



UNIVERSITY OF MASSACHUSETTS LOWELL

DEPT. OF MECHANICAL ENGINEERING & COMPUTER SCIENCE

COMP.7060 – Directed Research

**Development of A RADAR-IMU System for 3D-DIC Measurements Using
Remotely Paired UAVs**

Technical Report & User Guide

Supervisors

Prof. C. Niezrecki

Dr. A. Sabato

Students

M. Sameer Khan

Narasimha Prashanth C R

Samartha K Venkatramana

ACADEMIC YEAR 2017 - 2018

ABSTRACT

3D-DIC is a technique used to compare image data mathematically, using the statistical data and was extended from specular surface measurements. This report describes the process of setting up the hardware required for 3D-DIC measurements, calibrating it to get accurate readings and ultimately mounting this setup on two remotely paired UAVs. We use the IMU and Radar to measure the Acceleration, Gyro, and Compass readings - a total of 7 parameters, which is the basis of our experiment. Once the accuracy of the measurements is established, we are ready to connect the sensors to the drones using a mini-computer (in our case a Raspberry Pi 3 Model B). This is the final step in getting our experimental setup ready to perform 3D-DIC measurements with complete accuracy.

Keywords: *Radar, IMU, NUCLEO, Embedded, 3D, UAVs, FMCW Radar, Drones, Drone based 3D-DIC, sensors, data processing, sensor data, 3D capture, Digital Image Correlation, Raspberry pi.*

TABLE OF CONTENTS

S. No.	TOPIC	Page No.
1.	Abstract	<i>Page 2</i>
2.	List of Figures	<i>Page 4</i>
3.	List of Tables	<i>Page 5</i>
4.	Introduction	<i>Page 6</i>
5.	Sensor Board Development	<i>Page 8</i>
6.	Experimental Setup	<i>Page 10</i>
7.	Conclusions	<i>Page 15</i>
8.	References	<i>Page 16</i>
9.	Appendix A	<i>Page 19</i>
10.	Appendix B	<i>Page 21</i>
11.	Appendix C	<i>Page 30</i>
12.	Appendix D	<i>Page 31</i>
13.	Appendix E	<i>Page 34</i>

LIST OF FIGURES

Figure 1:	<i>Page 6</i>
a) Visualization of a typical 3D-DIC camera system and calibrated measurement volume	
b) Example of calibration target recognition on a coded calibration cross having dimension of 1m x 1m	
c) A very large wall mounted with optical targets being used for calibration of a 7m x 7m area	
Figure 2:	<i>Page 7</i>
a) calibration for the right camera requiring images taken from 12 different views of the target array	
b) calibration for the left camera requiring images taken from 12 different views of the target array calibration	
c) acquisition of the last images of the calibration panel array using both cameras in their final relative position	
Figure 3:	<i>Page 8</i>
Flow diagram of the proposed sensor board embedded on a Microcontroller Unit to be installed on a set of cameras	
Figure 4:	<i>Page 10</i>
Deployment of the three different systems used for comparison experiment procedure	
Figure 5:	<i>Page 10</i>
Calibration curve and scale factor for determining distance in meter from radar's readings	
Figure 6:	<i>Page 11</i>
Comparison of the distance measured using the laser tape measure and the 77 GHz radar	
Figure 7:	<i>Page 12</i>
Roll angle characterization setup sensing axis of the IMU	
Figure 8:	<i>Page 12</i>
Angle measured for different orientation of the translation stage difference between the angles measured using the IMU and the cellphone	
Figure 9:	<i>Page 13</i>
Angle measured for different orientation of the translation stage difference between the angles measured using the IMU and the cellphone	
Figure 10:	<i>Page 13</i>
Euler's Angles	

LIST OF TABLES

Table 1:
Used components and functions

Page 9

INTRODUCTION

Civil, mechanical, and aerospace structures continue to routinely be used even though they are approaching or have already exceeded their design life. Therefore, the need to assess structural integrity is more important than ever and efficient and low-cost assessment techniques are actively being sought. Recent developments in camera technology, optical sensors, and image-processing algorithms have made photogrammetry and three-dimensional (3D) Digital Image Correlation (DIC) an appealing tool for SHM and NDI of structures, as well as for structural dynamic testing. In particular, 3D-DIC has been shown to be an accurate method in providing quantitative information about full-field displacement, strain, and geometry profiles of a variety of structures (i.e., bridges [1], railroad tracks [2], wind turbine blades [3], rotating machinery [4], and composite materials [5]) from images acquired using a pair of synchronized stereo-cameras.

The fundamental principle of 3D-DIC relies on matching the same physical point between a reference state and the altered configuration [6]. Before performing stereo-photogrammetry type measurements, the position of the cameras relative to each other and the distortions of the individual lenses must be determined [7]. Calibration is performed on the cameras' useful measurement volume to obtain the radial distortion coefficient together with the extrinsic and intrinsic parameters for each vision system (see Figure 1a) [8]. The most straightforward technique used for calibration purposes only requires the camera(s) to observe a planar pattern shown at least two different orientations [9]. A calibration for field of views up to ~2 meters are performed by taking several pictures of NIST - traceable calibration objects (e.g., panels or crosses) containing optical targets (i.e., dots) whose positions are previously well-known (see Figure 1b). This procedure creates calibrated measuring volumes that are approximately the same width as the calibration object. A sequence of pictures of the panel at different distances and orientations is captured. These parameters are required for computing the three elementary transformations needed for the pinhole camera model (i.e., conversion of global coordinates of a target object to the camera system coordinates, projection transformation into the retinal plane, and transformation into the sensor coordinate system in pixel units) to obtain the 3D coordinate of any physical point using the triangulation theory [6, 10]. Then a photogrammetry process known as bundle adjustment is used to establish the precise relationship between the two cameras. Once a system is calibrated, the relative position of the cameras must not be altered. Otherwise, measurement errors will occur. Therefore, the camera pairs are rigidly mounted to a stiff bar (generally no longer than 3 m in length) or are fixed on stable tripods. As the dimensions of the targeted object increases, a more complex procedure needs to be performed (i.e., large-area calibration). This operation is arduous as it involves the use of coded and un-coded targets that must be placed on an area having dimensions comparable to that of the object to be tested (see Figure 1c). Moreover, calibration itself is time-consuming because the distance between several pairs of targets needs to be measured to be used as scale bars, and because each camera must be independently calibrated before acquiring the last picture with the two cameras placed in their final position for calculating the relative geometry of the stereo-vision system.

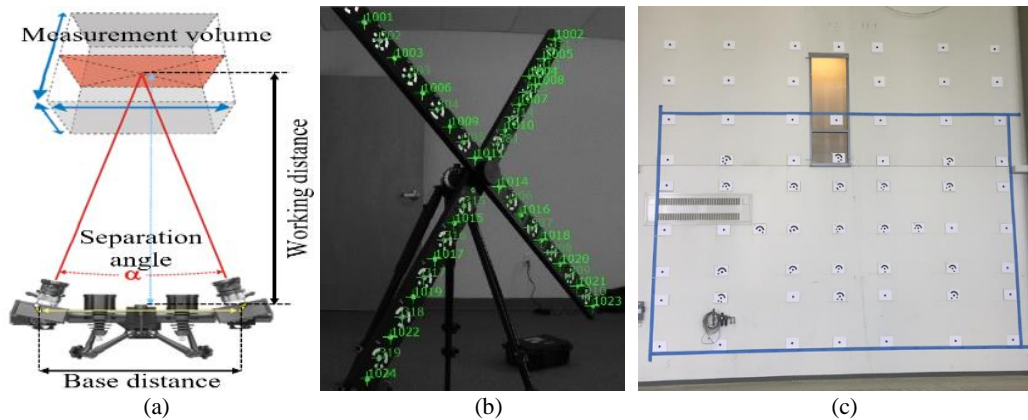


Figure 1. a) Visualization of a typical 3D-DIC camera system and calibrated measurement volume; b) example of calibration target recognition on a coded calibration cross having dimension of 1m x 1m; c) a very large wall mounted with optical targets being used for calibration of a ~7m x 7m area.

An example of this procedure is shown in Figure 2, where the procedure for obtaining a large-area calibration performed on the $\sim 7\text{m} \times 7\text{m}$ area shown in Figure 1c is summarized.

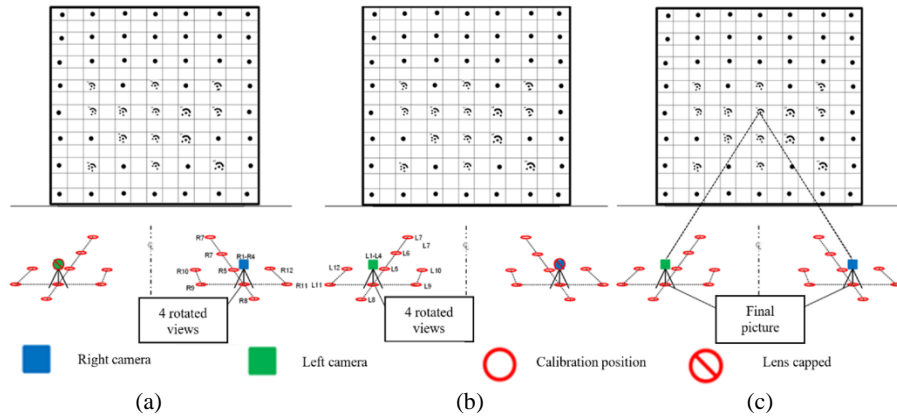


Figure 2. Schematic of a large-area calibration procedure using 14 coded and 42 un-coded targets over a calibration surface with dimensions approximately $7\text{m} \times 7\text{m}$: (a) calibration for the right camera requiring images taken from 12 different views of the target array; (b) calibration for the left camera requiring images taken from 12 different views of the target array calibration; (c) acquisition of the last images of the calibration panel array using both cameras in their final relative position.

Performing the calibration over this field of view requires: (1) fabrication of a calibration panel approximately the same size as the measurement test section (see Figure 1c), (2) manufacture of a very heavy and stiff calibration bar to mount the cameras, and (3) 25 separate camera images in various positions or orientations. Once the final picture was taken, the location of one camera concerning the other cannot change. Otherwise, it would result in a loss of calibration. Any change in cameras' orientation or position will affect the calculated extrinsic parameters values and will result in errors in the estimation of the displacement and strain fields. The fabrication of the calibration panel, the camera bar, and the sensitivity of the cameras to relative motion make the current large-area calibration procedure difficult and not practical to perform structural health monitoring (SHM), non-destructive inspection (NDI), and dynamic testing for larger-scale structures such as bridges, wind turbines, and buildings.

