux system (both BeagleBone Black and FreeScale). The graphical representation of the operational steps is shown in Figure 3.

2.1. Proposed Method. The GPS and EC both were sampled at 10 Hz while the IMU was sampled at 100 Mz. The data polled from these sensors were utilized for calculation of the heading through FAz (GPS data), classification and weighing-I (FAz data), classification and weighing-II (Faz, EC, and CnW-I data), and extended Kalman filter (FAz, CnW-I, and CnW-II data). The EKF prediction and update process involves a set of data each from GPS (FAz, GPSEC_yaw)

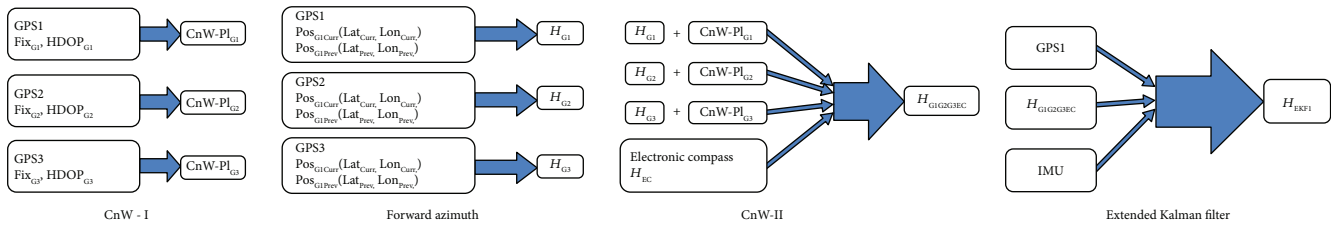


Figure 3: Graphical representation of the operational steps.

and IMU. The steps of the implemented algorithm are given as follows.

(1) Solve for individual heading values of the GPS devices (GPSi_yaw) through FAz, if there are valid GPS position values.

(2) Evaluate the derived individual GPS heading values through CnW-I.

(3) Fuse the solved individual GPS heading values with the EC heading value through CnW-II to derive GPSEC_yaw.

(4) Assign GPSEC_yaw as the heading value for any GPS device that does not have a valid position value (GPSi_yaw = GPSEC_yaw).

(5) Fuse the GPS and IMU values through EKF with the IMU accelerometer value treated as the local gravity vector measurement [9]. The inputs are lat$_i$, long$_i$, and GPSi_yaw while psi$_i$_EKF, lat$_i$_EKF, and long$_i$_EKF are the output values.

(6) Repeat steps 1 to 5 for the next set of IMU, EC, and GPS values.

2.2. Heading Derivation. Forward azimuth is described by the US Army as the angular measurement in clockwise direction of the line created by two points with the north set as the reference [34]. The equation for calculating the heading through forward azimuth is given in (1), taking note that the initial (lat$_2$, long$_2$) and succeeding (lat$_2$, long$_2$) position values are set so as to derive the value for initial heading. Other heading derivations were also tried such as centroid and direct arc tangent, but their performance was not as good as forward azimuth.

$$\begin{aligned} \arg_i &= \sin \Delta\text{long}^* \cos \text{lat}_2, \\ \arg_2 &= \cos \text{lat}_1{}^* \sin \text{lat}_2 - \sin \text{lat}_1 \cos \text{lat}_2 \Delta\text{long}, \\ \psi &= \tan^{-1}(\arg_1, \arg_2) + 180. \end{aligned} \tag{1}$$

2.3. Classification and Weighing. The classification and weighing process evaluates the quality of the polled GPS data by checking the values of the FIX type and the HDOP (Horizontal Dilution of Precision). The descriptive classification for the type of FIX that was adapted from http://www.gpsinformation.org/dale/nmea.html is given as follows:

(i) Invalid value is "0."

(ii) Standard Positioning Service or GPS mode is "1."

(iii) Differential GPS mode is "2."

(iv) Precise Positioning System is "3."

(v) Real-time kinematics is "4."

(vi) Float real-time kinematics is "5."

(vii) Dead reckoning or estimated fix is "6."

(viii) Manual input mode is "7."

(ix) Simulation mode is "8."

In the case of HDOP values, we utilized our own classification as follows:

(i) greater than 0 and less than or equal to 1 ($0 < \text{HDOP} \leq 1$), it is "IDEAL."

(ii) greater than 1 and less than or equal to 2 ($1 < \text{HDOP} \leq 2$), it is "EXCELLENT."

(iii) greater than 2 and less than or equal to 5 ($2 < \text{HDOP} \leq 5$), it is "GOOD."

(iv) greater than 5 and less than or equal to 10 ($5 < \text{HDOP} \leq 10$), it is "MODERATE."

(v) greater than 10 and less than or equal to 20 ($10 < \text{HDOP} \leq 20$), it is "FAIR."

(vi) greater than 20 and less than or equal to 1 ($20 < \text{HDOP} \leq 1$), it is "POOR."

The combined FIX and HDOP classification is now given as follows:

(i) If FIX is 2 to 5 and HDOP is 1 to 2, then it is IDEAL, with weight of "3."

(ii) If FIX is 1 and HDOP is 1 to 2, then it is EXCELLENT, with weight of "2."

(iii) If FIX is 2 to 5 and HDOP is 3 to 5, then it is EXCELLENT, with weight of "2."

(iv) If FIX is 1 and HDOP is 3 to 5, then it is GOOD, with weight of "1."

(v) If FIX is 2 to 5 and HDOP is 6 to 10, then it is GOOD, with weight of "1."

(vi) Else, it is BAD, with weight of "0."

The resulting weights are then used as inputs along with the GPS and EC heading values into (2) in order to solve for the CnW-II heading.

$$\begin{aligned} h_{\text{fused}} &= (h_{\text{allGPS}})(w_{\text{allGPS}}) + (h_{\text{EC}})(w_{\text{EC}}), \\ h_{\text{allGPS}} &= \frac{\sum_1^n (w_i h_i)}{\sum_1^n w_i}, \\ w_{\text{allGPS}} &= \frac{\sum_1^n w_i}{n^* \text{idealValue}}, \\ w_{\text{EC}} &= (1 - w_{\text{allGPS}}), \end{aligned} \tag{2}$$

where n is the number of GPS devices, idealValue is the weight value given for IDEAL, h_{fused} is the fused heading from electronic compass and GPS, w_{allGPS} is the weight assigned for the calculated GPS heading, w_{EC} is the weight assigned to the electronic compass, h_{allGPS} is the fused calculated heading for all the "n" GPS, w_i is the weight value given to the "ith" GPS, h_i is the derived heading of the "ith" GPS, and h_{fused} is assigned as the "GPSEC_yaw."

