DESIGN OF A COMPACT ACADEMIC COURSE FOR SATELLITE NAVIGATION AND CONTROL WITH A REAL-TIME OPERATING SYSTEM

Muhammad Faisal, Atheel Redah, Sergio Montenegro

Institute of Aerospace Information Technology, Julius-Maximilians-University of Wuerzburg (GERMANY)

Abstract

Aerospace navigation and control is one of the modern and fastest developing fields. It has created many avenues for research, development and businesses as large number of the systems are becoming satellite based such as communication, broadcasting, internet etc. Huge numbers of public and private companies are coming in aerospace business and that's why there is a large demand of professionals who can deal with the on-board satellite subsystems as well as its ground station. Universities all around the world are trying to build and use Nanosatellites to train the students for the emerging aerospace industry. In this paper we have presented a compact course for the Graduate and Undergraduate students to learn the basic development process for the satellite's navigation and control with proposed exercises.

Keywords: Satellite, Navigation and Control, Real-time Operating System, Ground Station.

1 INTRODUCTION

This course is based on an indigenously designed Floating Satellite (FloatSat) system which was developed in the Chair of Aerospace Information Technology at the University of Wuerzburg (Germany)[4]. The course is designed around this FloatSat system in order to help students understand and get familiar with basic satellite subsystems and also to develop and test different attitude control algorithms and strategies for small satellites in an almost frictionless environment similar to that in space. The first part of the course starts from the basics of a Real-time Object Oriented Dependable Operating system (RODOS) which is specially designed for Aerospace Applications jointly by German Aerospace Center (DLR) and University of Wuerzburg[1]. As an embedded operating system it is specially designed forspace applications, but fits perfectly to all applications demanding high dependability[2] .The second part deals with the Embedded System Programming under the Hardware Abstraction Layer (HAL) of the RODOS for the available on-board sensors and actuators of the FloatSat. In the third part, the students should develop the Control Algorithms for the basic Operating Modes of the Satellite and in the fourth part we have designed the experiments for the development of a ground station where the students will have to design simple graphical displays for the Telecommand and Telemetry data. This Graphical User Interface (GUI) is based on a desktop computer and connected with the FloatSat via the Bluetooth or the Wifi module where the communication is being done through a simple Gateway protocol layer provided by RODOS.

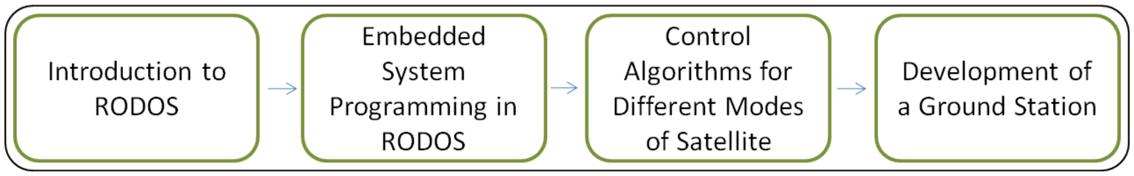

Introduction to RODOS
Embedded System Programming in RODOS
Control Algorithms for Different Modes of Satellite
Development of a Ground Station

Figure 1: Structure of the Course

2 HARDWARE PLATFORM

2.1 . Mechanical Structure

The FloatSat system is consists mainly from a mechanical structure that contains the basic satellite subsystems. This structure is placed into a hemisphere shell that it is floating inside a spherical air bearing unit. The air bearing unit requires compressed air input may vary depending on the mass of the floating unit.