In this report, the preliminary design of a novel wireless sensor board embedding a MEMS-based Inertial Measurement Unit (IMU) and a 77 GHz radar unit for determining the distance and orientation of cameras in space is presented. The design schematics, selected components, software adjustments, and the laboratory tests performed to evaluate the measurement accuracy are described. Each sensor is compared with its traditional counterpart to determine whether it can be used to provide data for determining the seven degrees-of-freedom (DOFs) needed to identify the cameras relative position. If fully developed, the proposed system would enable to achieve measurements to be made from both small to very-large scales on civil engineering structures and infrastructure that require periodic inspections.

SENSOR BOARD DEVELOPMENT

In order to perform the calibration for stereo-photogrammetry, the relative distance between the cameras and their orientation needs to be determined whenever images are recorded. So far, all calibration techniques rely on self-calibration via both targeted planar arrays and target less scenes [11, 12]. The proposed measurement system will streamline the calibration process by 1) enabling the determination of the cameras' relative position for each image sample (eliminating the need for a camera bar and calibrated panel); 2) removing the need to perform the cumbersome calibration prior to testing; and 3) allowing the cameras to be moveable during test, thereby opening the door for long-term monitoring (i.e., camera de-calibration is no longer an issue due to movement).

This research focuses on the development of a sensor-board installed on each of the cameras to determine the seven DOFs needed to identify the cameras relative position in space whenever images are recorded. It includes: (1) the distance between the cameras, (2, 3, 4) roll, pitch and yaw of camera #1, and (5, 6, 7) roll, pitch, and yaw of camera #2. The 3-axis accelerometer along with a compass embedded in the IMU will be used for determining the UAVs orientation, while the 77 GHz radar unit for measuring the distance between the cameras. Those data will be then used for determining the extrinsic parameters needed in the calibration process as described in the flow diagram shown in Figure 3, where the integration of the proposed system on a Microcontroller Unit (MCU) is summarized.

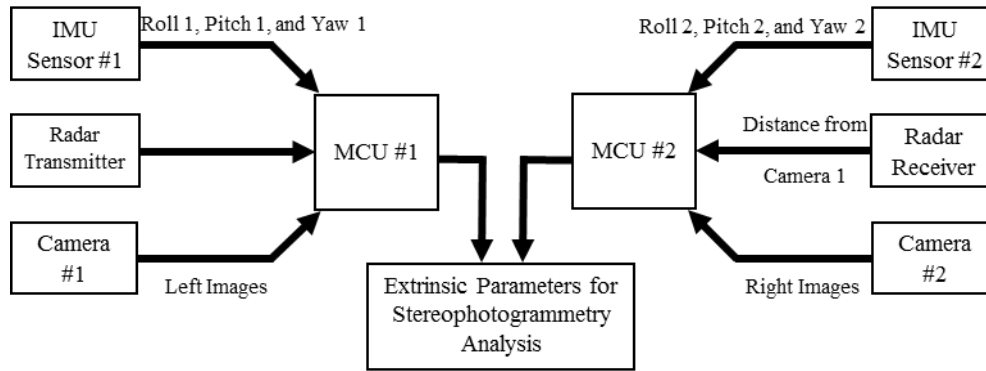


Figure 3. Flow diagram of the proposed sensor board embedded on a Microcontroller Unit to be installed on a set of cameras.

IMU sensors have been widely used for providing low-cost but accurate real-time navigation, guidance, and control of unmanned aerial vehicles focus on delivering reliable detection and localization to maintain a stable attitude and a correct heading direction during flight [13 - 15]. The use of IMU module for photogrammetry applications is still limited to few applications, mostly concerning the onboard georeferentiation of the captured images by a single UAV [16, 17] or the off-board georeferentiation using at least three points and an already calibrated camera [18]. On the other hand, radar units have been often used for determining the distance (or the change in distance) between two objects in space for dynamics analyses [19], modal identification [20], and structural deflection [21].

To realize the sensor package, primarily off-the-shelf components have been used and integrated on a MCU RaspberryPi 3 Model B developed by Raspberry Pi Foundation [22]. The IMU is an ICM-20948 sensor produced by InvenSense [23]. It is a low power 9-axis motion tracking device (suited for smartphones, tablets, wearable sensors, and Internet of Things applications) embedding 3-axis gyroscope, 3-axis accelerometer, and 3-axis compass. The sensor is installed on an evaluation board EVICM-20948 commercialized by InvenSense [24] for connecting the sensor itself with a daughterboard BRD_CARRIER [25] used for programming and controlling the operations. To finish, a MCU STM32F411RE manufactured by STMicroelectronics [26] is used for interfacing the IMU sensor to a laptop computer for data streaming. This interface provides a very reliable access to the component, through MotionLink software or C/C++ codes using customized libraries (e.g., Workbench). This whole setup includes an interface for the IMU itself, and a NUCLEO board, all powered individually, using USB Cables. No external power supply is required since the system can be powered using the $\pm 5V$ power supplied by any computer's Universal Serial Bus (USB) port. The radar unit is a 77 GHz sensor RS3400W/04 developed by SiversIMA [27], which is interfaced to a laptop for data streaming using a Controller Board -

CO1000A/01 provided by the same company [28]. The radar setup involves the usage of an external power supply of 12V to power the whole system. To collect backscattered signals a standard gain pyramidal horn antenna was used. A detail of the various components used is shown in Table 1.






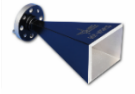


	Equipment	Purpose	Image	Equipment	Purpose	Image	
IMU	ICM-20948	Sensing of roll, pitch, and yaw angles (w/an electronic compass)		RS3400W/04	Distance measurement		Radar
	EVICM-20948	Sensor board with embedded IMU sensor		CO1000A/01	Interface radar sensor with computer for data streaming		
	BRD_CARRIER	Testing of the EVICM-20948		Antenna	Signal reception		
	STM32F411RE	Interface sensor board to computer for data streaming		Raspberry Pi 3 Model B	IMU and radar sensors data acquisition and synchronization		MCU

Table 1. Used components and functions.

To work, both IMU and radar have been programmed using suitable codes. The IMU relies on an ad-hoc Python library that allows the BRD_CARRIER to setup the EVICM-20948 (e.g., sampling frequency, dynamic range, number and type of sensors active) and receive the sampled data before transmitting them to a laptop via the STM32F411RE. The radar employs a set of customized Python codes for enabling measurements, data recording, and data streaming through a RS232 to USB interface.

EXPERIMENTAL SETUP

To evaluate the performance of the proposed systems, several laboratory tests were performed to: (i) demonstrate that radar accuracy in measuring distance relevant to 3D-DIC applications is comparable with the accuracy of traditionally used measurement system and (ii) evaluate IMU consistency in measuring angle and for obtaining cameras orientation and extracting the extrinsic parameters. The first set of experiments referred to stationary signal acquisition of the data recorded using the 77 GHz unit and a direct comparison with traditional length measurement devices, while the second set of tests employed a traditional back-to-back comparison with other devices, with data recorded as the IMU was subjected to different orientation in a three-dimensional space. A detailed description of the performed experiments is provided in the following sections.

Evaluation of the accuracy of the 77 GHz radar sensor in measuring distances

To evaluate the accuracy of the radar, a back-to-back comparison has been performed with a tape measure and a laser tape measure. Both auxiliary systems have an accuracy of $\pm 1 \cdot 10^{-3}$ m, while radar systems can measure distance with a theoretical accuracy of half of their wavelength λ . For a 77GHz radar, λ is equal to $3.896 \cdot 10^{-3}$ m; therefore, a measurement accuracy of $\sim \pm 2 \cdot 10^{-3}$ m should be expected. Since the 77GHz sensor does not provide results in engineering units (e.g., length, L or time, T), a calibration procedure has been developed that allows matching the readings from the radar with those from a tape measure. Then, once a reliable calibration curve has been obtained, a comparison with the readings from the laser tape measure has been carried out to determine the average measurement error. Figure 4 shows the experimental setup used for determining the calibration curve and performing the comparison with the laser tape measure.

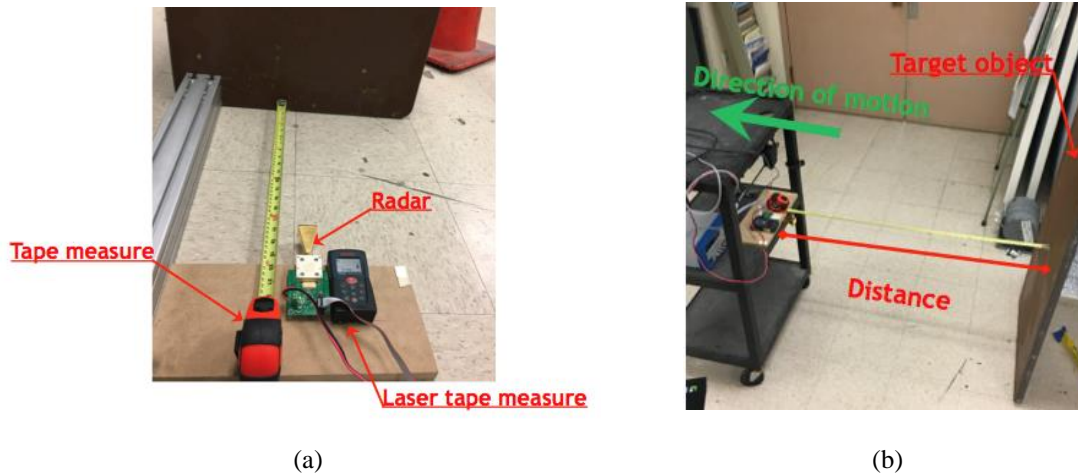


Figure 4. Experimental setup for radar calibration: (a) deployment of the three different systems used for comparison; (b) experiment procedure.

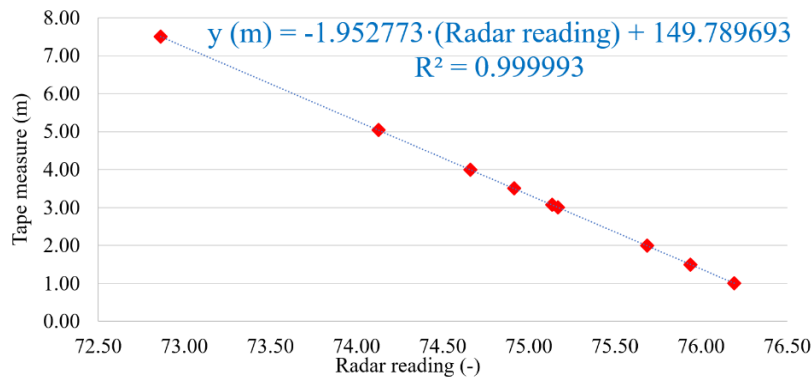


Figure 5. Calibration curve and scale factor for determining distance in meter from radar's readings.

