1-1-2011

Development Of Wearable Sensors For Body Joint Angle Measurement

Saba Bakhshi Khayani

University of Denver

Follow this and additional works at: https://digitalcommons.du.edu/etd

Recommended Citation
https://digitalcommons.du.edu/etd/49

This Thesis is brought to you for free and open access by the Graduate Studies at Digital Commons @ DU. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ DU. For more information, please contact jennifer.cox@du.edu.
DEVELOPMENT OF WEARABLE SENSORS FOR BODY JOINT ANGLE MEASUREMENT

A Thesis
Presented to
The Faculty of Engineering and Computer Science
University of Denver

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Saba Bakhshi Khayani
June 2011
Advisor: Dr. Mohammad H. Mahoor
Abstract

This thesis presents the development of three different methods for body joint angle measurement using wearable sensors. Continuous monitoring of patients’ movements and activities has recently become one of the active research areas in the field of body sensor network and telehealth monitoring. For many medical and rehabilitation applications, a continuous monitor of the patients’ daily activities at home without visiting the hospital is desirable. This type of monitoring is beneficial for the therapists and physicians as it does not require patients’ physical presence. Traditionally, measuring the range of motion (ROM) is performed in hospitals by utilizing standard tools such as a goniometer. This method needs to be fulfilled by a physiotherapist in the hospital and requires great deal of overhead. Thus, a remote sensing technique for monitoring the progress of body joint flexion during regular daily life activities becomes very beneficial.

The main focus of this thesis is on developing different methods for sensing body joint angle, although we developed some mechanisms for transmitting measurements from patient’s side to a remote server at the hospital. The first method for measuring the joint angle- specifically the knee joint- is based on an encoder attached to a brace. The second method is performed by utilizing a wearable cloth with flex-sensors. In the third method, inertial measurement units (IMUs) are employed to measure the desired joint angle. We conducted several experiments to compare the feasibility and accuracy of each method for angle measurement with ground truth measurements. The advantages
and disadvantages of each approach are discussed and explained in detail in the assigned section.
Acknowledgements

Words fall short to convey my gratitude to those who helped and supported me during my Master studies. I would like to take this opportunity to express my individualized thanks to all of those.

My special thanks and heartfelt gratitude go to my academic advisor Dr. Mahoor for his invaluable advice, patience, guidance, and consistent support throughout my studies. The accomplishment of this work would have been impossible without his help.

I would like to extend my sincere appreciation to the committee members: Dr. Mohammad Matin and Dr. Bradley Davidson for their valuable time and thoughtful comments. I would like to thank the Department of Electrical and Computer Engineering of the University of Denver for its financial support and providing a friendly and progressive work atmosphere. Furthermore, I would like to thank my officemates and friends for their encouragement and support, especially Babak Tousifar, Mu Zhou, Alejandro Gutierrez, Goncalo Fernandes Pereira Martins, Xiaoting Yang, Xin Li, and Robert Nawrocki for helping me during my experimental tests.
# Table of Contents

Acknowledgements ............................................................................................................. iv

Chapter One: Introduction .................................................................................................. 1  
1.1. Motivation ................................................................................................................. 1  
1.2. Knee Rehabilitation and Remote Monitoring ......................................................... 4  
1.3. Thesis Contributions .............................................................................................. 6  
1.4. Thesis Organization ............................................................................................... 6  

Chapter Two: Previous Work ............................................................................................ 7  
2.1. Wearable Cloth ....................................................................................................... 7  
2.2. Joint Angle Measurement using Accelerometers .................................................. 11  
2.3. Force and MARG sensor ...................................................................................... 13  

Chapter Three: Knee Joint Angle Measurement Using Encoder .................................... 16  
3.1. Introduction .......................................................................................................... 16  
3.2. Method .................................................................................................................. 16  
3.3. Experimental Result .............................................................................................. 19  

Chapter Four: Joint Angle measurement Using Flex-Sensor ........................................ 21  
4.1. Introduction .......................................................................................................... 21  
4.2. Flex-Sensor and Supportive Cloth ....................................................................... 22  
4.3. Angle Estimation Using Extended Kalman Filter .............................................. 23  
4.2. Experimental Result .............................................................................................. 30  

Chapter Five: Joint Angle Measurement Using IMU ...................................................... 33  
5.1. Introduction .......................................................................................................... 33  
5.2. Method .................................................................................................................. 33  
5.3. Experimental Result .............................................................................................. 38  

Chapter Six: Conclusion and Future Work ................................................................. 42  
6.1. Conclusion .......................................................................................................... 42  
6.2. Future Work ......................................................................................................... 43  

References ........................................................................................................................ 45  

Appendix A ....................................................................................................................... 47  
A.1. Microcontroller Schematic .................................................................................. 47  
A.2. Microcontroller Code .......................................................................................... 48  

Appendix B ....................................................................................................................... 50  
B.1. Kalman Filtering Code ......................................................................................... 50  
B.2. Microcontroller Code .......................................................................................... 51  

Appendix C ....................................................................................................................... 54  
C.1. Labview Program ................................................................................................. 54
C.2. Bluetooth mate Schematic................................................................. 55
C.3. IMU Code for Arduino IDE.............................................................. 56
Chapter One: Introduction

1.1. Motivation

Although in the U.S. and other countries medical and healthcare services are improving and people are living longer, medical costs are also increasing. There is specific concern with the aging of the “baby boomers” generation\(^1\) in the U.S., since maintaining a high quality of life for the whole population may stress the healthcare system. Based on released figures by the U.S. Centers for Medicare and Medicaid Services of the Health and Human Services administration in 2009 [1], the U.S. healthcare spend in 2007 was about $2.2T. In order to reduce the hospitalization cost and improve patients’ treatment, the idea of telehealth monitoring is growing rapidly. The essence of telehealth is to enable evaluation of an individual’s medical status in real time in spite of his/her location. It also allows a doctor to analysis the information anywhere, which leads to faster diagnosis of diseases. Such a potential can combine more accurate and up-to-date data gathering with better development analysis, while allowing the individual to stay in comfortable surroundings [1].

The importance of telehealth monitoring is growing fast since it is applicable in monitoring variety diseases such as congestive heart failure, coronary artery disease, asthma, diabetes, obesity and more.

\(^1\) The “Baby Boom” generation refers to 79 million American babies that were born during the years 1946-1964.
A remote health monitoring system consists of sensory measurements, data transmission over the network, and feedback system. The techniques for telehealth monitoring can be classified into three methods [2]:

1) **Store and forward telehealth**: In this method, clinical data from patients such as image, video and sound are obtained and stored on a client computer or mobile device. The data is then given to a specialist in the field for proper analysis. As an example, one of the common diseases currently being monitored with this type of telehealth is a skin problem. A digital image from the skin is sent to a server in a clinic for receiving feedback and recommendation from a dermatologist. This method is mostly used for monitoring the critical signs.

2) **Real-time telehealth**: In this type of telehealth, communication between patient and a specialist is performed by telecommunication. Videoconferencing is one of the most general forms of synchronous telemedicine. This method is very beneficial for those patients such as veterans who are abroad and cannot visit their doctors in person. Peripheral devices can also be attached to computers or to the videoconferencing device to help in an interactive assessment manner. With the accessibility of better and cheaper communication channels, direct two-way audio and video streaming between doctors and patients using computers is leading to lower costs. Examples of real-time clinical telehealth include: tele-audiology, tele-cardiology, tele-dentistry, tele-mental health, tele-nursing, tele-radiology, and tele-rehabilitation.
3) **Remote patient monitoring:** In remote monitoring, sensors are used to sense and transmit biometric data. For instance, a tele-EEG device monitors the electrical activity of a patient’s brain and then transmits that data to a specialist. This could be done in either real time or the data could be stored and then forwarded. The system presented in this thesis is categorized in this type of telehealth monitoring.

The objective of this thesis is to develop wearable sensors for body joint angle measurement that can be potentially used to remotely monitor patients’ activities while they stay at home. Specifically, in this thesis three different methods for measuring the knee joint angle were developed and tested. These sensory systems can be used by patients who have arthroplasty or knee-replacement surgery and need to be monitored on a regular basis by their physiotherapist to monitor the improvement of the joint movement. By utilizing such technology patients can stay at their home and be monitored remotely with telehealth monitoring technology. It not only saves the cost of the hospital visits and doctors’ time but it is also convenient for patients.

The three developed methods for measuring the knee joint flexion/extension are: a) a brace with an attached shaft encoder, b) flex-sensors attached to a wearable cloth, and c) inertial measurement units (IMUs) mounted on limb segments. The accuracy of each method is evaluated and compared with other available works. One of the important challenges that we faced with these methods is the placement of sensors on the body joint that yields to similar accurate measurements. In this thesis, the advantages and disadvantages of each developed method is discussed.
1.2. Knee Rehabilitation and Remote Monitoring

Based on a study published in the Journal of Bone and Joint Surgery, Health Leaders Media reports [3], knee-replacement patients who undergo an Internet-based postoperative rehabilitation program experience alike and sometimes better results than those who undergo traditional rehabilitation. One of the common knee surgeries in the United State is arthroplasty [4]. This procedure is performed for about 150,000 patients annually. The goal is to restore a pain-free joint, restore range of motion (ROM), and allow function that approaches normal for a patient.

ROM should be measured from the lateral side of the patient's leg with a goniometer. Full extension—i.e., an angle between the femur and the tibia shaft of 0°—should be recorded as 0°. The knee is then brought to full flexion and measured again from the lateral side of the patient's knee and this is recorded as a positive number between 0-135°. The stability of knee after knee replacement is checked by medial testing. The knee is checked throughout the ROM starting at full extension and then proceeding to 30°, 60°, and 90°. At each position, the patient's leg should stress medially and laterally [4]. This process is usually performed at the hospital. Figure 1.1 shows the joint angle ($\lambda$) which is aimed to be measured by sensors.