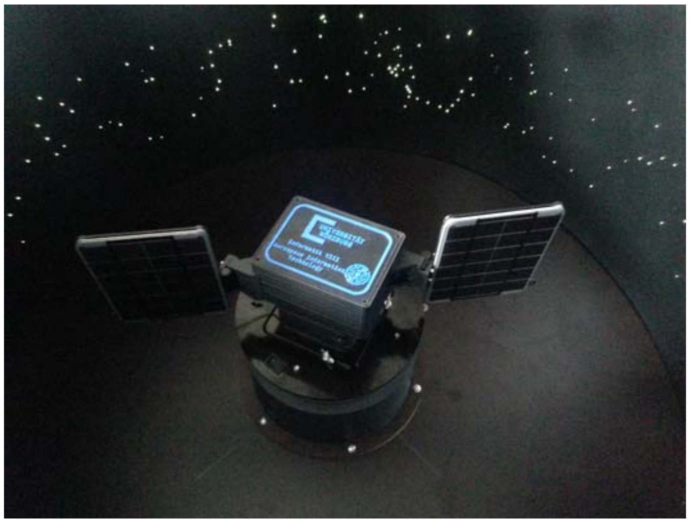


Figure 2: The FloatSat System

2.2 Avionics of the FloatSat

Our proposed electronic control of the FloatSat is based on the STM32F4DISCOVERY Development Board from ST-Microelectronics which is based on an ARM-Cortex M4 microcontroller. This board has all the required interfaces such as UART, I2C, SPI, DCMI etc to interface a wide range of sensors for related operation. An extension board for the STM32F4DISCOVERY developed in the Chair of Aerospace Information Technology at the University of Wuerzburg (Germany) which contains all the sensors, communication modules and power drivers necessary to drive the system is also being used. The whole avionics system is running under a Real-time On-board Dependable Operating System (RODOS). RODOS provides a seamless capability of interfacing the different sensors with its corresponding Hardware Abstraction Layer (HAL) which reduces the development time significantly.

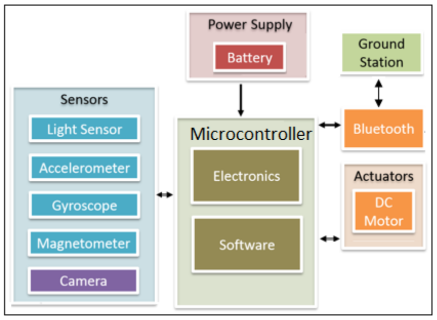


Figure 2: Avionics of the FloatSat

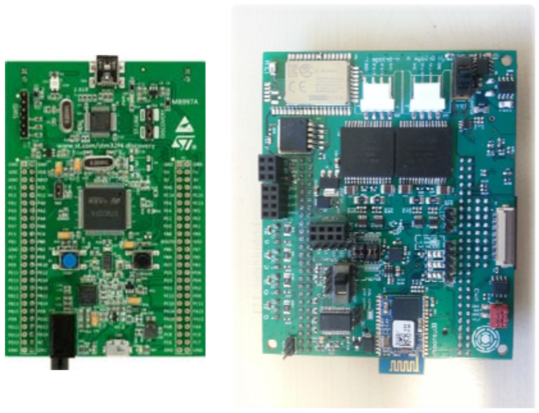


Figure 3: The STM32F4DISCOVERY Development Board with the Extension Board

3 INTRODUCTION TO THE RODOS

RODOS is a real-time operating system for embedded systems and was designed for application domains demanding high dependability. RODOS was developed at the German Aerospace Center (DLR) and has its roots in the operating system BOSS. It is used for the current micro satellite program of the DLR. The system runs on the operational satellite TET-1 and will be used for the currently developed satellite BiROS. RODOS is further enhanced and extended at the German Aerospace Center as well as the department for aerospace information technology at the University of Würzburg.

3.1 . Introduction of Threads and its priority in RODOS

(a) Write a thread class that outputs "Hello World" and the thread name:

- Thread name should be given to constructor.
- Create 3 or more threads (objects) to test the behaviour.

(b) Write a thread-class that outputs its name and the time every n seconds (n is given in the constructor)

- Output the time and the thread-name every 1, 2, 5 and 11 seconds by different threads

  -Thread-name and period should be given to the constructor.

- Output the time and the thread name once at 30 seconds by a fifth thread (can be implemented in new class, or the one-shot function can be added to the above class)

Thread name and time should be given to constructor

(c) Write a thread-class where the objects resume each other as shown in the picture.

