Appears in: 2012 IEEE/ION Position Location and Navigation Symposium (PLANS), April 23-26, 2012, Myrtle Beach, SC, pp140-145

Foot-mounted INS for Everybody – An Open-source Embedded Implementation

John-Olof Nilsson[1], Isaac Skog[1], Peter Händel[1], and K.V.S. Hari[2]
[1]Signal Processing Lab, ACCESS Linnaeus Centre, KTH Royal Institute of Technology, Stockholm, Sweden
[2]Statistical Signal Processing Lab, Department of ECE, Indian Institute of Science (IISc), Bangalore, India

***Abstract*—We present an open-source, realtime, embedded implementation of a foot-mounted, zero-velocity-update-aided inertial navigation system. The implementation includes both hardware design and software, uses off-the-shelf components and assembly methods, and features a standard USB interface. The software is written in C and can easily be modified to run user implemented algorithms. The hardware design and the software are released under permissive open-source licenses and production files, source code, documentation, and further resources are available at www.openshoe.org. The reproduction cost for a single unit is below $800, with the inertial measurement unit making up the bulk ($700). The form factor of the implementation is small enough for it to be integrated in the sole of a shoe. A performance evaluation of the system shows a position errors for short trajectories (<100 [m]) of ± 0.2-1 % of the traveled distance, depending on the shape of trajectory.**

I. Introduction

Foot-mounted inertial navigation is not rocket science and the path from a theoretical algorithm to a realtime embedded implementation might seem deceptively short. However, our experience is that the difficulties of developing a well-performing embedded foot-mounted inertial navigation system (INS), yet surmountable, are easily underestimated. The number of software and hardware components that need to work together is large enough such that getting them to work in harmony is not trivial. Further, obtaining sufficient computational power and versatility on an embedded processor and sufficiently low computational cost of the filter algorithms require care in platform selection and algorithm implementation. Moreover, attaining a form factor and mechanical durability, such that the system can be integrated into the sole of a shoe and still be able to use off-the-shelf components and easily available construction methods, is challenging. Few solutions exist on the market or in the literature with even fewer solutions that provide the documentation and modifiability desired by researchers.

Therefore, to give researchers, teachers, and system designers a shortcut to a working foot-mounted INS implementation suitable for further research, education, and rapid prototyping, and usable as a component in larger pedestrian navigation systems, we present an open-source, embedded, foot-mounted INS implementation containing both hardware design and software/algorithm implementations. Our hope is that such an implementation will save time, sweat, and tears for navigation researchers as well as facilitate the use of the technology by researchers not specialized in aided INS, e.g. in fields such as biomedical engineering, behavioral science, and ubiquitous computing. The value of the embedded implementation also lies in its modularity and in its small weight, bulk, and price in comparison with the typical sensor-plus-laptop research systems. These properties alleviate the work of integrating the foot-mounted INS in larger realtime navigation systems, and make it feasible to equip a larger number of users with foot-mounted INS units for field performance tests and cooperative navigation studies.

Fig. 1: Shoes with units of the presented foot-mounted INS implementation integrated in the heels and cabling with USB connectors at the shoe shafts. The cabling inside the shoes is normally protected by a leather patch but is here displayed for clarity.

The presented system is a zero-velocity-update (ZUPT)-aided INS built from off-the-shelf components and with a standard USB interface. The filtering is implemented on a microcontroller (μC) fitted with an OEM inertial measurement unit (IMU) in a casing, which, in turn, is integrated in the sole of a shoe. Figure 1 shows a pair of shoes with the system units integrated in the heel. The system can be configured to run any user implemented algorithm executable in $\sim 10^5$ clock cycles per time update. The software and hardware design are released under permissive open-source licenses. Available resources include ready-to-compile, as well as pre-compiled source code, hardware production files, a Matlab system interface, and documentation, with everything from electronic schematics and CAD-drawings of casing components to code documentation and installation instructions. All resources can be downloaded from **www.openshoe.org**.

