



**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***

01676411

***Narasimha Prashanth C R***

01669930

***Samartha K Venkatramana***

01654104

## INTRODUCTION

Civil, mechanical, and aerospace structures continue to routinely be used despite the fact that 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. This information is 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 have to 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.

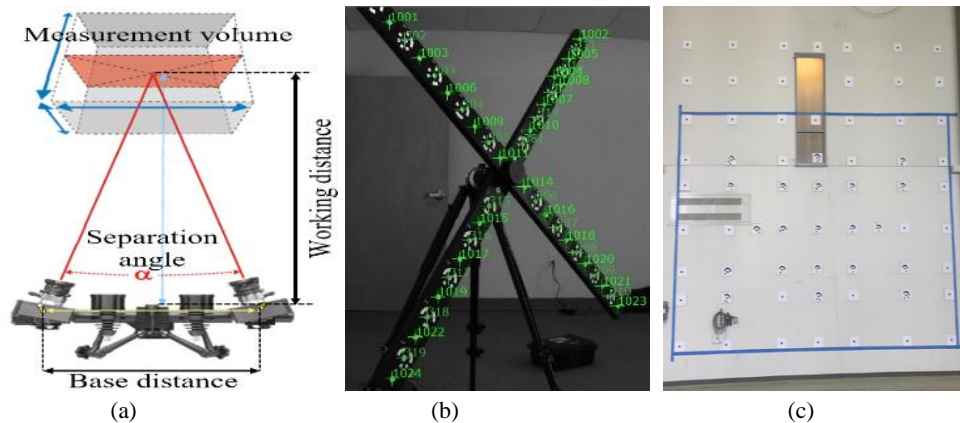


Figure 1. a) Visualization of a typical 3D-DIC camera system and calibrated measurement volume; b) example of calibration target recognition on a coded calibration cross having dimension of 1m x 1m; c) a very large wall mounted with optical targets being used for calibration of a  $\sim 7\text{m} \times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

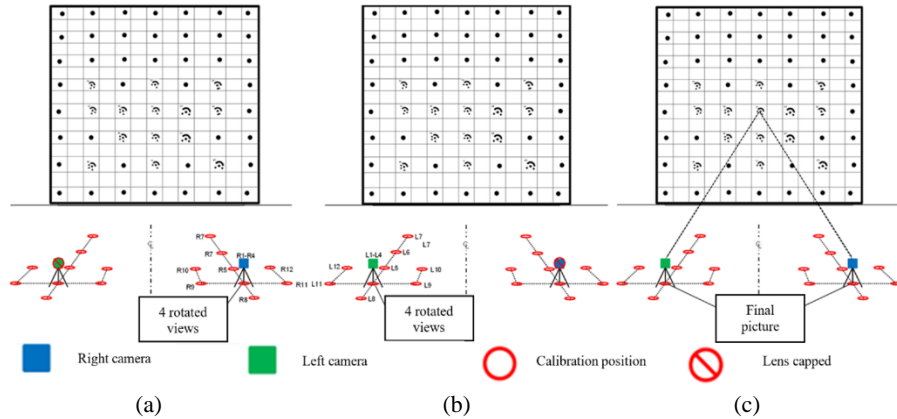


Figure 2. Schematic of a large-area calibration procedure using 14 coded and 42 un-coded targets over a calibration surface with dimensions approximately  $7\text{m} \times 7\text{m}$ : (a) calibration for the right camera requiring images taken from 12 different views of the target array; (b) calibration 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 or not 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.

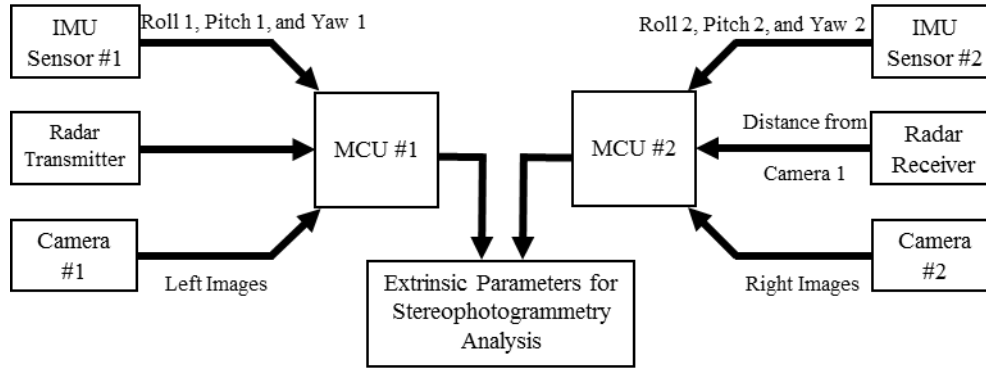


Figure 3. Flow diagram of the proposed sensor board embedded on a Microcontroller Unit to be installed on a set of cameras.







