DEPARTMENT OF MECHANICAL ENGINEERING
&
DEPARTMENT OF COMPUTER SCIENCE

University of Massachusetts Lowell

COMP.7060 – Directed Research

Development of a Radar-IMU system for 3D-DIC measurements using remotely paired UAVs

Technical Report

STUDENTS: ***SUPERVISORS:***
M. SAMEER KHAN (01676411) **PROF. C. NIEZRECKI**
NARASIMHA PRASHANTH C R(01669930) **DR. A. SABATO**
SAMARTHA K VENKATRAMANA(01654104)

Academic year 2017-18

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

1 - INTRODUCTION

Digital Image Correlation is a newer technique which uses sensitive optics, to find the deformation, peaks etc in large deformation problems. However, in our project, we use this technique to capture 3D-DIC so that we can measure and analyse the structure in question. Further, this method is useful for different applications, largely towards very wide and tall structures. 3D-DIC is an offset of specular light analysis, where the specs are arbitrarily done and statistical data processing is fully utilized to interpretation of data. The data received from this technique needs to be post processed, for us to interpret it. In this project, we utilize this technique to implement drones enabled with cameras or IMU and Radar, in swarms of 2 or more, to stereotypically capture the data and use the 3D-DIC to process it, and analyse the results.

The objective of this project involves the use of a RADAR-IMU system for 3D-DIC measurements by using two remotely paired drones. The aim of this project is twofold - first, getting the RADAR and IMU calibrated and working synchronously and second, mounting them on two separate drones that are remotely paired. Through our project we found that calibration itself was a strenuous process. The distance between the drones and any other objects in the environment is measured using the radar, which provides with a multi-tuple value. This set of values is then processed through MATLAB by plotting Fast Fourier Transforms and getting the peak values from the graphs generated. Similarly, for the IMU, we use the required libraries and softwares to generate real-time data. The process involved in this is that we need to move the IMU slowly under small increments. We capture each data element for all the angles one by one and generate corresponding graphs to calibrate the IMUs accuracy.

After verifying the results through the graphical data carefully, we are ready to assume that the sensors are now accurate. Next, we mount them on a drone and attempt to perform the same measurements so the two drones can work synchronously without any problems. This report describes the entire process, equipment involved, techniques used and further, analyses the result and determines the accuracy of the calibration.

2 - METHODS

2.1 List of equipment used:

• ***Hardware***:

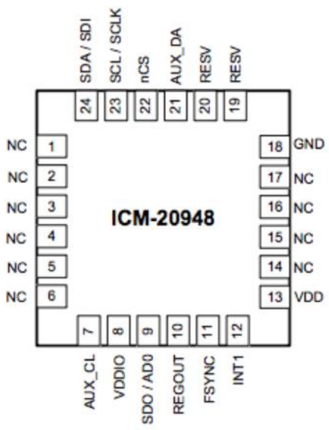
- Nucleo STM32F411RE development board
- InvenSense carrier board
- ICM20948
- FMCW Radar – RS3400W
- Voltage Regulator

• ***Software***:

- IAR Embedded Workbench 7.70
- Anaconda (for Python 2.7)
- Embedded Motion Driver (eMD) ICM-20948 v1.0 for Nucleo Board
- MotionLink v2.0.9
- MATLAB for computing Fast Fourier Transform for Radar
- TeraTerm for UART interface with Radar

2.2 The Sensors

First, for receiving data from surroundings we have various sensors placed strategically, in the system we are implementing, as we require different type of sensor measurements. The first sensor in our system is the Inertial Measurement Unit (IMU). Many Inertial Navigation Systems (INSs) including ours use an IMU to measure acceleration and direction of motion (through gyroscope). The IMU we are using for this project is the InvenSense ICM20948. The Embedded Motion Driver is an embedded software stack of the sensor driver layer that easily configures and leverages many of the features of InvenSense motion tracking solutions. The ICM-20948 is the world's lowest power 9-axis MotionTracking device that is ideally suited for Smartphones, Tablets, Wearable Sensors, and IoT applications. It uses only 1/3rd the power of existing 9-axis devices and has 3-axis gyroscope, 3-axis accelerometer, 3-axis compass, and a Digital Motion Processor (DMPTM) in a 3x3x1mm (24-pin QFN) package.