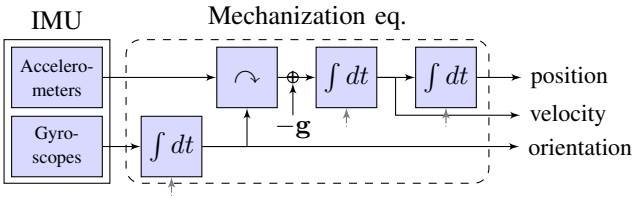


Fig. 2: Block diagram of the INS and the internal filtering. The gray arrow indicates the insertion point of correction from the feedback of Fig. 3.

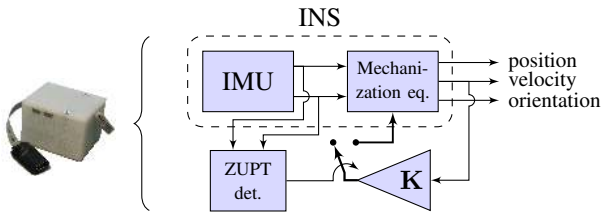


Fig. 3: Block diagram of the ZUPT-aided INS. The zero-velocity detector (2) switches the feedback on and off. The feedback will affect all INS states since they are coupled with the velocity error. The gain $\mathbf{K}$ of the feedback is given by the Kalman gain.

II. Foot-Mounted INS

A foot-mounted INS is simply an INS mounted on the foot combined with additional filtering that improves the accuracy of the INS by using properties of the motion related to the IMU mounting point (the foot), i.e. regularly reoccurring stationary periods. For a tutorial introduction to foot-mounted INS see [1]. The principles of inertial navigation are simple. The derivative of the position is the velocity and the derivative of the velocity is the acceleration. Consequently, starting at stationary, integrating the acceleration once and twice renders the velocity and the change in position, respectively. To obtain the acceleration in the navigation coordinate frame, the measurement vector from the 3-axis accelerometer must first be transformed to the navigation coordinate frame, and thereafter have the gravity acceleration component subtracted from it. This is made possible by a 3-axis gyroscope, which measures the rotational rate of the system. Integrating the rotational rate gives the relative orientation of the IMU, which can be used to transform the accelerometer measurements. Together, this gives the position, the velocity, and the orientation of the IMU. A block diagram in Fig. 2 illustrates the basic filtering. A discretization (first order) of the continuous block diagram gives

$$\begin{bmatrix}\mathbf{x}_k\\ \mathbf{v}_k\\ \mathbf{q}_k\end{bmatrix} = \begin{bmatrix}\mathbf{x}_{k-1} + \mathbf{v}_{k-1}dt_k\\ \mathbf{v}_{k-1} + (\mathbf{q}_{k-1}\mathbf{f}_k\mathbf{q}_{k-1}^{-1} - \mathbf{g})dt_k\\ \boldsymbol{\Omega}(\boldsymbol{\omega}_k dt_k)\mathbf{q}_{k-1}\end{bmatrix} \tag{1}$$

where k is a time index, dt_k is the time difference between measurement instants, $\mathbf{x}_k$ is the position, $\mathbf{v}_k$ is the velocity, and $\mathbf{q}_k$ is the quaternion describing the orientation of the system relative to the navigation coordinate frame, $\mathbf{f}_k$ is the accelerometer measurements, $\mathbf{g}$ is the gravity, $\boldsymbol{\omega}_k$ is the gyroscope measurements (all in 3 dimensions), and $\boldsymbol{\Omega}(\cdot)$ is the quaternion update matrix. The IMU together with the filtering

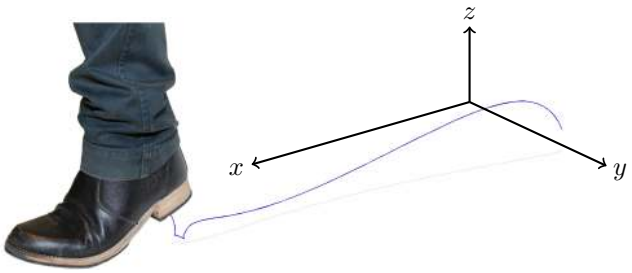


Fig. 4: Close-up of the position tracking of a single step given by the foot-mounted INS of Fig. 3. The system is mounted in the heel. The small discontinuity below the heel is the correction (ZUPT) provided by the feedback.