As can be observed from Figure 4b, the three measurement systems have been placed on a four-wheel cart that was moved away from a target object consisting of a wood panel placed perpendicular to the direction of propagation of the radar's signal. The cart has been moved away from the target object with random increments and, for each of the positions used, a total of seven measurements have been recorded with the radar and the laser tape measure.

A significant number of measurements allowed for data averaging and statistical parameters determination (e.g., average and standard deviation, σ). The average of data measured using the radar for each position (in radar units) have been compared to the distances measured by the measure tape. This procedure allowed determining a curve that can be used for converting the radar readings in engineering units (i.e., meter). Figure 5 shows the calibration curve obtained for measurements in the range 1 – 8 m. For instance, as the radar is providing a reading of 75.6866 it corresponds to a distance of $1998 \cdot 10^{-3} \text{ m} \pm 1 \cdot 10^{-3} \text{ m}$ measured using the tape measure.

As can be observed from data plotted in the graph, the relation between the radar readings and the distance measured using the tape measure is linear and the correlation between data set is extremely good (i.e., $R^2 = 0.999993$). To further validate the accuracy of the 77 GHz radar and defined calibration curve, a back-to-back comparison with data recorded using the laser tape measure has been performed. The results of this experiment are summarized in Figure 6.

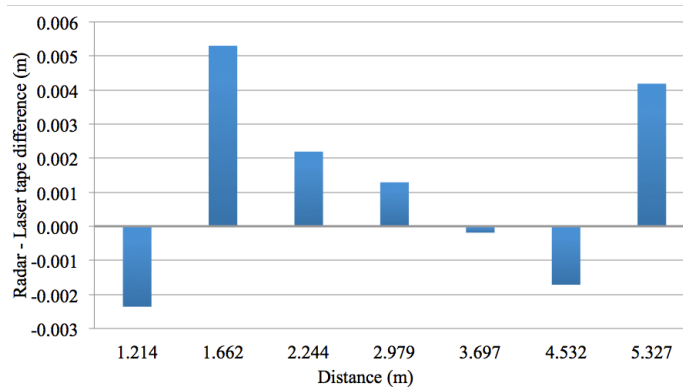


Figure 6. Comparison of the distance measured using the laser tape measure and the 77 GHz radar.

It is observed that the difference between the two sets of data is on average equal to $2.46 \cdot 10^{-3} \text{ m}$ (on the same order of magnitude of the theoretical accuracy of a 77 GHz radar system), with a maximum error equal to $5.29 \cdot 10^{-3} \text{ m}$ recorded for a distance of 1.662 m. Error in the measurement can be due to the not perfect alignment of the cart to the target object resulting in an angle between the laser tape and the radar sensors. Further investigations are required to refine those findings. However, the data showed in Figure 6 are a confirmation of the accuracy of the used radar sensor in measuring distances as data are compared to traditional measurement systems.

Evaluation of the accuracy of the IMU sensor in measuring orientation

An IMU is an electronic device that measures a body's specific force, angular rate, and magnetic field surrounding the body itself, using a combination of accelerometers, gyroscopes, and magnetometers. The accelerometer provides detection of gravity and can measure pitch and yaw. The gyroscope's data provide a rate of rotation, but not an absolute position. Therefore, it can be integrated to estimate change from a known attitude, while the magnetometer can be used for detecting the change in magnetic field. The numerical relationship used for determining the pitch, yaw and tilt angle from the IMU reading are shown below:

$$\theta_{pitch} = \tan^{-1} \frac{Acc_x}{(\sqrt{Acc_y^2 + Acc_z^2})} \quad (\text{Eq.1})$$

$$\theta_{roll} = \tan^{-1} \frac{Acc_y}{(\sqrt{Acc_x^2 + Acc_z^2})} \quad (\text{Eq.2})$$

$$\theta_{yaw} = \tan^{-1} \frac{Acc_z}{(\sqrt{Acc_x^2 + Acc_y^2})} \quad (\text{Eq.3})$$

To determine the accuracy of the IMU, the sensor has been rotated to being subjected to different values of the gravity acceleration, and readings have been used in a back-to-back comparison with a tiltmeter embedded in a cellphone. To facilitate the rotation, the IMU has been attached to a wood support and to a rotational stage in a setup as shown in Figure 7. During the test, IMU has been rotated several times in random increments. For each position, 30 seconds of readings have been recorded to allow data averaging and significant statistical parameters evaluation (e.g., average and standard deviation, σ).

During the tests, only a single parameter (i.e., pitch or roll) at the time was supposed to change, with the other to remain stable within a tolerance of a few hundreds of a degree. Nevertheless, due to the setup used, change in the other parameter was observed that might have affected the outcome of the experiments. Static acceleration data recorded have been used for calculating the values of pitch and roll using the equations shown above and compared with the reading from the cellphone as summarized in Figure 8 and 9.

Results plotted in Figure 8a and 9a show an excellent agreement between the IMU and cellphone data.

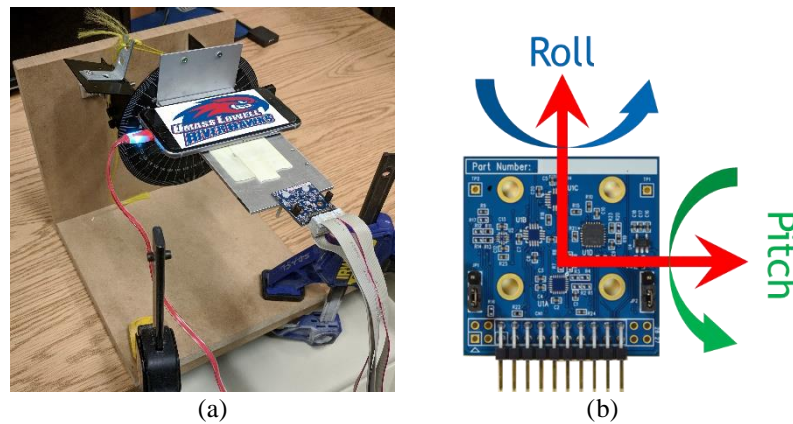


Figure 7. Experimental setup for IMU characterization: a) roll angle characterization setup and b) sensing axis of the IMU.

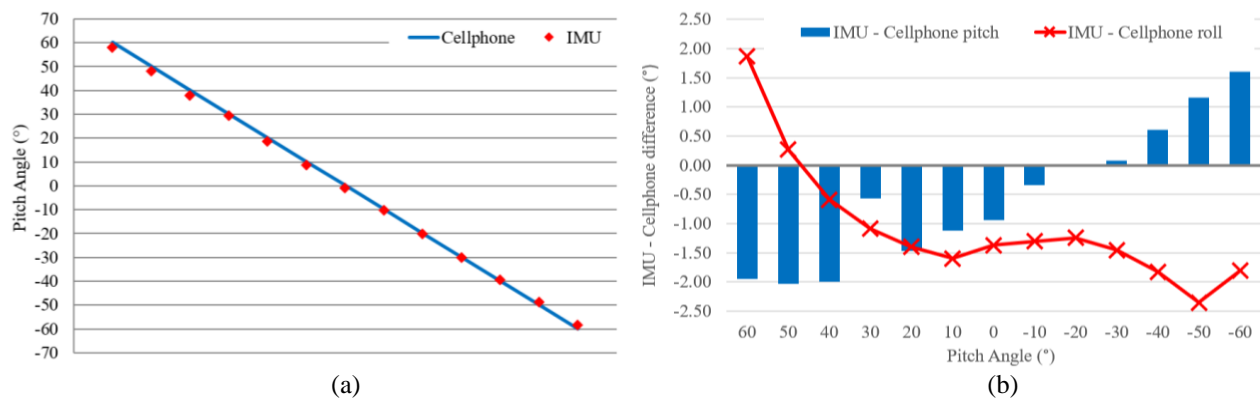


Figure 8. Comparison of pitch angles measured using cellphone and the IMU sensor: a) Angle measured for different orientation of the translation stage and b) difference between the angles measured using the IMU and the cellphone.

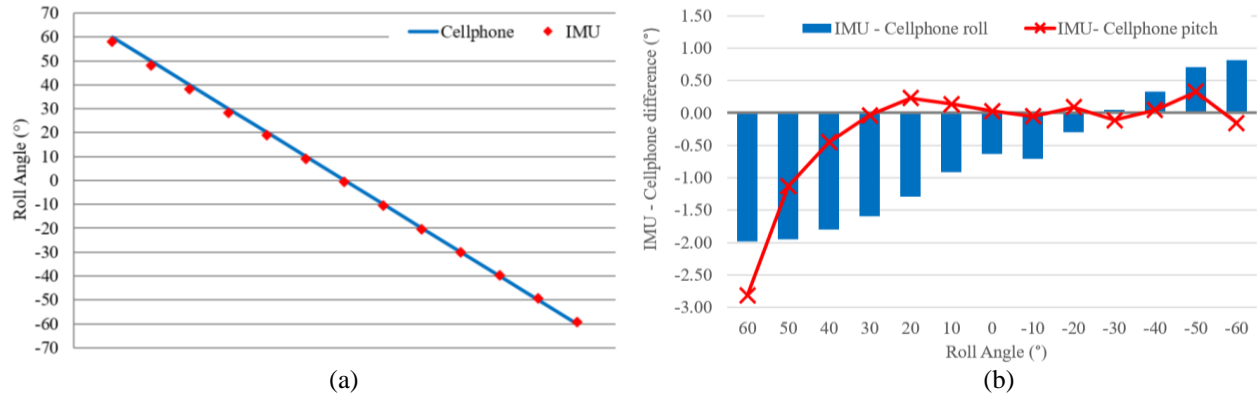


Figure 9. Comparison of roll angles measured using cellphone and the IMU sensor: a) Angle measured for different orientation of the translation stage and b) difference between the angles measured using the IMU and the cellphone.

From more accurate analyses (see Figure 8b and 9b) it is observed that the difference between the two data sets is usually below 1° , but this difference increases as the non-dominant angle (i.e., roll for data reported in Figure 8 and pitch for data reported in Figure 9) rises. With reference to experiments summarized in Figure 8, the tests were performed to subject the IMU and the cellphone to increasing values of the roll angle. This means that as the roll angles increases with 10° steps, the value of the pitch angle should stay close to zero. As reported from data shown in the Figure, the pitch angle varies significantly as well, and this may cause a larger difference between the roll angle values measured using the two systems. Increase in the roll angle from an ideal value of zero may depend from (1) a non-perfect alignment of the experimental setup used, (2) the fact that the IMU and the cellphone are not closely located, and (3) error in the measurements performed by the cellphone. Similar conclusions can be drawn for the data shown in Figure 9. To finish, no information about the yaw angle has been determined at this time.

