An Educational Platform for Testing and Evaluating Satellite Control Algorithms in a Real-Time and Frictionless Environment.

Khubaib Ahmad[1], Atheel Redah[2], Arslan Arif[1], Awais A Khan[3], M Kamran Saleem[1], and Sergio Montenegro[2]

[1]Electical Engineering Department, Faculty of Engineering, University of Central Punjab, Lahore, Pakistan.
[2]Institute of Aerospace Information Technology, Julius-Maximilians University, Wurzburg, Germany.
[3]Mechanical Engineering Department, University of Engineering and Technology, Lahore, Pakistan.

***Abstract* – Floating satellite (FloatSat) platform is utilized for testing and evaluation of Pico/Nano satellite in nearly a frictionless environment. In this paper, we have presented its basic setup and applications regarding communication and control of its speed and velocity. This proposed educational hardware setup serves as a benchmark for students to learn basic satellite systems/subsystems. Furthermore, the setup can be utilized to test and evaluate control algorithms for various satellite systems and subsystems.**

Index Terms: satellite subsystem, RODOS, Aerospace, Operational modes of satellite, satellite test, and evaluation

I. Introduction

In the field of Aerospace technology, various projects were done to simulate a small satellite or test/evaluate satellite subsystems such as the 3DoF test bench for CubeSats [1], simulation satellite [2], the open prototype for educational nanosats [3], robotic test bench [4], CubeSat simulator [5], and satellite balancer [6]. The focus of this communication is on the Floating Satellite (FloatSat) [7] platform specifically designed for students to get familiar with basic satellite subsystems. Furthermore, the proposed educational platform gives a unique opportunity to develop and test various attitudes and control algorithms of a satellite in a frictionless space-like environment.

The proposed FloatSat platform is shown in Fig. 1. It consists of the basic satellite subsystems. To control the orientation of the satellite in one dimension a reaction wheel is mounted at the center of the horizontal plane of the mechanical structure. Various electrical and mechanical components are integrated to simulate a basic satellite

Section II describes the hardware incorporated in FloatSat which is controlled by Real-Time On-Board Dependable Operating System (RODOS). Section III provides a brief introduction to RODOS. Followed by a brief description of various applications such as extraction of sensor data, wireless connectivity of hardware in real-time, satellite attitude control along with results in section IV. Finally, a conclusion is drawn at the end in section V.

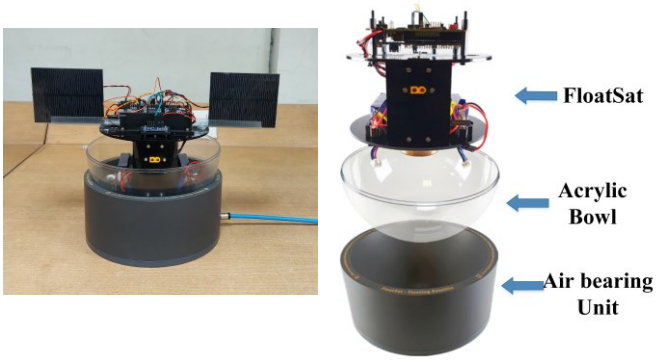

Fig 1. Floating Satellite (FloatSat) platform.

II. Hardware Platform

The spherical Air Bearing Unit (SABU) is the most important part of the FloatSat. This air-bearing unit has a very smooth finished inner surface and holes. These holes provide the pressurized Air to lift the mass up in the Air to about 1mm. The weight of the spherical air bearing unit is 2.75 kg and its inner diameter is 18.22 cm. The two acrylic hemispheres cover the FloatSat avionic system. These shells are made according to the curve surface of the air bearing unit. It moves freely to achieve a frictionless environment. The diameter of the hemisphere shell is about 20cm and its weight of 155g. The FloatSat is the frame that contains the avionics of the system. The FloatSat with basic modules has a total weight of 1.19kg. These modules interact with each other to achieve the basic functionality of a real satellite. The different modules utilized in FloatSat are shown in Fig. 2.

The Reaction wheel attached to the bottom end of the FloatSat is used to generate torque when the rotational speed of the wheel is changed. The weight of the reaction wheel on FloatSat is 280g. The moment of inertia produced by the wheel is 1.175×10^{-4} kgm^2. A brushed dc motor is attached to run the reaction wheel. At 12V the motor is running at 11000 revolutions per minute (RPM) with a current of 300mA. The generated torque is 0.3 kgm^2. In FloatSat, this motor is powered by a 5V dc. The maximum motor current at 5V is 170mA and the motor speed is 4616 RPM.