(1) constitute the INS. For a detailed treatment of inertial navigation see [2].

Correctly initialized, the INS can give state estimates for all future times. Unfortunately, for low-cost sensors, the quality of the estimates deteriorates swiftly. Due to the integrations in the inertial navigation, small measurement errors accumulate. For the microelectromechanical system (MEMS) type of inertial sensors which are commonly used for foot-mounted inertial navigation, this renders the position estimates useless after $\sim 10\,[\mathrm{s}]$. However, the fact that the foot-mounted IMU will return to stationary at regular intervals can be exploited to circumvent this. The stationarity can be detected from the inertial measurements by hypothesis testing [3]. The system is considered stationary at time instant k if

$$\frac{1}{N}\sum_{\ell\in W_k}\left(\frac{1}{\sigma_f}\left\|\mathbf{f}_\ell - \mathbf{g}\frac{\bar{\mathbf{f}}_k}{\|\bar{\mathbf{f}}_\ell\|}\right\| + \frac{1}{\sigma_\omega}\|\boldsymbol{\omega}_\ell\|^2\right) < \gamma, \tag{2}$$

where $\|\cdot\|$ is the 2-norm, $\bar{\mathbf{f}}_k$ is the mean accelerometer measurements over the time window W_k of length N samples centered around k, σ_f and σ_ω are measurement error standard deviations, and γ is a zero-velocity detection threshold. If the detector (2) has declared the system stationary, the INS should give a zero-velocity estimate. Due to the accumulated errors it most likely will not. This discrepancy can be used to correct the INS, giving the so-called ZUPTs, which greatly increases the quality of the estimates. A Kalman filter feedback (correction) appears as

$$\begin{bmatrix}\mathbf{x}_k\\ \mathbf{v}_k\\ d\boldsymbol{\theta}_k\end{bmatrix} \Leftarrow \begin{bmatrix}\mathbf{x}_k\\ \mathbf{v}_k\\ 0\end{bmatrix} + \mathbf{K}_k\mathbf{v}_k \quad \text{and} \quad \mathbf{q}_k \Leftarrow \boldsymbol{\Omega}(d\boldsymbol{\theta}_k)\mathbf{q}_k, \tag{3}$$

where $\mathbf{K}_k$ is the Kalman gain, and $d\boldsymbol{\theta}_k$ is the correction in orientation. The block diagram of Fig. 3 illustrates the ZUPT-aiding of the INS. For a detailed treatment of aided INS see [4].

Equations (1)-(3) constitute a minimal filtering for a foot-mounted INS and are the formulation we used for the presented implementation. As seen in Fig. 4, this gives an accurate position tracking of the foot with [cm] to [mm] accuracy over a step. The behavior of the system over longer trajectories is presented in Section V. Note that the aiding works for all types of motions (not just walking), but for the system to perform

well, it should return to stationary with intervals of ~ 2 [s] or less.

III. Embedded implementation

For an embedded, foot-mounted INS implementation, the hardware required is: an IMU that provides the inertial measurements, an embedded processing platform that provides the computation capability and external hardware interfaces, and a shoe or equivalent that provides a mounting point for the system. In addition, software implementations of (1)-(3) and other auxiliary functions for initialization and system control are needed. The hardware, software, and footware of the current implementation are described in the following three subsections.