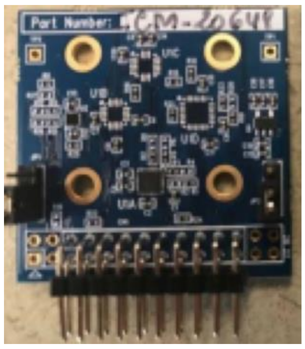


Fig: Pin out Diagram and actual image for ICM-20948 3 mm x 3 mm x 1 mm QFN

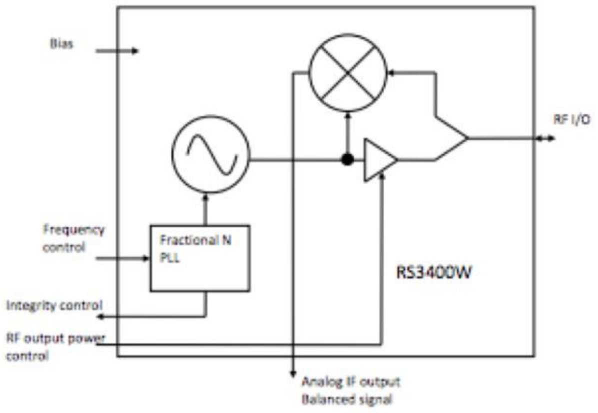
To use the ICM20948, we connect it to an STM ARM Cortex M4 board, in our case, a STM32 Nucleo-64 development board with STM32F411RE MCU. This interface provides a very reliable access to the component, through MotionLink software or C/C++ using their Workbench. This whole setup includes an interface for the IMU itself, and a NUCLEO board, all powered individually, using USB Cables. No external power supply is required. The setup is easy to achieve and work with, as show in the figure below.

The second sensor in this system is the FMCW Radar – RS3400W. Frequency Modulated Continuous Wave radar differs from pulsed radar in that an electromagnetic signal is continuously transmitted. FMCW radars in the GHz spectrum provide excellent distance measurement performance in applications where high accuracy, repeatability and reliability are needed. Because of the short wavelength, resolutions on the order of 2 cm can be achieved over distances of 20-30 meters.

Fig: STM32F411RE Development Board

Due to the non-contact nature of the measurement system, and due to the nature of the microwave, FMCW radar operating in the GHz range also exhibit excellent resistance to dust, steam, heat, etc. This allows for use in demanding conditions and hence we decided to use it for our project. The RS3400W/04 is a W-band FMCW radar front end featuring synthesized frequency sweeps. A fast sweep mode enables complete sweeps below 1ms. The system provides one Tx/Rx with a WR12 waveguide interface suitable for attaching compact high directivity antennas.

The setup involves the usage of an external power supply of 12V to power the radar system. Measurements of the radar are received on the terminal window of the attached computer. Connections are through USB on the PC and a visual cable(VGA) on the interfacing board. The radar component is installed on the tightly coupled interface board. Commands can be sent and data can be received through the USB connection to PC and a U ART Term. The data received must be processed to be able to calibrate the device. The setup is shown below.



<- Fig: Block diagram for RS3400W/04

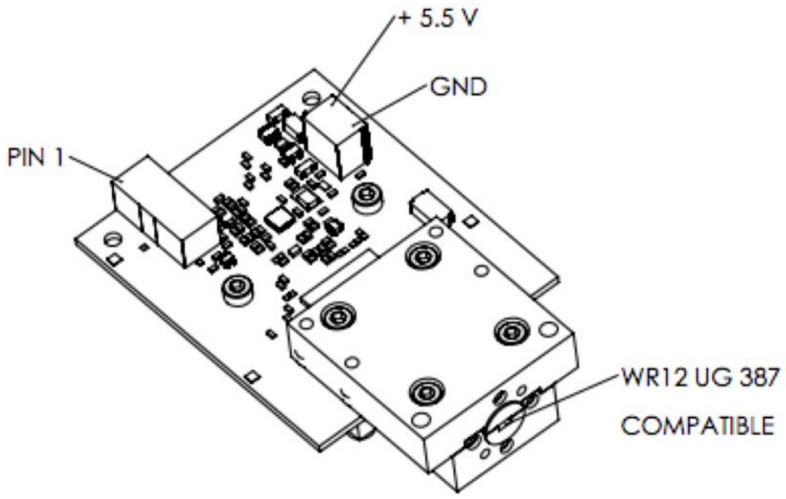


Fig: RS3400W/04 Interface **->**

2.3 *Procedure*

IMU MotionLink measurements:

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. The relevant code can be found at https://github.com/prasanthnarasimha/COMP-7060-731-/tree/master/ICM20948

The results for the graph plots can be seen in the figure below.