The power source of the FloatSat is the two Lithium Iron Phosphate (LiFePo4) batteries. There are 2 cells in a single battery with a capacity of 2100mah. The battery is fully charged at 7.2V with the balance charge. There are 2 cells in a single battery with a capacity of 2100mah. The battery is fully charged at 7.2V with the balance charge.

STM32F407G DISC-1 is the microcontroller used to control all modules' functionality in FloatSat. HC-06 Bluetooth device is used to send and receive data wirelessly over a distance of fewer than 100 meters. LSM9DS1 is an IMU used to calculate the object velocity, orientation, and gravitational forces using the combination of an accelerometer, gyroscope, and magnetometer. A voltage regulator is a DC-DC step-down converter used to power up all the electrical components in the FloatSat system. The input operating voltage is 6-38V and the output voltage is 5V. Motor Driver is used for controlling the motor speed and direction with a PWM signal and two GPIO signals respectively.

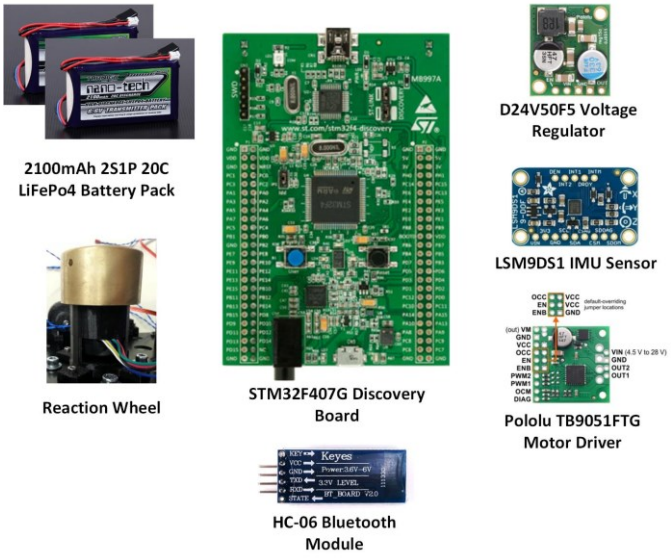

Fig 2. FloatSat Modules

III. Software Platform

The software architecture of floating satellites is based on Real-Time On-Board Dependable Operating System (RODOS) [8]. Every thread inside RODOS runs parallel to each other in Real-time. The current version of RODOS is developed for the STM32F407G discovery board. However, it can be tailored for other devices. The software components in RODOS adjust each other to provide dependable computing [9]. RODOS framework is illustrated in Fig. 3. RODOS control both the operating system (OS) and microelectromechanical system (MEMS). On the top, there is the software middleware (MW), around MW the user implements its application program (AP). Each node provides a gateway to communicate with the external network and input/output (I/O) device. There are around 48 library header files, to govern the RODOS operating system. Application Programmable Interface (API) accesses the data and interacts with external hardware and software components.

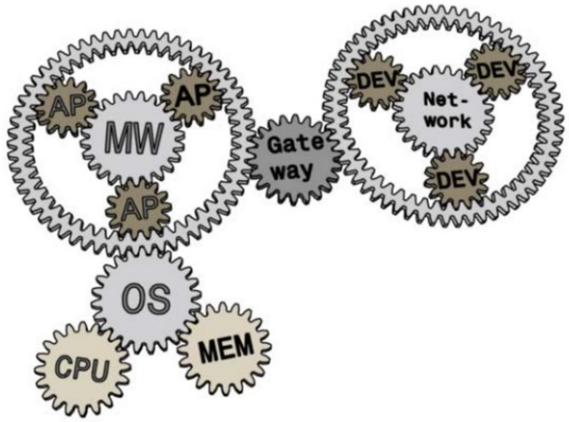

Fig. 3. RODOS framework.

IV. Applications

In this section, we will describe some of the applications tested on top of RODOS on FloatSat hardware. The results of each are discussed along with its implementation.