A. Hardware

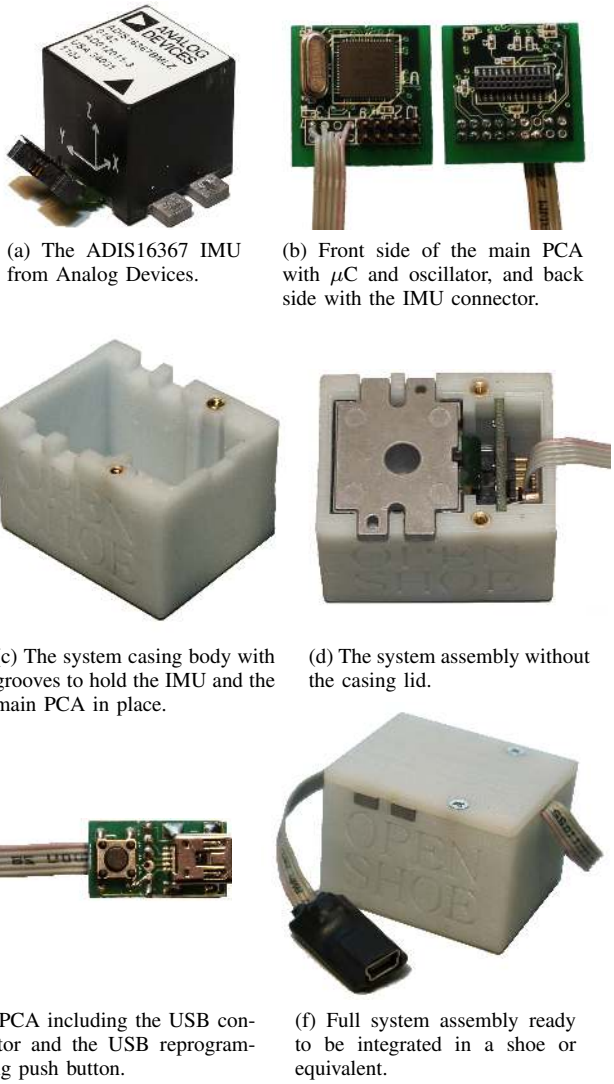
The main hardware components of the system is the IMU, a printed circuit assembly (PCA) with the μC and auxiliary components, and a casing. In addition, there is cabling and an external USB connector on another small PCA. Figure 5 shows the hardware components and assemblies.

An IMU contains three orthogonally mounted accelerometers and gyroscopes. The IMU of the implementation is the ADIS16367 *i*Sensor from Analog Devices seen in Fig. 5a. The dynamic range of the accelerometers is ± 18 [g] and for the gyroscopes, it is $\pm 1200\,[^\circ/s]$. The bandwidth and sample rate of the IMU is 330 [Hz] and 820 [Hz], respectively. The IMU has a standard serial peripheral interface (SPI). The casing of the IMU, with its 23.5 [mm] sides, limits the minimum height of the system assembly.

The IMU is connected to the main PCA shown in Fig. 5b. The PCA is based on a 23×23 [mm] 4-layer printed circuit board (PCB). The main component of the PCA is a 32-bit AT32UC3C2512 (QFN64 package) μC from Atmel. The μC provides hardware floating point arithmetic, facilitating the implementation of many filtering algorithms, and is clocked at 42 [MHz] (clocking frequencies up to 66 [MHz] are supported). The μC has a built-in USB slave device controller, which means that it can work as a USB slave device without any additional components; USB OTG is also possible to implement. In addition to the μC, the PCA contains connectors, an oscillator, and decoupling capacitors. A JTAG interface to the μC is available on the PCA. An external oscillator is needed since the internal oscillators have too low quality for the INS. A systematic errors in the time differentials dt_k of a fraction of a percent gives significant scale errors in the position estimates. Note that the IMU has interface compatibility with all IMUs in the *i*Sensor serie. Hence, a side-effect of the implementation is that the main PCA constitute an open-source USB interface to the *i*Sensor IMU serie.