The relative compass direction and yaw can be calculated using the game rotation vector which can be read as a serial output from the IMU. The magnetometer readings can also be used for calculating the yaw, however, it is very susceptible to change with even the smallest magnetic fields around it, and hence is not very reliable. The game rotation vector does not rely on the magnetometer and instead uses sensor fusion to combine the readings of the accelerometer and gyroscope to get the yaw. The output from the game rotation vector is a set of quaternions each of which represents the roll, pitch and yaw.

Quaternions are a number system that extends the complex numbers and applied to mechanics in three-dimensional space. A feature of quaternions is that multiplication of two quaternions is noncommutative. The general representation for quaternions is as follows: $a + bi + cj + dk$.

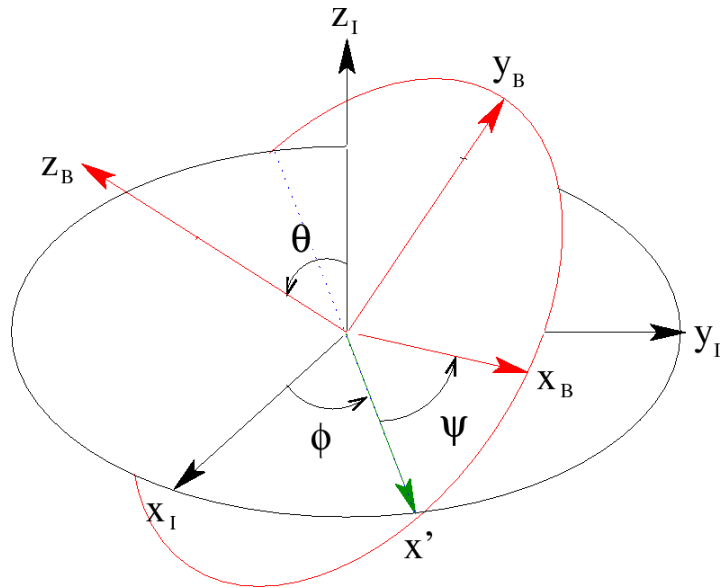


Figure 10. Euler's angles

Quaternions find uses in both theoretical and applied mathematics and in particular for calculation involving three-dimensional rotations. For this project, they are being used to get the respective Euler's Angle for roll, pitch and yaw.

The Euler angles are three angles introduced by Leonhard Euler to describe the orientation of a rigid body with respect to a fixed coordinate system. Euler angles are typically denoted as α, β, γ , or ϕ, θ, ψ . Any target orientation can be reached,

starting from a known reference orientation, using a specific sequence of intrinsic rotations, whose magnitudes are the Euler angles of the target orientation.

- α (or ϕ) represents a rotation around the z axis,
- β (or θ) represents a rotation around the x axis,
- γ (or ψ) represents a rotation around the y axis.

For α and γ , the range is defined modulo 2π radians. For instance, a valid range could be $[-\pi, \pi)$. for β , the range covers π radians (but can't be said to be modulo π). For example, it could be $[0, \pi]$ or $[-\pi/2, \pi/2]$.

The Euler's angles can be obtained from quaternions via the following relation:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0 q_1 + q_2 q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan \frac{2(q_0 q_3 + q_1 q_2)}{1 - 2(q_2^2 + q_3^2)} \end{bmatrix}$$

However, the arctan and arcsine functions implemented in the computer languages only produce results between $-\pi/2$ and $\pi/2$, and for three rotations between $-\pi/2$ and $\pi/2$ one does not obtain all possible orientations. To generate all the orientations, one needs to replace the arctan functions in computer code by atan2:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \text{asin}(2(q_0 q_2 - q_3 q_1)) \\ \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

The yaw obtained using the game rotation vector is quite accurate.

CONCLUSIONS

A novel sensor board prototype is proposed for measuring the seven degrees of freedom (i.e., the distance between the cameras, roll, pitch and yaw of camera #1, and roll, pitch, and yaw of camera #2) necessary for determining the extrinsic parameters of a set of paired cameras to be used for performing three-dimensional digital image correlation (3D-DIC). The system consists of an inertial measurement unit (IMU) and a 77 GHz radar unit embedded on a Raspberry Pi 3 microcontroller unit (MCU) board. In this study, the characterization of the two sensor units is introduced, and comparison with traditional measurement devices is presented to determine the accuracy of the proposed system.

Several laboratory experiments have shown that the accuracy of the radar in measuring distances in a range 1 to 8 meter is equal, on average, to $2.46 \cdot 10^{-3}$, while the IMU can detect change in the orientation with an accuracy of $\sim 1^\circ$ when a back-to-back comparison with an inclinometer embedded in a smartphone is performed. A better set of experiments must be planned to better characterize the performance of the IMU and radar systems that can get rid of all the systematic errors that may have described the current version of the tests.

Nonetheless, results are extremely encouraging and may pave the road to further development of this research. The proposed system has the potentiality to transform the way existing small-scale photogrammetry and DIC measurements are made and will also enable quantitative analyses to be made at very-large-scale (>100 m) from multiple angles and positions. The proposed calibration system will also be insensitive to camera movement and therefore can be attached to a pair of unmanned aerial vehicles (UAVs) to enable measurement from multiple locations and fields of view. The use of the proposed system will allow the self-calibration of cameras as 3D-DIC analyses have to be performed eliminating the need for a rigid bar connection between the cameras, streamlining the calibration process, and extending the range of applicability that stereo photogrammetry and DIC can have. No comparable system currently exists.

REFERENCES

- [1] Reagan, D., Sabato, A., & Niezrecki, C. (2017). Unmanned aerial vehicle acquisition of three-dimensional digital image correlation measurements for structural health monitoring of bridges. In *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*, 1016909.
- [2] Sabato, A., & Niezrecki, C. (2017). Feasibility of digital image correlation for railroad tie inspection and ballast support assessment. *Measurement*, 103, 93-105.
- [3] LeBlanc, B., Niezrecki, C., Avitabile, P., Chen, J., & Sherwood, J. (2013). Damage detection and full surface characterization of a wind turbine blade using three-dimensional digital image correlation. *Structural Health Monitoring*, 12(5-6), 430-439.
- [4] LeBlanc, B., Niezrecki, C., Avitabile, P., & Tribikram, K. (2010). Structural health monitoring of helicopter hard landing using 3D digital image correlation. In *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*, 76501.
- [5] Asl, M. E., Niezrecki, C., Sherwood, J., & Avitabile, P. (2017). Experimental and theoretical similitude analysis for flexural bending of scaled-down laminated I-beams. *Composite Structures*.
- [6] Sutton, M. A., Wolters, W. J., Peters, W. H., Ranson, W. F., & McNeill, S. R. (1983). Determination of displacements using an improved digital correlation method. *Image and vision computing*, 1(3), 133-139.
- [7] Clarke, T. A., & Fryer, J. G. (1998). The development of camera calibration methods and models. *The Photogrammetric Record*, 16(91), 51-66.
- [8] Schmidt, T., Tyson, J., & Galanulis, K. (2003). Full-field dynamic displacement and strain measurement using advanced 3d image correlation photogrammetry: part 1. *Experimental Techniques*, 27(3), 47-50.
- [9] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11), 1330-1334.
- [10] Sturm, P. (2014). Pinhole camera model. In *Computer Vision* (pp. 610-613). Springer US.
- [11] Chen, F., Chen, X., Xie, X., Feng, X., & Yang, L. (2013). Full-field 3D measurement using multi-camera digital image correlation system. *Optics and Lasers in Engineering*, 51(9), 1044-1052.
- [12] Luhmann, T., Fraser, C., & Maas, H. G. (2016). Sensor modelling and camera calibration for close-range photogrammetry. *ISPRS Journal of Photogrammetry and Remote Sensing*, 115, 37-46.
- [13] Kim, J. H., Sukkari, S., & Wishart, S. (2003, July). Real-time navigation, guidance, and control of a UAV using low-cost sensors. In *Field and Service Robotics* (pp. 299-309). Springer, Berlin, Heidelberg.
- [14] Chao, H., Cao, Y., & Chen, Y. (2010). Autopilots for small unmanned aerial vehicles: a survey. *International Journal of Control, Automation and Systems*, 8(1), 36-44.
- [15] Jean, J. H., Liu, B. S., Chang, P. Z., & Kuo, L. C. (2016). Attitude Detection and Localization for Unmanned Aerial Vehicles. *Smart Science*, 4(4), 196-202.
- [16] Chiang, K. W., Tsai, M. L., Naser, E. S., Habib, A., & Chu, C. H. (2015). New calibration method using low cost mem imus to verify the performance of uav-borne mms payloads. *Sensors*, 15(3), 6560-6585.
- [17] Shahbazi, M., Sohn, G., Théau, J., & Menard, P. (2015). Development and evaluation of a UAV-photogrammetry system for precise 3D environmental modeling. *Sensors*, 15(11), 27493-27524.
- [18] Masiero, A., Fissore, F., & Vettore, A. (2017). A Low Cost UWB Based Solution for Direct Georeferencing UAV Photogrammetry. *Remote Sensing*, 9(5), 414.
- [19] Pieraccini, M., Fratini, M., Parrini, F., Macaluso, G., & Atzeni, C. (2004). High-speed CW step-frequency coherent radar for dynamic monitoring of civil engineering structures. *Electronics Letters*, 40(14), 907-908.
- [20] Gentile, C., & Bernardini, G. (2008). Output-only modal identification of a reinforced concrete bridge from radar-based measurements. *Ndt & E International*, 41(7), 544-553.
- [21] Gentile, C., & Bernardini, G. (2010). An interferometric radar for non-contact measurement of deflections on civil engineering structures: laboratory and full-scale tests. *Structure and Infrastructure Engineering*, 6(5), 521-534.
- [22] Raspberry Pi 3 Model B - Single-board computer with wireless LAN and Bluetooth connectivity, Available online: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> (Accessed January 18).
- [23] ICM-20948 - World's lowest power 9-axis MotionTracking device, Available online: <https://www.invensense.com/products/motion-tracking/9-axis/icm-20948/> (Accessed January 18).