A. Inertial Data Extraction and Establishing Connectivity.

The first step is to establish a basic configuration and set up the wireless communication link between the FloatSat and PC. In this regard, the LSM9DS1 Inertial Measurement Unit (IMU) is utilized in FloatSat for measuring the object's velocity, orientation, and gravitational force with 9 Degree of Freedom, 3 degrees for each gyroscope, accelerometer, and magnetometer. The IMU uses the I2C protocol to interact with the controller device. Each axis in 9DOF has a 16-bit data output. The sensitivity range for each sensor in IMU is configurable and has an embedded temperature sensor inside it. IMU Data includes 3-axis gyroscope values, 3-axis accelerometer values, 3-axis magnetometer values, and pitch, roll, and yaw values of the system. Initial calibration for the gyroscope and accelerometer can be done by taking the 1000 samples for each axis in a standstill position. These samples are then averaged to generate bias values corresponding to 0 deg/sec measurement. The angular velocity and linear acceleration can be calculated by using the calibrated bias values and the corresponding equations for pitch, roll, and yaw. Next, the partial derivatives of pitch, roll, and yaw are calculated. Finally, the orientation angles of the system concerning the fixed coordinate system can be calculated using the Euler convention. The calibration process for the magnetometer can be done by calculating the minimum and maximum value for all three axes by rotating the sensor in all directions.

To establish wireless communication HC-06 Bluetooth module is used for short-range wireless data communication between the microcontroller and PC. The data rate is 2.1 Mb/s. FTDI32-TTL module is used to configure the baud rate of the HC-06 module. This is done to connect Bluetooth

using the UART protocol with the RODOS library. Pair the Bluetooth Module with the PC and link it with Hyper-Terminal with the correct COM port to examine the data. The algorithm for the purpose is as follows:

- Make a write function to send the data on the UART port inside Bluetooth Thread.
- Make a Bluetooth thread to process the data.
- Initialize the peripheral parameters.
- Loop inside Bluetooth thread runs every 1 second.
- Data is printed on Hyper-terminal by writing the data into a string.

After implementing the algorithm on eclipse IDE, we get 9-DoF values, 3 for each gyroscope, magnetometer, accelerometer, and orientation angles pitch, roll, and yaw on hyper-terminal as shown in fig. 4.

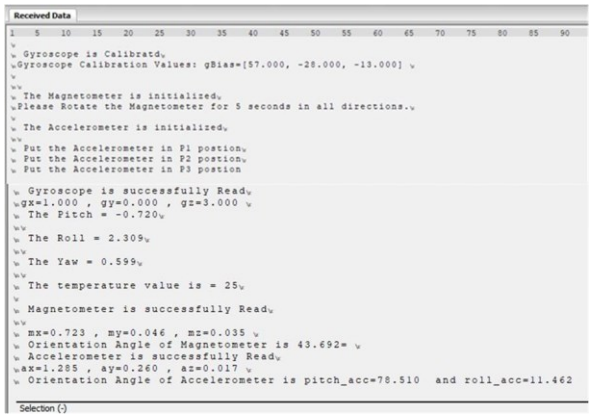

Fig 4. IMU Data Extraction

B. Attitude Control Algorithm

The control system is a crucial part of any dynamic system in which the output of the system tracks the desired input and output is feedback to compare with the desired input to reduce error. This closed-loop system is very important against unmeasured disturbances to keep track of input. The attitude control algorithm in FloatSat dynamics is completely based on the published subscribed protocol as shown in Fig. 5. In this system, some threads represent the publisher sending the data on communication channels called topics, and some threads represent subscribers to get the data from the topics after getting registered to it.

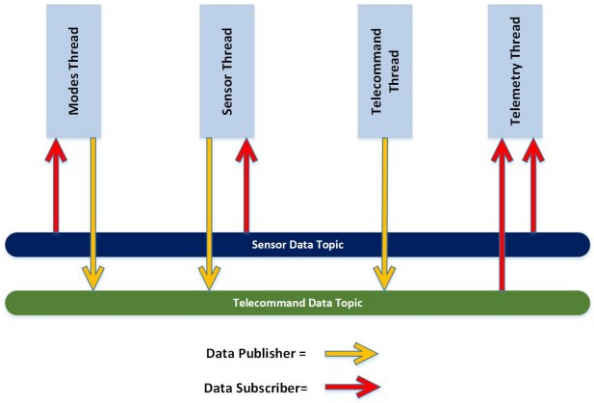

Fig 5. Publish Subscribed Network

The Attitude Control Algorithm for the FloatSat has four main threads that communicate with each other via communication channels i.e. sensor thread, telecommand thread, telemetry thread, and mode thread. The Description for these threads is as follows:

a. Sensor Thread

The first thread is the “Sensors” thread that runs at a 5ms time interval. This thread collects sensor information, processes them, and then publishes the data stored in the "sensor Data" structure into the "Sensor Data Topic" to be received by other threads. All the values of sensors i.e. angular velocity, linear acceleration, Magnetic Flux density, pitch, roll, yaw, temperature, and motor speed are computed under this thread. Next, filters are used to minimize the error of attitude estimation. Mahoney and Madgwick filters are implemented instead of simple complementary filters because under complementary filters attitude error is ignored and the case drifts over time resulting in wrong measurements. The flow chart of the sensor thread is shown in Fig. 6 and its working algorithm is as follows:

- Sensor Thread class computes the IMU parameters.
- Initialize peripherals i.e. SPI, ADC, and Encoder. This is done to link peripherals with the RODOS library.
- If IMU is not properly connected, send the error message "Failed to communicate with IMU" and end the thread.
- If IMU data is ready to compute parameters. Calibrate the Gyroscope, to remove the initial error.
- Time-Loop is running every 5ms to collect the (Attitude Heading Reference system) AHRS values. The AHRS contains a different filter to find IMU data with minimum error.
- Calculate Motor speed and motor current.
- All the sensor data is being published on the sensor data topic to be received by subscribers.

b. Telecommand Thread

The second thread is the “Telecommand” thread that runs once data is received. This thread decodes the received telecommand messages and publishes the data stored in the telecommand data structure into the telecommand data topic to be received by other threads. This is done only if the message is valid. The message body of the telecommand should have the following format “$Xdata#”

Where,

“$” is the Message Start Character [1 byte].

“#” is the Message End Character [1byte].

“X” is the Message-ID Character [1 byte].

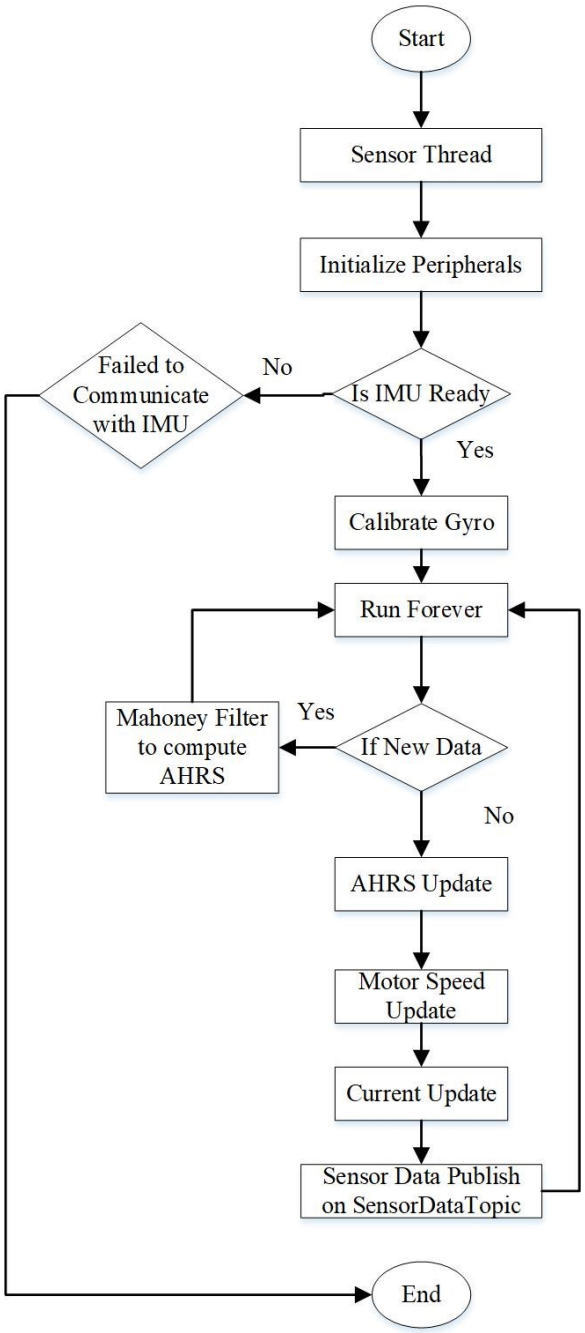

Fig. 6. Algorithm of Sensor Thread

Data are ASCII characters that correspond to a valid floating-point number [0 –12 bytes]. The message ID character includes:

- ON and OFF the Telemetry data bypassing T1 and T0 as a message ID respectively.
- Attitude Heading Reference System (AHRS) Mode from A1 through A8 for Computing Sensor Data through different Filters. By Default, AHRS mode is set to Mahoney Filter if the command is not sent manually.
- System Mode form M0 through M2 for going to Standby mode, Speed mode, and Velocity mode respectively. After entering the respective mode, pass the value of motor speed and FloatSat Velocity e.g. S+200 and V+20 respectively.

The sample telecommand sequence for velocity mode and then set velocity to 50 deg/sec is "$M2#" and "$V+50#" respectively. Fig. 7 describes the working of the Telecommand thread, which is explained as follows:

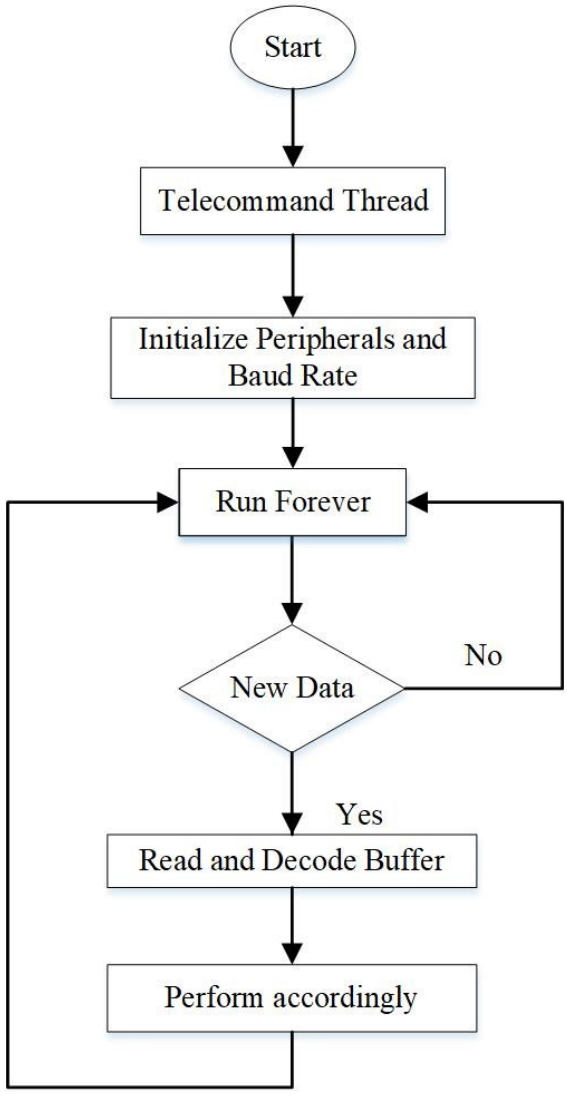

Fig. 7. Algorithm of Telecommand Thread

- Initialize the peripherals for Telecommand i.e. LED and a baud rate.
- The while loop is running to decode the data.
- Suspend until the new data come to the receive buffer. The new data is the command given by the user.
- If new data is received in the buffer, the thread will read the data.
- Decode the received data according to the telecommand pattern discussed earlier.

c. Telemetry Thread

The third thread is the "Telemetry" thread runs at a 1000ms time interval. Fig. 8 shows telemetry data including Accelerometer Axis, Gyroscope Axis, Magnetometer Axis, yaw, pitch, roll, motor speed, and motor current. This thread publishes the sensor's information to the ground station i.e. Hyper-Terminal.

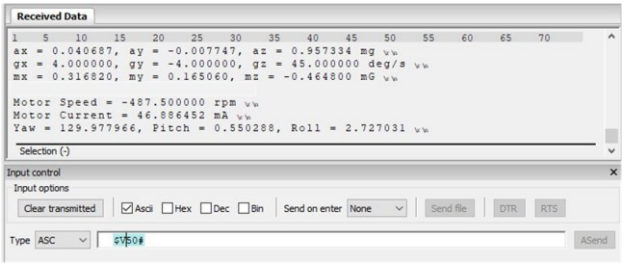

Fig. 8. Telemetry On Hyper-Terminal

Fig. 9 describes the working of the Telemetry thread, which is explained as follows:

- The time loop is running and prints data on the terminal every 1 second.
- This thread receives sensor data and telecommand data.
- If this thread receives telecommand related to telemetry, the data is printed on Hyper-terminal.