The IMU and the main PCA are enclosed in a plastic casing shown in Fig. 5c. Grooves in the casing and the lid hold the IMU and the PCA in place. Figure 5d shows the assembly of the casing body, the IMU, and the main PCA.

A 5-wire ribbon cable, carrying the USB pins and an additional pin for USB reprogramming, is soldered directly to the main PCB. The other end of the ribbon cable is soldered directly in a small additional PCB, shown in Fig. 5e. This PCB features a mini-USB connector and a push-button for the reprogramming pin. Potentially, the USB connector could be mounted in the system casing but the permanently mounted cable allow integrations of the system in which the main system assembly is not directly accessible.



(a) The ADIS16367 IMU from Analog Devices.

(b) Front side of the main PCA with μC and oscillator, and back side with the IMU connector.

(c) The system casing body with grooves to hold the IMU and the main PCA in place.

(d) The system assembly without the casing lid.

(e) PCA including the USB connector and the USB reprogramming push button.

(f) Full system assembly ready to be integrated in a shoe or equivalent.

Fig. 5: Main hardware components and assemblies of the presented foot-mounted INS implementation. The dimensions of the full assembly are $28.5 \times 32 \times 40.5$ [mm].

A full system assembly is shown in Fig. 5f. The dimensions of the full system assembly are $28.5 \times 32 \times 40.5$ [mm]. The system assembly is powered directly via the USB *vbus* pin. Optionally a battery can be added. The power consumption is < 0.75 [W] (< 150 [mA] at 5 [V]).

B. Software

The μC software consists of a *runtime framework*, *software communication interfaces*, and *filter algorithm implementations*. The software has been written in C, and compiled and tested with GCC and the *avr32gcc* compiler utility. The

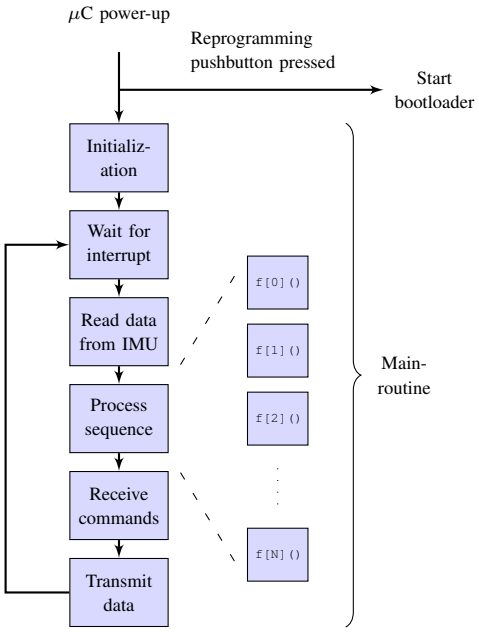


Fig. 6: Flow chart of the runtime framework. The runtime framework runs in an infinite loop that reads data from the IMU, cycles through the process sequence, and receives commands from and transmits data to, the user.

Atmel Software Framework (ASF), which is openly available from Atmel, provides many of the low-level functionalities. The μC program memory contains a USB-bootloader, which means that the system can be reprogrammed via the USB interface. The programming mode is entered by pressing the reprogramming push button (see Fig. 5e) on power-up.

The runtime framework contains the main-routine and is directly or indirectly responsible for calling all other runtime routines. Figure 6 shows a flow chart of the runtime framework and its components. The runtime framework calls initializations routines after which it enters an infinite loop in which it: waits for an interrupt, read data from the IMU, runs through a process sequence, and receives commands from and sends data to the user. The initialization routines setup and configure necessary μC components. After power up, the IMU regularly sends interrupts, signaling that new inertial measurements are available. These are the interrupts that the runtime framework is waiting for. The process sequence is a dynamic sequence (pointer array) of arbitrary filtering functions (function pointers). Functions can be inserted or removed from the process sequence based on user commands (by command response functions) or by functions in the sequence itself. This way the runtime framework can be configured to run user implemented algorithms. During normal operations, this sequence contains the functions for the ZUPT-aided INS, i.e. implementations of equations (1)-(3). Upon receiving data from the user, the runtime framework parses them, and executes command response functions. All valid commands are associated with a response function, which is executed on the μC. The transmitted data can be any state. States can be transmitted a single time, continuously with the rate determined by a rate divider, or based on some condition detected in the process sequence. The commands and the transmitted data have standard formats with a header, payload, and a checksum. Following the transmit stage, the system listens for the next interrupt.