- [24] EV_ICM-20948 - Motion Sensor Evaluation Board from TDK InvenSense, Available online: <https://store.invensense.com/ProductDetail/EVICM20948-TDK-InvenSense/597422/> (Accessed January 18).
- [25] BRD_CARRIER - Motion Sensor Evaluation Board from TDK InvenSense, Available online: <https://store.invensense.com/ProductDetail/BRDCARRIER-TDK-InvenSense/607644/> (Accessed January 18).
- [26] NUCLEO-F411RE - STM32 Nucleo-64 boards, Available online: <http://www.st.com/en/evaluation-tools/nucleo-f411re.html> (Accessed January 18).
- [27] RS3400W/04 - 77 GHz Radar Sensor, Available online: <https://www.siversima.com/product/rs3400w04/> (Accessed January 18).
- [28] CO1000A/00 - Power and Controller Board, Available online: <https://www.siversima.com/product/co1000a00/> (Accessed January 18).
- [29] Wikipedia contributors, 'Euler angles', *Wikipedia, The Free Encyclopedia*, 14 March 2018, 15:18 UTC, <https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=830393287> [accessed 16 March 2018]
- [30] Wikipedia contributors, 'Quaternion', *Wikipedia, The Free Encyclopedia*, 10 March 2018, 19:38 UTC, <<https://en.wikipedia.org/w/index.php?title=Quaternion&oldid=829779456>> [accessed 16 March 2018]
- [31] Wikipedia contributors, 'Conversion between quaternions and Euler angles', *Wikipedia, The Free Encyclopedia*, 4 March 2018, 08:47 UTC, <https://en.wikipedia.org/w/index.php?title=Conversion_between_quaternions_and_Euler_angles&oldid=828712576> [accessed 16 March 2018]
- [32] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, *Computing in Science & Engineering*, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37, website: <http://www.numpy.org/>, <http://www.scipy.org/> [accessed March 2018]
- [33] Wes McKinney. Data Structures for Statistical Computing in Python, *Proceedings of the 9th Python in Science Conference*, 51-56 (2010), website: <http://pandas.pydata.org/pandas-docs/stable/index.html>
- [34] John D. Hunter. Matplotlib: A 2D Graphics Environment, *Computing in Science & Engineering*, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55. [accessed 16 March 2018]
- [35] Liechti, C. (2013). Welcome to pySerial's documentation--pySerial 2.6 documentation. website: <https://pythonhosted.org/pyserial/index.html> [accessed 16 March 2018]
- [36] Python package manager PIP - <https://pip.pypa.io/en/stable/> [accessed 16 March 2018]
- [37] Technical Note SiversIMA RS3400W, web: <https://www.siversima.com/wp-content/uploads/Usage-instructions-RS3400W-PC1.pdf> [accessed March 2018]

APPENDIX

APPENDIX A – HOW TO WORK WITH RADAR

The initial configuration for the RADAR as per the datasheet is as follows and be sure to verify the same in the Device Manager and the main program since communication with the Radar may fail if they are not correct.

- ✓ *Bits per second: 115200*
- ✓ *Data bits: 8*
- ✓ *Parity: None*
- ✓ *Stop bits: 1*
- ✓ *Flow control: None* [37]

To work with Radar, primarily we require *python 2.7* and all the next steps required to get the distance from radar are as provided below:

1. Go to <https://www.python.org/downloads/> and click on **Download Python 2.7.14** button.
2. Open the downloaded file and install Python 2.7 by following the on-screen wizard prompt.
3. Open command prompt by typing **cmd** in the Start application
4. Once the installation is done, to verify that it was installed correctly, on command line execute the command ***python --version***
5. Next, type ***python*** and press enter. You should see a window like the one shown in the screenshot below.

6. Depending on the version of python installed, you might have to manually install *pip* as it is already installed if you're using *python 2 >=2.7.9* or *python 3 >=3.4* binaries downloaded from *python.org* [36]
7. Next, check if *pip* is installed by typing ***pip*** into the terminal window. Some text should be shown about *pip* or else, install *pip* from their official website. *Proceed if pip is installed.* [36]
8. Next in the same command prompt and type the following commands to install all the dependencies:
 - ***pip install pyserial***
 - ***pip install numpy***
 - ***pip install scipy***
 - ***pip install pandas***
9. These are all the external libraries that we are going to use to communicate with the radar which are easily installed through the python package installer.
10. Then, go to *Device Manager* (steps mentioned Appendix E) and note down the *port number* (usually **COMXX**) of **USBSerialDevice**.
11. Open ***rad.py*** program in any editor (preferably python IDLE) and change the ***port number in line 14***.

12. These steps are one-time setup only. When you are done with all these steps, you're ready to communicate with the radar.
13. To communicate with the radar, open rad.py in python IDLE and run the program by pressing F5. Else, from the source directory, type *python rad.py* to generate the output onto the console window.

ONE SHOULD GET THE RESULT in (m) ON THE CONSOLE EVERYTIME THE PROGRAM IS RUN.

APPENDIX B – HOW TO COMMUNICATE WITH IMU

There are many ways to communicate and read sensor outputs from IMU. **IAR workbench** is required to program the IMU component. One can also use the **MotionLink** software to view real-time readings. IAR Workbench is not a free software. They come *in trial edition and full version*. The trial is enough to run and debug the program. The following are the one-time setup steps to be done to initiate communication with IMU.

Steps to setup the IAR Workbench:

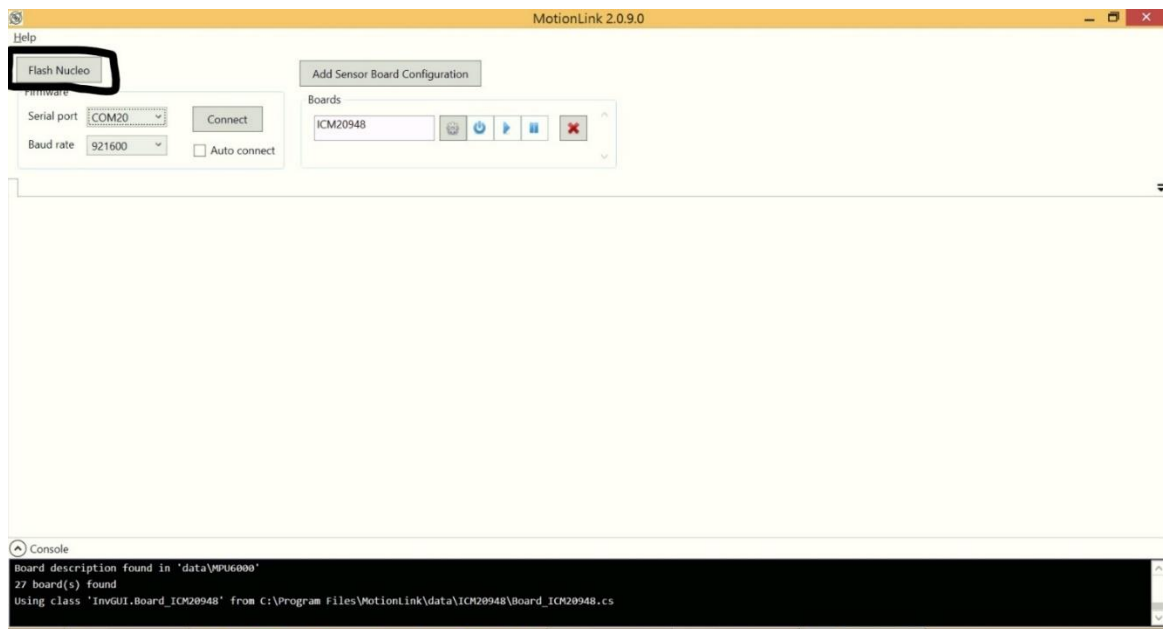
1. Download the **IAR Embedded Workbench for ARM** from www.iar.com/iar-embedded-workbench, by choosing the architecture as “**ARM**”.
2. Once completed, please run the setup program to install the workbench and click **Install IAR Embedded Workbench for Arm** button to start the installation.
3. Click next to choose location of installation. Then make sure **STM** is checked in the next page where you’re asked to *select drivers to install*.
4. Next, download the **STM32 ST-LINK utility**, from <http://www.st.com/en/development-tools/stsw-link004.html>. *Note: You will need to create an account with www.st.com to access these files.*
5. *Extract* the above downloaded file and open **STM32 ST-LINK Utility v4.2.0 setup.exe** to install **ST-Link**.
6. Install the drivers downloaded from <http://www.st.com/en/development-tools/stsw-link009.html> by choosing *the appropriate architecture x86/64*. Extract the above downloaded file and open *dpinst_amd64.exe* for amd64 architecture computer and operating system to install required USB drivers.
7. An optional step involves the firmware upgrade using the software downloaded from <http://www.st.com/en/development-tools/stsw-link007.html>. This step is not a compulsion as the firmware we have is updated to the current latest version v15.0. In future if there are any firmware upgrades by STM, one must download and install the file from the above link. [26]

To run **MotionLink**:

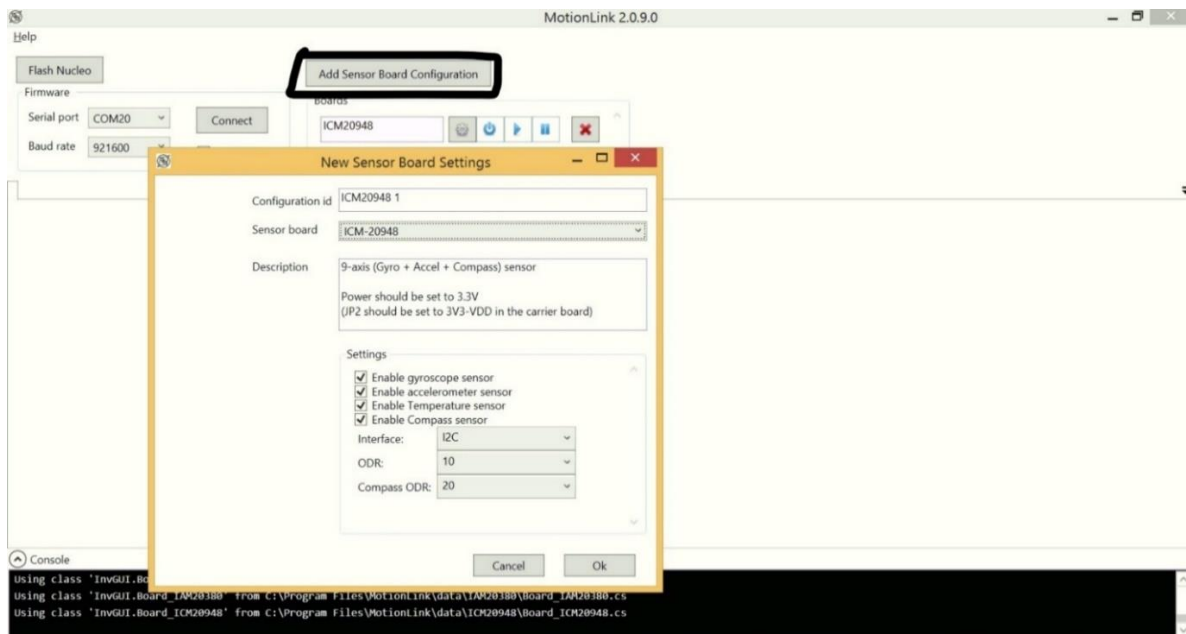
1. Go to <https://www.invensense.com/developers/software-downloads/> . Download and install **MotionLink**.
2. *Right click* on **MotionLink** icon and click on **Run as Administrator**.
3. Click on **Flash NUCLEO** button as shown in the picture below to flash the IMU with binary file

Refer to the next page for pictures and further directions.