2.4. Extended Kalman Filter. The extended Kalman filter is utilized to derive the state estimate through the following general steps: (i) predicting the state and error covariance, (ii) deriving the Kalman gain, (iii) finding the time update of estimate, and then (iv) solving for the time update of error covariance. The observability and controllability of the Kalman filter or extended Kalman filter have been extensively studied and proved as can be read in the works of Trzuskowsky et al. [16], Bustamante et al. [15], and Simonetti et al. [14].

The nonlinear measurement and dynamic models are given as follows:

$$\begin{aligned} x_{k+1} &= f(x_k) + w_k, \\ z_k &= h(x_k) + v_k, \end{aligned} \tag{3}$$

where x_k and z_k are the state estimate and measurement, respectively. $\mathbf{R}_k$ and $\mathbf{Q}_k$ are the measurement noise and process noise covariance matrix and are assumed as positive definite. It is the linearity of the system which is why EKF was chosen for this work. The predicted state $\hat{x}_{k+1}^-$ and error covariance of the predicted state P_{k+1}^- are derived as follows:

$$\begin{aligned} \hat{x}_{k+1}^- &= f\left(\hat{x}_k^+\right), \\ P_{k+1}^- &= \mathbf{F}_k \mathbf{P}_k F_k^T + \mathbf{Q}_k, \end{aligned} \tag{4}$$

where F_k is the nonlinear dynamic model Jacobian matrix and P_k is the updated covariance matrix of the state for the previous time step k. The predicted measurement $\hat{z}_{k+1}^-$ is calculated as follows:

$$\hat{z}_{k+1}^- = h(\hat{x}_{k+1}^-). \tag{5}$$

The innovation covariance P_{k+1}^{vv} of the residual error between predicted and observed measurements is derived through

$$P_{k+1}^{vv} = P_{k+1}^{yy} + R_{k+1} = H_{k+1} P_{k+1}^- H_{k+1}^T + R_{k+1}, \tag{6}$$

where $P_{k+1}^{yy} = H_{k+1} P_{k+1}^- H_{k+1}^T$ is the output covariance, H_{k+1} is the Jacobian of the measurement function $h(\hat{x}_{k+1}^-)$ evaluated about the state prediction $\hat{x}_{k+1}^-$, and R_{k+1} is the measurement noise covariance of the sensor at time $k+1$.

The Kalman gain is now derived through

$$K_{k+1} = P_{k+1}^{xy} (P_{k+1}^{vv})^{-1} = P_{k+1}^- H_{k+1}^T (P_{k+1}^{vv})^{-1}. \tag{7}$$

The Kalman gain is computed by deriving the product of the predicted cross-correlation matrix with inverse of the innovation covariance matrix. The state distribution in the EKF algorithm is approximated by a Gaussian random variable that is propagated by first-order linearization of nonlinear functions.

The state vector $\mathbf{x}_k$ of the model is composed of the latitude, longitude, and heading as elements, $\mathbf{x}_k = [x_N, y_E, \psi]^T$, which is described as

$$\begin{aligned} \dot{x} &= \begin{Bmatrix} \dot{x}_N \\ \dot{y}_E \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} u \\ v \\ r \end{Bmatrix} + w \\ &= \begin{bmatrix} u\cos\psi - v\sin\psi \\ u\sin\psi + v\cos\psi \\ r \end{bmatrix} + w = f(x) + w. \end{aligned} \tag{8}$$

On the other hand, the model for measurement $h(x_k)$ is given as

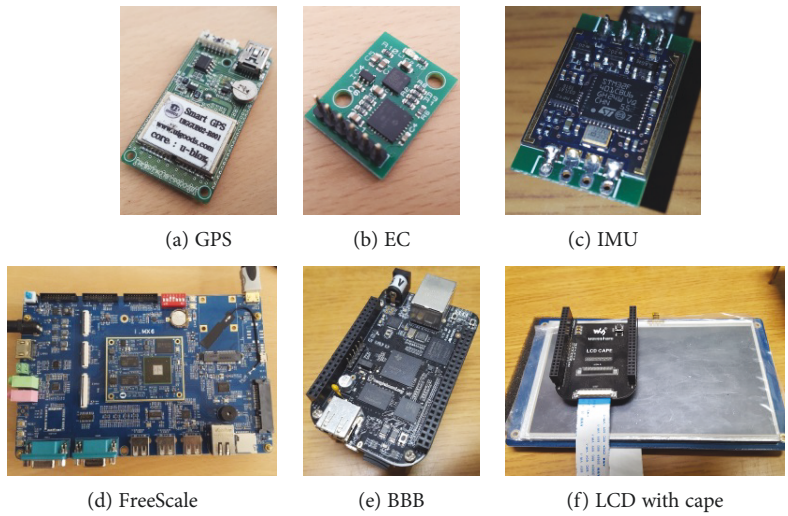
$$y_k = \begin{Bmatrix} x_{N,k} \\ y_{E,k} \\ \psi_k \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_{N,k} \\ y_{E,k} \\ \psi_k \end{pmatrix} = H_k x_k + v_k. \tag{9}$$

The measured values for estimating the state (latitude, longitude, and heading angle) are from three individual GPS devices and an EC. Process noise covariance (Q) and measurement noise covariance (R) values are set depending on the measured values as follows:

$$\begin{aligned} Q &= \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 10 \end{bmatrix}, \\ R_k &= \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.07 & 0 \\ 0 & 0 & 0.03 \end{bmatrix}. \end{aligned} \tag{10}$$

3. Design and Implementation

The system was designed and programmed using RBCDWBPA (Rapid By-Customer Demand with Business Process Approach) in tandem with the SFA (Systems



(a) GPS (b) EC (c) IMU

(d) FreeScale (e) BBB (f) LCD with cape

Figure 4: Devices utilized.