The software communication interface provides the means of communicating between the IMU and the μC and between the μC and a user platform (USB host). The ASF provides low-level SPI and USB functionalities while higher-level functionalities, such as commands and command level parsing and responses, are encoded in the communication interface. The USB device controller in the μC is configured to appear as a virtual serial port (CDC device).

The filter algorithm implementations are the processing algorithms corresponding to (1)-(3) together with auxiliary functions. With an update rate of 820 [Hz] and a clock frequency of 42 [MHz], the filtering is limited to $\sim 5 \cdot 10^4$ clock cycles. However, this can be tweaked by reducing the update rate and increasing the clock frequency. The current implementation of (1)-(3) executes within $\sim 2 \cdot 10^4$ clock cycles.

The system has 3 main modes of operation: 1) For data collection, it can work as a pure IMU. Low-pass filters can be added, and the output downsampled from the maximum sampling rate of 820 [Hz] to a desired output rate. 2) For stand-alone navigation, the system can work as a stand-alone ZUPT-aided INS (or run any user defined algorithms). Any system states can be output in realtime up to 820 [Hz] or based on user defined conditions. 3) For integration in a larger pedestrian navigation system, it can work as a displacement and heading change sensor. In short, this means that the system transmit relative displacement and heading changes, together with error covariances, for each step.

C. Footware

Although the shoes we have used were especially made for the system, there is nothing special about them except for the casing and cabling integration. The custom made shoes were made possible at a reasonable cost ($\sim$\$30) by the Indian partner in the project. However, any standard shoe with a carved out hole in the sole will do. Note that the mounting point is important. We stress that, as exemplified in Fig. 1, *the system should be integrated in the sole to get as close as possible to the contact surface between the user and the ground*. If the system is attached to the uppers of the shoes, system parameters need to be changed and the performance will be worse. Since the system has a permanently mounted extension cable and can be programmed via the USB, the units themselves do not need to be easily accessible. Note that the system assembly is strong enough to support the weight of a user.

IV. System Reproduction and Modification

All software and hardware design are licensed under permissive open-source licenses and can be freely used, copied, modified, integrated, and/or redistributed. Reproducing the

implementation only requires basic engineering skills. For reproduction, the following four steps have to be done:

1) Acquire an IMU.
2) Have the PCAs produced by a PCB print and population service.
3) Have the casing printed by a rapid prototyping service.
4) Connect the different parts and program the μC.

After completing these steps, the system will be ready for use. Optionally, one may chose to print and populate the PCBs oneself. However, this requires some special equipment and soldering skills since currently the implementation is based on a 4-layer PCB design and the components are surface mounted. The reproduction cost is less than $800, with the IMU making up the bulk of the cost ($700), and the printing and population of the PCBs and printing of the casing making up the rest.

Likewise, modifying the system is simple. PCB and casing designs are available in standard formats and can readily be changed to suit ones needs. The software is documented with tagged comments (Doxygen), and the code is available together with AVR Studio 5 project files, making linking with new user-implemented algorithms straight forward. The procedure to implement and run new algorithms is as follows:

1) Divide the algorithm into a suitable set of processing functions. The functions can take no arguments and most therefore use global variables (states) to communicate.
2) Implement the functions in C and configure the runtime environment to recognize (link with) the functions.
3) Setup a new command with a response function which inserts the new functions into the process sequence.
4) Compile and link the code for the AVR32 platform and program the μC.
5) Power up the system and send the newly defined command over the USB.