There are several recommended home exercises for patients who have had a knee surgery [5]. For instance: sitting knee extension, standing knee bending, standing hip bending and heels slide while lying flat (see Figure 1.2). In all of these exercises, changes in the knee joint are can be monitored remotely to see the improvement.
Figure 1.1. Knee joint angle. \( \beta \) and \( \alpha \) are roll of thigh and shank segment respectively.

To save doctors’ time and clinic’s cost, wearable sensors can be developed and utilized to measure knee joint angle remotely. According to the clinicians’ suggestion, the design requirement for remotely measuring the extension or flexion between the fibula and tibia should be repeatable with a resolution of \( \pm 2^\circ \).

Figure 1.2. Recommended home exercises for patients who had a knee surgery [5].
Although in this thesis the main focus is on sensing the biometric data, the ultimate goal is to transmit data to the clinic for remote monitoring.

1.3. **Thesis Contributions**

The major contributions of this thesis are as follows:

1) Developing a system for the knee joint angle measurement using encoder sensor and brace and evaluating the percentage error.

2) Developing a method for transmitting angle measurements via smartphone in real time.

3) Developing a system for the knee joint angle measurement using multiple flex-sensors attached to a wearable cloth; utilizing Kalman filtering to improve the accuracy of the system.

4) Developing a system for the knee joint angle measurement using IMU sensors and assessing the accuracy of the system by comparing the results with Vicon motion capture system.

1.4. **Thesis Organization**

The remainder of this thesis is organized as follows. Previous work is presented in Chapter two. Knee joint angle measurement using encoder is described in Chapter three. Knee joint angle measurement using flex-sensor is explained in Chapter four. Knee joint angle measurement using IMU sensors is presented in Chapter five. Finally, conclusions and future work are given in Chapter six.
Chapter Two: Previous Work

Some researchers have studied the utilization of wearable sensors on measuring body movements for different applications. In most of these studies, the sensors such as conductive fiber [6], Hall effect sensor [7], accelerometer sensor [8] are sewed or attached on a piece of fabric and then mounted on person’s cloth. In this chapter, related work on body joint angle measurement using wearable sensors is reviewed.

2.1. Wearable Cloth

Gibbs and Harry developed a system for continuous day-to-day monitoring of body joint movement using conductive fiber sewed on a wearable and comfortable garment [6]. As shown in Figure 1.2, there is one conductive fiber which is employed as a sensor. One end of this fiber is permanently attached to the nonconductive fiber (point A). Along the conductive fiber, there is a wire contact point at B that is sewed into the fabric. The other end of the conductive fiber, point C, is kept in tension by a coupled elastic cord. It is attached to the isolated side of the joint, point D. The length of the elastic cord is changed if the joint moves. Since the length of conductive thread between points A and B changes as the joint rotates, the resistance, which is linearly related to length, is measured constantly across these two points A and B.

The goal in designing these wearable sensors is to make a tool that is eventually self-registering for subsequent uses after the first one-time calibration experiments. This means that no extra equipment is needed to register the sensors for every use.
Furthermore, it is critical that any procedures that are needed for self-registration are uncomplicated, and capable to be performed by the patient without supervision. To attain these goals, a multi-thread sensor array design was presented. An array of $M$ sensors covered a single-axis joint as shown in Figure 1.3, each sensor thread is separated from the adjacent sensor thread by a known, constant distance, $d$. This multi-thread sensor array was employed to approximate a single-axis joint angle, $\theta_j$, lower body joint angles.

To develop a registration procedure, first calibration of each sensor thread individually was performed. Creating a denser sensor array in this way leads to more accurate estimates of sensor sensitivities, which in turn leads to more accurate estimates of $\theta_j$. The registration algorithm takes place in real time as the sensor is in use. The only task that a patient needs to do for the first using of these sensors is to first "zero" the sensor output with the joint entirely extended in the $0^\circ$ position, and then without restraint move the

![Figure 1.2. Sensor design Schematic [6].](image)
joint to obtain non-zero data. This non-zero data then allows the self-registration to take place.

![Diagram of sensors over knee joint](image)

Figure 1.3. Array of sensors over knee joint in an equal distance [6].

The pants sensing garment was first employed to calculate single-axis knee angle measurements. For the single-axis experiments, a rotary potentiometer firmly attached to the leg was utilized as a goniometer, and this was the standard for which to compare joint angles. In every trial, the potentiometer was "zeroed" with the leg in the complete extension position. The average root mean square (RMS) error between the pants sensor approximation and the potentiometer using the linear predictor was 5.4°.

Despite the success of this method in measuring the joint angle, the authors [6] mentioned several uncertainties in measuring the resistance across the conductive fiber which resulted in incorrect sensor output. For example, joint movement produces small changes in the fiber tension resulting in a change of fiber resistance which leads to incorrect sensor output.
Laura Dipietro et al. measured the angle of hand joints using Humanglove [7]. The aim of their work was to assess the performance of a commercial glove system, the Humanglove, version 1, to explore its appropriateness as a computerized hand-capture motion system for goniometric applications. The glove has 20 Hall Effect sensors. Each sensor computes data related to a DOF of the hand. The nominal sensor characteristics are resolution, 0.4° over a range up to 90° and accuracy about 1°.

The glove control unit is connected to the host computer through a standard RS-232 at 38400 baud. Data acquisition is executed through a software package called Graphical Virtual Hand (GVH). Using this program calibration of the glove and displaying an animated hand that mirrors movements of the user’s hand is done, as shown in Figure 1.4. Data acquisition and storage in ASCII format can be achieved both with and without the GVH interface (with a non-graphical version of the data acquisition software).

They recruited six right-handed adult subjects. The calibration process was done for each subject. Every subject was requested to lay her or his hand flat on a tabletop, with the wrist in a neutral position to determine the reference position for each joint at 0°.
The subject was inquired to move her or his hand until the greatest extension and flexion were achieved for all DOFs. A therapist assisted the subject to passively execute the same movement. It is known that active and passive movements are characterized by different ROMs. Measurements in the interval between zero and maximum values reached during the procedure described were then normalized.

However, no information about the sensors performance when they are mounted on the elastic fabric glove has been reported. The overall error in measuring angles using this method is 6.17° compared to 5.6° error obtained using Data Glove which is based on fiber-optic technology. This method is fast in time since measurement of all the hand joints together.

![Human glove for measuring hand joint angle](image)

Figure 1.5. Human glove for measuring hand joint angle [7].

2.2. Joint Angle Measurement using Accelerometers

Luinge et al. [8] focused on the ambulatory measurement of the human body and the orientation of joints. They evaluated the impact of neuromuscular disorders affecting the upper extremities; the practical utilization of the arm needs to be evaluated during daily activities. In their method, they calculated the orientation of upper arm with
respect to the forearm by using accelerometer and gyroscope. To assess the orientation of inertial measurement unit (IMU), they executed a sensor-to-segment calibration. The authors intended to improve the potential error of sensor-to-segment calibration by using different calibration movements. The forearm IMU was placed on the dorsal side of the forearm, near the wrist. The upper arm IMU was located on the lateral side of the upper arm close to the elbow. The orientation of the IMU with respect to the forearm was found as the subject executed a pronation – supination movement, while the palm of the hand faced downwards at the beginning and end of the measurement. The upper arm was to be held upright. It was assumed that the angular velocity is in the direction of the $y$-axis during pronation. By holding the palm of the hand downwards, it is assumed that the $z$-axis of the forearm coordinate system points in the vertical direction at the beginning and end of each trial. This vertical direction was measured using the 3D accelerometer in the IMU. Sensor to segment calibrations was conducted. Initial pose was with the arms along the body and the thumbs forward. Each movement needed for the calibration was conducted five times and averaged.

The method was tested on one subject by comparing elbow orientations achieved using the IMUs with the orientations as concluded using a laboratory optical motion capturing system (Vicon). The subject completed 2 tasks: mimicking eating routines and mimicking morning routines. The eating task consisted of the following activities: pouring a glass, eating soup, eating spaghetti, eating meat, drinking. The morning routines task consisted of: splashing water on face and drying it using a towel, applying deodorant, buttoning a blouse, combing hair, brushing teeth.
The accuracy of the reference measurements relies on the precision of the position measurement of the markers. The accuracy of the position measurement was estimated by considering the distance between two markers. The standard deviation of the fluctuation in measured distance was 1mm. This corresponds to a standard deviation in marker-frame orientation of less than 1°. The assumption that the adduction angle is zero was tested using the video camera reference system. Their result showed that the adduction angle during a morning routine task has 8° the RMS value of the angle.

2.3. **Force and MARG sensor**

Kobashi *et al.* proposed a way to monitor knee joint angle by using a Magnetic, Angular Rate, Gravity sensor called MARG[9]. This sensor is a combination of magnetometer, gyroscope and accelerometer from XSENS technology. The attached sensors are shown in Figure 1.7. The three types of sensor systems attain three kinds of
biomechanics values, and an intelligent processing system derives the knee joint angles based on system-of-systems (SoS) technology.

Two MARG sensors are attached to the thigh and shank surfaces, and the foot pressure sensor is affixed into the shoes. By evaluating the measured signals at the signal analysis system, 3-DOF knee joint angles are approximated. Two acceleration vectors at the knee joint center are estimated from each of the thigh and the shank sensors. The expected acceleration vector is dissimilar because they are measured in each local coordinate system of the sensors. Thus, they can estimate the differences between the two local coordinate systems by assessing the differences between the acceleration vectors at the knee joint center.

To approximate the performance of estimating three knee angles composed from flexion/extension (f/e) angle, internal/external rotation (i/e) angle and varus/valgus (v/v) angle, the proposed SoS was applied to three healthy male subjects (22.0 ± 0.0 years old,
173.7 ± 5.0 cm height, 63.7 ± 1.7 Kg weight). The subjects were taught to rotate their right shank for f/e direction, i/e direction and v/v direction by supported active movement. Simultaneously, to evaluate the estimated angles using the proposed SoS, reference angles were measured by using an optical motion system (Micron Tracker H40, Claron Technology, Canada). The mean error of measuring f/e angle was -1.61 ± 3.09 degrees; of measuring i/e angle was 0.93 ± 1.75 degrees; and of measuring v/v angle was 1.83 ± 1.79 degrees.

