**Generic RIS Controller Detailed Development Document**

### **Contents**

1. Introduction........................................................................................................... 4

2. Architecture............................................................................................................ 5

3. Work Flow.............................................................................................................. 7

3.1Initialization................................................................................................ 7

3.2 Operation................................................................................................... 7

4. Key Functionalities................................................................................................ 9

4.1 CE.py......................................................................................................... 9

4.2 LUT.py..................................................................................................... 10

4.3 GRC.py.................................................................................................... 11

4.4 RSC.py..................................................................................................... 12

5. Deployment and Execution ................................................................................. 15

### **List of Figures**

1. Testbed Architecture .............................................................................................. 6

2. Message flow in Lookup Table Calibration........................................................... 7

3. Generic RIS Control Operation Message Flow .................................................... 8

### **List of Tables**

1.Function Table CE ................................................................................................. 9

2. API and Interface Table CE ................................................................................ 10

3.Function Table CE ................................................................................................ 10

4. API and Interface Table CE ................................................................................ 11

5.Function Table CE ................................................................................................ 12

6. API and Interface Table CE ................................................................................ 12

7.Function Table CE ................................................................................................ 13

8. API and Interface Table CE ................................................................................ 14

9. Endpoints Consumed Across Files...................................................................... 14

### **Abbreviations**

1. RIS- Reconfigurable Intelligent Surface (RIS)

2. CE- Channel Estimator

3. GRC- Generic RIS Controller

4. RSC – RIS Specific Controller

5. LUT - Lookup table

6. API – Application programming Interface

### **1. Introduction**

This document outlines a comprehensive testbed for Reconfigurable Intelligent Surface (RIS) control. The testbed is designed to boost the signal quality at the receiver, through efficient beamforming and magnitude optimization in a RIS environment.

**Scope:**  
 This document describes the implementation details of the Generic RIS Controller Application, that will consist of four major components: Channel Estimator , Lookup table, Generic RIS Controller, and RIS specific controller.

This document will also guide external developers in integrating their RIS specific algorithms using the Generic RIS Controller. This document includes refined schematics and detailed descriptions of system components and user integration for the RIS test bed which follows TSDSI architecture.

### **2. Testbed Architecture**

### The testbed integrates hardware and software components to iteratively find the best beamforming angle and apply corresponding configurations. The core components communicate through REST APIs for modular functionality. The components of the testbed and their roles are:

**1. Channel Estimator (CE.py) :** Measures the signal magnitude from a receiver and provides it via an API.

**Features:**

* + - Computes signal magnitude from complex data from receiver, in real time.
    - Exposes the magnitude via a Flask API.

**2.** **Generic RIS Controller (GRC.py):** Determines the optimal beamforming angle by iterating over valid angles and comparing magnitudes. Retreives and exposes data via API.

**Features:**

* + - Retrieves valid angles from the RIS, through capability query.
    - Sends each angle to the RIS specific controller module, for RIS configuration.
    - Measures the magnitude using CE.py.
    - Finds and sends the best angle for RIS configuration via RIS specific controller module, using API.

### **3. RIS Specific Controller (RSC.py):** Handles RIS configurations and communicates with the hardware using serial communication.