-Insert a small break before or after every output

Structure of Class:	class RingElement : public Thread {...};
Implementation:	RingElement a("A"), b("B"), c("C");
Output:	A B C D E F G H A B C D E F G H ... (or vice versa)

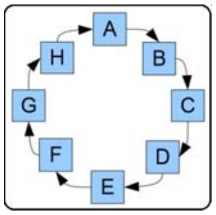


Figure 3: Ring of the objects of Class

3.2 Inter-Thread Communication

In any Real-Time Operating System different threads always require to share information and messages. RODOS provides inter-thread communication and synchronization by using its special data buffer CommBuffer. Not using a CommBuffer is risky, because may be the data is half written in the shared variable while the thread is interrupted. In this case the receiver thread gets inconsistent data. In order to avoid simultaneous access of critical section, semaphore mechanism has to be used. We have proposed two experiments to clear the concept of Inter-Thread Communication and Deadlock Avoidance.

3.2.1 Sharing Information

(a). Write two Theads or Tasks where TaskOne sends a string message (”Message from taskOne”) to taskTwo using the CommBuffer (data buffer of RODOS) as shown in figure 6. The taskTwo prints the received message and its length.

(b). Implement a simple User and Calculator Application. The user thread sends two operands and the mathematical operation (+, - or *) to the Calculator thread which performs the desired operation on the operands and prints the Result.

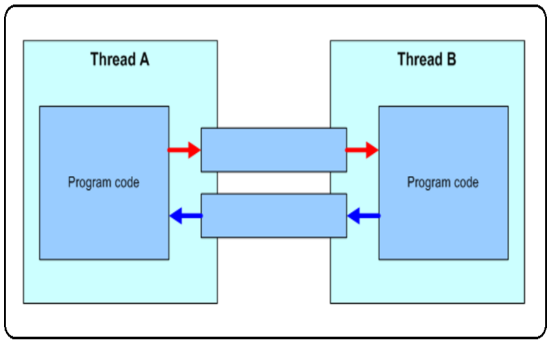


Figure 5: Thread Communication

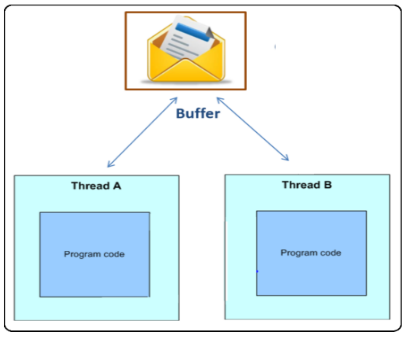
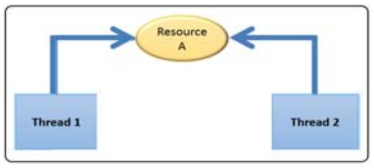


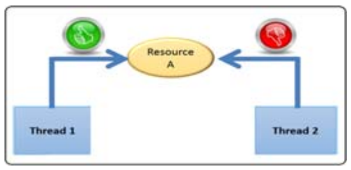
Figure 6:Thread Communication in RODOS

3.2.2 Deadlock-Handling

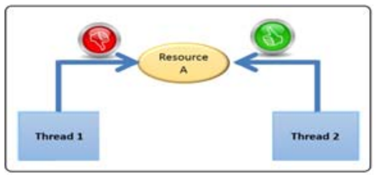
Write a small programme that shows the increment-errors when two or more threads access the same global variable. (**note**: Declare the global variable ***volatile***). Protect the global variable with a semaphore as shown in figure 7.



(a)



(b)



(c)

Figure 7 (a),(b)(c) : Deadlock Handling via Semaphore

4 EMBEDDED SYSTEM PROGRAMMING THROUGH HAL OF RODOS

RODOS provides a set of classes to directly use the hardware and its resources with ease and simplicity. This set of classes in RODOS is referred as Hardware Abstraction Layer (HAL) of RODOS which can be used to access all the available interfaces on the hardware such as I2C, UART, SPI, GPIO etc. Without The HAL interface the developer has to go through a lengthy and tidy process of defining each and every initialization and setting on his own which off course increases the development time and fragility to the errors.

Seven Experiments have been suggested here for embedded system programming with RODOS under its HAL interface. These experiments cover all the details about the UART, I2C, GPIO, PWM Middleware & Gateway interfaces of RODOS. These experiments are believed to provide the students with clear understating about Reading and Writing of the numerous sensors with different interfaces.