Though the accuracy of their system is very high, the MARG sensors are expensive, and may not be appropriate for home applications and large scale use in the medical fields.
Chapter Three: Knee Joint Angle Measurement Using Encoder

3.1. Introduction

In this chapter, our first method for measuring the knee joint angle is presented. Wearing a rehabilitation brace by patients who are recovering from knee surgery is usually recommended by doctors and therapist. Hence, designing an angle measurement system that can be simply attached to a brace can be convenient to monitor patients’ knee joint range of motion. Therefore, we used an absolute magnetic kit encoder and a Microcontroller to develop our first angle measurement system.

3.2. Method

Motivated by the fact that a rehabilitation brace is often used after knee surgery due to its significant role in restoring knee extension [11], we used a wearable brace with an absolute magnetic kit encoder attached to the shaft for angle measurement. The encoder with 12 bit resolution is the core of measurement; it generates different pulse widths based on the various angles during the exercise. This encoder is connected to the CSMB12 microcontroller. The microcontroller counts the pulses and adds them in the accumulator. According to the description of the encoder, the maximum angle of 359.91 degrees has the 4097 µs pulse width. Since the output is linear, we can measure all different angles. The benefits of using optical encoders is you avoid the “contact-point
degradation” present in the potentiometer design, however, software is required to process the readings from the sensor.

![Image](image.png)

Figure 3.1. Absolute Magnetic Kit Encoders

The clock frequency of microcontroller 4MHZ and for the LCD is 2MHZ; we increased the clock frequency of the microcontroller to 32 MHZ by using phase-locked loop (PLL). We configured the 9600 baud rate for both of them. Microcontroller is programmed with C programming language.

The next step was to transmit data which is the current angle from the microcontroller to the smart phone. Bluetooth “BlueSMiRF Gold” from Sparkfun Company was employed. Specification for this Bluetooth is listed as follows:

1. Dimension: 51.5x15.8x5.6mm
2. Very robust link both in integrity and transmission distance 100m (Class1)
3. Low power consumption : 25mA avg
4. Frequency: 2.4~2.524 GHz
5. Operating Voltage: 3.3V-6V
6. Serial communications: 2400-115200bps
7. Built-in antenna
The smart phone that we used is HTC dash from T-mobile which has Windows mobile 6, it is also provided with two serial ports. The entire configuration for the smart phone was done by C#. The “Receive” pin of Bluetooth connects to the TXT pin of SCI in microcontroller, the value of angle is ready to send to the smart phone by Bluetooth. The graphical user interface (GUI) of smart phone monitor for the patients is shown in Figure 3.3. Due to the fact that just maximum and minimum of the angle is critical for the doctors, we designed the primary page of the cell phone to track these two values with the Bluetooth.
In order to have a package for the system, we designed a box included the microcontroller, Bluetooth and LCD. There are two push buttons on the box for starting and stopping the communication with the smart phone. First, patient wears the brace and turns on the microcontroller using a turn off/on switch. Then, s/he will see a message on the LCD asked to press the start button on the box. By pressing that button the connection between the microcontroller and the smart phone through a Bluetooth is started. Patients can see the measurements both on the LCD and the smart phone. After finishing the exercise, s/he needs to press the stop button to terminate the communication.

Therefore, when the patient opens the program, s/he should first select the port number of the Bluetooth. And whenever they press the “Connect”, the data is transmitted from microcontroller to the smart phone by holding the “send” push button.

3.3. Experimental Result

The accuracy of the brace and angle encoder was tested by attaching a protractor to the joint and simultaneously the angle values from the protractor and the encoder were read. This comparison was performed on two subjects 26 year-old female and 28 year-old male for three times. Subjects were asked to wear the brace properly, and sit on a chair. S/he flexed/extent the knee joint while sitting; we tested different angles, from 0° to 130° with steps of 5° and read data from the encoder output and measured the knee joint angle using protractor (like traditional method). As Table 1.1 shows, an average error of 3.25°, standard deviation of 2.74° and correlation of 0.99 was obtained by utilizing the brace for the knee angle measurement. The result for wearable brace
completely depends on how the subject wears the brace. Every time a subject wears the brace, s/he has to try to place it in the same position as the previous exercise; otherwise the measured angle will not be accurate.

<table>
<thead>
<tr>
<th>Average Error(degree)</th>
<th>Standard Deviation(degree)</th>
<th>Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.25</td>
<td>2.74</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 1.1 Results of angle measurement using wearable brace for two subjects.

Therefore, to count this disadvantage of the brace system, results based on the placement of the shaft and encoder on the joint vary. In addition it is not comfortable wearing the brace for more than few minutes due to heaviness.

Figure 3.3.a presents a subject wearing the wearable brace and Figure 3.3.b shows the brace, Smart-Phone and a box that contains the LCD and microcontroller.

Figure 3.3. a) Subject with worn a brace; b)Wearable brace, LCD and smart-phone.
4.1. Introduction

In this chapter, another approach for measuring the knee joint angle is presented. This system consists of flex-sensors (as the sensor is flexed, the resistance across the sensor increases) and elastic supportive cloth. Specifically, in our design two flex-sensors mounted underneath a supportive cloth to measure the inner angle of the knee joint. In order to measure the flexion of the flex-sensors, we designed and built a microcontroller-based system using Freescale microcontroller [10]. For driving the flex-sensors we built a constant current source for each of the sensors using bipolar junction transistors (BJTs) (Figure 4.1.).

Figure 4.1. Designed constant current sources to derive flex-sensors. Current sources (0.75 mA) operate separately and keep the current constant for a wide range of voltage (1.5V-4V).
4.2. **Flex-Sensor and Supportive Cloth**

Two flex-sensors were mounted on a supportive cloth to measure the range of knee motion. Figure 4.2 illustrates the size and the shape of the flex-sensor compared to a quarter. The resistivity of the flex-sensor changes when the sensor is flexed. According to the manufacturer’s datasheet, the range of the sensor resistivity is between 10 kΩ and 110 kΩ [12]. Based on the size of a joint, a different number of flex-sensors can be employed.

![Figure 4.2. A sample flex-sensor used in this work](image)

For example, for the knee joint the flex-sensors can be affixed on top of the knee or can be affixed underneath the knee joint. Whenever the knee bends, the flex-sensors affixed on the supportive cloth worn by a subject will bend and cause the resistance of flex-sensors to gradually increase. By developing an electronic system, the change in the resistivity of the flex-sensors can be measured.

We utilized a supportive cloth that is made of stretch fabric. Figure 4.3 illustrates an example of the supportive cloth with attached sensors for studying the knee joint worn by a subject. The supportive cloth has a placement for the knee (there is a hole in the middle of the cloth as shown in Figure 4.3), which helps in reducing the placement error. Furthermore, lightness of the cloth helps reduce patients’ discomfort associated
with prolong usage. As the following figure shows, two sensors are mounted on the supportive cloth underneath the knee.

![Supportive cloth for monitoring the knee joint](image)

Figure 4.3. Top and Bottom views of supportive cloth for monitoring the knee joint.

Changes in the resistivity of the flex-sensors alter the voltage across the collector of the BJTs that collect the constant currents. Thus, change in the joint flexion is converted to change in resistivity of the flex-sensor and then converted to voltage by the constant current source. The collector voltage (0-5V) is then measured by the Analog to Digital Converter (ADC) on the micro-controller board with 8-bit resolution. The measured voltage by the micro-controller board is used in the extended Kalman filter (EKF) to estimate the angle.

### 4.3. Angle Estimation Using Extended Kalman Filter

In order to measure the joint angle using multiple flex-sensors and reducing the measurement uncertainty due to sensor noise, we use EKF. Kalman filter, comprised of two fundamental phases, *Predictor* and *Updater* (Corrector), is used to estimate the state of a system with noisy measurements. In the prediction phase, the current state of the system is estimated based on the prior state information (system dynamic model),
whereas in the update phase, the predicted state of the system is updated using the measurements obtained by the sensors. Originally Kalman filtering was developed to track linear systems. It was then extended to track nonlinear systems as well. The nonlinearity can happen either in the dynamic model of a system, or in the measurement part, or in both dynamic and observation parts. Following is the explanation of Kalman Filtering.

The state vector of a system, $x_k$ at time $k$, is related to the state of the system at time $k-1$ using Equation 4.1. $U_k$ is a control input to the system and $w_k$ is the process noise (white Gaussian noise, $(0, Q)$).

$$x_{k|k-1} = f(x_{k-1}, u_k, W_{k-1}) \quad (4.1)$$
$$z_k = h(x_k, V_k) \quad (4.2)$$

The non-linear function, $h$ in Equation 4.2, relates the state $x_k$ to the measurement $z_k$. $W_k$ and $V_k$ are process noise and the measurement noise, respectively with covariance matrices $Q$ and $R$ which can be either constant or varying based on the time step.

The following equations are used to linearize the estimation phase:

$$x_k \approx \bar{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + W_k \quad (4.3)$$
$$z_k \approx \bar{z}_k + H(x_k - \bar{x}_k) + V_k \quad (4.4)$$

$x_k$ is the actual state and $\bar{x}_k$ is the approximate state and $\hat{x}_k$ is a posteriori estimation of the state at step $k$. $z_k$ is a measurement vector and $\bar{z}_k$ is an approximate measurement vector. $A$, in Equation 4.3, is the Jacobian matrix of partial derivatives of $f$ with respect to $x$, which means:
\[ A = \frac{\partial f}{\partial x}(\hat{x}_{k-1}, u_k, 0) \quad (4.5) \]

\( H \) (Equation 4.6) is the Jacobian matrix of partial derivatives of \( h \) with respect to \( x \):

\[ H = \frac{\partial h}{\partial x}(\bar{x}_k, 0) \quad (4.6) \]

For calculating updated measurement, we use the following equation:

\[ \hat{x}_k = \bar{x}_k + k_k(z_k - \bar{z}_k) \quad (4.7) \]

where, \( k_k \) (Kalman filter gain) is calculated as follows:

\[ k_k = \bar{P}_k H^T (H \bar{P}_k H^T + R)^{-1} \quad (4.8) \]

\( \bar{P}_k \) can be computed by:

\[ \bar{P}_k = A P_{k-1} A^T + W_k Q_{k-1} W_k^T \quad (4.9) \]

\( \bar{P}_k \) is a priori estimate error covariance and \( P_k \) is a posteriori estimate error covariance.