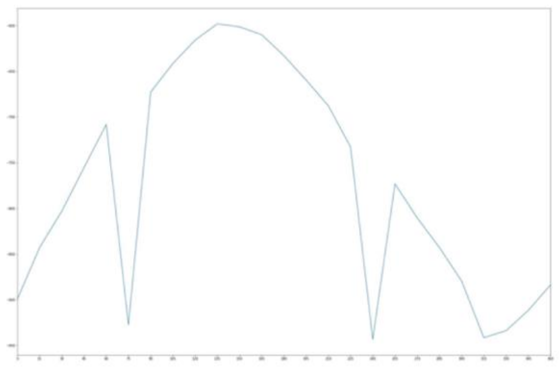


Fig: AccelX graph plot

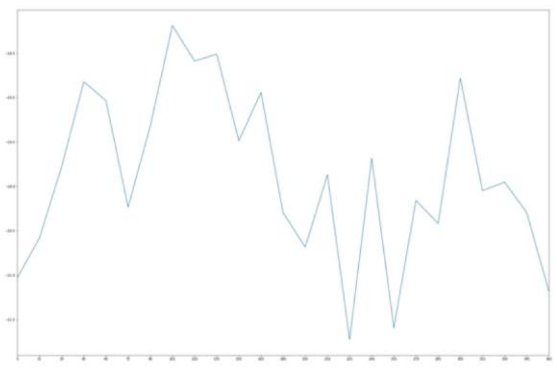


Fig: GyroX graph plot

Radar Measurements:

However, for the RADAR, the process is slightly different. 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 relevant code can be found at Appendix B.

The initial configuration for the RADAR is as follows:

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

Some of the command that can be sent to the system are:

- **INIT:** To initialize the system
- **SWEEP: MEASURE ON**
- **SWEEP: NUMBERS <X>**
- **TRIG:ARM**
- **TRIG:DATA ?**

The last command TRIG:DATA ? gives us the measurements that we need based on the parameters that we have set above. We save the measurement data to text/csv file so we can use it later to plot the Fast Fourier Transforms (FFT).
For plotting the FFTs we first use the pandas library in Python to order the data received as output and then use the scipy library to plot the graphs.

2.4 Steps of Operation

2.4.1 ICM connection and running the code

Initial Steps:

1. Clone this repo and extract source.zip file
2. Inside the extracted folder, we have an already compiled and built bin file ready in release/bin/.bin
3. To start right away, download ST LINK and open the installed software.
4. After opening ST LINK, click on Target -> Connect.
5. Then, Click on Target -> Program and select the bin file in release/bin folder and Click program.
6. Once the programming is done, the flash memory restarts and is ready to use.

To Get Sensor data

We have setup two UARTs, one for logging and one for getting the sensor data. The configuration of both UARTs are as follows:

LOG UART

```
uart_config.uart_num = LOG_UART_ID;
uart_config.irqs_on = 1;
uart_config.use_for_printf = 1;
uart_config.use_dma_for_tx = 0;
uart_config.use_dma_for_rx = 0;
uart_config.tx_buffer = uart_logTx_buffer;
uart_config.rx_buffer = NULL;
uart_config.tx_size = UART_LOG_TX_FIFO_SIZE;
uart_config.rx_size = 0;
uart_config.baudrate = 921600;
uart_config.rx_interrupt_cb = NULL;
uart_config.tx_interrupt_cb = NULL;
uart_config.rx_context = NULL;
uart_config.tx_context = NULL;
```

Main UART

```
uart_config.uart_num = MAIN_UART_ID;
uart_config.irqs_on = 1;
uart_config.use_for_printf = 0;
uart_config.use_dma_for_tx        = 1;
uart_config.use_dma_for_rx        = 0;
uart_config.tx_buffer             = NULL;
uart_config.rx_buffer             = uart_mainRx_buffer;
uart_config.tx_size               = 0;
uart_config.rx_size               = UART_MAIN_RX_FIFO_SIZE;
uart_config.baudrate              = 2000000;
uart_config.rx_interrupt_cb       = NULL;
uart_config.tx_interrupt_cb       = NULL;
uart_config.rx_context            = NULL;
uart_config.tx_context            = NULL;
uart_init(&uart_config);
```

We can connect to these UART's using putty or hyper terminal or even python using pyserial.

Sensor-cli

1. To implement the cube program, we use sensor-cli. It is located in the extracted folder tools/sensor-cli.exe
2. Navigate to the above specified folder through command prompt and type the following command: sensor-cli --target=emdwrapicm20x48,port=\\.\COMxx --adapter=dummy COMXX being the comport of daughter board
3. If the device is correctly connected and flashed, this will display sensor-cli> prompt waiting for user input.
4. help command provides list of commands available.
5. type cube on grv and then en grv 20 to start the cube program.
6. To stop the sensor output, type dis grv

To make changes to the code

1. To modify the existing code, open the project in IAR Embedded workbech and open example.c
2. Make the necessary changes and then click on Project->build all.
3. Once the build is completed, go to Project->Download and Debug. The modified code is compiled and downloaded to the flash memory. Once you're certain that the code is running as expected, click on run.