4.1 Reading and Writing the GPIO using HAL_GPIO.

- In this Experiment we use the same thread-class we proposed under section 3 where the objects resume each other but this time each object corresponds to each led which should switch on and then off and resumes the next one as shown in the picture.

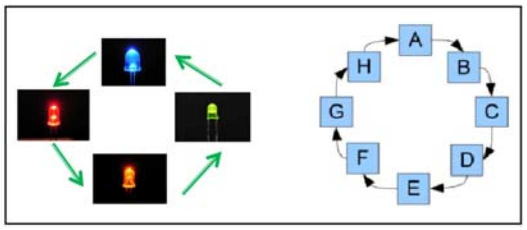


Figure 8: Ring of LEDs Using RODOS HAL_GPIO

4.2 Serial Communication through HAL_UART for debugging.

- Write a thread to pass a message to the satellite on the serial port and receive a corresponding message on your computer via any serial terminal tool.

4.3 Motor Control through HAL_PWM.

- Generate a PWM signal using HAL_PWM to drive the DC motor as show in Figure 9.

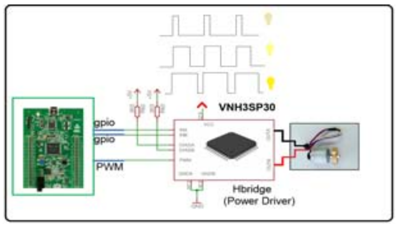


Figure 9: Motor Control through HAL_PWM

4.4 Power Monitoring of the Satellite.

To monitor the power consumption of the FloatSat, INA219 Current Sensors are being used to both measures the high side voltage and DC current draw over I2C.

- Write a C++ Class Implementation for the INA219 Current Sensor using HAL_I2C.

  • Write a thread to monitor the Current and Voltage of the Battery.

  • Write a thread to monitor the Current and Voltage of the Motor.

  • Verify your results by attaching the board with external power supply and use Multimeter to verify the validity of your implementation.

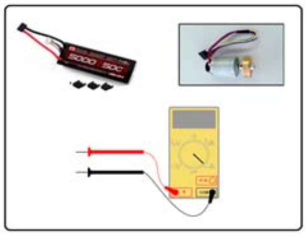


Figure10: Battery and Motor Power Consumption

4.5 Temperature Monitoring through HAL_I2C.

The FloatSat system uses an infrared thermopile sensor TMP006 from Texas Instrument to monitor the temperature of the objects in front of the satellite. It is a non-contact infrared (IR) sensor with a digital interface which is I2C [5].

• Initialize the temperature sensor with number of samples and Conversion time.

• Read and send an object temperature to your computer via any serial terminal tool.

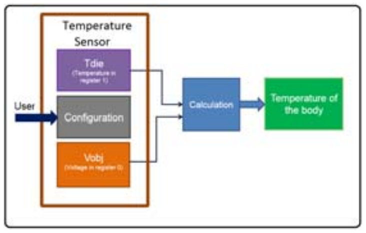


Figure 11: Thread Communication

4.6 Satellite Angular Velocity Monitoring through the Gyroscope Data.

Create a Signal Processing thread that needs to run with following requirements:

• A simple calibration process for the Gyroscope sensor should be performed by taking 1000 samples for each axis in standstill condition and then average them to get the offset value correspond to 0 deg/sec measurement for each axis.

• A data gathering process to get the updated values of the three axis of the Gyro, should be performed continuously at 5 millisecond time interval using the starting data register address of the sensor.

• Get the angular velocity in degree per second and the angular position in degree for the three axis of the sensor by using the sensor sensitivity value.

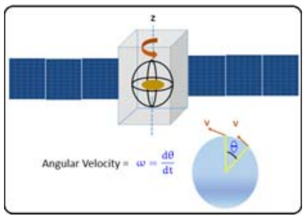


Figure 12: Reading the Angular Velocity using the Gyroscope

4.7 Image Sensor Interfacing.

The FloatSat system uses an OV9665 CMOS image sensor from Omni Vision. It is a 1.3 Megapixel camera. This camera has a dedicated DCMI interface to capture the image on 8 parallel lines. Camera can be configured for image resolution, Picture Format etc through a SCCB protocol which is equivalent to I2C protocol as shown in the figure 13:

- Using the HAL_I2C interface, configure the camera module to capture a VGA image and set the resolution to 120*120 and select the YUV format by setting the appropriate registers.
- Capture the image through the HAL_DCMI interface and save the image on the DMA buffer.