Over all for prediction we have:

Project the state ahead

\[ \bar{x}_k = f(\bar{x}_{k-1}, u_k, 0) \quad (4.10) \]

Project the error covariance ahead

\[ \bar{P}_k = A P_{k-1} A^T + W_k Q_{k-1} W_k^T \quad (4.11) \]

And for update phase we have:

Compute the Kalman gain

\[ k_k = \bar{P}_k H^T (H \bar{P}_k H^T + V_k R V_k^T)^{-1} \quad (4.12) \]

Update the estimate with measurement \( z_k \)

\[ \hat{x}_k = \bar{x}_k + k_k(z_k - h(\bar{x}_k, 0)) \quad (4.13) \]
Update the error covariance

\[ P_k = (I - k_k H_k) \bar{P}_k \]  

(4.14)

In this work, the observation data, \( z \), is the voltage, and \( h \) is a coefficient or a function that maps the joint angle to the voltage (Equation 4.2). In this work, we used curve fitting to find the relation of angle versus observation (voltage). For example, the result for one of the sensor is as following:

![Figure 4.4. Relation Between Angles and Output of Debugger](image)

The corresponding equation by applying derivative for this curve is:

\[ H_k = 0.009735 * z_k - 0.8805 \]  

(4.15)

In order to find the dynamic model of the knee flexion, we used a goniometer to measure the knee movement of a subject versus time which is shown in Figure 4.5 Based on Equation (4.1),

\[ x_{k|k-1} = A * x_{k-1} + U_k + W_k \]  

(4.16)

In our experiment, \( A = 1 \) and \( U_k \) is the slope of the graph. Because to find changes from \( x_{k-1} \) to \( x_k \) in \( \Delta t = 1 \) we have:
\[
\frac{x_k - x_{k-1}}{\Delta t} = \tan \theta = \text{Slope} \tag{4.17}
\]

then,
\[
x_k = x_{k-1} + \Delta t \cdot \tan \theta = x_{k-1} + 1 \cdot U_k \tag{4.18}
\]

There are several ways to fuse data acquired by multiple sensors [13]. One of the methods is to fuse the measurements corresponding to each sensor, and then use the output to estimate the state vector, \(x_k\), by using Kalman filtering. Another method is track-to-track fusion methods first proposed by Bar-Shalom et al.[14,15] and used it in this work. The \(\hat{x}_k\), a posteriori estimation, of each sensor are fused to estimate a new state vector. The reasons that we used this approach are because 1) each sensor acts separately and if any of the sensors fails; it does not affect the fusion process. 2) We can readily add other sensors and fuse the results without modifying the design and the measurements performed by each individual sensor. 3) Moreover, the approach is computationally efficient. Figure 4.6 shows the schematic of the fusion process. A further technique is to modify track-to-track method by feeding back the final fused estimation, \(\hat{x}_{k|k}\) to the single state predictor.

![Figure 4.5. Angles versus Time (Dynamic Model)](image-url)
Then, the output of predictor is fed to correction phase of each sensor separately.

Although the Kalman gain is different with track-to-track fusion method, the computation cost is the same as Bar-Shalom’s track-to-track fusion method. An additional method is a track fusion model with fused prediction. This method has two fusion parts. First, the prediction states of each sensor are fused together and the output is fed to the correction phase of the sensors individually. Then, before fusing the final result of correction phase, the outputs are used for the next prediction procedure.

The estimated angles by separate sensors (sensor 1: $\hat{x}_k^1$ and sensor 2: $\hat{x}_k^2$), (i.e., Kalman filter) are fused as follows:

$$\hat{x}_{k|k} = \hat{x}_{k|k}^1 + \left[ p_{k|k}^1 - p_{k|k}^{12} \right] \left[ p_{k|k}^1 + p_{k|k}^2 - p_{k|k}^{12} - p_{k|k}^{21} \right]^{-1} * (\hat{x}_{k|k}^2 - \hat{x}_{k|k}^1)$$
Where $P_{k|k}^1$ and $P_{k|k}^2$ are the covariance matrix for the tracked estimate $\hat{x}_{k|k}^1$ and $\hat{x}_{k|k}^2$, and $P_{k|k}^{12} = (P_{k|k}^{21})^T$ is the cross covariance matrix between $\hat{x}_{k|k}^1$ and $\hat{x}_{k|k}^2$ which is calculated by:

$$
P_{k|k}^{12} = (I - k_k^1 H_k^1) A_{k-1} P_{k-1|k-1}^{12} A_{k-1}^T (I - k_k^2 H_k^2)^T + (I - k_k^1 H_k^1) G_{k-1} Q_{k-1} G_{k-1}^T (I - k_k^2 H_k^2)^T
$$

(4.20)

where $G_k$ is a gain of the state noise, $W_k$.

Our experiments show that the relation between the voltage (resistivity) and the flexion of the flex-sensor is nonlinear (Figure 4.4). Figure 4.7 indicates changes of voltage during full range flexion and extension versus time of a subject’s knee for two sensors. As the figure shows, the two sensors have different voltage/resistivity responses even though the current sources were kept constant for a wide range of voltage where the current sources operate. In addition, flex-sensors have a nonlinear behavior that varies slightly from one sensor to another sensor.

We designed an experiment to derive the characteristic function of each flex-sensor and its corresponding current source. The characteristic function describes flexion versus voltage that will be used in Equation 4.2.
4.2. Experimental Result

We used the developed Kalman filter tracking and fusion system to estimate the knee flexion of a subject who wore the supportive cloth and exercised her knee. Figure 4.8 illustrates a snapshot of the voltage (observation) in blue, predicted angle in red, and the update angle in green using the EKF for two sensors for a period of 36 seconds (several rounds of bending the knee joint). During this experiment, the range of motion was decreased and increased to show that the Kalman filter can track the changes in the movement. Figure 4.9 shows the fusion of the output of two sensors after Kalman filtering.
Figure 4.8. (a): Sensor 1, (b) Sensor 2. Outputs of sensors after Kalman filter. Blue shows observations, red: predicted values, green: updated values by Kalman filter.

These results are based on the voltage that we have received directly from ADC before sampling it; we can easily convert them to angles. With this result, we assume that the straight leg without any bending is 180°, and angle is decreased by bending the knee. Range of motion for healthy person is 180° to 60° (or 0° to 120°). The measured angles using this approach were compared with the angles simultaneously measured by a goniometer. We recorded the experiment session and particularly the goniometer measurements using a camera. Then, we observed the videos and read and save the goniometer values frame by frame. These measured values were used as ground truth to evaluate the error rate of our system. Based on our experiments, the correlation between
the fused measurement and the output of the goniometer is 0.98, the range of error is 0.076° to 11.32°, and the average error is 6.92°. The average error in measuring the joint angle using Sensor 1 and Sensor 2, are 10.99° and 9.36°, respectively. However, when we fuse the two sensors using the EKF, the average error rate reduces to 6.92°. As we discussed previously, mounting more than two sensors is not practical and comfortable due to the space limitation underneath the knee. We also studied mounting the flex sensor on the top of the knee, but it prohibited the knee to be flexed/bent freely and hence it is not practical. Figure 4.10 shows the comparison of errors between sensor 1(in red) and sensor 2 (in green) and fused data (in blue).

Figure 4.10. Comparsion of errors (two sensors and fused data together).
Chapter Five: Joint Angle Measurement Using IMU

5.1. Introduction

This chapter presents a method for measuring the knee joint angle using IMU sensors. These tiny electronic sensors are fabricated by Micro Electro Mechanical System (MEMS) technologies to compute motions of an object in free space relative to an inertial frame with relatively low power consumption [16]. Accelerometers and gyroscopes are two primary types of IMU sensors for inertial measurement. This study proposes a method to measure the range of motion of the knee joint using two IMU sensors mounted on the body shank and thigh. The measurements are transmitted to a computer via Bluetooth protocol for further data analysis. The acquired data are processed using LabView to calculate the knee angle in real time.

5.2. Method

Two IMUs (SparkFun Electronics, Boulder, CO), each with dimensions 49.53 × 27.94 × 1.5 mm were used in this study. Each sensor consists of triple axis accelerometer with 13-bit resolution and three degrees of freedom gyroscope and magnetometer. The outputs of all sensors were processed using an on-board ATmega328 microcontroller. The two sensors were mounted on simple straps and located on the shank and thigh (Figure 5.1).
Figure 5.1. The two IMUs’ location on the shank and thigh attached to the strips (side and front view).

Attached to each IMU is a Bluetooth radio that transmits all sensors outputs processed by an on-board microcontroller via a serial stream to a PC wirelessly. To test the IMU, at first communication with PC through computer USB port was done using FTDI Basic Breakout (Appendix C.2) since output pins of IMU match up with it. Figure 5.2 illustrates the stream of data using Bray program.

Figure 5.2. Stream of data from IMU sensor to PC using serial port
Sparkfun Company provides customers with the sample Graphical User Interface (GUI) code that helps to estimate the rotation angle around each axis (Figure 5.3).

Bluetooth Mate Gold from Sparkfun Company which is very similar to BlueSMiRF modem is used (Appendix C.1). Any serial stream from 9600 to 115200bps can be passed seamlessly from the computer to a target. The Bluetooth Mate has on-board voltage regulators, so it can be powered from any 3.3 to 6VDC power supply. It has the same pin out as the FTDI Basic, and is meant to plug directly into an Arduino Pro.