The following figure shows the **MotionLink** software main window. It is a very useful tool which provides tools to read, log and save data from the different sensors of the IMU device.

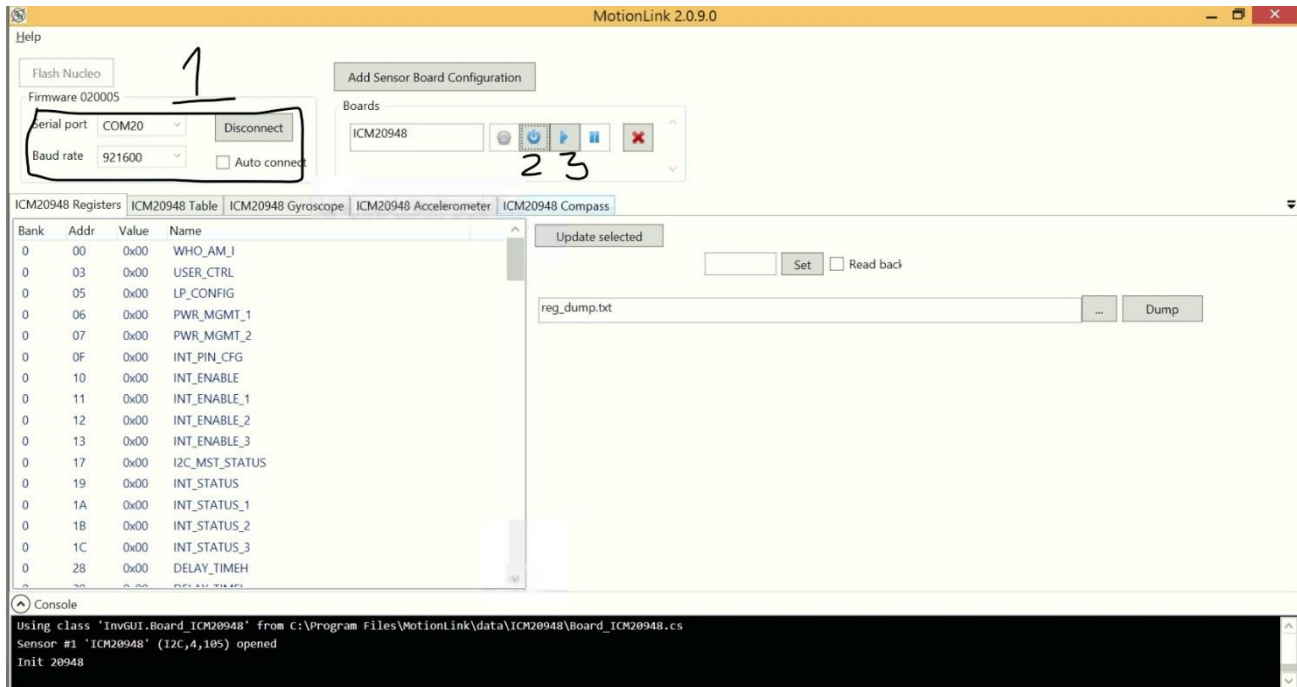


4. Click on Add Sensor Board Configuration button as shown in the picture below and choose **ICM20948** from the list and check all the options as shown below.



5. You can also change ODR (Output Data Rate) in this screen. You can also select which sensors to activate at this point.
6. Make sure that the selected port number is correct, and the baud rate is set to 921600.

7. Click on **Connect** and *power up* the ICM by clicking on power button and then play button as highlighted in the picture below in the same order, as shown by steps 1, 2, 3 in the figure below.



8. Click on different sensors to get real time readings of each one. The following picture shows the output of Gyroscope. In a similar way, we can get real time readings for all the other available sensors as well.



9. To store these readings in a text file, go to **ICM20948 Table tab**, select the *location* in which you're interested in storing the text file, and *click enable* whenever you're ready. We can use the text file to do various analytics on the obtained data.

To run Sensor-cli (Cube Program):

sensor-cli is a command line application used to control InvenSense device from a PC for evaluation and testing purpose.

Included features, non-exhaustive list:

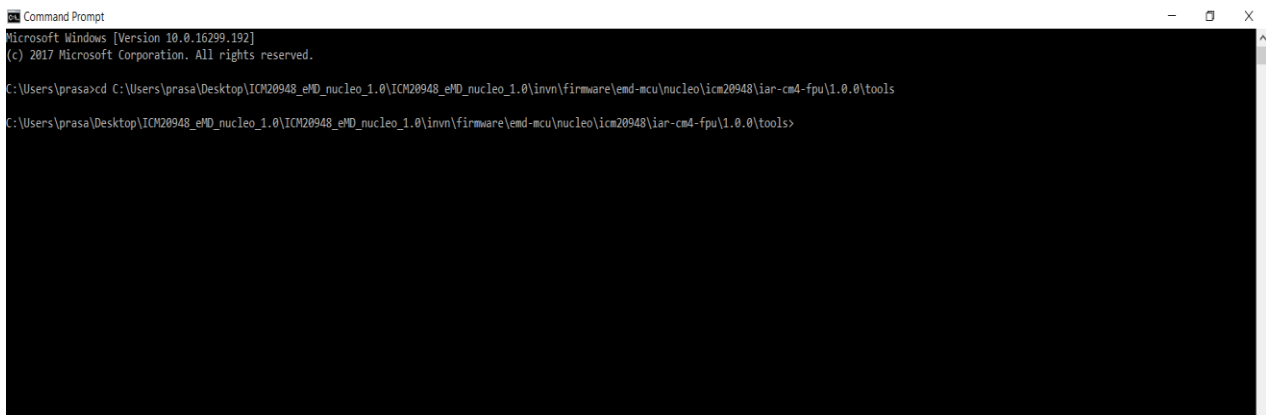
- a) Start/stop/configure sensors for an InvenSense devices
- b) Display data on the screen
- c) Log sensor data to file

The optimal steps are provided below:

1. Go to <https://www.invensense.com/developers/software-downloads/> and download **Embedded MotionDriver (eMD) ICM-20948 v1.0 for Nucleo Board** and then extract the downloaded file. These extracted files in the folder are the tools we need.
2. Open *command prompt* (cmd) and navigate to *tools* (Wherever you extracted the files) folder using *cd* (Change directory) command. It is located deep within the extracted folder under
...\\ICM20948_eMD_nucleo_1.0\\invn\\firmware\\emd-mcu\\nucleo\\icm20948\\iar-cmd-fpu\\1.0.0\\tools
The syntax for *cd* command is

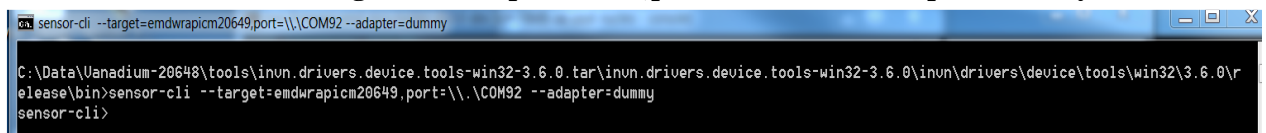
cd <space> <path_to_tools_folder>

If you've extracted on the desktop, navigating to tools folder will look something like the picture below:



3. Now type

sensor-cli --target=emdwapiem20x48,port=\\.\COMxx --adapter=dummy



in the command prompt and press enter. *If correctly configured, you will be taken to sensor-cli prompt.*

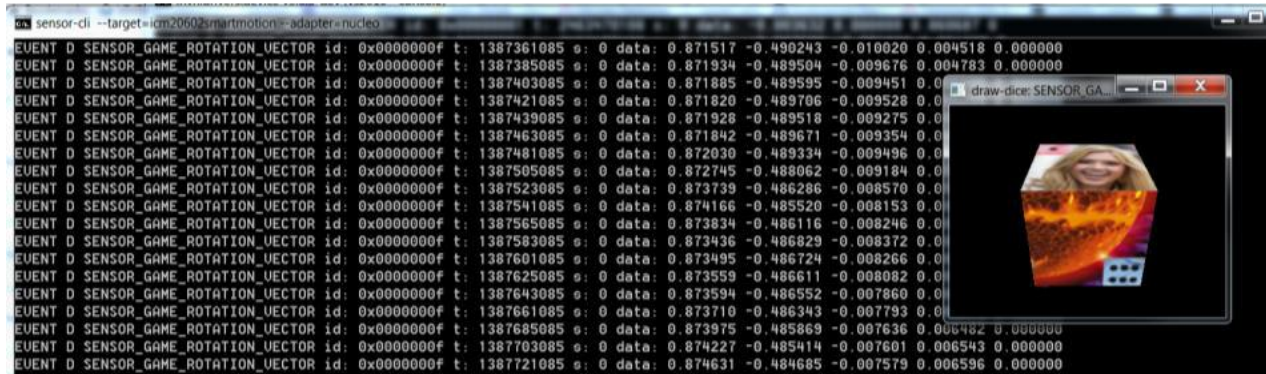
4. Now enable the cube display for *Game Rotation Vector Sensor* (GRV):

sensor-cli> cube on grv

A new window displaying a cube will be opened.

5. Now enable the GRV with a data output period of 20 ms by typing:

sensor-cli> en grv 20



```
sensor-cli --target=icm20602smartmotion --adapter=nucleo
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387361085 s: 0 data: 0.871517 -0.490243 -0.010020 0.004518 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387385085 s: 0 data: 0.871934 -0.489504 -0.009676 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387403085 s: 0 data: 0.871885 -0.489595 -0.009451 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387421085 s: 0 data: 0.871820 -0.489706 -0.009528 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387439085 s: 0 data: 0.871928 -0.489518 -0.009275 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387463085 s: 0 data: 0.871842 -0.489671 -0.009354 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387481085 s: 0 data: 0.872030 -0.489334 -0.009496 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387505085 s: 0 data: 0.872745 -0.488062 -0.009184 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387523085 s: 0 data: 0.873739 -0.486286 -0.008570 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387541085 s: 0 data: 0.874166 -0.485520 -0.008153 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387565085 s: 0 data: 0.873834 -0.486116 -0.008246 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387583085 s: 0 data: 0.873436 -0.486029 -0.008372 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387601085 s: 0 data: 0.873495 -0.486724 -0.008266 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387625085 s: 0 data: 0.873559 -0.486611 -0.008082 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387643085 s: 0 data: 0.873594 -0.486552 -0.007860 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387661085 s: 0 data: 0.873710 -0.486343 -0.007793 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387685085 s: 0 data: 0.873975 -0.485869 -0.007636 0.004783 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387703085 s: 0 data: 0.874227 -0.485414 -0.007601 0.006543 0.000000
EVENT D SENSOR_GAME_ROTATION_VECTOR id: 0x0000000f t: 1387721085 s: 0 data: 0.874631 -0.484685 -0.007579 0.006596 0.000000
```

Game Rotation Vector's data will be displayed, and the cube will move according to the IMU motion.