2.4.2 RADAR connection and running the code

Hardware setup:

1. Connect a suitable antenna (pyramidal horn antenna in this case) to the 77GHZ Radar(RS3400W) and supply 5.5v (+-1.5v)
2. Connect this radar to a controller board (CO1000A/01) through appropriate cable and supply 12v to controller board.
3. Connect this whole setup to a pc/raspberry pi using a RS232 to USB cable.

Software setup:

1. Install Python 2.7 from this link

2. Once python is downloaded and installed, install pyserial, scipy, numpy, pandas libraries from pip using the following commands:

- pip install pyserial
- pip install scipy
- pip install numpy
- pip install pandas

3. Open device manager and note down the port number of usbserial device and change the port number in line 14.

4. Navigate to current folder through command prompt and type `python rad.py`

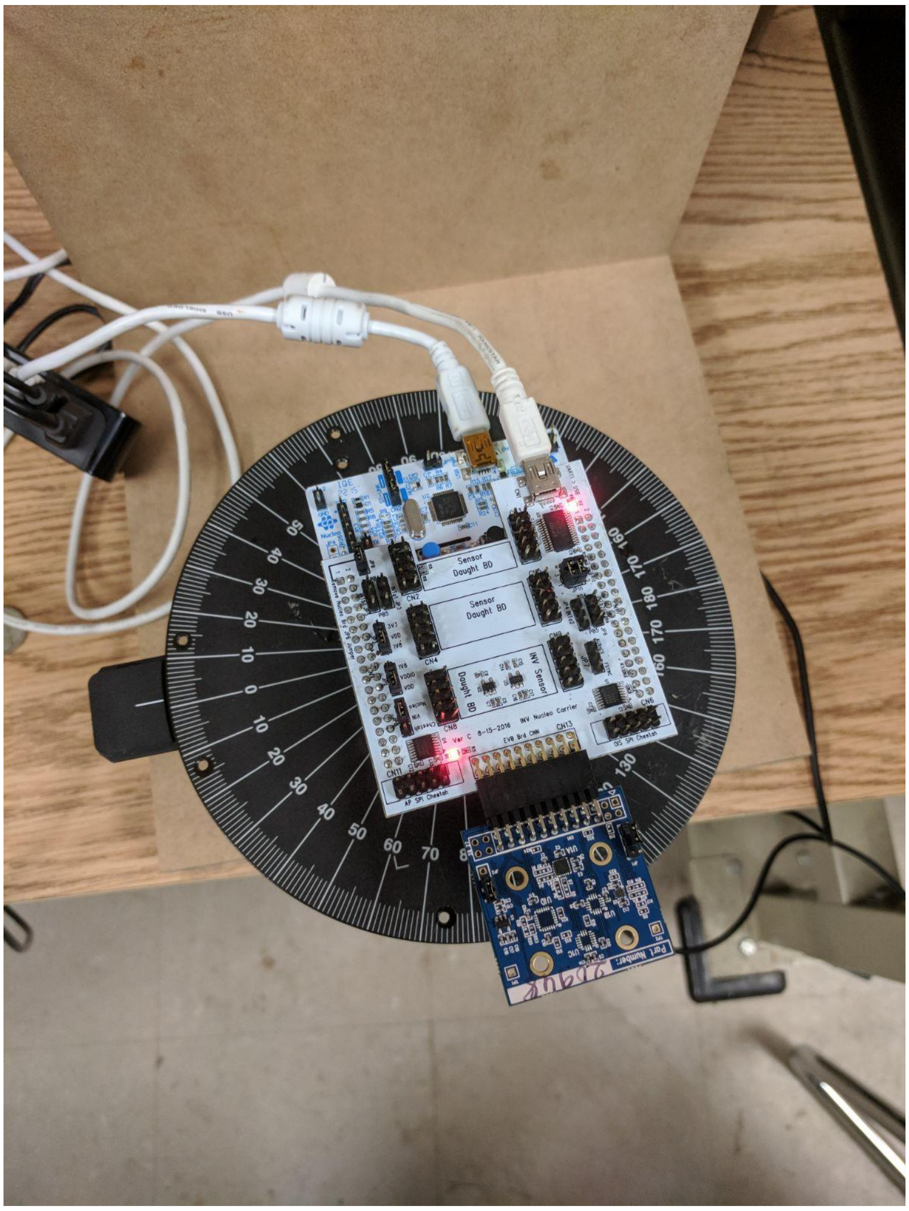
5. You can see the distance as output in the command prompt

3 - TEST CASES STUDIED / TESTING PERFORMED

Since all we get from the devices is raw data, we need to do a lot of calibrating to get accurately measure each sensor.