![Figure 5.3. GUI for IMU sensor.](image)

For setting the baud rate of Bluetooth, command mode is used by typing “$$” from either the remote Bluetooth connection or the local serial port connection. It is necessary to enter command mode within 60 seconds (configurable by setting the config timer). “U,9600,E” command is effective immediately and does not require rebooting the microcontroller. This sets the baud rate to 9600 and parity even. The baud rate for both Bluetooth and Microcontroller was set to 9600.
Before powering and wearing the sensors, they were calibrated on a smooth surface, parallel to the ground only one time. In this case, both sensors have the same zero reference coordinator.

Before powering and wearing the sensors, the IMUs were calibrated on a flat surface that was parallel to the ground. In this case, both sensors have the same zero reference coordinator. The assumption that thigh and shank segments are in the same plane was considered.

We used the accelerometer for finding flexion angles, and gyroscope to eliminate the effect of vibrations on the accelerometer. As Figure 5.4 shows, $\gamma$ is the absolute tight angle measured by IMU #1 and is the angle between the gravity vector and a perpendicular vector to the femur. This vector is exactly equal to the sense of gravity by accelerometer. Also $\lambda$ is absolute shank angle measured by IMU #2 and is the angle between the gravity vector and a perpendicular vector to the tibia. The flexion angle, $\theta$, is calculated by: $\theta = \gamma - \lambda$. The following equations show how $\beta$ (Roll of thigh segment) and $\gamma$ are calculated based on Figure 5.4.

$$\beta = \gamma$$  \hspace{1cm} (5.1)

$$Acc = \cos(\gamma) \times g$$ \hspace{1cm} (5.2)

where $Acc$ and $g$ indicate accelerometer and gravity, respectively.

$$\gamma = \arccos\left(\frac{Acc}{g}\right)$$ \hspace{1cm} (5.3)

By substituting $\gamma$ in (5.), we obtain:

$$\beta = \arccos\left(\frac{Acc}{g}\right)$$ \hspace{1cm} (5.4)
Figure 5.4. Schematic of the Knee shows the angle configuration with respect to the reference point

$\alpha$ (roll of shank segment) and $\lambda$ are calculated the same way as $\beta$ and $\gamma$. These calculations are performed by implementing python programming language in the microcontroller of the IMUs. Therefore, we have the corresponding roll angle of each segment, $\alpha$ and $\beta$ relative to the calibrated orientation.

As shown in Figure 5.4, we are interested in finding $\lambda$, i.e. the knee flexion angle where full extension is equal to zero degrees. Because, knee flexion occurs in one plane, the roll data from IMU #1 ($\beta$ angle) and roll data from IMU # 2 ($\alpha$ angle) can be combined to provide a knee flexion angle, $\theta = 180 - (\alpha + \beta)$.

Each IMU sends the Euler angles to separate serial ports of the PC using the Bluetooth module. Calculation of $\theta$ is performed using a custom program written using Labview 2010. Data from each IMU is received by the program through the serial port in the same time. Appendix C.1 shows the schematic of the developed Labview program.
5.3. Experimental Result

We tested the accuracy of the IMU measurement system by comparing the concurrent recorded knee flexion angle with the calculations from a passive infrared motion capture system (Vicon Motion Systems, Centennial, CO). This comparison was performed on a subject 26 year-old male. We mounted the IMU sensors onto the thigh and shank along with series of reflective spheres placed in a modified Helen-Hayes configuration, a standard set for accurate lower body motion capture[17] (Figure 5.6). The subject performed four tasks involving knee movement: 1) swinging the lower leg while in a seated position, 2) unilateral hip and knee flexion in a standing position, 3) sitting down and standing up, and 4) combined movement patterns of gait and squatting.

Using the motion capture and anthropometrics collected from the subject, lower limb kinematics was calculated using the Newington Model through Plug-In Gait [18]. Location of the hip, knee, and ankle joint centers were estimated using marker and anthropometric data. Orientation of the thigh and shank segments were modeled by orthonormal coordinate systems attached to planes passing through the joint centers.

![Markers attached to one the subjects under test to measure the body joint angles.](image)

Standard Euler angles were [19] used to define the knee angle in relation to the thigh coordinate system with the following rotation order: flexion, adduction, rotation.
Knee flexion angle comparisons to the IMU system were performed with the first rotation (flexion). Figure 5.7 shows a graphical comparison of knee joint angle measurements using the two systems for the subject under test.

The sampling rates of infrared system and the IMU system were 100 Hz and 5 Hz, respectively. Therefore, the data from the infrared system was down sampled to 5Hz for comparison. To synchronize the IMU system with the infrared motion capture system, the subject remained still at the beginning and the end of the trials. Data was then synchronized at the start and end of the motion. The results demonstrate that the knee flexion angle calculated with the IMU system approximates the angle calculated using the infrared motion capture system.
Table 5.1 demonstrates the results for the subject for four tasks which are mentioned in the first paragraph of this section. For each task the average error, standard deviation, and correlation coefficient were calculated. The error is almost zero in the primary range of knee flexion during these motions. However, when a change in direction occurs, the deviation between the modes of measurement is larger.

<table>
<thead>
<tr>
<th>Task</th>
<th>Average Error (degrees)</th>
<th>Standard Deviation</th>
<th>Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.08</td>
<td>6.55</td>
<td>0.99</td>
</tr>
<tr>
<td>2</td>
<td>3.06</td>
<td>7.24</td>
<td>0.97</td>
</tr>
<tr>
<td>3</td>
<td>1.68</td>
<td>4.67</td>
<td>0.98</td>
</tr>
<tr>
<td>4</td>
<td>2.40</td>
<td>13.30</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Table 5.1. Experimental Results.

Bland-Altman plots (Figure 5.8) demonstrated good agreement between the two systems with slightly larger variation toward the middle of the range of motion. These
residuals correspond to larger angular velocities that occur in the middle of the range of motion and are likely related to a small time lag between the devices. Tasks 1, 2, 3 and 4 demonstrated approximate biases of 0.38, 2.88, 2.28 and 1.05 deg, respectively, which are clinically insignificant.

Figure 5.8. Bland-Altman Plot for all tasks.
Chapter Six: Conclusion and Future Work

6.1. Conclusion

We developed three different systems to measure the knee joint angle using encoders attached to a brace, flex-sensors, and IMU sensors. Each system has advantages and disadvantages as discussed below.

Although, the combination of brace and shaft encoder yields to a system that can precisely measures the knee joint angle without any calibration, it has several disadvantages as follows: it is uncomfortable to be worn by patients while they exercise or performing daily activities. In addition the accuracy of measurement is affected every time the brace is replaced on the body. In other words, the same results cannot be reproduced if the brace is not mounted exactly on the same position. This is due to the fact that the center of the brace hinge must be aligned with the knee axis of rotation to measure the knee angle without any error.

Implementing flex-sensors on a wearable cloth is comfortable, light, and inexpensive (approximately $150). But flex-sensors can be damaged if they are not worn properly or gently by the subject. In addition, the lifetime of the sensors is not high as bending may result in breaking. Furthermore, the available flex sensors are wide and affixing more than two sensors on a supportive cloth is not practical.

IMU system are convenient, inexpensive (approximately $400), and light. There is no wiring involved; data from the sensors are sent to the PC through Bluetooth radio.
The system is accurate and similar results can be reproduced over time. The only drawback of using IMUs could be the drift accumulation in the accelerometer that can be readily compensated using signal-processing techniques. Table 6.1 summarizes the comparison between the three methods developed in this thesis for the same experiment (flexion/extension the knee joint while sitting).

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Error</th>
<th>Standard Deviation</th>
<th>Correlation</th>
<th>Price</th>
<th>Sensor Life time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brace &amp; Encoder</td>
<td>3.25°</td>
<td>2.74°</td>
<td>0.99</td>
<td>$400</td>
<td>High</td>
</tr>
<tr>
<td>Wearable cloth &amp; flex-sensor</td>
<td>6.92°</td>
<td>13.84°</td>
<td>0.98</td>
<td>$150</td>
<td>Low</td>
</tr>
<tr>
<td>IMU System</td>
<td>0.08°</td>
<td>6.55°</td>
<td>0.99</td>
<td>$400</td>
<td>High</td>
</tr>
</tbody>
</table>

### 6.2. Future Work

- We plan to study flex-sensors with better quality and smaller size and utilize them in the design of our flex-sensor system as well as evaluating how the error in the angle measurement can be reduced by fusing more than two sensors.
- We plan to extend the IMU approach and use multiple IMUs to measure multiple body joint angles/flexions. We will use a smart phone to conveniently store the measurements, calculate important body kinematics, and send this information to a server over the internet for further monitoring or processing by clinicians. Using smart phone makes
the portability of the system much easier. In this case patients do not necessarily need a PC and monitoring can be performed outdoor.
References


Appendix A

A.1. Microcontroller Schematic

#include <hidef.h>        /* common defines and macros */
#include "derivative.h"   /* derivative-specific definitions */

void MCU_init (void);     // Device initialization function declaration
int DutyCycle (void);
void main (void)
{
  int i;
  unsigned int angle = 0;
  //int angle2 = 0;
  unsigned int temp = 0;
  unsigned char data[3];
  MCU_init();       // call Device Initialization
  // Initialize the SCI (Bluetooth)
  DDRS = 8;         // configure PS3 for TxD1
  SCIBD = 13;       // set baud rate for 9600 for 2MHz clock
  SCICR1 = 0;
  SCICR2 = 12;      // enables timer interrupt
  SCISR2 = 2;       // allows TxD1 to be output
  // Initialize the Pulse Accumulator
  TSCR1_TEN = 1;    // Turn on timer
  TSCR1_TFFCA = 1;  // Allow for fast clearing
  TSCR2 = 3;        // Prescaler of 8
  DDRT = 0;         // Make Port T input
  PACTL = 103;      // Pulse Acc System enabled for Gated Acc Mode w/ interrupts enabled
  PACNT = 0;        // Clear register
  PAFLG = 1;        // Clear flag
  // Begin Measurement Code in endless loop
  for (;;)
  {
    angle = 0;
    PACNT = 0x0000;    // Be sure this is clearing register
    PAFLG = 1;         // Clear flag
    while(PAFLG_PAIF == 0); // Wait for interrupt flag (data)
    angle = PACNT;
    data[0] = ((char)((angle%1000)/100))+'0';
    data[1] = ((char)((angle%100)/10))+'0';
  }
data[2] = ((char)(angle%10))+'0';
while(!SCISR1_TDRE);  //Reset the cursor to line 0 position 0
SCIDRL = 128;
for(i=0;i<3;i++)
{
    while(!SCISR1_TDRE);  // Allow the shift reg to load data
    SCIDRL = data[i];
}
}
Appendix B