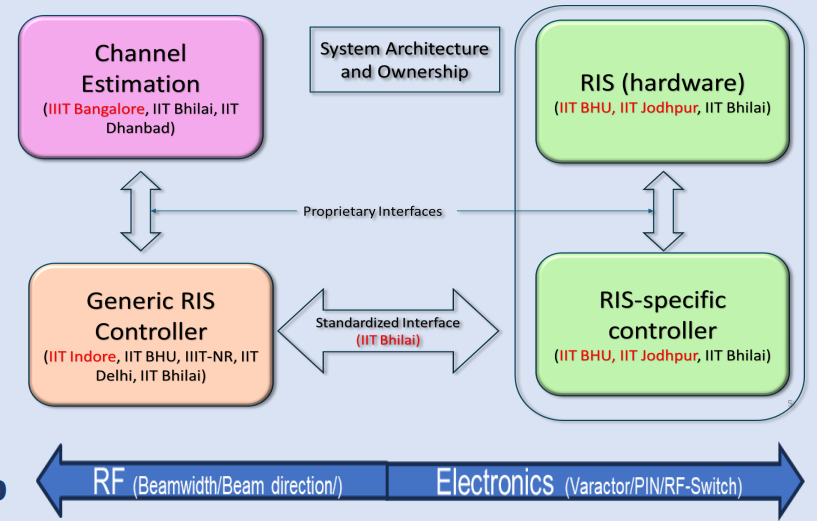
**Features:**

* + - Queries valid beamforming angles from a lookup table (LUT).
    - Configures RIS based on the received beamforming angle, as per the corresponding configuration in the lookup table.
    - Sends configuration patterns to RIS via serial communication.

**4. Lookup Table Generator (LUT.py):** Populates a LUT with beamforming angles and corresponding patterns to maximize signal magnitude.

**Features:**

* + - Iterates through possible beamforming angles.
    - For each angle, generates random patterns and evaluates their performance.
    - Stores the optimal angle-pattern pair in the LUT.



**Figure 1: Testbed Architecture**

**3. Workflow**

**3.1 Initialization:**

* + - Run CE.py to start the magnitude measurement service.
    - Use LUT.py to populate the LUT by determining optimal beamforming patterns for different angles.