Features Analysis) development method on a 64-bit desktop computer with the Ubuntu Linux operating system using the Qt-anywhere 4.8-5 open-source version. It was initially developed on the desktop Linux box and then cross-compiled on the same Linux box for deployment to an ARM-based embedded board. It is not possible to directly connect the EC we are using to the serial port of both the Linux box and the embedded board due to the high-power output of their serial ports which would overload the serial port of the board. The EC therefore was connected to a simple Arduino-based system that was in turn polled by the Linux box or the embedded board.

3.1. Data and Devices. Three UIGGUB02-R001 from u-blox were utilized as inexpensive GPS receivers and were polled using the NMEA protocol. GPS data utilized in the sensor fusion process are timestamp (time when fix was taken), latitude, longitude, and COG (course over ground) which are all extracted from the RMC sentence (Recommended Minimum). On the other hand, the HDOP (Horizontal Dilution of Precision) and FIX (type of fix) are extracted from the GGA sentence which is the source for essential fix information. The EC used in this work was a Devantech Ltd. SEN0183 CMPS11 tilt-compensated model while the IMU was an EBIMU-9DOFV3 heading and attitude referencing system. Two embedded boards were utilized to demonstrate the generic design of the real-time sensor fusion system: namely, a FreeScale i-MX6 and a BeagleBone Black (BBB). A touchscreen with cape for a BeagleBone Black from Waveshare was used to serve as display while the FreeScale was connected to a HDMI-compatible display device. All the devices used in the implementation of the real-time sensor fusion system are shown in Figure 4.

3.2. Setting Up the System for Programming and Deployment. There are several steps that need to be done in setting up and configuring the desktop system for designing and programming the sensor fusion system as well as for cross-compiling and then deploying it on an embedded board such as the BBB. These important configurations and settings are broken down into four major substeps as follows:

(i) Configure TSLIB for Qt-desktop and Qt-arm on freshly installed Ubuntu 15.04-LTS.

(ii) Configure Qt-anywhere for the Qt-arm setup (cross-compile).

(iii) Configure fresh BBB for deployment.

(iv) Configure Qt-Creator for writing and deploying the program (both desktop and cross-compile versions).

The Qt utilized in this work is the Qt-4.8.5-anywhere open-source version while the toolchain used for compiling the fusion algorithm into an ARM-compatible system is the gcc-linaro-arm-linux-gnueabihf. TSLIB was also utilized to enable the touchscreen mode of the LCD display connected to a BBB.

A detailed step-by-step procedure for the given configuration settings is given in the Appendix. These steps are the result of various experimentations and tests that were done in order to be able to easily replicate the system configuration and settings needed to design, develop, and deploy a system onto an embedded system.

3.3. TinyEKF Library. The TinyEKF library by Simon D. Levy, which is available from GitHub (https://github.com/simondlevy/TinyEKF), was chosen to implement the EKF on an embedded board system. It is a straightforward C/C++ execution of the EKF that is generally enough for the implementation on STM32, Arduinos, and other microcontrollers. It is very suitable for utilization especially due to its utilization of static or compile-time memory allocation instead of "`malloc`" or "`new`."

There were customizations done on the TinyEKF library for it to be implemented with our sensor fusion system. The first thing was to make sure that the library would be recognized and compiled on a Qt environment

(a) SUBSYSTEM==”tty” value

(b) KERNELS==”1-4.1” value

Figure 5: A part of the result for “udevadm info –a –n /dev/tty/ACM0.”

wherein they were initially converted to cpp files and then additional Qt-related header files were included especially “math.h” and “qmath.h.”

For the tinyekf.cpp source file, the following changes were implemented:

(i) Comment out the line typedef struct ekf_t declaration.

(ii) Insert the line “mulmat(ekf.H, ekf.fx, ekf.hx, n, n, 1)” after the “/* \hat{x}_k = \hat{x_k} + G_k(z_k - h(\hat{x}_k)) */” line of the ekf_step function since the current implementation starts from “H” which is multiplied with “fx” to derive “hx.”

For the Tinyekfconfig.h header file, the number of states (“Nsta”) is set as “3” while the number of observables (“Mobs”) is also set into “3.”

3.4. Operating System Configuration. It is recommended to assign persistent names to all sensor devices so that the compiled system can be uploaded to similar embedded boards without the need for unique setting up and configuration of each fusion system as long as the same embedded board systems are utilized. This is to avoid the problem of setting the correct serial port to which a specific sensor is connected, that is, GPS, EC, and IMU. The Ubuntu Linux operating system usually assigns the next available serial port device name to the currently connected serial device, such as “/dev/ttyACM0,” “/dev/ttyACM1,” “/dev/ttyACM2,” “/dev/ttyACM3,” and “/dev/ttyUSB0” for the GPS1, GPS2, GPS3, EC, and IMU devices, respectively. A problem arises when the serial device gets disconnected and gets reconnected again since then there is a distinct probability that it will now be assigned a different available serial device name such as “/dev/ttyACM4,” “/dev/ttyACM5,” and “/dev/ttyACM6.” When this happens, the correct information from the specified sensor cannot be retrieved due to naming convention in the fusion system, that is, if the serial port device name in the fusion system for GPS1 is set as “/dev/ttyACM0” but the device is actually now recognized as “/dev/ttyACM4” by the embedded board system.

This situation is addressed by using persistent names to the sensor devices connected via the serial port so that even if the sensors get disconnected and connected, the necessary information can be extracted from the correct specified sensor. Given as follows are the steps in setting up persistent names in the Ubuntu Linux system which is applicable for both desktop and embedded board systems.

1. Open a terminal and run the command “udevadm info –a –n /dev/tty/ACM0.”

   *In our system, ACM0 is for GPS1, ACM1 is for GPS2, ACM2 is for GPS3, ACM3 is for EC, and ttyUSB0 is for IMUc.

   *Search for the needed values that will be encoded into the rule file (Figure 5).