If needed, new state variables can be setup in the runtime environments. Note that the inertial measurements are available for the processing functions as state variables.

V. Performance Evaluation

The nonlinear and unstable nature of a foot-mounted INS filtering makes a general performance analysis difficult. The performance is dependent on the true trajectory in a non-trivial way. The main obstacle is that the position errors are strongly coupled with the heading errors via the true (relative) position. A heading error of $0.5°$ gives a relative position error of 1% of the traveled end-to-end point distance. However, if the user then walks back the same distance, the position errors cancel out. Scale errors are canceled out likewise. Two extreme trajectory types can be identified: a straight-line trajectory in which the heading error couples the strongest with the position error; and a closed-loop symmetric trajectory in which the heading and other trajectory induced errors largely cancel themselves out. By studying the errors in such trajectories, we can get a rough separation of the position errors induced by the heading errors and the other error sources.

In Fig. 7, 100 straight-line and 100 figure-of-eight trajectories are shown. Close-ups of the final position estimates of

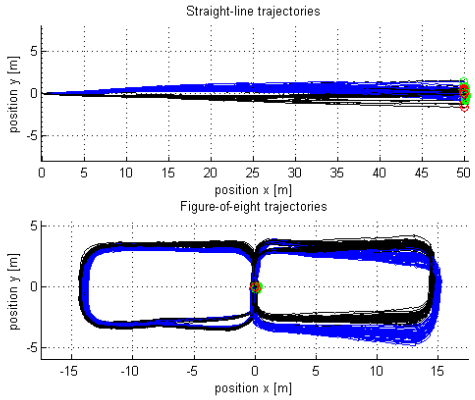


Fig. 7: The straight-line and the figure-of-eight evaluation trajectories. Blue lines correspond to right foot trajectories and black lines correspond to left foot trajectories. Close up of the final positions are found in Figs. 8 and 9.

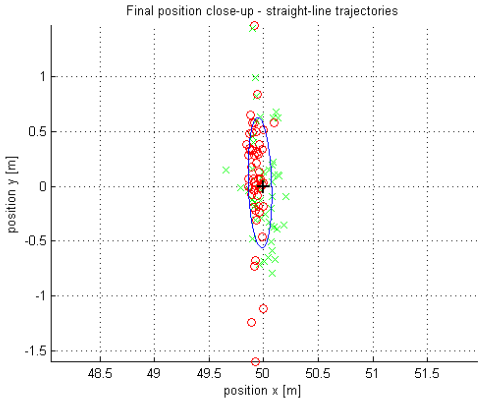


Fig. 8: Scatter plots of the position errors at the end of the straight-line trajectories shown in Fig. 7. Green crosses correspond to right foot end positions and red circles correspond to left foot end positions. The black cross indicates the reference final position.

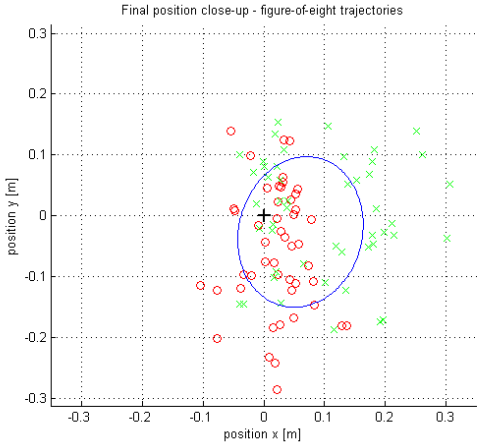


Fig. 9: Scatter plots of the position errors at the end of the figure-of-eight trajectories shown in Fig. 7. Green crosses correspond to right foot end positions and red circles correspond to left foot end positions. The black cross indicates the reference final position.