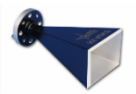

IMU sensors have been widely used for providing low-cost but accurate real-time navigation, guidance, and control of unmanned aerial vehicles focus on delivering reliable detection and localization to maintain a stable attitude and a correct heading direction during flight [13 - 15]. The use of IMU module 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 of-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 object 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.

Table 1. Used components and functions.

IMU			Radar			MCU		
Equipment	Purpose	Image	Equipment	Purpose	Image	Equipment	Purpose	Image
ICM-20948	Sensing of roll, pitch, and yaw angles (w/an electronic compass)		RS3400W/04	Distance measurement		Raspberry Pi 3 Model B	IMU and radar sensors data acquisition and synchronization	
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							

To work, both IMU and radar have been programmed using suitable codes. In particular, 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  $\lambda$ . For a 77GHz radar,  $\lambda$  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.

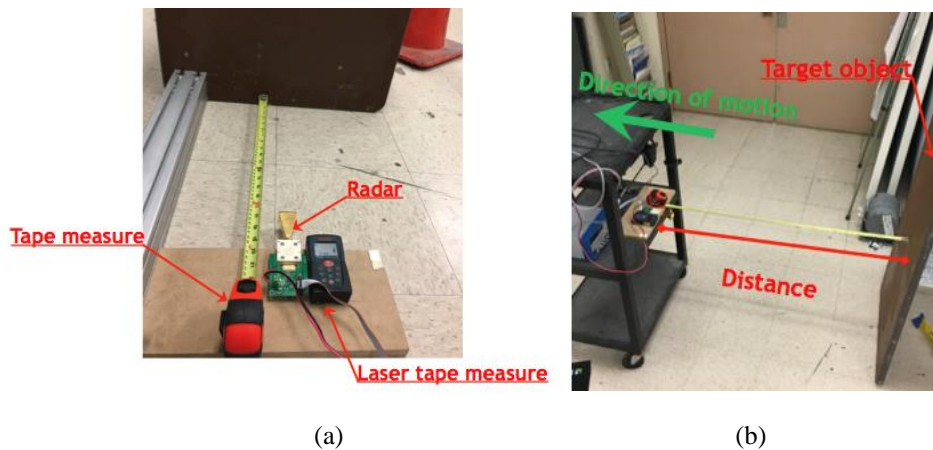


Figure 4. Experimental setup for radar calibration: (a) deployment of the three different systems used for comparison; (b) experiment procedure.

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.

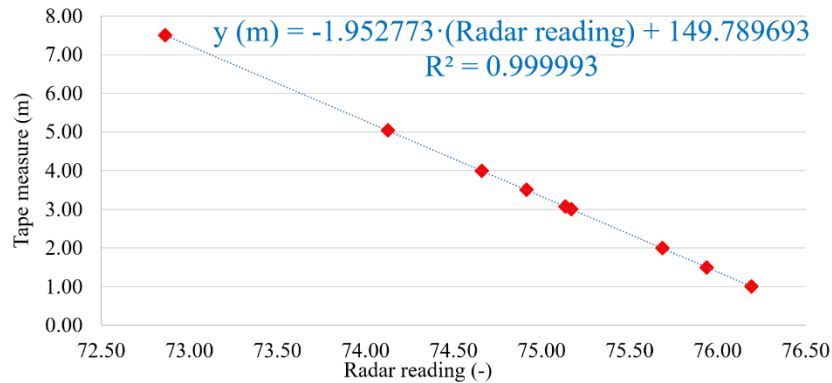


Figure 5. Calibration curve and scale factor for determining distance in meter from radar's readings.

A significant number of measurements allowed for data averaging and statistical parameters determination (e.g., average and standard deviation,  $\sigma$ ). 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.

[ADD COMMENT TO THE FIGURE]

#### 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:

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

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

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

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 similar to that 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,  $\sigma$ ).



Figure 7. Experimental setup for IMU characterization.