2. Create the /etc/udev/rules.d/99-usb-serial.rules file.

3. Insert the following into the newly created file:

   SUBSYSTEM==”tty”, ATTRS{idVendor}==”2a03”, ATTRS{idProduct}==”0043”, ATTRS{product}==”Arduino Uno”, ATTRS{serial}==”85439303033351819221”, SYMLINK+=”fjEC”

   SUBSYSTEM==”tty”, ATTRS{idVendor}==”10c4”, ATTRS{idProduct}==”ea60”, ATTRS{serial}==”0001”, SYMLINK+=”fjIMU”

   SUBSYSTEM==”tty”, KERNELS==”1-4.1”, ATTRS{idVendor}==”1546”, ATTRS{idProduct}==”01a5”, ATTRS{product}==”u-blox 5 - GPS Receiver”, SYMLINK+=”fjGPS1”

   SUBSYSTEM==”tty”, KERNELS==”1-4.2”, ATTRS{idVendor}==”1546”, ATTRS{idProduct}==”01a5”, ATTRS{product}==”u-blox 5 - GPS Receiver”, SYMLINK+=”fjGPS2”

   SUBSYSTEM==”tty”, KERNELS==”1-4.3”, ATTRS{idVendor}==”1546”, ATTRS{idProduct}==”01a5”, ATTRS{product}==”u-blox 5 - GPS Receiver”, SYMLINK+=”fjGPS3”

   *Note that the “KERNELS” information is especially necessary when setting up a BBB embedded board. It is used to uniquely identify each of the connected similar model GPS devices.

   *The SYMLINK+= fjGPS3 value will be the one utilized in the program instead of the default device name to identify the sensor device, that is, “/dev/fjGPS3” instead of “/dev/ttyACM2.”

3.5. Multithreading. The multithreading capability of the Ubuntu system and the embedded boards was harnessed to simultaneously retrieve data from the various sensor devices as well as implement the various individual processes such as CnW-I, FAz, CnW-II, and EKF. There are a total of eight (8) threads running simultaneously. These threads and their specific jobs are described as follows:

(1) Thread 1 is the main thread for the fusion system.

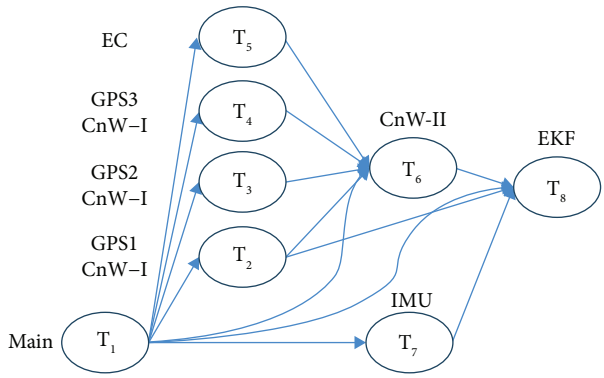


Figure 6: Multithreading design.

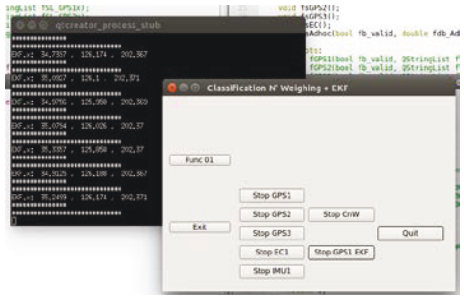


Figure 7: Real-time on a desktop.

(2) Thread 2 polls the data and performs CnW-I and then FAz on GPS1.

(3) Thread 3 polls the data and performs CnW-I and then FAz on GPS2.

(4) Thread 4 polls the data and performs CnW-I and then FAz on GPS3.

(5) Thread 5 polls the data from EC.

(6) Thread 6 performs CnW-II.

(7) Thread 7 polls the data from IMU.

(8) Thread 8 performs EKF on the data from GPS1 (those from GPS2 or GPS3 can also be used).

This multithreading design and the interconnection between the threads are graphically represented in Figure 6.

4. The Resulting Real-Time Sensor Fusion System

The sensor fusion was successfully implemented in real time on a Linux system desktop whose result is given in Figure 7. The same system was then cross-compiled in Qt for deployment on an ARM-based FreeScale board as well as on a BBB. The fusion system was able to perform as desired with the display shown in the LCD (Figure 8) and the output monitored through a remote login console on the embedded board (Figure 9). The main button for starting and stopping the whole sensor fusion process is labeled as "Start Sensor Fusion" which switches to "Stop Sensor Fusion" when the fusion process is running.

Figure 8: Program display on the embedded board.

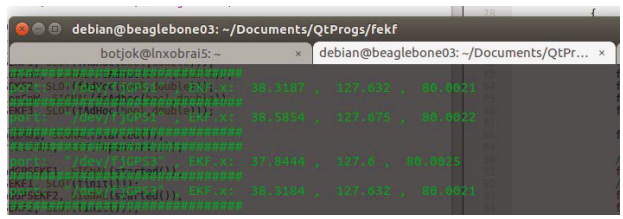


Figure 9: Fused heading and position on the embedded board.

The sensor fusion system developed also allowed the user to be able to control the polling of data from each of the sensors (start/stop GPS1, start/stop GPS2, start/stop GPS3, start/stop EC1, and start/stop IMU1). This was done so that it is possible to test the performance of the system by artificially introducing loss of data from either one or more of the sensors. It is also possible to start/stop the CnW-I and CnW-II processes as well as start the EKF process.

The system was able to run and still perform sensor fusion correctly when the sensor devices are disconnected and then reconnected or even switching the serial ports they are attached to. This is the main reason why it is important to implement the assigning of persistent names in the Ubuntu Linux systems of the embedded boards. This was tested for both the FreeScale and BBB with the same program deployed into it.

5. Conclusion

The real-time implementation of classification and weighing plus extended Kalman filter sensor fusion for the derivation of more accurate heading has been described. The theoretical background of the sensor fusion algorithm to the setting up and configuration of the desktop system and then the design and development using Qt-anywhere were described. The target embedded boards are then set up with the Ubuntu Linux system and then configured with the necessary libraries and drivers to be able to run the fusion system. The real-time sensor fusion system performed as designed, and the same configuration steps given in this paper allowed for the ease of deployment of the same system on an embedded board with very minimal changes that need to be done. Future work should focus on the addition of the covariance intersection algorithm into the real-time system to further improve the accuracy of the designed system.