the two different types of trajectories together with 1σ error covariance ellipsoids are shown in Figs. 8 and 9. The trajectories are recorded from the right (blue) and the left (black) foot with two different IMUs (four IMUs in total). The gyroscopes were calibrated prior to recording the trajectories but not between individual trajectory measurements. The trajectories were taken over a period of roughly 1 hour for each IMU pair. The conditions of the trajectories are normal walking speed (~ 1.6 [m/s]) on a solid floor. The system works fine for running motion (not displayed). Running motion even gives a somewhat better performance for the same trajectories. This is due to the fact that less gyro measurement errors accumulate since the travel times of the trajectories become shorter. For a closed-loop trajectory, the initial heading alignment errors cancel out, but for a straight-line trajectory they will not. Therefore, the mean of the final positions for the straight-line trajectories was used to calculate the initial orientation. A plate with imprints of the shoes was used to get the same initial orientation for each trajectory. The length of the straight-line trajectory was set to 50 [m] with a Class III measurement tape ($<$2 [cm] error on 50 [m]).

The Figs. 7, 8, and 9 are intended to give an idea of the behavior and performance of the system. A detailed performance analysis is beyond the scope of this article. However, it can be noted that the errors perpendicular to the trajectory for the 50 [m] straight-line trajectory is around ± 0.5 [m]. This corresponds to 1% of the traveled distance. This is probably a combination of initial heading alignment errors, integrated gyroscope measurement errors, and filter induced errors. We expect to have an accuracy of approx. ± 1 [mm], between heel and toe of the shoe, of the initial alignment. This corresponds to ± 0.15 [m] induced error over 50[m] ($\pm 0.3\%$) and consequently the rest is judged to be due to measurement and filter induced errors. The errors along the straight-line trajectory (distance errors) is around ± 0.15 [m] corresponding to $\pm 0.3\%$ of the traveled distance. The closed-loop errors on the figure-of-eight trajectory is around ± 0.15 [m] corresponding to $\pm 0.2\%$ of the traveled distance (approx. 80 [m] long trajectory). Since the systematic errors are consistent between different IMUs, they are most likely caused by couplings to the trajectory itself. We can get a verification of the cancelation of errors in the closed-loop trajectory by noting that the variance in the length direction of the straight line trajectory is roughly the same as the variance in the symmetric closed-loop case. Position errors in the height direction (not displayed) are also around $\pm 0.3\%$ of the traveled distance.

VI. Conclusions

We have presented an open-source, embedded, foot-mounted INS implementation containing both hardware design and software. The system can easily be reproduced and used without deeper understanding of the technology. Due to the openly available software and hardware design, the implementation sets a reference for the field. Evaluation of the system performance shows that the system has heading related errors in the order of percent and residual errors in the order of tenth of a percent of the traveled distance. Consequently, the heading related errors dominate the system errors. The focus of the implementation has been simplicity. However, the performance results are in line or even better than many results published in the literature.

Acknowledgment

We would like to acknowledge Syam Krishnan (ECE, IISc), who helped us with the main PCB design, and Afzal Ameen (APDAP, IISc), who helped us with review and production of the casings.

Parts of this work have been funded by The Swedish Governmental Agency for Innovation Systems (VINNOVA) and Department of Science and Technology (DST), Government of India.

References

[1] C. Fischer, P. T. Sukumar, and M. Hazas, "Tutorial: implementation of a pedestrian tracker using foot-mounted inertial sensors," *IEEE Pervasive Computing*, vol. 99, no. PrePrints, 2012.

[2] C. Jekeli, *Inertial Navigation Systems with Geodetic Applications*. de Gruyter, 2001.

[3] I. Skog, P. Händel, J.-O. Nilsson, and J. Rantakokko, "Zero-velocity detection: An algorithm evaluation," *Biomedical Engineering, IEEE Transactions on*, vol. 57, pp. 2657 –2666, nov. 2010.

[4] J. A. Farrell, *Aided Navigation*. Mc Graw Hill, 2008.