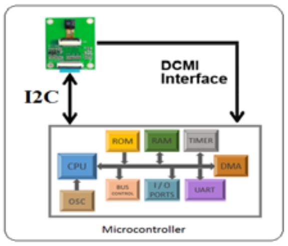


Figure 13: Image Sensor Interfacing

5 CONTROL MODES OF THE SATELLITE

Since our mission requirements are fully depend on the determination and the control of the satellite's orientation, the orientation of the satellite should be supervised all the time. This is realized via the IMU (Inertial Measurement Unit). IMU provides attitude information of the satellite, including roll, pitch and yaw using a combination of accelerometers, gyroscopes, and magnetometers sensors.

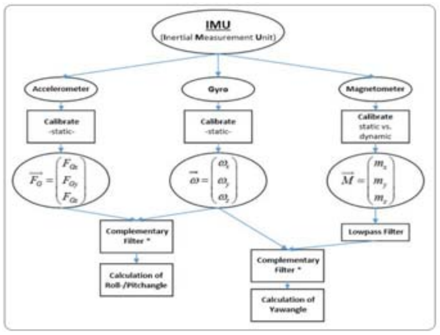

IMU
(Inertial Measurement Unit)
Accelerometer
Gyro
Magnetometer
Calibrate -static-
Calibrate -static-
Calibrate static vs. dynamic
$\overrightarrow{F_G} = \begin{pmatrix} F_{Gx} \\ F_{Gy} \\ F_{Gz} \end{pmatrix}$
$\vec{\omega} = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$
$\vec{M} = \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix}$
Complementary Filter *
Lowpass Filter
Complementary Filter *
Calculation of Roll-/Pitchangle
Calculation of Yawangle

Figure 14: Pitch, Roll and Yaw Measurement

Control Objectives: The control loop must assure stability of our satellite. The initial input is processed and compared to the data collected from the sensors. If any adjustment in the attitude is needed, the Actuators will respond accordingly. The process is repeated until we reach steady-state stability. A PID controller will be utilised to achieve the objectives.

5.1 Satellite Angular Velocity Control

A PI controller for controlling the FloatSat Angular Velocity needs to be implemented. Tune the **PI** controller online to have good performance characteristics of faster response with less overshoot by sending corresponding telecommands.

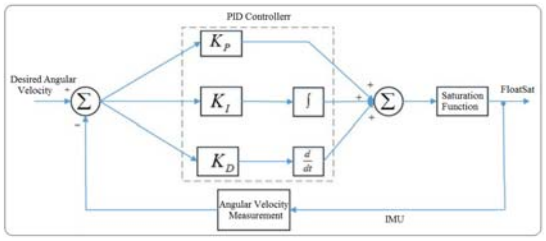


Figure 15: Angular Velocity Control

5.2 Satellite Angular Position Control

A PID controller for controlling the FloatSat Angular Position needs to be implemented. Tune the PID controller online to have good performance characteristics of faster response with no overshoot by sending corresponding telecommands.

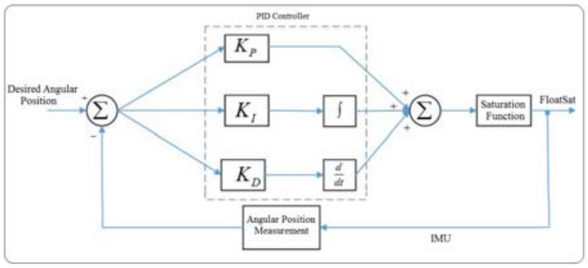


Figure 16: Angular Position Control

6 MESSAGE PASSING OVER THE NETWORK.

RODOS uses so-called Topics to enable communication between threads and over gateways between different systems. A Topic represents a message of a certain kind. A thread can publish Topics as well as subscribe to a Topic to receive all messages that belong to a specific type of message. The message system refers as publish subscribe pattern.