B.1. Kalman Filtering Code

clear all; close all;
% Initial Conditions
x(1) = 18.5860; % Our real plant initial condition
x_(1) = 18.6; % Our estimate initial condition (they might differ)
xc = x_; % Set the ideal model as we think it should start
P = 0.01; % Set initial error covariance for position & frec, both at sigma 0.1,
P=diag([sigma_pos_init^2 sigmaw_frec_init^2])
sigmav = 0.1; % the covariance coefficient for the position error, sigma
sigmaw = 0.5; % the covariance coefficient for the frequency error, sigma
Q = sigmav*sigmav; % the error covariance constant to be used, in this case just a scalar unit
R = sigmaw*sigmaw; % the error covariance constant to be used, in this case just a scalar unit
G = 1; % G is the Jacobian of the plant transfer functions due to the error.
H = [1 0]; % H is the Jacobian of the sensor transfer functions due to the variables involved
W = 1; % W is the Jacobian of the sensor transfer functions due to the error.

steps = 1000; % Amount of steps to simulate
sw = 1;
alp = 0.3595;
bet = 18.226;
for i = 2:steps % start @ time = 2
    if (mod(i, 400) == 0)
        if (~sw)
            alp = 0.3595;
            bet = 18.226;
            sw = 1;
        else
            alp = -0.3595;
            bet = 290.28;
            sw = 0;
        end
    end
end

x(i) = (8/pi^2)*(105/2)*((sin((i-1)*pi*2/567)-(1/9)*sin((i-1)*pi*3/567)+(1/25)*sin((i-1)*pi*10/567)))+(8/pi^2)*(105/2+10)+randn*sigmav; % dynamic Model
z(i) = 0.0001*x(i)*x(i)*x(i)-0.02*x(i)*x(i)-0.53*x(i)+180.5 + randn*sigmaw; % observation
x_(i) = (8/pi^2)*(105/2)*((sin((i-1)*pi*2/567)-(1/9)*sin((i-1)*pi*3/567)+(1/25)*sin((i-1)*pi*10/567)))+(105/2+10);% observation
z_(i) = x_(i);
h(i) = 0.0001*x(i)*x(i)*x(i)-0.02*x(i)*x(i)-0.53*x(i)+180.5;

% H = h(i)-h(i-1)/x(i,i)-x(i,i-1)
H = 0.0003*x(i,i)^2-0.04*x(i,i)-0.53;
% F = 1
F = x(i)-x(i-1)/(i+1);
P = F*P*F' + G*Q*G'; % Prediction of the plant covariance
S = H*P*H'+R; % Innovation Covariance
K(1,i) = P*H'*inv(S) ; % Kalman's gain

\[
x_{(1,i)} = x_{(1,i)} + K(1,i)*(z(i)-z_{(i)})\] % State check up and update

\[
P = (1-K(1,i)*H)*P;  \] % Covariance check up and update

\[
sigmaP(:,i)=sqrt(diag(P)); \] %sigmap is for storing the current error covariance for plotting purposes
end

figure(10);
plot(x,'-m');                       %plot the real plant behavior
figure(1);
plot(x_,'-g');                      %plot the Kalman filter prediction over the plant
Title('magenta: dynamic model, Green: Updated by Kalman')
figure(2)
plot(z,'.r');                       %plot the observations over this plant
Title('Observation, Voltage')
figure(3), plot(K,'*'), title('Kalman Gain')

B.2. Microcontroller Code

#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#include <MC9S12C128.h>

void MCU_init (void);     // Device initialization function declaration
void sendDataToSCI (char* array, int length);
void delay_ms(void);
unsigned int i = 0,j,q,k;
float avg =0;
float atd_value=0;             //output of sensor 1
float atd_value2=0;            //output of sensor 2
float atd_value3=0;            //output of sensor 3
float atd_value4=0;            //output of sensor 4
float tetha1=0;                //created angle with the sensor 1
float tetha2=0;                //created angle with the sensor 2
float tetha3=0;                //created angle with the sensor 3
float tetha4=0;                //created angle with the sensor 4
unsigned int x;
unsigned int angles[320];
unsigned int angles2[320];
unsigned int angles3[320];
unsigned int angles4[320];
unsigned int sec =0;
char data[32] ={' ',' ',' ',' ',',',' ',' ',' ',',',' ',' ',' ',' ',' ',' ',' ','','
',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '} ;
void ATD_Init(void)
{   
    ATDCTL2 = 0xE0;
// 1 - ATD power down
// 1 - ATD fast flag clear
// 1 - ATD power down in wait mode
// 0 - External trigger level/edge control
// 0 - External trigger polarity
// 0 - External trigger mode enable
// 0 - ATD sequence complete interrupt enable
// 0 - ATD sequence complete interrupt flag

ATDCTL3 = 0x22;
// 0 - reserved
// 0100 - Conversion sequence length = 4 (we are looking for four channels)
// 0 - Result register FIFO mode
// 10 - Background Debug Freeze Enable

ATDCTL4 = 0xC7;
// 1 - A/D resolution select - 8bit
// 10 - Sample time select - 8 A/D conversion clock periods
// 00111 - ATD clock prescaler
  // bus clock 8MHz, PRS = 7
  // ATD clock = BusClk/(PRS + 1)*0.5 = 500kHz

ATDCTL5 = 0xB0; //start of continuous conversion
// 1 - Result register data justification - right justified - bits 0-7
// 0 - Result register data signed or unsigned - unsigned
// 1 - Continuous conversion sequence mode - continuous mode
// 1 - Multi-channel sample mode - Multi channel
// 000 - Analog input channel select mode - channel AN0 - PAD00 -> start of conversion

} void main (void)
{
  MCU_init();       // call Device Initialization
  ATD_Init();
  // Initialize the SCI
  DDRS = 8;         // configure PS3 for TxD1
  SCIBD = 13;       // set baud rate for 9600 for 2MHz clock
  SCICR1 = 0;
  SCICR2 = 12;      // enables timer interrupt
  SCISR2 = 2;       // allows TxD1 to be output

  for (;;)
  {
    for(q=0;q<320;q++)
    {
      while(ATDSTAT0_SCF == 0);     //wait for the end of conversion
      for (k= 0;k<4;k++) delay_ms();  // create 1 second delay by calling delay_ms and then increase second;
      atd_value = ATDDRL0;        //the result of first sensor
      atd_value2 = ATDDRL1;         // The result of second sensor
      atd_value4 = ATDDRL3;        //the result of fourth sensor
      if (tetha1<0) tetha1=0;
      if (tetha2<0) tetha2=0;
      if (tetha3<0) tetha3=0;
      tetha1= (int) (tetha1);
angles[q] = theta1;
angles2[q] = theta2;
angles3[q] = theta3;
angles4[q] = theta4;
data[0] = ((char)(((int) ATDDR0L/100)) + '0'); // to show the value on the LCD
data[1] = ((char)(((int) ATDDR0L%100)/10)) + '0';
data[2] = ((char)(((int) ATDDR0L%10)) + '0';
data[4] = ((char)(((int) ATDDR1L/100)) + '0';
data[5] = ((char)(((int) ATDDR1L%100)/10)) + '0';
data[6] = ((char)(((int) ATDDR1L%10)) + '0';
data[8] = ((char)(((int) ATDDR2L/100)) + '0';
data[9] = ((char)(((int) ATDDR2L%100)/10)) + '0';
data[10] = ((char)(((int) ATDDR2L%10)) + '0';
sendDataToSCI(data, 32);
}
}
}

void sendDataToSCI (char* array, int length)
{
    while(!SCISR1_TDRE); // Reset the cursor to line 0 position 0
    SCIDRL = 128;
    for(j=0; j<length; j++)
    {
        while(!SCISR1_TDRE); // Allow the shift reg to load data
        SCIDRL = array[j];
    }
}

// Timer delay
void delay_ms(void)
{
    int count_loop = 65536/32768;
    while(TCNT!= 0) count_loop=count_loop+TCNT;
}
Appendix C

C.1. Labview Program
C.2. Bluetooth mate Schematic

C.2. FTDI breakout
C.3. IMU Code for Arduino IDE

#define GRAVITY 248 //this equivalent to 1G in the raw data coming from the accelerometer
#define Accel_Scale(x) x*(GRAVITY/9.81)/ //Scaling the raw data of the accel to actual
acceleration in meters for seconds square
#define ToRad(x) (x*0.01745329252) //pi/180
#define ToDeg(x) (x*57.2957795131) //180/pi

// LPR530 & LY530 Sensitivity (from datasheet) => (3.3mv at 3v) at 3.3v: 3mV/s, 3.22mV/ADC
step => 0.93
// Tested values : 0.92
#define Gyro_Gain_X 0.92 //X axis Gyro gain
#define Gyro_Gain_Y 0.92 //Y axis Gyro gain
#define Gyro_Gain_Z 0.92 //Z axis Gyro gain
#define Gyro_Scaled_X(x) x*ToRad(Gyro_Gain_X) //Return the scaled ADC raw data of the
gyro in radians for second
#define Gyro_Scaled_Y(x) x*ToRad(Gyro_Gain_Y) //Return the scaled ADC raw data of the
gyro in radians for second
#define Gyro_Scaled_Z(x) x*ToRad(Gyro_Gain_Z) //Return the scaled ADC raw data of the gyro
in radians for second

#define Kp_ROLLPITCH 0.02
#define Ki_ROLLPITCH 0.00002
#define Kp_YAW 1.2
#define Ki_YAW 0.00002