Appendix

A. Configuration of TSLIB for Qt-Desktop and Qt-Arm on Freshly Installed Ubuntu 15.04-LTS

(1) Download toolchain (we use linaro)

∗ For gcc-linaro-arm-linux-gnueabihf-4.8-2013.10

$ wget -c https://launchpadlibrarian.net/155358238/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux.tar.xz

$ tar xJf gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux.tar.xz

$ chmod 777 - R gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux

∗ For gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf

$ wget -c https://releases.linaro.org/archive/15.02/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf.tar.xzz

$ tar xJf gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf.tar.xz

$ chmod 777 - R gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf

(2) Download tslib

∗ For tslib-1.2

$ wget -c https://github.com/kergoth/tslib/releases/download/1.2/tslib-1.2.tar.gz

$ tar xvf tslib-1.2.tar.gz

∗ For tslib-1.3 (with multitouch support)

$ wget -c https://github.com/kergoth/tslib/releases/download/1.3/tslib-1.3.tar.gz

$ tar xvf tslib-1.3.tar.gz

(3) Configure environment

∗ For gcc-linaro-arm-linux-gnueabihf-4.8-2013.10

$ export CC=/home/botjok/Documents/Qt/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-gcc

$ export CXX=/home/botjok/Documents/Qt/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-g++

$ export CROSS_COMPILE=/home/botjok/Documents/Qt/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-

∗ For gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf

$ export CC=/home/botjok/Documents/Qt/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc

$ export CXX=/home/botjok/Documents/Qt/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++

$ export CROSS_COMPILE=/home/botjok/Documents/Qt/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-

(4) Install necessary packages for compiling tslib

$ sudo apt-get install autoconf autogen intltool libtool

$ sudo apt-get install libc6-i386 lib32stdc++6

$ sudo apt-get install lib32ncurses5 lib32z1

(5) Cross-compile tslib

∗ For tslib-1.2 and 4.8-2013.10 toolchain

$ cd tslib-1.2

$ ./autogen.sh

$ sudo ./configure CC=/home/botjok/Documents/Qt/tmp/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-gcc CXX=/home/botjok/Documents/Qt/tmp/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-g++ –host=arm-linux –prefix=/usr/local/Cross tslibSE

$ sudo make

$ sudo make install

∗ For tslib-1.2 and 4.9-2015.02-3 toolchain

$ cd tslib-1.2

$ ./autogen.sh

$ sudo ./configure CC=/home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc CXX=/home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++ –host=arm-linux –prefix=/usr/local/Cross tslibSE4.9

$ sudo make

$ sudo make install

∗ For tslib-1.3 and 4.8-2013.10 toolchain

$ cd tslib-1.3

$ ./autogen.sh

$ sudo ./configure CC=/home/botjok/Documents/Qt/tmp/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-gcc CXX=/home/botjok/Documents/Qt/tmp/gcc-linaro-arm-linux-gnuea-

bihf-4.8-2013.10_linux/bin/arm-linux-gnueabihf-g++ –host=arm-linux –prefix=/usr/local/Crosstslib1.3bSE

$ sudo make

$ sudo make install

∗ For tslib-1.3 and 4.9-2015.02-3 toolchain

$ cd tslib-1.3

$ ./autogen.sh

$ sudo ./configure CC=/home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc CXX=/home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++ –host=arm-linux –prefix=/usr/local/Crosstslib1.3SE4.9

$ sudo make

$ sudo make install

(6) Check if the compiled library is ARM-compatible

$ file /usr/local/Crosstslib1.3SE4.9/bin/ts_calibrate

– result should be something similar as follows, especially the ", ARM,"

/usr/local/Crosstslib1.3SE4.9/bin/ts_calibrate: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.16, BuildID[sha1]=2dc789dcb60d6d9758c4d90b4c92c722bc661a34, not stripped

∗ The generated library will be used when compiling Qt and will also be copied to the BBB or other boards

B. Configuration of Qt-Anywhere for Qt-Arm Setup (Cross-Compile)

(1) Download Qt-4.8.5-everywhere open-source version

$ wget -c http://download.qt.io/archive/qt/4.8/4.8.5/qt-everywhere-opensource-src-4.8.5.tar.gz

$ tar xzvf qt-everywhere-opensource-src-4.8.5.tar.gz

(2) Download toolchain (we use linaro)

∗ For gcc-linaro-arm-linux-gnueabihf-4.8-2013.10

$ wget -c https://launchpadlibrarian.net/155358238/gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux.tar.xz

$ tar xJf gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux.tar.xz

$ chmod 777 - R gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux

∗ For gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf

$ wget -c https://releases.linaro.org/archive/15.02/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf.tar.xzz

$ tar xJf gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf.tar.xz

$ chmod 777 - R gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf

(3) Configure mkspec file for ARM of Qt-everywhere

$ vi qt-everywhere-opensource-src-4.8.5/mksepcs/qws/linux-arm-gnueabi-g++/qmake.conf {use gedit instead of vi for graphical method}

∗ This is the original file content

#####################QMAKE.CONF##########################

#

qmake configuration for building with arm-none-linux-gnueabi-g++

#

include(../../common/linux.conf)

include(../../common/gcc-base-unix.conf)

include(../../common/g++-unix.conf)

include(../../common/qws.conf)

modifications to g++.conf

QMAKE_CC = arm-none-linux-gnueabihf-gcc

QMAKE_CXX = arm-none-linux-gnueabihf-g++

QMAKE_LINK = arm-none-linux-gnueabihf-g++

QMAKE_LINK_SHLIB = arm-none-inux-gnueabihf-g++

modifications to linux.conf

QMAKE_AR = arm-linux-gnueabihf-ar cqs

QMAKE_OBJCOPY = arm-linux-gnueabihf-objcopy∗

QMAKE_STRIP = arm-linux-gnueabihf-strip∗

load(qt_config)

################END_OF_QMAKE.CONF######################

∗ This will be the updated file content for gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf

#####################QMAKE.CONF##########################

#

qmake configuration for building with arm-none-linux-gnueabi-g++

#

include(../../common/linux.conf)

include(../../common/gcc-base-unix.conf)

include(../../common/g++-unix.conf)

include(../../common/qws.conf)

modifications to g++.conf

QMAKE_CC = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc

QMAKE_CXX = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++

QMAKE_LINK = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++

QMAKE_LINK_SHLIB = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++

modifications to linux.conf

QMAKE_AR = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-ar cqs

QMAKE_OBJCOPY = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-objcopy

QMAKE_STRIP = /home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-strip

include the compiled TSLIB library

QMAKE_INCDIR +=/usr/local/Crosstslib1.3SE4.9/include

QMAKE_LIBDIR +=/usr/local/Crosstslib1.3SE4.9/lib

flags to successfully compile Qt4.8.5 linked with TSLIB

QMAKE_LFLAGS += -Wl,-rpath-link=/usr/local/Crosstslib1.3SE4.9/lib

load(qt_config)