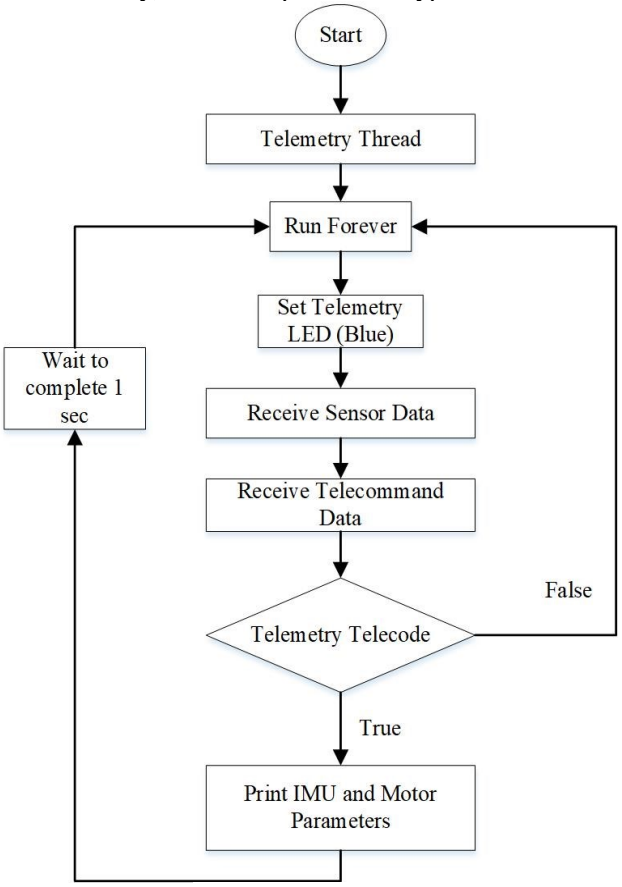

Fig. 9. Algorithm of Telemetry Thread

d. Modes Thread

The fourth thread is the "Modes" thread runs at a 10ms time interval. This thread runs a different operation mode for the FloatSat. There are two main modes of operation i.e.

- Speed control mode
- Velocity control mode

The PI controller is implemented to control the motor speed and a complete description of the code is given in Fig. 10. The reference speed is given by the user by sending a related telecommand. The speed calculation from the encoder is the actual speed. Reference speed is subtracted from the actual speed to generate the error value. This error is computed in the PI controller and a control signal is generated. This signal passes to the actuator signal generator to compute the direction and PWM signals accordingly. The PWM and direction signal goes to the H-Bridge motor driver and outputs the desired current. Then the encoder calculates the actual speed and feeds it back to compute the error. This feedback loop runs until zero error is achieved and also counteracts unmeasured disturbances.

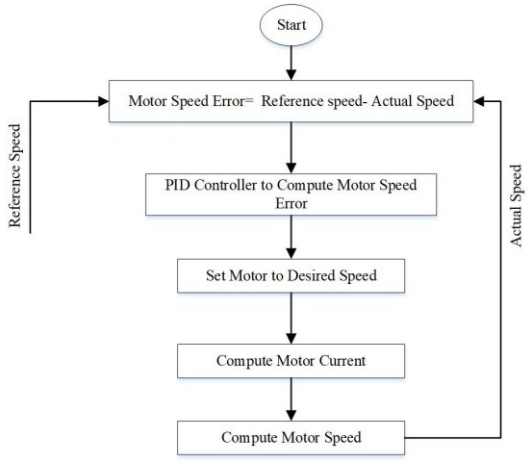

Fig. 10. Algorithm of Speed Mode

After implementing this we get the result in which the telecommand sends to achieve the motor speed of 1000 rpm and the motor speed is set to our desired value as shown in Fig. 11.