/*For debugging purposes*/
#define OUTPUTMODE 1 //Will print the corrected data,
#define OUTPUTMODE 0 //Will print uncorrected data of the gyros (with drift)
#define OUTPUTMODE 1

#define PRINT_DCM 0 //Will print the whole direction cosine matrix
#define PRINT_ANALOGS 0 //Will print the analog raw data
#define PRINT_EULER 1 //Will print the Euler angles Roll, Pitch and Yaw
#define ADC_WARM_CYCLES 50
#define STATUS_LED 13

int8_t sensors[3] = {1,2,0}; // Map the ADC channels gyro_x, gyro_y, gyro_z
int SENSOR_SIGN[9] = {-1,1,-1,1,1,-1,-1,-1,-1}; //Correct directions x,y,z - gyros, accels,
magnetometer

float G_Dt=0.02; // Integration time (DCM algorithm) We will run the integration loop at 50Hz
if possible

long timer=0; //general purpose timer
long timer_old;
long timer24=0; //Second timer used to print values
int AN[6]; //array that store the 3 ADC filtered data (gyros)
int AN_OFFSET[6]={0,0,0,0,0,0}; //Array that stores the Offset of the sensors
int ACC[3]; //array that store the accelerometers data
int accel_x;
int accel_y;
int accel_z;
int magnetom_x;
int magnetom_y;
int magnetom_z;
float MAG_Heading;
int a = 65;

float Accel_Vector[3] = {0,0,0}; //Store the acceleration in a vector
float Gyro_Vector[3] = {0,0,0}; //Store the gyros turn rate in a vector
float Omega_Vector[3] = {0,0,0}; //Corrected Gyro_Vector data
float Omega_P[3] = {0,0,0}; //Omega Proportional correction
float Omega_I[3] = {0,0,0}; //Omega Integrator
float Omega[3] = {0,0,0};

// Euler angles
float roll;
float pitch;
float yaw;

float errorRollPitch[3] = {0,0,0};
float errorYaw[3] = {0,0,0};

unsigned int counter=0;
byte gyro_sat=0;

float DCM_Matrix[3][3] = {
    {1,0,0 },
    {0,1,0 },
    {0,0,1 }
};
float Update_Matrix[3][3] = {{0,1,2},{3,4,5},{6,7,8}}; //Gyros here

float Temporary_Matrix[3][3] = {
    {0,0,0 },
    {0,0,0 },
    {0,0,0 }
};

//ADC variables4
volatile uint8_t MuxSel=0;
volatile uint8_t analog_reference;
volatile uint16_t analog_buffer[8];
volatile uint8_t analog_count[8];
union u_tag {
    byte b[4];
    float f_val;
};

union u_tag1 {
    byte by[2];
    int i_val;
};

void SendFloat(float f) {
    u_tag u;
    u.f_val = f;
    Serial.print(u.b[0], BYTE);
    Serial.print(u.b[1], BYTE);
    Serial.print(u.b[2], BYTE);
    Serial.print(u.b[3], BYTE);
}

void ReceiveFloat(float &res) {
    u_tag u;
    u.b[0] = Serial.read();
    u.b[1] = Serial.read();
    u.b[2] = Serial.read();
    u.b[3] = Serial.read();
    res = u.f_val;
}

void setup() {
    Serial.begin(9600);
    pinMode (STATUS_LED, OUTPUT);  // Status LED
    Analog_Reference(DEFAULT);
    Analog_Init();
    I2C_Init();
    Accel_Init();
    Read_Accel();
    Serial.println("Sparkfun 9DOF Razor AHRS");
    digitalWrite(STATUS_LED, LOW);
    delay(1500);
    // Magnetometer initialization
    Compass_Init();

    // Initialize ADC readings and buffers
    Read_adc_raw();
    delay(20);
    for(int i=0;i<32;i++)  // We take some readings...
    {
        Read_adc_raw();
        Read_Accel();
        for(int y=0; y<6; y++)  // Cumulate values
            AN_OFFSET[y] += AN[y];
    }
delay(20);
}
for(int y=0; y<6; y++)
    AN_OFFSET[y] = AN_OFFSET[y]/32;
AN_OFFSET[5]=GRAVITY*SENSOR_SIGN[5];
for(int y=0; y<6; y++)
    Serial.println(AN_OFFSET[y]);
delay(2000);
digitalWrite(STATUS_LED,HIGH);
Read_adc_raw();    // ADC initialization
timer=millis();
delay(20);
counter=0;
void loop() //Main Loop
{
    if((millis()-timer)>=20)  // Main loop runs at 50Hz
    {
        counter++;
        timer_old = timer;
        timer=millis();
        if (timer>timer_old)
            G_Dt = (timer-timer_old)/1000.0;    // Real time of loop run. We use this on the DCM
algorithm (gyro integration time)
else
            G_Dt = 0;
    // *** DCM algorithm
    // Data adquisition
    Read_adc_raw();    // This read gyro data
    Read_Accel();     // Read I2C accelerometer
    if (counter > 5) // Read compass data at 10Hz... (5 loop runs)
    {
        counter=0;
        Read_Compass();  // Read I2C magnetometer
        Compass_Heading(); // Calculate magnetic heading
    }
    // Calculations...
    Matrix_update();
    Normalize();
    Drift_correction();
    Euler_angles();
    printdata();
    //Turn off the LED when you saturate any of the gyros.
    if((abs(Gyro_Vector[0])>=ToRad(300))||(abs(Gyro_Vector[1])>=ToRad(300))||(abs(Gyro_Vector[2])>=ToRad(300)))
    {
        if (gyro_sat<50)
            gyro_sat+=10;
    }
else
    {
}
if (gyro_sat>0)
gyro_sat--;  
}  
if (gyro_sat>0)
digitalWrite(STATUS_LED,LOW);
else
digitalWrite(STATUS_LED,HIGH);
}

// We are using an oversampling and averaging method to increase the ADC resolution
// Now we store the ADC readings in float format
void Read_adc_raw(void)
{
  int i;
  uint16_t temp1;
  uint8_t temp2;

  // ADC readings...
  for (i=0;i<3;i++)
  {
    do{
      temp1 = analog_buffer[sensors[i]]; // sensors[] maps sensors to correct order
      temp2 = analog_count[sensors[i]];
    } while (temp1 != analog_buffer[sensors[i]]);  // Check if there was an ADC interrupt during readings...
    if (temp2>0) AN[i] = (float)temp1/(float)temp2; // Check for divide by zero
  }

  // Initialization for the next readings...
  for (int i=0;i<3;i++)
  {
    do{
      analog_buffer[i]=0;
      analog_count[i]=0;
    } while (analog_buffer[i]!=0); // Check if there was an ADC interrupt during initialization...
  }
}

float read_adc(int select)
{
  if (SENSOR_SIGN[select]<0)
    return(AN_OFFSET[select]-AN[select]);
  else
    return(AN[select]-AN_OFFSET[select]);
}

//Activating the ADC interrupts.
void Analog_Init(void)
{
  ADCSRA|=(1<<ADIE)|(1<<ADEN);
  ADCSRA|= (1<<ADSC);
void Analog_Reference(uint8_t mode)
{
    analog_reference = mode;
}

// ADC interrupt vector, this piece of code is executed everytime a conversion is done.
ISR(ADC_vect)
{
    volatile uint8_t low, high;
    low = ADCL;
    high = ADCH;

    if(analog_count[MuxSel]<63)
    {
        analog_buffer[MuxSel] += (high << 8) | low;   // cumulate analog values
        analog_count[MuxSel]++;
    }
    MuxSel++;
    MuxSel &= 0x03;   //if(MuxSel >=4) MuxSel=0;
    ADMUX = (analog_reference << 6) | MuxSel;
    // start the conversion
    ADCSRA|= (1<<ADSC);
}

/**************************************************************************
// Local magnetic declination
// I use this web : http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp
#define MAGNETIC_DECLINATION -6.0    // not used now -> magnetic bearing
void Compass_Heading()
{
    float MAG_X;
    float MAG_Y;
    float cos_roll;
    float sin_roll;
    float cos_pitch;
    float sin_pitch;

    cos_roll = cos(roll);
    sin_roll = sin(roll);
    cos_pitch = cos(pitch);
    sin_pitch = sin(pitch);
    // Tilt compensated Magnetic filed X:
    MAG_X =
magnetom_x*cos_pitch+magnetom_y*sin_roll*sin_pitch+magnetom_z*cos_roll*sin_pitch;
    // Tilt compensated Magnetic filed Y:
    MAG_Y = magnetom_y*cos_roll-magnetom_z*sin_roll;
    // Magnetic Heading
    MAG_Heading = atan2(-MAG_Y,MAG_X);
}
**************************************************************************/

void Normalize(void)
{
    float error=0;
    float temporary[3][3];
    float renorm=0;

    error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*0.5; //eq.19

    Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq.19
    Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq.19

    Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]); //eq.19
    Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]); //eq.19

    Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x b
    //eq.20

    renorm= 0.5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //eq.21
    Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

    renorm= 0.5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //eq.21
    Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

    renorm= 0.5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //eq.21
    Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}

/***************************************************************************/
void Drift_correction(void)
{
    float mag_heading_x;
    float mag_heading_y;
    float errorCourse;

    // Compensation the Roll, Pitch and Yaw drift.
    static float Scaled_Omega_P[3];
    static float Scaled_Omega_I[3];
    float Accel_magnitude;
    float Accel_weight;

    //*****Roll and Pitch***************
    // Calculate the magnitude of the accelerometer vector
    Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] + Accel_Vector[1]*Accel_Vector[1]
                          + Accel_Vector[2]*Accel_Vector[2]);
    Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.

    // Dynamic weighting of accelerometer info (reliability filter)
    // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)
    Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);  //

    Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0]); //adjust the
ground of reference
    Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);

    Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_weight);
Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);

//*****YAW***************
// We make the gyro YAW drift correction based on compass magnetic heading

mag_heading_x = cos(MAG_Heading);
mag_heading_y = sin(MAG_Heading);
errorCourse=(DCM_Matrix[0][0]*mag_heading_y) - (DCM_Matrix[1][0]*mag_heading_x);
//Calculating YAW error
Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw correction to the
XYZ rotation of the aircraft, depending the position.

Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01proportional of YAW.

Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001Integrator
Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the Omega_I
}
/***************************************************************************/
/*
void Accel_adjust(void)
{

Accel_Vector[1] += Accel_Scale(speed_3d*Omega[2]); // Centrifugal force on Acc_y =
GPS_speed*GyroZ
Accel_Vector[2] += Accel_Scale(speed_3d*Omega[1]); // Centrifugal force on Acc_z =
GPS_speed*GyroY
}
void Matrix_update(void)
{
    Gyro_Vector[0]=Gyro_Scaled_X(read_adc(0)); //gyro x roll  
    Gyro_Vector[1]=Gyro_Scaled_Y(read_adc(1)); //gyro y pitch  
    Gyro_Vector[2]=Gyro_Scaled_Z(read_adc(2)); //gyro Z yaw  
    Accel_Vector[0]=accel_x;  
    Accel_Vector[1]=accel_y;  
    Accel_Vector[2]=accel_z;  
    Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);  //adding proportional term  
    Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]); //adding Integrator term  
    //Accel_adjust();  //Remove centrifugal acceleration. We are not using this function in this version - we have no speed measurement
    #if OUTPUTMODE==1
    Update_Matrix[0][0]=0;  
    Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z  
    Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y  
    Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z  
    Update_Matrix[1][1]=0;  
    Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x  
    Update_Matrix[2][0]=-G_Dt*Omega_Vector[1]; //x  
    Update_Matrix[2][1]=G_Dt*Omega_Vector[0]; //y  
    Update_Matrix[2][2]=0;  
    #else                    // Uncorrected data (no drift correction)
    Update_Matrix[0][0]=0;  
    Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z  
    Update_Matrix[0][2]=G_Dt*Gyro_Vector[1]; //y  
    Update_Matrix[1][0]=G_Dt*Gyro_Vector[2]; //z  
    Update_Matrix[1][1]=0;  
    Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];  
    Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];  
    Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];  
    Update_Matrix[2][2]=0;  
    #endif
    Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a*b=c
    for(int x=0; x<3; x++) //Matrix Addition (update)
    {
        for(int y=0; y<3; y++)
        {
            DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
        }
    }
}
void Euler_angles(void)
{
    pitch = -asin(DCM_Matrix[2][0]);
    roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}

**************************************************************************
/* I2C code for ADXL345 accelerometer */
/* and HMC5843 magnetometer */
**************************************************************************
int AccelAddress = 0x53;
int CompassAddress = 0x1E;  //0x3C //0x3D;  //(0x42>>1);

void I2C_Init()
{
    Wire.begin();
}

void Accel_Init()
{
    Wire.beginTransmission(AccelAddress);
    Wire.beginTransmission(AccelAddress);  // power register
    Wire.send(0x2D);  // power register
    Wire.send(0x08);  // measurement mode
    Wire.endTransmission();
    delay(5);
    Wire.beginTransmission(AccelAddress);  // Data format register
    Wire.send(0x31);  // Data format register
    Wire.send(0x08);  // set to full resolution
    Wire.endTransmission();
    delay(5);
    // Because our main loop runs at 50Hz we adjust the output data rate to 50Hz (25Hz bandwidth)
    Wire.beginTransmission(AccelAddress);
    Wire.send(0x2C);  // Rate
    Wire.send(0x09);  // set to 50Hz, normal operation
    Wire.endTransmission();
    delay(5);
}

// Reads x, y and z accelerometer registers
void Read_Accel()
{
    int i = 0;
    byte buff[6];

    Wire.beginTransmission(AccelAddress);
    Wire.send(0x32);        // sends address to read from
    Wire.endTransmission(); // end transmission

    Wire.beginTransmission(AccelAddress); // start transmission to device
    Wire.requestFrom(AccelAddress, 6);    // request 6 bytes from device

    while(Wire.available()) // ((Wire.available())&&(i<6))
    {
        buff[i] = Wire.receive(); // receive one byte
        i++;
    }
Wire.endTransmission(); //end transmission

if (i==6)  // All bytes received?
{
    ACC[1] = (((int)buff[1]) << 8) | buff[0];    // Y axis (internal sensor x axis)
    ACC[0] = (((int)buff[3]) << 8) | buff[2];    // X axis (internal sensor y axis)
    AN[3] = ACC[0];
    AN[4] = ACC[1];
    AN[5] = ACC[2];
    accel_x = SENSOR_SIGN[3]*(ACC[0]-AN_OFFSET[3]);
    accel_y = SENSOR_SIGN[4]*(ACC[1]-AN_OFFSET[4]);
    accel_z = SENSOR_SIGN[5]*(ACC[2]-AN_OFFSET[5]);
}
else
    Serial.println("!ERR: Acc data");

void Compass_Init()
{
    Wire.beginTransmission(CompassAddress);
    Wire.send(0x02);
    Wire.endTransmission(); //end transmission
}

void Read_Compass()
{
    int i = 0;
    byte buff[6];

    Wire.beginTransmission(CompassAddress);
    Wire.send(0x03);        //sends address to read from
    Wire.endTransmission(); //end transmission

    Wire.beginTransmission(CompassAddress);
    Wire.requestFrom(CompassAddress, 6);    // request 6 bytes from device
    while(Wire.available())   // ((Wire.available())&&(i<6))
    {
        buff[i] = Wire.receive();  // receive one byte
        i++;
    }
    Wire.endTransmission(); //end transmission

    if (i==6)  // All bytes received?
    {
        // MSB byte first, then LSB, X,Y,Z
        magnetom_x = SENSOR_SIGN[6]*((int)buff[2]) << 8) | buff[3];    // X axis (internal sensor x axis)
        magnetom_y = SENSOR_SIGN[7]*((int)buff[0]) << 8) | buff[1];    // Y axis (internal sensor y axis)
        magnetom_z = SENSOR_SIGN[8]*((int)buff[4]) << 8) | buff[5];    // Z axis
    }
void printdata(void)
{
    //Serial.print("!"v);

    #if PRINT_EULER == 1
    Serial.print("Roll:");
    Serial.print(abs(ToDeg(roll)));
    Serial.print(" Pitch:");
    Serial.print(ToDeg(pitch));
    Serial.print(" Yaw:");
    Serial.print(ToDeg(yaw));
    //Serial.print(",");
    #endif

    #if PRINT_ANALOGS==1
    Serial.print(AN[sensors[0]]);  //(int)read_adc(0)
    Serial.print(AN[sensors[1]]);
    Serial.print(AN[sensors[2]]);
    Serial.print(ACC[0]);
    Serial.print (ACC[1]);
    Serial.print (ACC[2]);
    Serial.print(magnetom_x);
    Serial.print(magnetom_y);
    Serial.print(magnetom_z);
    #endif

    /*#if PRINT_DCM == 1
    Serial.print ("DCM:");
    Serial.print(convert_to_dec(DCM_Matrix[0][0]));
    Serial.print (" DCM:");
    Serial.print(convert_to_dec(DCM_Matrix[0][1]));
    Serial.print (" DCM:");
    Serial.print(convert_to_dec(DCM_Matrix[0][2]));
    Serial.print (" DCM:");
    Serial.print(convert_to_dec(DCM_Matrix[1][0]));
    Serial.print (" DCM:");
    Serial.print(convert_to_dec(DCM_Matrix[1][1]));
    Serial.print (" DCM:");
    Serial.print(convert_to_dec(DCM_Matrix[1][2]));
    Serial.print (" DCM:");
    #endif
*/
}

else
    Serial.println("!ERR: Mag data");
}
Serial.print(convert_to_dec(DCM_Matrix[1][2]));
Serial.print (",");
Serial.print(convert_to_dec(DCM_Matrix[2][0]));
Serial.print (",");
Serial.print(convert_to_dec(DCM_Matrix[2][1]));
Serial.print (",");
Serial.print(convert_to_dec(DCM_Matrix[2][2]));
#endif*/
Serial.println();
}

long convert_to_dec(float x)
{
    return x*10000000;
}

long convert_to_dec(float x)
{
    return x*10000000;
}

long convert_to_dec(float x)
{
    return x*10000000;
}

//Computes the dot product of two vectors
float Vector_Dot_Product(float vector1[3], float vector2[3])
{
    float op=0;
    for(int c=0; c<3; c++)
    {
        op+=vector1[c]*vector2[c];
    }
    return op;
}

//Computes the cross product of two vectors
void Vector_Cross_Product(float vectorOut[3], float v1[3], float v2[3])
{
    vectorOut[0]= (v1[1]*v2[2]) - (v1[2]*v2[1]);
    vectorOut[1]= (v1[2]*v2[0]) - (v1[0]*v2[2]);
    vectorOut[2]= (v1[0]*v2[1]) - (v1[1]*v2[0]);
}

//Multiply the vector by a scalar.
void Vector_Scale(float vectorOut[3], float vectorIn[3], float scale2)
{
    for(int c=0; c<3; c++)
    {
        vectorOut[c]=vectorIn[c]*scale2;
    }
}

void Vector_Add(float vectorOut[3], float vectorIn1[3], float vectorIn2[3])
{
    for(int c=0; c<3; c++)
    {
        vectorOut[c]=vectorIn1[c]+vectorIn2[c];
    }
}
Multiply two 3x3 matrices. This function developed by Jordi can be easily adapted to multiple n*n matrix's. (Pero me da flojera!).

```c
void Matrix_Multiply(float a[3][3], float b[3][3], float mat[3][3])
{
    float op[3];
    for(int x=0; x<3; x++)
    {
        for(int y=0; y<3; y++)
        {
            for(int w=0; w<3; w++)
            {
                op[w]=a[x][w]*b[w][y];
            }
            mat[x][y]=0;
            mat[x][y]=op[0]+op[1]+op[2];
        }
        float test=mat[x][y];
    }
}
```