3.1 Testing for ICM-20948:

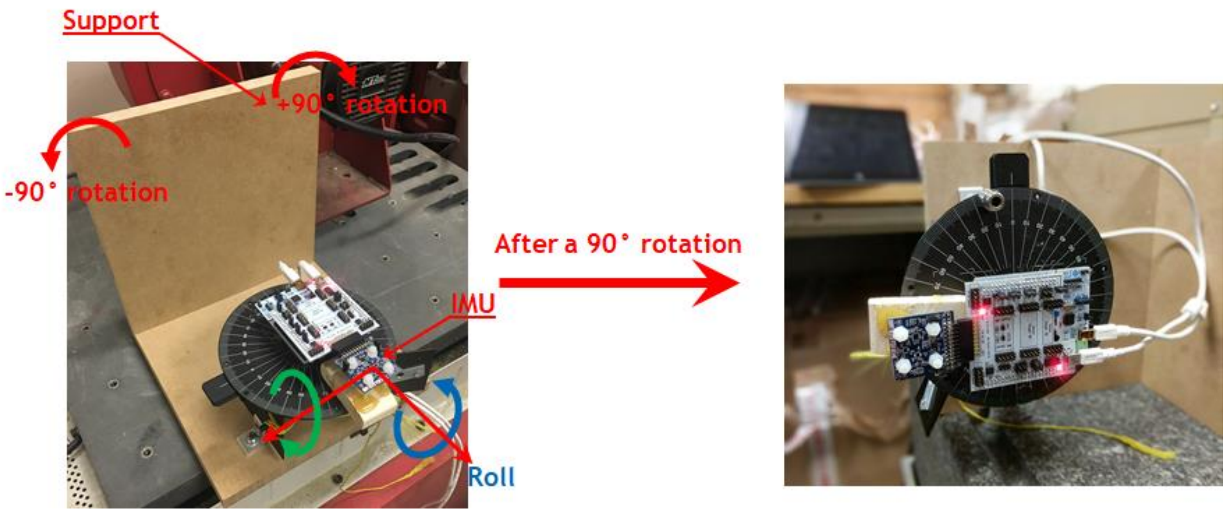
The ICM 20948 is a 9-axes device. It contains 3-axes Gyroscope, 3-axes Accelerometer and 3-axes Compass. In order to calibrate the ICM 20948, the device is fixed to a dial with marked readings from 0 degrees to 360 degrees like the picture below.



To make sure we do not affect the readings in any way, we have fixed the device to the board using hot glue and placed the entire setup on a heavy object which will prevent small fluctuations.

We have taken readings for all the three sensors(Gyroscope, Accelerometer and Compass) for every 5 degrees. The readings taken are the average of readings for one minute. In this process, we assume the temperature is constant at the room temperature.

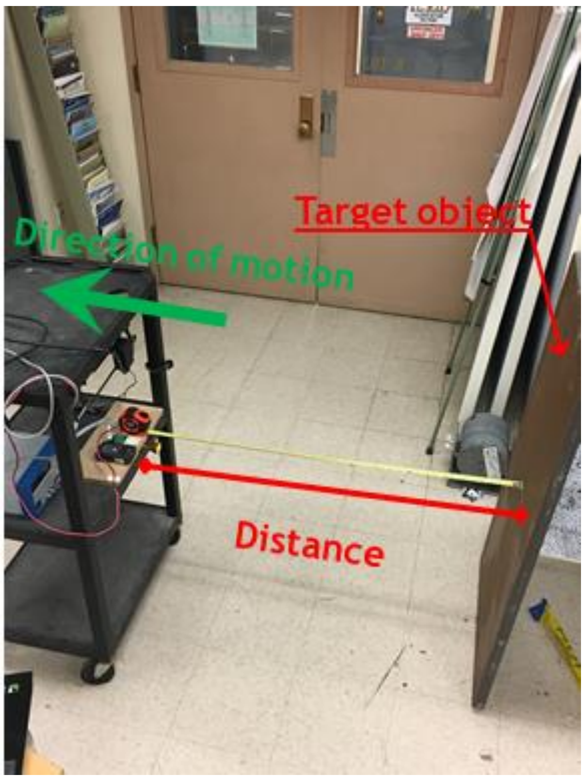
To correctly calibrate pitch and roll, we had to test and analyse the sensor readings in various orientations like the figures below.