6. Then, to stop the sensor, use the following command:

sensor-cli> dis grv

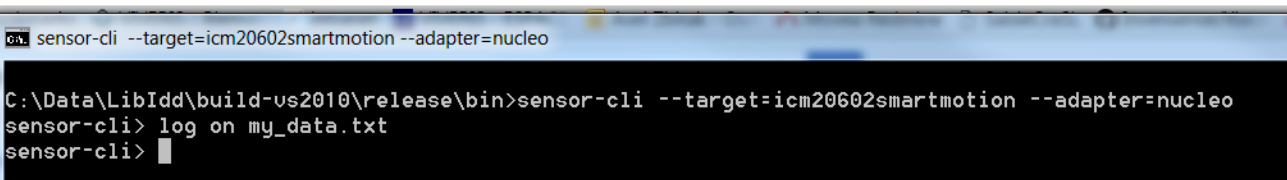
Data output will stop after this command.

7. We can also enable each sensor individually and log the data to a text file for further processing.

8. First, to enable logging to a pre-selected text file, use the following command.

sensor-cli> log on my_data.txt

If you do not type full path to my_data.txt, the file will be created in the current directory



```
sensor-cli --target=icm20602smartmotion --adapter=nucleo
C:\Data\LibIdd\build-us2010\release\bin>sensor-cli --target=icm20602smartmotion --adapter=nucleo
sensor-cli> log on my_data.txt
sensor-cli>
```

9. Then, you can *enable each sensor individually* with specific data rate interval (in milliseconds).

10. For example, to enable accelerometer with 10 ms data rate interval, use the following command

sensor-cli> en acc 10

Output from accelerometer sensor will start flowing in at the specified rate.

11. After performing some tests, we can *disable* the sensor by using the following command:

sensor-cli> dis acc 10

This command disables the sensor and output will stop flowing.

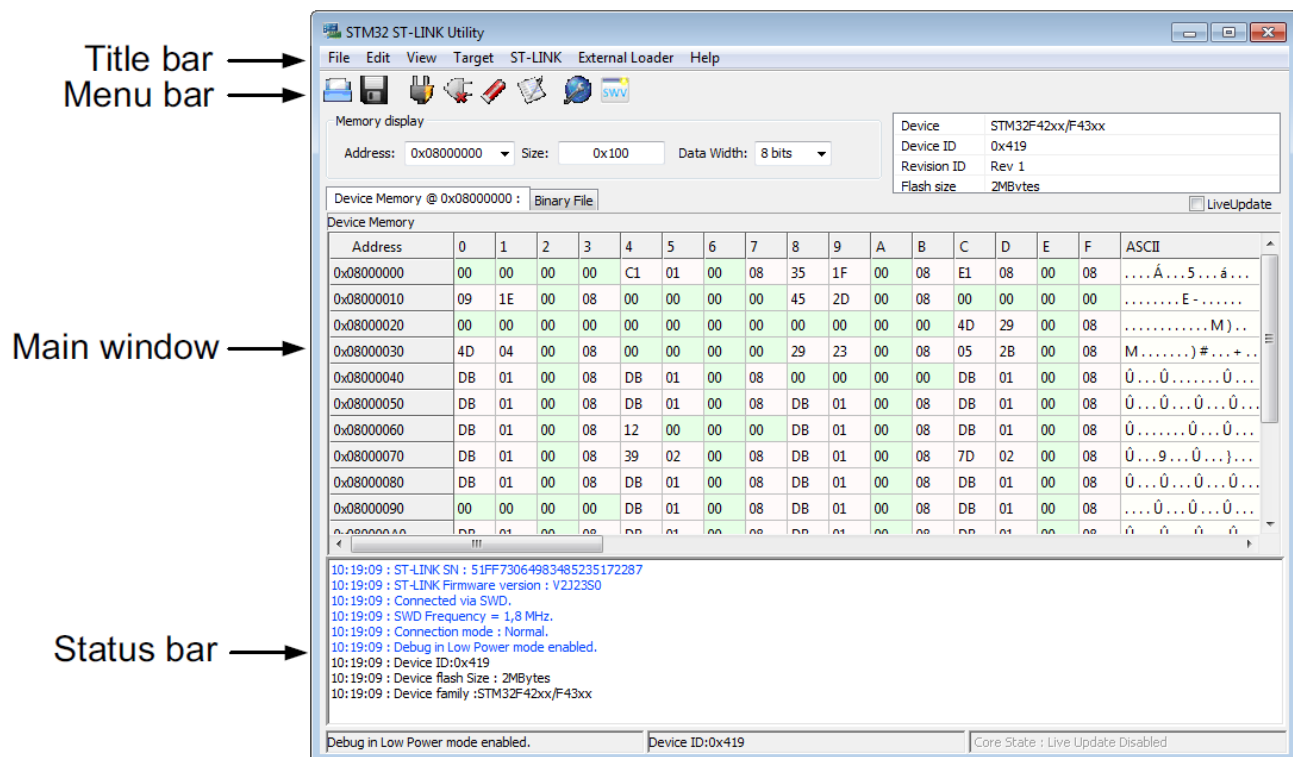
12. Finally, we can check the data from **my_data.txt** file.

```
my_data.txt
1 D SENSOR_ACCELEROMETER 0x00000001 0 2059052085 -0.018799 -0.002563 0.883972 0
2 D SENSOR_ACCELEROMETER 0x00000001 0 2059052085 -0.022644 -0.005981 0.868591 0
3 D SENSOR_ACCELEROMETER 0x00000001 0 2059060085 -0.020081 -0.005127 0.872437 0
4 D SENSOR_ACCELEROMETER 0x00000001 0 2059068085 -0.019653 -0.006836 0.877991 0
5 D SENSOR_ACCELEROMETER 0x00000001 0 2059076085 -0.021362 -0.003845 0.876709 0
6 D SENSOR_ACCELEROMETER 0x00000001 0 2059086085 -0.021790 -0.005554 0.869446 0
7 D SENSOR_ACCELEROMETER 0x00000001 0 2059096085 -0.022217 -0.002991 0.877991 0
8 D SENSOR_ACCELEROMETER 0x00000001 0 2059106085 -0.023071 -0.010254 0.875427 0
9 D SENSOR_ACCELEROMETER 0x00000001 0 2059116085 -0.019226 -0.005127 0.872864 0
10 D SENSOR_ACCELEROMETER 0x00000001 0 2059126085 -0.020508 -0.007690 0.869873 0
```

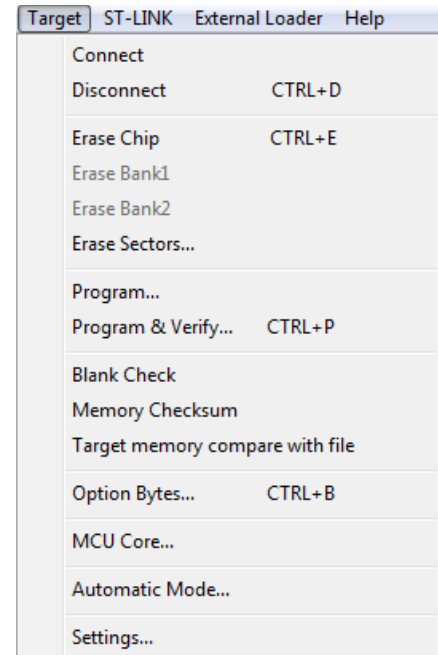
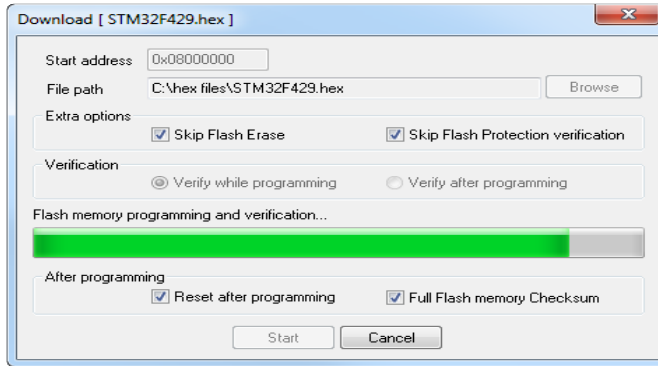
To run **STM32 ST-LINK Utility**:

The STM32 ST-LINK utility software facilitates *fast in-system programming* of the *STM32 microcontroller* families in development environments. This software can be used to *flash the NUCLEO board* with pre-compiled binary files, without the use of Integrated Development Environment(IDE) like IAR Workbench.

The main window of ST-Link looks something like the picture below:



1. From the *Menu* bar, click on **Target** and click **Connect**
2. This connects the IMU with the software and is now ready to be *flashed*.
3. From the same *Target* menu, select **Program** to flash the Nucleo board. Then, choose the **bin/hex file** and click on start.
4. You should see the flashing process begin and complete like shown below.