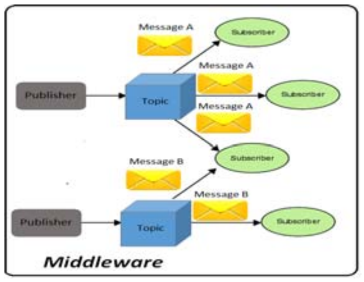


Figure 17: RODOS publisher Subscriber Protocol.

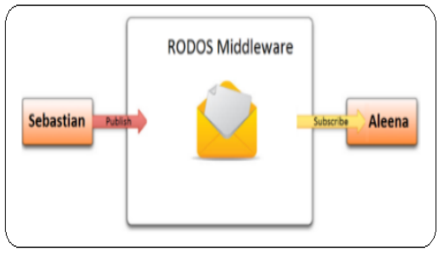


Figure 18: RODOS Middleware

6.1 Telecommand & Telemetry Threads Implementation.

- Implement a Telecommand thread that runs upon receiving a message.
- Implement a Telemetry thread runs at 1000ms time interval. This thread should publish the sensors information to the ground station.

6.2 Gateway Usage.

- Initialize the Gateway of the RODOS with UART link interface.
- Initialize the UART with the desired Baud Rate.
- Write the structure of the telemetry messages with proper ID and transmit it on the Gateway.

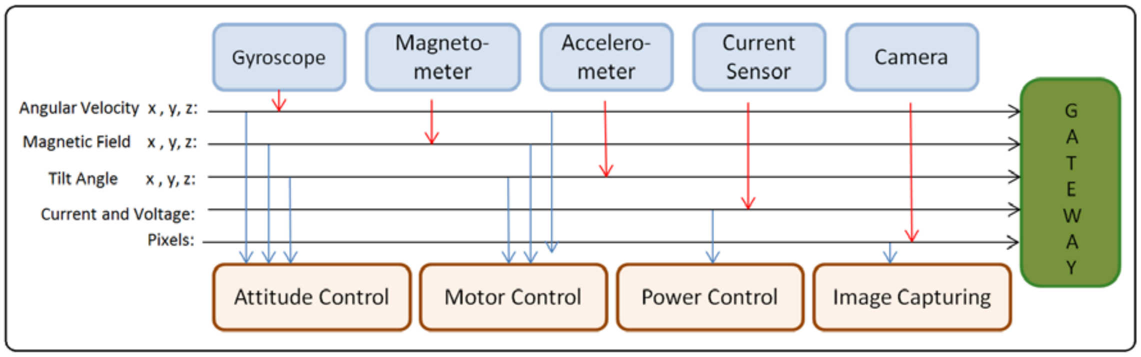


Figure 19: RODOS Middleware and Gateway

7 GROUND STATION DEVELOPMENT IN QT THROUGH RODOS-GATEWAY

The data transmission and reception between the FloatSat and the ground station is achieved through the Bluetooth communication protocol. We propose to use Bluetooth to serial converter chip with the microcontroller in FloatSat and a USB to Bluetooth dongle that establishes a Virtual Serial Port on the PC to transmit and receive the data as depicted in the Figure 20. In order to develop a Ground station We propose to use the RODOS Wrapper for Qt, This way you can transmit and receive the data through the same data structure and its corresponding Topic ID what you did for the on-board programming and you don't have to care about separating the data bytes from its whole message frame, RODOS wrapper can extract the data just on the basis of Topic ID of the message. Each expected message is listed in the code of the ground station with its data structure and corresponding Topic ID. You only have to initiate the RODOS UART Gateway on the Board side as well as Ground Station as shown in the Figure 21.