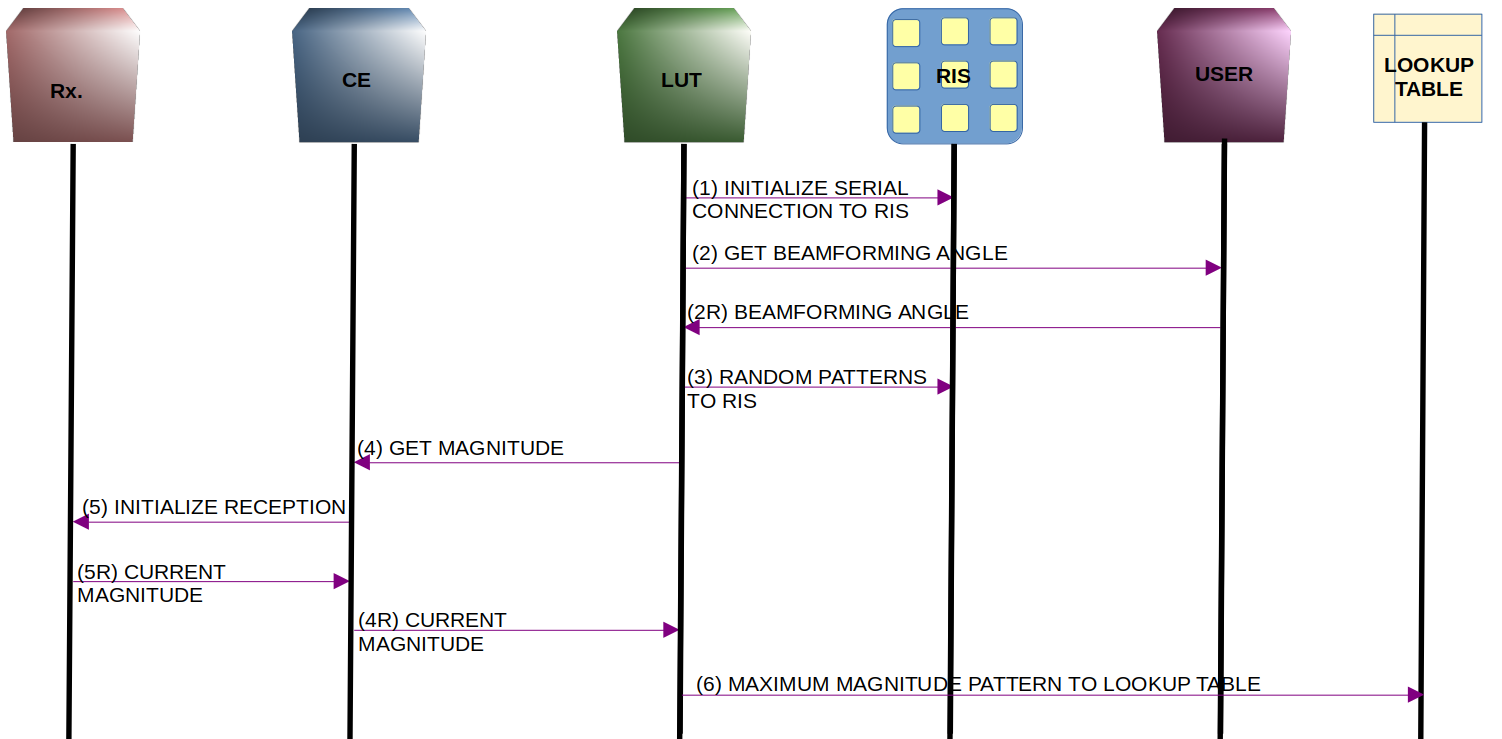
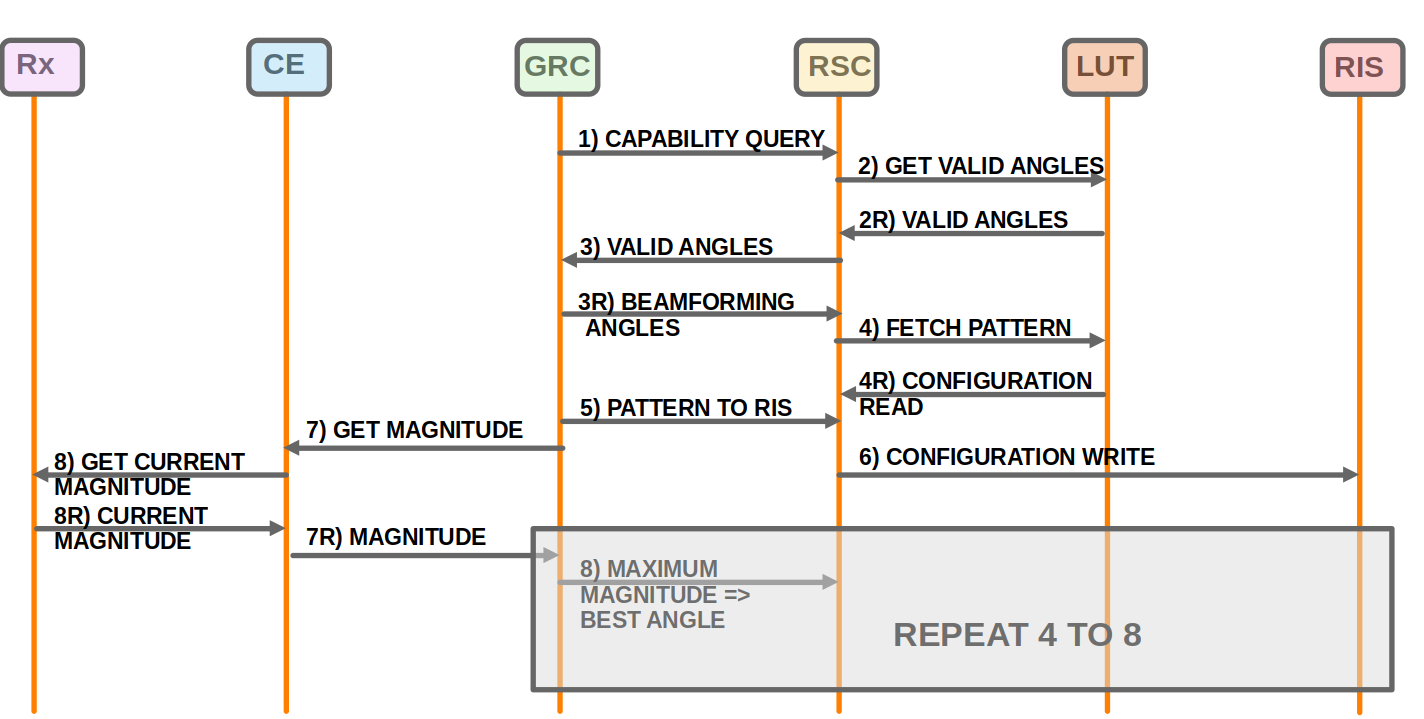


Figure 1: Lookup Table Calibration Message Flow

**Figure 2: Message flow in Lookup Table Calibration**

**3.2 Operation:**

* + - Run CE.py to start the magnitude measurement service.
    - Start RSC.py to handle RIS hardware configuration.
    - Run GRC.py to optimize and apply beamforming angles dynamically during operation.