In the first picture, we observed that roll is ~ 0 and pitch is also ~ 0. In the second picture, we can see that the sensor setup is rolled about 90 degrees. We observed that roll is ~90 and pitch is still ~ 0, which is what we expect the readings to be. In the same way, the sensor setup is rotated 90 degrees to measure the pitch as 90 degrees.

3.2 Testing for 77GHz FMCW Radar :

The radar used is 77GHz radar manufactured by Sivers IMA. To accurately measure Radar's readings, we have taken the average of readings for 30 seconds. We have used two other means to calibrate the measurements from Radar. We used a tape measure and digital measure.



The radar is calibrated using the above testing method.

4 – ANALYSIS OF THE RESULTS

After performing the experiment, we have generated a set of tables to capture the analysis. We have measured it against the physical systems for their dimensions.

4.1 IMU Results

The IMU was measured against a smartphone, equipped with the required sensors, Accelerometer and Gyroscope. These sensors have a very low margin of error on a commercially configured smartphone. Upon verification with the mobile device, we were able to witness the error to be very low, in the order of less than 2 degrees. The following tables allow us to visualize the results.

This is a very low value, however due to difficulty conducting this test, we believe that the error might be due to human error, more than anything else. Hence, we believe the system is very accurate. The sensors on the phone and IMU are subject to interference, from electrostatic charges, electromagnetic waves of high intensity and other such attenuation or deflection of values.
These cases can be studied in further detail. However, we believe the noise from the environment is marginal and can be ignored.

Roll angle		
Cellphone (°)	**IMU (°)**	**Cellphone - IMU (°)**
-13.40	*-13.03*	*-0.37*
-1.10	*-0.25*	*-0.85*
-38.10	*-37.22*	*-0.88*
-87.60	*-86.81*	*-0.79*
10.10	*10.76*	*-0.66*
35.10	*35.62*	*-0.52*

88.10	*87.09*	*1.01*
	Average	**0.72**

Pitch angle		
Cellphone (°)	**IMU (°)**	**Cellphone - IMU (°)**
-12.10	-11.68	-0.42
1.10	3.24	-2.14
-31.30	-29.91	-1.39
62.52	63.32	-0.80
90.60	87.45	3.15
	Average	**1.58**

4.2 RADAR Results

The RADAR was targeted at a distant solid metal wall, and was moved away slowly in increments of 0.5m. This test was performed by taking multiple readings at each step, which allowed for significant data average and statistical parameters.

Tape		**Laser**		**Radar**	
x (10^{-3}m)	**σ (10^{-3}m)**	**x (10^{-3}m)**	**σ (10^{-3}m)**	**x (-)**	**σ (-)**
500.0000	**0.0000**	**495.8571**	**0.3499**	**76.7275**	**0.4517**

1000.0000	**0.0000**	**997.5714**	**0.7284**	**76.1912**	**0.0003**
1500.0000	**0.0000**	**1498.0000**	**0.9258**	**75.9374**	**0.0006**
2000.0000	**0.0000**	**1996.4286**	**0.7284**	**75.6866**	**0.0005**
2500.0000	**0.0000**	**2496.1429**	**0.3499**	**75.4353**	**0.0019**
3000.0000	**0.0000**	**2996.6667**	**0.4714**	**75.1702**	**0.0013**
3067.1429	**0.3499**	**3066.8571**	**0.3499**	**75.1349**	**0.0003**
3500.0000	**0.0000**	**3499.0000**	**1.0690**	**74.9139**	**0.0007**
4000.0000	**0.0000**	**3997.4286**	**0.4949**	**74.6596**	**0.0006**
5047.0000	**0.0000**	**5045.4286**	**0.4949**	**74.1265**	**0.0005**
7503.0000	**0.0000**	**7497.8571**	**1.3553**	**72.8619**	**0.0005**

Upon receiving these values, we were able to infer that the RADAR has an error margin of less than 4x10^{-3}m. This is not a very large error. RADAR has been tested against a tape measure and a digital measure. The chart above helps to visualize the data, which we have converted to (m) later. By capturing these values, we interpolated the values to derive an equation. The accuracy of this equation is around 99%. The following graph helps to explain this.
The RADAR provides readings in a raw number format, which needs processing. After processing, we receive the reading in meters. This has been plotted below, which shows that the error in meters is less than 4mm.