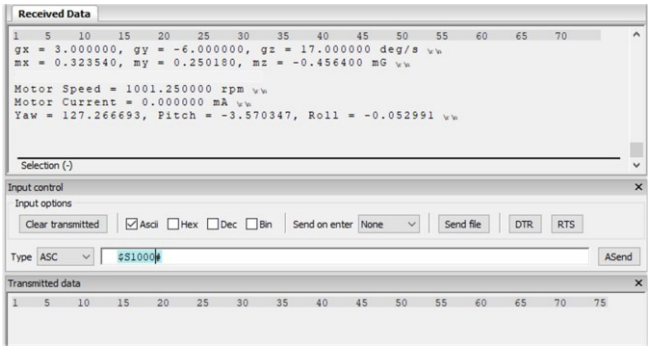

Fig. 11. Output on hyper terminal for motor speed mode.

For controlling the FloatSat velocity, the PI controller is implemented and Fig. 12 describes the velocity mode behavior. The input velocity is given by the user by sending related telecommands. The IMU measures the angular velocity which is the actual velocity of the Floating Satellite. The actual velocity is subtracted from the reference velocity to calculate the error. This error is then computed in the PI controller and a control signal is generated. The PWM signal, direction signal, and currents are generated from the H-bridge motor driver. The Floating Satellite system stabilizes itself to the desired velocity by feeding back the IMU data of the moving Satellite until zero error is achieved.

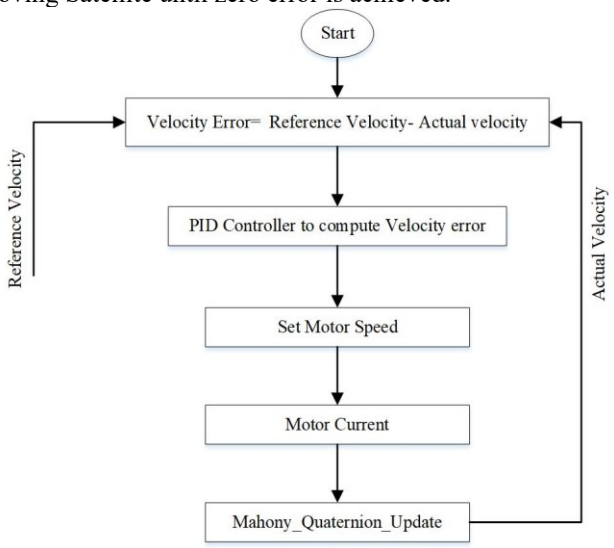

Fig. 12. Algorithm of Velocity Mode

After completing this on Eclipse IDE, (as in Fig. 13) the following result in which the telecommand related to velocity is sent and achieves the same velocity as the value of g_z which is 30 deg/s on the hyper-terminal.

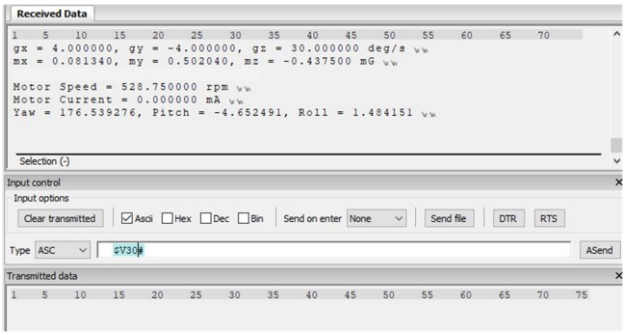

Fig. 13. Output on hyper terminal for velocity mode.

IV. Debris Detection Mission

In this section the space debris detection mission is briefly explained. The space debris detection mission is designed, developed and implemented for students to understand working of a basic radar system which serves to protect the satellite from space junk present in satellite surroundings. Initially, the application is designed to identify the space junk covering 180° in azimuth plan. The hardware setup is shown in Fig. 14. Where, arduino board along with ultrasonic sensor and dc motor can be seen.

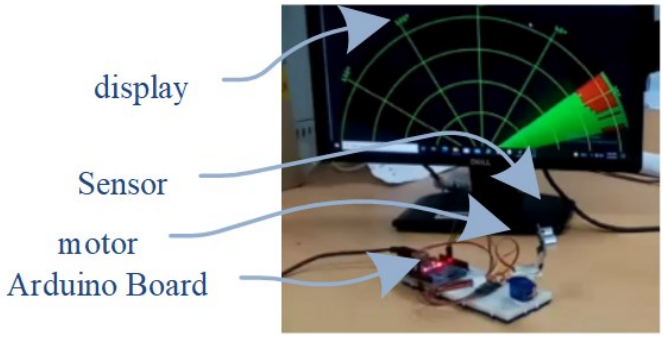

Fig. 14. Space debris detection mission with 180° coverage.