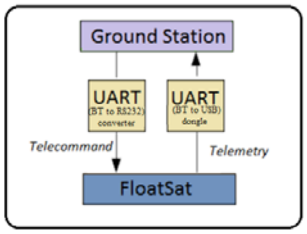


Figure 20: Communication between the FloatSat and the GroundStation

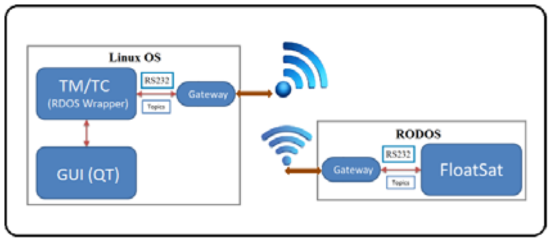


Figure 21: RODOS Middleware Wrapper

7.1 Experiments.

7.1.1 Gateway Initialization

- Initialize the Gateway of the RODOS inside the Qt with the proper ID of serial port.
- Write a function to poll the messages and connect this function with Qt's Timer object with every 5000ms through the Qt's mechanism of Connect Signal and Slot

7.1.2 Reception of Telemetry data and Sending the Telecommand.

- Extract the pitch, roll and yaw from the corresponding data structure and get it plotted in any appropriate widget.
- Use the RODOS's available functions to send the Telecommand to the FloatSat to set the PID Parameters.

7.1.3 Real-Time Plots

In order to plot the real time data we propose to use QCustomPlot library. It is a Qt C++ widget for plotting and data visualization. It has no further dependencies and is well documented. This plotting library focuses on making good looking, publication quality 2D plots, graphs and charts, as well as offering high performance for real-time visualization applications [4].

- Use QtCustomplot Library to plot the real-time data for temperature and Power Monitoring of the FloatSat.

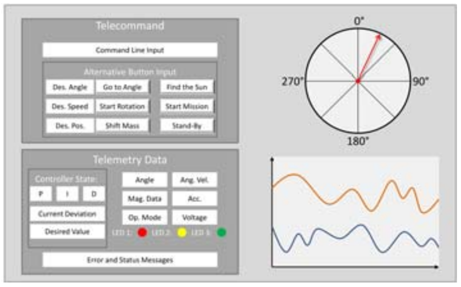


Figure 22: GUI for the Ground-Station

7.1.4 Perform Missions

- Make a button in your GUI to issue the Telecommand to receive the picture from the satellite.
- Receive the image pixel by pixel and save the picture data in a picture array.
- Use the QImage and QScene class to display the image in the QLable of Qt as shown below.

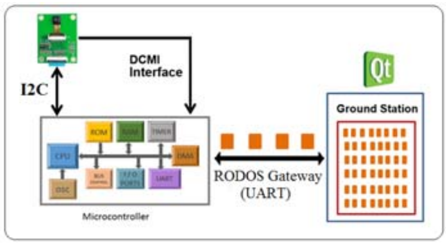


Figure 23: Image Capturing in GS

8 CONCLUSION

A comprehensive and in-detail outline has been presented for a basic course on Satellite Navigation and Control. We have suggested a sequential range of experiments that covers the basic introduction of a Real-time operating system which controls the whole operation of the experimental satellite. Exercises regarding the embedded systems programming of the control unit has been suggested that covers all the required knowledge of embedded system protocols which are proposed to be used under the Hardware Abstraction Layer of the RODOS. RODOS is an open-source real-time operating system which is very flexible to get and use[3]. This Hardware Abstraction Layer of RODOS will reduce the overhead in development process and give the students a simple picture how the embedded systems are programmed on the satellite. In order to communicate with the ground station, a simple mechanism of Middleware and Gateway is presented with the corresponding exercises. In order to develop a real-time Ground Station, exercises are presented that uses Qt wrapper which can be used to receive and display the Telemetry and send Telecommands between the FloatSat and the Ground Station.

REFERENCES

[1] Sergio Montenegro and Frank Dannemann. RODOS - Real Time Kernel Design for Dependability. In DASIA (Data Systems In Aerospace), 2009-

[2] Frank Dannemann and Sergio Montenegro. Embedded Logging FrameWork For SpaceCrafts. In DASIA (Data Systems In Aerospace), 2013.

[3] RODOS Download: http://www.montenegros.de/sergio/rodos/

[4] Floating Satellite Project : http://www8.informatik.uni-wuerzburg.de/en/wissenschaftforschung/floatsat_floating_satellite/

[5] Infrared Thermopile Contactless Temperature Sensor Data sheet http://www.ti.com/product/TMP006

[6] QCustomPlot : http://www.qcustomplot.com/