Tape	**Laser**	**Radar**	**Radar - Tape**	**Radar - Laser**

(10^{-3}m)	**(10^{-3}m)**	**(10^{-3}m)**	**(10^{-3}m)**	**(10^{-3}m)**
1000.000	997.571	1001.624	1.624	4.053
1500.000	1498.000	1499.803	-0.197	1.803
2000.000	1996.429	1991.792	-8.208	-4.636
2500.000	2496.143	2484.139	-15.861	-12.004
3000.000	2996.667	3002.909	2.909	6.242
3067.143	3066.857	3072.052	4.909	5.194
3500.000	3499.000	3504.199	4.199	5.199
4000.000	3997.429	4000.937	0.937	3.509
5047.000	5045.429	5040.677	-6.323	-4.752
7503.000	7497.857	7498.882	-4.118	1.025
Average Error (10^{-3}m)			4.928	4.842

Average error is on the order of $4*10^{-3}$ m, comparable with the minimum theoretical value of $2*10^{-3}$ m. Another chart explains the maximum probability of error over a range of 0.5m to 8m.

5 – CONCLUSIONS

The goal of this report and the directed study associated with it was to develop a Radar-IMU system for 3D-DIC measurements using remotely paired UAVs. The most important part of the system was to have absolutely accurate readings and measurements and so most of our work went into calibrating the IMU and RADAR system. This process was critical and worth the effort as now the system is recording measurements in our dimensions. Upon verification with other instruments, we can say that the system is very accurate. If the system succeeds in higher scale applications, or experiments, then it can be recommended for industry production. Further experimentation is required and is in the cards.

Finally, we can conclude that a novel sensor board embedding an IMU and a radar sensor has been designed and the software architecture for operating it has been prepared. The employed sensors have been characterized and calibrated in a controlled environment. The radar unit has shown to be capable to measure distances with an accuracy on the order of ~$4*10^{-3}$ m. The IMU sensor has been used for measuring pitch and roll angle with an accuracy of ~0.72° for roll and ~1.58° for pitch.

Future work can be devoted to completing the characterization of the compass in the IMU and porting our work to a single-board computer such as a raspberry pi or better.

REFERENCES

1. https://www.morf.lv/mems-part-1-guide-to-using-accelerometer-adxl345

2. http://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html

3. https://www.invensense.com/developers/login/

4. http://www.st.com/en/evaluation-tools/nucleo-f411re.html

5. https://store.invensense.com/datasheets/invensense/App-Note-eMD-20x48-User-Guide.pdf