The space debris detection mission is further improved by implementing it utilizing STM32 so that it could be integrated with FlaotSat hardware. Furthermore, the coverage is also extended to 360° in azimuth plan. The required necessary threads are added in RODOS to achieve required functionality. The complete mission is integrated with FloatSat as shown in Fig. 15. The radar integrated with floatsat can detect the object in 50 cm range around the FloatSat. The range of 50 cm is expected to be enhanced by utilizing Frequency modulated continues wave (FMCW) radar in future. Where, instead of ultrasonic sensors a highly directive planar antenna will be utilized along with complete RF/microwave transceiver.

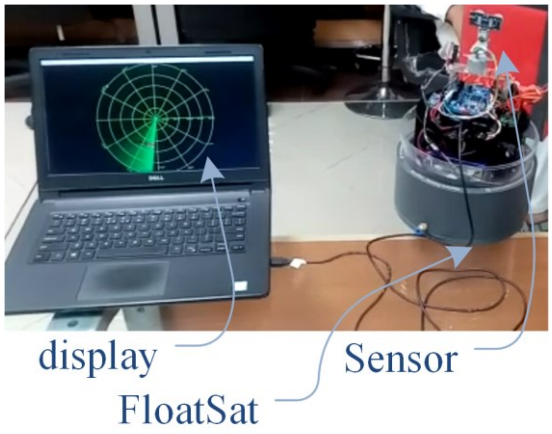

Fig. 15. Space debris detection mission with 360° coverage and integrated with FloatSat hardware.

V. Conclusion

In this paper, we have discussed the basic hardware and software setup of FloatSat. The FloatSat platform in basic configuration can be utilized to implement various satellite missions. Furthermore, it can be very efficiently utilized to verify customized algorithms for test and evaluation of satellite systems and subsystems such as satellite telemetry

and telecommand, satellite attitude control, satellite speed, velocity control, etc. Furthermore, a space debris detection mission is designed and integrated with floatsat hardware.

VI. Acknowledgment

The authors would like to extend their sincere appreciation to chair of computer science VIII, Aerospace information technology, University of Wurzburg, Germany for their support and providing the FloatSat hardware and RODOS operating system.

VII. References

1. Gavrilovich, Irina, Sébastien Krut, Marc Gouttefarde, François Pierrot, and Laurent Dusseau. "Test bench for nanosatellite attitude determination and control system ground tests." In *4S: Small Satellites Systems and Services Symposium*. 2014.
2. Tragesser, Steven, and Gregory Agnes. "Simsat: A ground based platform for demonstrating satellite attitude dynamics and control." In *Annual Conference*, pp. 7-999. 2002.
3. Berk, Josh, Jeremy Straub, and David Whalen. "The open prototype for educational NanoSats: Fixing the other side of the small satellite cost equation." In *IEEE Aerospace Conference*, pp. 1-16. 2013.
4. Gavrilovich, Irina, Sébastien Krut, Marc Gouttefarde, François Pierrot, and Laurent Dusseau. "Robotic test bench for CubeSat ground testing: Concept and satellite dynamic parameter identification." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5447-5453. 2015.
5. Suhandinata, and M. N. Setiawan. "Development of 1U CubeSat attitude determination and control system simulator." In *AIP Conference Proceedings*, vol. 2366, no. 1, p. 030003. AIP Publishing LLC, 2021.
6. Curatolo, Andrea, Anton Bahu, and Dario Modenini. "Automatic Balancing for Satellite Simulators with Mixed Mechanical and Magnetic Actuation." *Aerospace* 9(4), p.223. 2022.
7. Redah, Atheel, Muhammad Faisal, and Sergio Montenegro. "The Floating Satellite System as an Educational Platform for Space Applications." In *IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, pp. 1-3. 2020.
8. Montenegro, Sergio, and Frank Dannemann. "RODOS-real time kernel design for dependability." *DASIA 2009-DAta Systems in Aerospace*. 2009.
9. Dannemann, Frank, and Sergio Montenegro. "Embedded logging framework for spacecrafts." In *DASIA (Data Systems In Aerospace)*. 2013.