################END_OF_QMAKE.CONF##########

(4) Install packages needed to compile Qt-4.8.5-everywhere

∗ Follow http://doc.qt.io/qt-4.8/requirements-x11.html

$ sudo apt-get update

$ sudo apt-get install libxrender-dev libxrandr-dev libxcursor-dev libxfixes-dev libxinerama-dev libfontconfig-dev libfreetype6-dev libxi-dev libxt-dev libxext-dev libx11-dev libsm-dev libice-dev libglib2.0–0 libc6-amd64 libxft-dev

∗ Install package to utilize virtual framebuffer for Qt for 32-bit ARM

$ sudo apt-get install libxtst-dev

∗ Install additional packages to configure Qt for 32-bit ARM on a 64-bit Ubuntu 16.04 LTS

$ sudo apt-get install lib32stdc++6 zlib1g:i386

(5) Compile Qt-4.8.5-everywhere for Qt-4.8.5-arm

∗ Configure Qt-4.8.5-everywhere using the following command

$ cd qt-everywhere-opensource-src-4.8.5

$ ./configure -opensource -confirm-license -prefix /usr/local/Qt-4.8.5-arm -embedded arm -little-endian -no-pch -no-webkit -xplatform qws/linux-arm-gnueabi-g++ -qt-mouse-tslib -qt-kbd-tty -qt-kbd-linuxinput -qt-mouse-linuxinput

∗ Compile and install Qt-4.8.5-arm

∗∗ check the number of processors of your linux box

$ cat /proc/cupinfo | grep processor | wc -l

∗∗ use option “-j<n+1>”, where “n” is the number of processors

$ make -j5 ARCH=arm CROSS_COMPILE=/home/botjok/Documents/Qt/tmp/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-

$ make install

C. Configuration of Fresh BBB for Deployment

Using Windows system:

(1) Prepare the SD card

∗ Use Debian-sdcard-beaglebone-2015.02.27-LCD-cape-LCD7-v2.0.img file

∗ Follow http://www.waveshare.com/wiki/LCD_CAPE_(7inch)#Introduction

(2) Extract the system image file

(3) Format the SD card

∗ Use HPUSBDisk.exe to format the TF card. Note: the TF card capacity should be 4 GB or above!

∗ Choose the device as your TF card and File system as FAT32. Then, click “Start”.

(4) Write the system image file to the SD card

∗ Use Win32DiskImager.exe to write the system image

∗ Launch Win32DiskImager.exe and select the extracted system image. Then, click "Write".

(5) Insert the SD card into BeagleBone and boot

(6) Configure network properties

∗ Connect debugger cable to the board and use PUTTY.EXE to remotely login to BeagleBone Board or SSH to the BeagleBone Board using the IP address assigned from the router

On BeagleBone Black board:

(1) Expand the file system partition on a microSD

∗ Follow http://elinux.org/Beagleboard:Expanding_File_System_Partition_On_A_microSD

$ sudo su

fdisk /dev/mmcblk0

d

2

n

p

2

w

∗ Reboot -> (after reboot)

$ sudo resize2fs /dev/mmcblk0p2

(2) Disable boot to GUI

∗ Follow http://unix.stackexchange.com/questions/143855/disable-gui-on-beaglebone-black-running-debian

$ sudo vi/boot/uboot/uEnv.txt {or you can use gedit instead of vi}

∗ Update /boot/uboot/uEnv.txt file by appending the following lines

for BBB debian OS, this disables lightdm run from "/etc/init.d/lightdm"

commwent it to enable GUI

optargs=text

(3) Setup TSLIB

∗ Follow http://wiki.mentorel.com/doku.php/tslib

∗ Copy compiled TSLIB(CrosstslibSE) folder to host

∗ Copy contents of TSLIB/lib/∗.∗ to /usr/lib/

cp –R /home/debian/Qt/CrosstslibSE/lib/∗ /usr/lib/

∗ Copy contents of TSLIB/bin/∗.∗ to /usr/bin/

cp –R /home/debian/Qt/CrosstslibSE/bin/∗ /usr/bin/

∗ Update following environment settings

export TSLIB_CONSOLEDEVICE=none

export TSLIB_FBDEVICE=/dev/fb0 {the display device}

export TSLIB_TSDEVICE=/dev/input/event1 {the touchscreen device}

export TSLIB_CALIBFILE=/etc/pointercal

export TSLIB_CONFFILE=/etc/ts.conf

export TSLIB_PLUGINDIR=/usr/lib/ts

export QWS_MOUSE_PROTO = tslib:/dev/input/event1 {the mouse/touchscreen device}

∗ Run the ts_calibrate program to test TSLIB setup and calibrate the touch screen; it can only be run as ROOT

/usr/bin/ts_calibrate

(4) Setup Qt-4.8.5 library and plugins on the BBB

∗ Follow https://sites.google.com/site/timpicuc/Down home/beaglebone-black/setting-up-qt-for-bbb-with-debian

∗ create /opt/Qt-arm folder

mkdir /opt/Qt-arm

∗ copy Qt-4.8.5-arm/lib and Qt-4.8.5-arm/plugins folders to /opt/Qt-arm/

cp –R /home/debian/Qt/Qt-4.8.5-arm/lib /home/debian/Qt/Qt-4.8.5-arm/plugins/ /opt/Qt-arm/

∗ update LD_LIBRARY_PATH

export LD_LIBRARY_PATH=/opt/Qt-arm/lib

∗ export QT_QWS_FONTDIR path to solve font problem

export QT_QWS_FONTDIR=/opt/Qt-arm/lib/fonts

D. Configuration of Qt-Creator for Writing and Deploying the Program (Both Desktop and Cross-Compile Versions)

(1) Install Qt-Creator

∗ We use Qt-Creator-3.4.2 since it works for Qt-4.8.5

$ wget -c https://download.qt.io/official_releases/qtcreator/3.4/3.4.2/qt-creator-opensource-linux-x86_64-3.4.2.run

$ chmod +x qt-creator-opensource-linux-x86_64-3.4.2.run

$ sudo qt-creator-opensource-linux-x86_64-3.4.2.run

∗ Run Qt-Creator from the launcher or from the terminal

$/home/botjok/qtcreator-3.4.2/bin/qtcreator

(2) Configure Qt-Creator

* Configure "Build & Run"—"Qt Versions" option

(i) Go to "Tools"->"Options" menu.