5. This will *flash the Nucleo board with the selected file and is now ready to operate*.

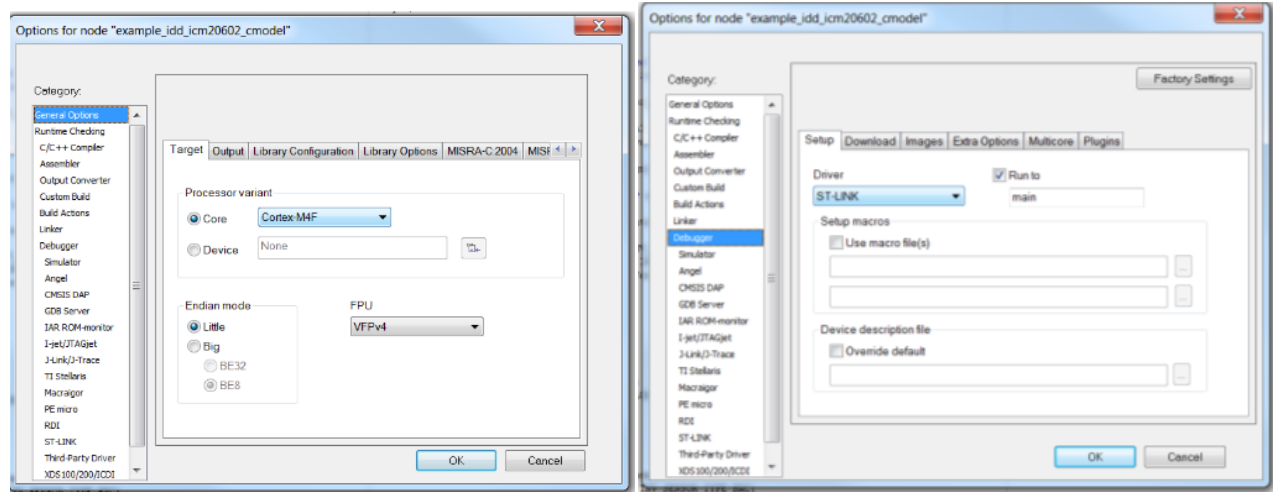
Please refer to the documentation for further details like *firmware upgrades, hex editing and many other advanced services*, which can be found here

http://www.st.com/content/ccc/resource/technical/document/user_manual/e6/10/d8/80/d6/1d/4a/f2/CD00262073.pdf/files/CD00262073.pdf/jcr:content/translations/en.CD00262073.pdf

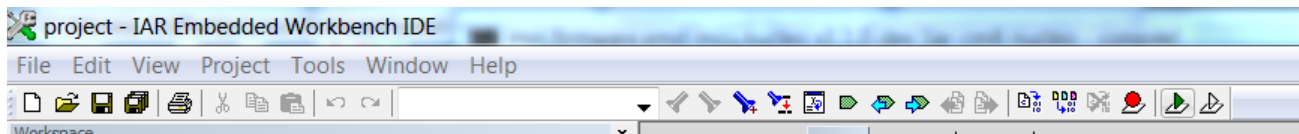
To run **IAR Workbench**:

IAR Workbench is an IDE to develop and debug the source code to work with IMU. The steps on how to download IAR Workbench has been discussed previously in this document.

1. Once the software is installed, we should be able to open the project saved in **EWB** file format in the IDE.
2. On the left side, we see all the available *source files including headers*. The high-level code can be found in **example.c** file.
3. To check the configuration of the project, left click on project options in “**General Options**” => “**Target**” that the core is properly set to **Cortex-M4F**. And in **debugger Category**, check that **ST-LINK** is selected.



4. To compile and debug the application, first connect the micro USB to the PC linked to the IMU.
5. On the Menu bar of IAR IDE, select **Download and Debug**.



6. This will download the application on the ST Nucleo board. We can debug the software line-by-line or block-by-block and fix bugs and errors.
7. Please refer to IAR Workbench *documentation* for further details.

APPENDIX C – HOW TO WORK WITH BOTH RADAR AND IMU SIMULTANEOUSLY

Raspberry Pi comes pre-installed with Python. The final python script is titled **app.py**. This script can produce real time readings from both IMU and radar simultaneously. Prerequisites for the said python script is a python library called **pySerial**. To install pySerial, open terminal (cmd) and type the following command.

sudo pip install pyserial



```
LXTerminal
File Edit Tabs Help
root@raspberrypi: /home/pi# sudo pip install pyserial
Requirement already satisfied (use --upgrade to upgrade): pyserial in /usr/lib/python2.7/dist-packages
Cleaning up...
root@raspberrypi: /home/pi#
```

To run the python script, open terminal and type following command,

python app.py

Where the app.py holds the consolidated program to measure real-time readings and can be found on a supporting Compact Disk or Web resource.

APPENDIX D – DEVELOPING CUSTOMIZED SOLUTIONS

General Instructions

If one seeks to develop the software with custom requirements fulfilled, then one should use the *Development pathway* described below to complete the task.

- Downloading **Anaconda** –
 1. Proceed to Anaconda's download website to equip with the IDE (Integrated Development Environment) required by us to configure and use the RADAR.
 2. Select python 2.7 and the Anaconda's package will be loaded.
 3. The website is <https://www.anaconda.com/download/?lang=en-us>
 4. Documentation: <https://conda.io/docs/user-guide/install/index.html>
- Follow the steps described in the documentation of anaconda or the onscreen setup wizard to complete the installation.
- Verification of installation
 1. Once installed, open a console window on windows, by searching for it on Start Menu.
 2. We should verify if conda was installed by typing "conda" in command prompt.
 3. If installed, it should show some auto-generated text.
 4. If you see error, please visit the troubleshooting section of the guide. Easy fix: Possible error with PATH environment variable. Consoles need to be restarted to access conda.
- Installing **pyserial**:
 1. To install *pyserial*, we should re-open a console window, and type the command
`conda install pyserial`
- Launching **spyder** (Python IDE)
 1. Launch *spyder* either from command prompt or from anaconda navigator.
- Checking device COM port & initialization:
 1. Follow the steps provided below to start *Device Manager on Windows*.
 2. In the *Device Manager*, note the port of the *USB Serial Device* attached, which reads the RADAR Components device ID or name.
 3. In the sample code, note the initialization of the serial library and COM port. Here you must change the COM port to the one noted in step 2.
- Reading data from RADAR
 1. Setup the device to begin acquiring readings.
 2. Next, navigate to the folder containing sample program file "rad.py"
 3. Hold *Shift + Right* click to open the context menu in the folder containing the file. Next, select open Command Prompt here.
 4. Once inside the CMD line, enter "python rad.py" to receive the distance as the output, in meters.

This next section provides the details of using the RADAR. We demonstrate the code related to setting up and using the RADAR Data. The RADAR data requires post-processing to generate any real data. Hence, we use the following imports in our rad.py python program. Some of the imports can be skipped if the calibration equation has been derived previously.


```

####-I-M-P-O-R-T-S---#####
#####
import time #built-in Time library, sleep function to pause CPU

#####
import math #built-in Math library, math functions like sine, cos etc

#####
# py-serial library provides various support functions for working with serial port devices which help automate our job.
import serial

#####
# pandas is a data analysis support library that provides the necessary tools such as data structures and math functions on such
# strpductures
import pandas as pd

#####
# SciPy provides a discrete fourier transform library
from scipy.fftpack import fft

#####
# mathPlot library provides a function for plotting values on a graph
import matplotlib.pyplot as plt

#####
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along
# with a large collection of high-level mathematical functions to operate on these arrays.
import numpy as np

#####
# SciPy is an open source Python library used for scientific computing and technical computing.
import scipy
#####

```

Next, we setup the *pyserial* object with appropriate port number (**port-number/baudrate can be set/verified in *Control Panel/Device Manager***), along with its baudrate. These values are available in the data sheets of the devices.

```

#####
#Setting up the Port and Data channel speed. Initialization of communications port.
ser = serial.Serial(
    port='COM4',
    baudrate=115200,
)
#####
# Printing the connection to serial port status, if OK, then proceed.
print ser.isOpen()
#####

```

Then, we initialize the variables required by the program.


```
#####
# Initialize a small list of commands to be sent over the serial connection, which help to initialize, sweep, trace and collect
# the radar data.
list_of_commands = [
    'hard:syst rs3400w',
    'hard:syst ?',
    'INIT',
    'SWEEP:MEASURE on',
    'SWEEP:NUMBERS 1',
    'TRIG:ARM',
    'TRACE:DATA ?',
    'TRACE:FFT:CALCulate',
    'FREQUENCY:IMMEDIATE ?',
    'TRACe:READ:DIStance ?',
    'Trace:read:FINDeX ?',
    'Trace:read:PINDeX ?'
]
#####
# Initialize certain variables
measuredDist = 0.0 #Floating value initialization
#####
```

Finally, here is the sample code for the main portion of the program:

```
#####
#Start a try-catch block to handle exceptions and prevent program crash
try:
    # Loop over the list of commands one by one
    for command in list_of_commands:
        ser.write(command + '\r\n') # Push the command to RADAR device and expect to receive feedback from it.
        out = ''
        print command # Display the current command to the user for feedback.
        if(command == 'TRACE:DATA ?'):
            print 'if' # Tracing for debugging purposes
            time.sleep(1)
        else:
            print 'else' # Tracing for debugging purposes
            time.sleep(.4 )
        # Check status and wait to receive the output from the RADAR.
        while ser.inWaiting() > 0:
            out += ser.read(1)
        print out # This object holds the output data strings.
        # Showing the measuredData from RADAR.
        if(command == 'TRACe:READ:DIStance ?'):
            out = out.replace("\r\n", "")
            x = float(out)
        else:
            print out
    # The serial library allows to close the serial port connection once done
    ser.close()
    # Accurate distance generation using curve fitted calibration formula
    dist = -0.003754*x*x - 1.391992*x + 128.851509
    print 'dist:::'
    print dist
except:
    #Error handling
    print "error"
    ser.close()
```

This program code is just for reference, the actual curve fitted calibration equation needs to be calculated one time using some basic tests as mentioned in the report. The structure of the program remains the same but there might be some changes required to produce the expected results.

APPENDIX E – PYTHON LIBRARY DETAILS

✓ *pandas*

pandas is a [Python](#) package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. [33]

✓ *numpy*

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the [BSD license](#), enabling reuse with few restrictions. [32]

✓ *scipy*

SciPy refers to several related but distinct entities:

- The *SciPy ecosystem*, a collection of open source software for scientific computing in Python.
- The *community* of people who use and develop this stack.
- Several *conferences* dedicated to scientific computing in Python - SciPy, EuroSciPy and SciPy.in.
- The [SciPy library](#), one component of the SciPy stack, providing many numerical routines. [32]

✓ *py-serial*

This module encapsulates the access for the serial port. It provides backends for [Python](#) running on Windows, OSX, Linux, BSD (possibly any POSIX compliant system) and IronPython. The module named “serial” automatically selects the appropriate backend. [35]

✓ *matplotlib*

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and [IPython](#) shell, the [jupyter](#) notebook, web application servers, and four graphical user interface toolkits. [34]

Device Manager

These can be setup in the Device Manager section of the Control Panel in Windows. To access the Device Manager, please follow these steps.

- Click the bottom-left Start button on desktop, type “*device manager*” in the search box and tap **Device Manager** on the menu.
- Or else, Open **Device Manager** from Quick Access Menu. Press **Windows** + X to open the menu and choose **Device Manager** on it.
- One might also Access **Device Manager** in Control Panel. [Google]

Once device manager is visible, we can find the device under **Ports (COM & LPT)**. The RADAR should be listed under this section only.

Note: Device Manager is used to setup both the devices.