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 Table 2/Figure 8.





Figure 8. Comparison of the roll and pitch angles measured using cellphone and the IMU sensor.

[ADD COMMENT TO THE FIGURE]. No information about the yaw angle has been determined at this time.

## 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 a comparison with traditional measurement devices is presented to determine the accuracy of the proposed system.

A number of laboratory experiments have shown that the accuracy of the radar in measuring distances in a range 1 to 8 meter is equal to  $\text{XXX} \cdot 10^{-3}$  m, while the IMU can detect change in the orientation with an accuracy of  $\text{XXX}^\circ$  when a back-to-back comparison with a tiltmeter embedded in a smartphone is performed.

The proposed system will 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 camera as 3D-DIC measurements 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 stereophotogrammetry 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] 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/>
- [30] 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>
- [31] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55
- [32] Liechti, C. (2013). Welcome to pySerial's documentation--pySerial 2.6 documentation. website: <https://pythonhosted.org/pyserial/index.html>

## APPENDIX A – USER GUIDE

### ICM20948 – Inertial Measurement Unit Setup and Usage

The Embedded Motion Driver (eMD) is designed as a solution that can be easily ported to most MCUs. The eMD 20x48 release package contains the example project for STM ARM Cortex M4 Core. Using IAR Workbench, we compile the files needed to flash the IMU for setting it up to get the 9-axis measurements. We then go into MotionLink and configure the correct connection port before starting the measurement log. MotionLink provides us with graphical plots to visualize the sensor data as well as read/write to hardware registers. We log the data to external files, 72 measurements – 1 for every 5 degrees in a 360-degree motion of the setup; each of the 72 measurements containing the readings for every axis of the 9-axes. We take these 72 measurements in 3 orientations – along the x-axis, y-axis and z-axis, tilting the setup as required. Once we get the measurements for every orientation, we need to calibrate them. We use the pandas library in Python to compute the mean and total deviation. We can then plot the graphs for each of the axis at every degree using the matplotlib library in Python.

#### ICM Setup

IAR workbench is required to program the IMU component. One can also use the MotionLink software to view realtime 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 very limitedly.

Steps to setup the IAR Workbench:

- Download the IAR Embedded Workbench for ARM from [www.iar.com/iar-embedded-workbench](http://www.iar.com/iar-embedded-workbench), by choosing the architecture as “ARM”.
- Once completed, please run the setup program to install the workbench.
- While installing the drivers, please make sure the STM driver is installed.
- 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](http://www.st.com) to access these files.*
- Extract the file and install ST-LINK v4.2.0
- Install the drivers downloaded from <http://www.st.com/en/development-tools/stsw-link009.html> by choosing the appropriate architecture x86 or amd64.
- 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 latest version. In future if there are any firmware upgrades by STM, we have to download and install the file from the above link.
- {{MORE STEPS}}

The relevant code is shown below:

[CODE]

#### RADAR – Setup and Usage

The first thing we had to decide when using this sensor was to determine the choice of antenna. The antenna choice has a significant impact on the range and field of view of the final sensing system. For our application, we are using a horn antenna since this type of antenna is best suited for outdoor use. The communicate with the development board connected to the RADAR, we need to use a terminal emulation program such as *putty*. However, since we must take many readings, we cannot rely on inputting commands every time and, so we wrote a python program using the ‘serial’ library that does this work for us.

The initial configuration for the RADAR as per the datasheet is as follows:

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

## 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.

If one seeks to use the existing code to test or as a sample for production, be advised, we are not responsible for any damages, as it is just a sample. Please use it at your own risk. The *Production pathway* is described below.

- Install python from their official website. Download version 2.7,
- Verify that it was installed by checking on command line by executing the command "python --version"

- 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.
- Install all the dependencies by executing the following commands:
  - ✓ pip install pyserial
  - ✓ pip install scipy
  - ✓ pip install numpy
  - ✓ pip install pandas
- Start the device manager, note the port of the RADAR device, listed under COM & PORTS as a USB Serial Device and make the change in rad.py at the line where pyserial library is initialized.
- Open the current working folder and run the command prompt. Next run the program by typing “python rad.py”
- You should now be able to generate the measured distance on the screen by repeating this step.

This 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 buadrate. 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()
```

## 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.



## APPENDIX B – 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. [30]

### ✓ *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. [29]

### ✓ *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. [29]

### ✓ *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. [32]

### ✓ *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. [32]