(ii) Click "Build & Run" on the left window pane.

(iii) Select "Qt Versions" tab of the "Build & Run" window plane.

(iv) Select "Manual" option then click then "Add" button on the right. Browse to the location of the compiled Qt-for-embedded earlier, in our case it was "/usr/local/Qt-4.8.5-arm/bin/qmake." Press "Open" button or double-click on "qmake" file in the browser window.

** there will be warning: "No compiler can produce code for this Qt version. Please define one or more compilers."

(v) Leave "Version Name" as it is and then Click "Apply" button.

* Configure "Build & Run"—"Compilers" option

(i) Go to "Tools"->"Options" menu.

(ii) Click "Build & Run" on the left window pane.

(iii) Select "Compilers" tab of the "Build & Run" window plane.

(iv) Click "Add"->"GCC."

(v) Select "Browse" and then go to the location of "/home/botjok/Documents/Qt/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++". Press "Open" button or double-click the "arm-linux-gnueabihf-g++" file.

(vi) Update "Name" to "armGCC".

(vii) Click the "Apply" or "Ok" button.

* Configure "Build & Run"—"Debuggers" option

(i) Go to "Tools"->"Options" menu.

(ii) Click "Build & Run" on the left window pane.

(iii) Select "Debuggers" tab of the "Build & Run" window plane.

(iv) Click "Add."

(v) Select "Browse" and then go to the location of "/home/botjok/Documents/Qt/gcc-linaro-4.9-2015.02-3-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gdb." Press "Open" button or double-click the "arm-linux-gnueabihf-gdb" file.

(vi) Update "Name" to "armGDB."

(vii) Click the "Apply" or "Ok" button.

* Configure "Devices" option

(i) Go to "Tools"->"Options" menu.

(ii) Click "Devices" on the left window pane.

(iii) Click "Add", select "Generic Linux Device", and then click "Start Wizard."

(iv) Fill up the information in the "New Generic Linux Device Configuration Setup" window.

** Name: "BeagleBoneBlack03"

** Device hostname/IP Address: "beaglebone03" or "192.168.0.93" {hostname set or IP address assigned)

** Username: "debian" {userID to login to the device}

** Authentication type:

**** Choose "Password" if using password authentication, encode Password: "********

" {password for userID}

**** Choose "Key" if choosing token key, then select "Browse" button to select the private key file

(v) Click "Next."

(vi) Click the "Finish" button.

** The connection to the device will be tested; press the "Close" button.

(i) Click "Apply" or "Ok."

* Configure "Build & Run"—"Kits" option

(i) Go to "Tools"->"Options" menu.

(ii) Click "Build & Run" on the left window pane.

(iii) Select "Kits" tab of the "Build & Run" window plane.

(iv) Click "Add."

(v) Fill up the following information

** Name: "BeagleBoneBlack"

** File system name: "" {leave blank}

** Device type: "Generic Linux Type"

** Device: "BeagleBoneBlack03" {select the one we created earlier, Click "Manage" to create/update device}

∗∗ Sysroot: "" {leave blank}

∗∗ Compiler: "armGCC" {select the one we created earlier, click "Manage" to create/update compiler}

∗∗ Environment: {leave it as it is}

∗∗ Debugger: "armGDB" {select the one we created earlier, click "Manage" to create/update debugger}

∗∗ Qt version: "Qt 4.8.5 (Qt-4.8.5-arm)" {select the one we created earlier, click "Manage" to create/update Qt version}

∗∗ Qt mkspec: "" {leave it as it is}

(vi) Press the "Apply" or "Ok" button.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research is supported by the Brain Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, & Future Planning (Grant no. NRF-2017M3C7A1044815). This was also supported by the EUREKA project (Korea Institute for Advancement of Technology and Ministry of Trade, Industry & Energy with EUREKA Project ID 10 735).

References

[1] T. Yairi, K. Hori, and S. Nakasuka, "Sensor space discretization in autonomous agent based on entropy minimization of behavior outcomes," in *Proceedings. 1999 IEEE/SICE/RSJ. International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI'99 (Cat. No.99TH8480)*, pp. 111–116, Taipei, Taiwan, August 1999.

[2] M. Joerger and B. Pervan, "Measurement-level integration of carrier-phase GPS and laser-scanner for outdoor ground vehicle navigation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 131, no. 2, article 021004, 2009.

[3] F. Samadzadegan and G. Abdi, "Autonomous navigation of unmanned aerial vehicles based on multi-sensor data fusion," in *2012 20th Iranian Conference on Electrical Engineering (ICEE)*, pp. 868–873, Tehran, Iran, May 2012.

[4] M. A. A. Akhoundi and E. Valavi, *Multi-Sensor Fuzzy Data Fusion Using Sensors with Different Characteristics*, ArXiv Prepr. ArXiv10106096, 2010.

[5] B. Li, C. Rizos, H. K. Lee, and H. K. Lee, "A GPS-slaved time synchronization system for hybrid navigation," *GPS Solutions*, vol. 10, no. 3, pp. 207–217, 2006.

[6] Y. L. Zhang, J. H. Park, N. O. Sel, and K. T. Chong, "Robot navigation using a DR/GPS data fusion," *Applied Mechanics and Materials*, vol. 392, pp. 261–266, 2013.

[7] Z. Ercan, V. Sezer, H. Heceoglu et al., "Multi-sensor data fusion of DCM based orientation estimation for land vehicles," in *2011 IEEE International Conference on Mechatronics (ICM)*, pp. 672–677, Istanbul, Turkey, April 2011.

[8] E. Borràs, J. Ferré, R. Boqué, M. Mestres, L. Aceña, and O. Busto, "Data fusion methodologies for food and beverage authentication and quality assessment – a review," *Analytica Chimica Acta*, vol. 891, pp. 1–14, 2015.