6. https://store.invensense.com/ProductDetail/EVICM20948-InvenSense-Inc/597422/

7. https://siversima.com/wp-content/uploads/FMCW-Radar-App-Note.pdf

Appendix B

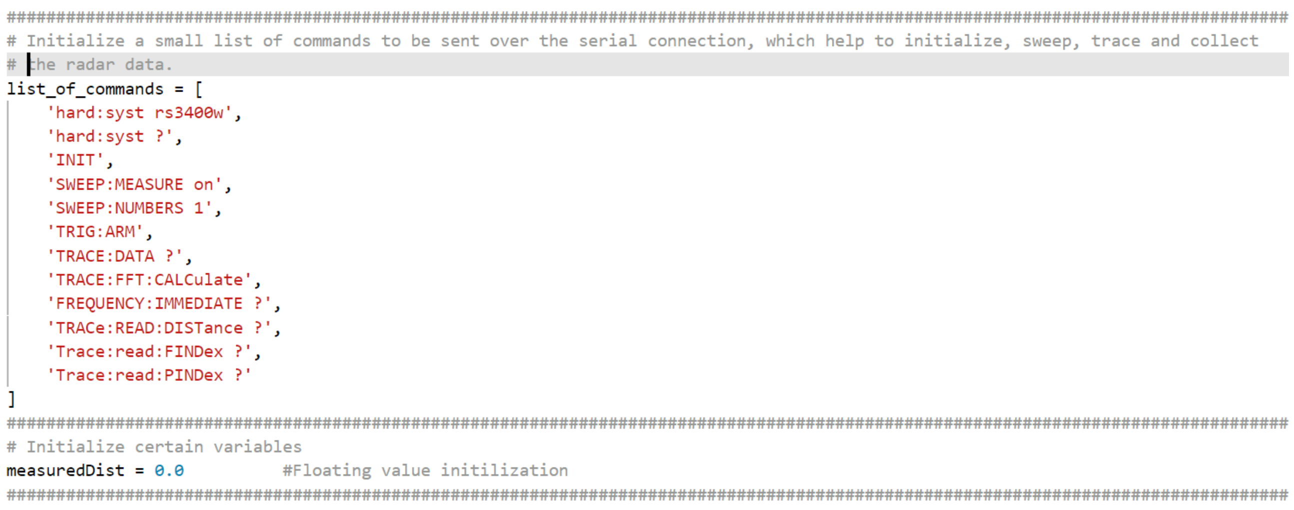
This section provides the details of using the RADAR. We demonstrate the code related to setting up and using the RADAR Data. The RADAR data requires post-processing to generate any real data. Hence we use the following imports in our radar.py python program. Some of the imports can be skipped if the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

```
###################################################################################################################################
#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()
###################################################################################################################################
####--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 py serial 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.

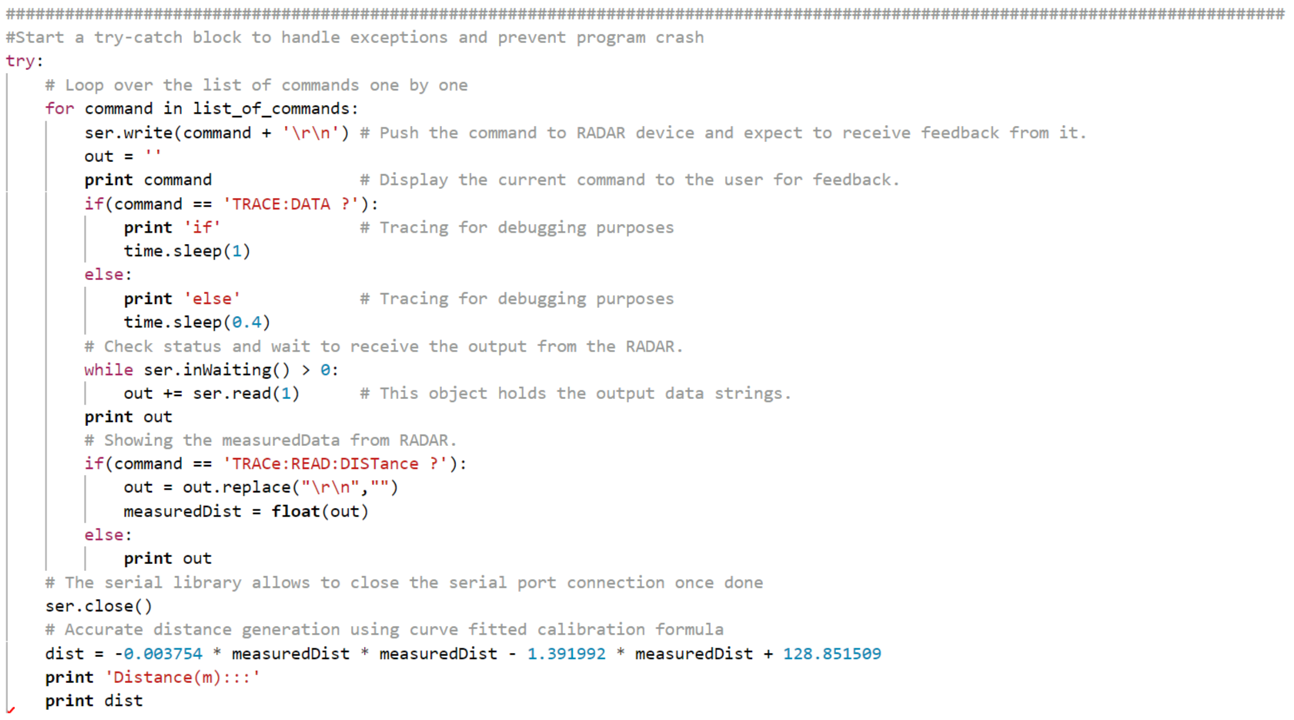
```
###################################################################################################################################
#Setting up the Port and Data channel speed. Initialization of communications port.
ser = serial.Serial(
    port='COM4',
    baudrate=115200,
)
###################################################################################################################################
# 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 initilization
##################################################################################################################################
```

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(0.4)
        # Check status and wait to receive the output from the RADAR.
        while ser.inWaiting() > 0:
            out += ser.read(1)       # This object holds the output data strings.
        print out
        # Showing the measuredData from RADAR.
        if(command == 'TRACe:READ:DISTance ?'):
            out = out.replace("\r\n","")
            measuredDist = 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 * measuredDist * measuredDist - 1.391992 * measuredDist + 128.851509
    print 'Distance(m):::'
    print dist
```