[9] E. Cardarelli, L. Sabattini, C. Secchi, and C. Fantuzzi, "Multisensor data fusion for obstacle detection in automated factory logistics," in *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 221–226, Cluj Napoca, Romania, September 2014.

[10] J. A. Duro, J. A. Padget, C. R. Bowen, H. A. Kim, and A. Nassehi, "Multi-sensor data fusion framework for CNC machining monitoring," *Mechanical Systems and Signal Processing*, vol. 66-67, pp. 505–520, 2016.

[11] C. Chen, R. Jafari, and N. Kehtarnavaz, "A real-time human action recognition system using depth and inertial sensor fusion," *IEEE Sensors Journal*, vol. 16, no. 3, pp. 773–781, 2016.

[12] A. Colombo, D. Fontanelli, D. Macii, and L. Palopoli, "Flexible indoor localization and tracking based on a wearable platform and sensor data fusion," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 4, pp. 864–876, 2014.

[13] Y. Dobrev, S. Flores, and M. Vossiek, "Multi-modal sensor fusion for indoor mobile robot pose estimation," in *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 553–556, Savannah, GA, USA, April 2016.

[14] A. Simonetti, D. Accardo, D. S. Domenico, and P. Calcagni, *Developing an Attitude and Heading Reference System Based on Advanced MEMS Gyros*, AIAA Infotech@ Aerospace, 2015.

[15] A. L. Bustamante, J. M. M. López, and J. G. Herrero, "Player: an open source tool to simulate complex maritime environments to evaluate data fusion performance," *Simulation Modelling Practice and Theory*, vol. 76, pp. 3–21, 2017.

[16] A. Trzuskowsky, C. Hoelper, and D. Abel, "ANCHOR: navigation, routing and collision warning during operations in harbors," *IFAC-PapersOnLine*, vol. 49, no. 23, pp. 220–225, 2016.

[17] W. C. Bruhn, H.-C. Burmeister, M. T. Long, and J. A. Moræus, "Conducting look-out on an unmanned vessel: Introduction to the advanced sensor module for MUNIN's autonomous dry bulk carrier," in *The 10th International Symposium ISIS 2014 "Integrated Ship's Information Systems"*, Hamburg, Germany, September 2014.

[18] A. L. Flåten and E. F. Brekke, "Performance prediction of tracking sensors for surface vehicle collision avoidance," in *2016 19th International Conference on Information Fusion (FUSION)*, pp. 1894–1900, Heidelberg, Germany, July 2016.

[19] D. Chen, C. Dai, X. Wan, and J. Mou, "A research on AIS-based embedded system for ship collision avoidance," in *2015 International Conference on Transportation Information and Safety (ICTIS)*, pp. 512–517, Wuhan, China, June 2015.

[20] P. Hu and C. Huang, "Shipborne high-accuracy heading determination method using INS- and GPS-based heading determination system," *GPS Solutions*, vol. 21, no. 3, pp. 1059–1068, 2017.

[21] J. C. Juang and C. F. Lin, "A sensor fusion scheme for the estimation of vehicular speed and heading angle," *IEEE*

Transactions on Vehicular Technology, vol. 64, no. 7, pp. 2773–2782, 2015.

[22] K. Jaroś, A. Witkowska, and R. Śmierzchalski, "Data fusion of GPS sensors using particle Kalman filter for ship dynamic positioning system," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pp. 89–94, Miedzyzdroje, Poland, August 2017.

[23] T. H. Bryne, *Nonlinear Observer Design for Aided Inertial Navigation of Ships*, Norges Teknisk-Naturvitenskapelige Universitet, 2017.

[24] J. M. Núñez, M. G. Araújo, and I. García-Tuñón, "Real-time telemetry system for monitoring motion of ships based on inertial sensors," *Sensors*, vol. 17, no. 5, p. 948, 2017.

[25] Q. Feng-de, W. Feng-wu, L. Jiang, and T. Guan-jun, "A method for the measurement of ship attitude based on multi-sensor data fusion," in *2015 Ninth International Conference on Frontier of Computer Science and Technology (FCST)*, pp. 196–199, Dalian, China, August 2015.

[26] F. Cappello, S. Ramasamy, R. Sabatini, and J. Liu, "Low-cost sensors based multi-sensor data fusion techniques for RPAS navigation and guidance," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 714–722, Denver, CO, USA, June 2015.

[27] F. Cappello, R. Sabatini, S. Ramasamy, and M. Marino, "Particle filter based multi-sensor data fusion techniques for RPAS navigation and guidance," in *2015 IEEE Metrology for Aerospace (MetroAeroSpace)*, pp. 395–400, Benevento, Italy, June 2015.

[28] F. Cappello, S. Ramasamy, and R. Sabatini, "Multi-sensor data fusion techniques for RPAS navigation and guidance," in *AIAC16: 16th Australian International Aerospace Congress*, pp. 64–72, Barton, ACT: Engineers Australia, May 2015.

[29] J. Choi, S. Kim, Y. Lee, T.-J. Kim, and H.-T. Choi, "Relative pose estimation of underwater robot by fusing inertial sensors and optical image," in *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 204–208, Kuala Lumpur, Malaysia, November 2014.

[30] J. L. Odom and J. L. Krolik, "Passive towed array shape estimation using heading and acoustic data," *IEEE Journal of Oceanic Engineering*, vol. 40, no. 2, pp. 465–474, 2015.

[31] K. L. Prasanna, S. K. Rao, B. O. L. Jagan, A. Jawahar, and S. B. Karishma, "Data fusion in underwater environment," *International Journal of Engineering and Technology*, vol. 8, no. 1, pp. 225–234, 2016.

[32] P. Scherz, A. Haderer, K. Pourvoyeur, and A. Stelzer, "Embedded sensor fusion system for unmanned vehicle navigation," in *2008 IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pp. 192–197, Beijing, China, October 2008.

[33] F. P. Vista IV, D.-J. Lee, and K. T. Chong, "Design of an EKF-CI based sensor fusion for robust heading estimation of marine vehicle," *International Journal of Precision Engineering and Manufacturing*, vol. 16, no. 2, pp. 403–407, 2015.

[34] US Army, *Advanced Map and Aerial Photograph Reading*, U.S. Army Headquarters, War Department, 1941.