**Figure 3: Generic RIS Control Operation Message Flow**

### 4. **Key Functionalities**

### 4.1 **CE.py**

CE.py performs channel estimation in Reconfigurable Intelligent Surfaces (RIS) environments.

#### Key Features:

* Lookup Table Calibration: Pre-calculates channel responses using LUT.py for RIS optimization.
* Runtime Usage in GRC: Provides real-time signal magnitudes for RIS adjustments.

#### Channel Estimation Algorithm:

* Calculates channel responses between RIS, and source
* Inputs: Transmitter/receiver coordinates, signal properties (frequency, power).
* Output: Estimated channel response.

### **Table1: Function Table CE**

| **Function Name** | **Description** | **Inputs** | **Outputs** | **Example Usage** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | Initializes the cos\_rx class and sets up signal processing flowgraph. | None | None | flowgraph = cos\_rx() |
| get\_current\_magnitude | Retrieves the current signal magnitude from the USRP source and logs it. | None | Float (current magnitude) | magnitude = flowgraph.get\_current\_magnitude() |
| main | Starts the signal processing flowgraph and Flask server in parallel threads. | None | None | python CE.py |
| get\_magnitude | Flask API function to provide the current magnitude via an endpoint. | HTTP GET Request | JSON (magnitude value) | Access via /api/v1/magnitude |

### **Table 2: API and Interface Table CE**

| **API Endpoint** | **Method** | **Description** | **Response Code** | **Response Body** | **Example Call** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/magnitude | GET | Provides the current signal magnitude. | 200 (Success) | {"magnitude": <float>} | curl -X GET http://localhost:5000/api/v1/magnitude |

Customization and Extension for Advanced Applications  
To adapt CE.py for RIS environments, modify the \_\_init\_\_ function. For cellular communication, incorporate path loss models (e.g., 3GPP), MIMO support, and Doppler shifts. For WiFi, integrate IEEE 802.11 standards and optimize for high-frequency effects (e.g., 5 GHz, 6 GHz). These changes enable handling multi-user scenarios, long delay spreads, and short-range communications, ensuring seamless channel estimation.

## 4.2. LUT.py

LUT.py computes signal magnitudes and populates the Lookup Table for optimal RIS configurations.

#### Key Features:

* Signal Processing: Computes magnitudes from signal data.
* Data Communication: Sends data to LUT for beamforming decisions.

### **Table3: Function Table LUT**

| **Function Name** | **Description** | **Inputs** | **Outputs** | **Example Usage** |
| --- | --- | --- | --- | --- |
| get\_magnitude | Fetches magnitude via API from CE.py. | None | Float (magnitude) | magnitude = get\_magnitude() |
| send\_to\_lut | Writes the beamforming angle and pattern to the lookup table. | Beamforming angle, Best pattern | None | send\_to\_lut(90, '!0XFFFF0000') |

### **Table4: API and Interface Table**

| **API Endpoint** | **Method** | **Description** | **Response Code** | **Response Body** | **Example Call** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/magnitude | GET | Fetches the signal magnitude from CE.py. | 200 (Success) | {"magnitude": 23.5} | curl -X GET http://localhost:5000/api/v1/magnitude |

1. **Customization and Extension for Advanced Applications**
2. The resolution of beamforming angles and the resulting beamforming patterns can be configured based on the specifications of the RIS board. The communication interface with the RIS board is also board-specific; for example, it can utilize a serial port, as in the current setup, or alternative modes such as Bluetooth or other protocols, depending on the implementation requirements.

## **4.3. GRC.py**

Orchestrates beamforming optimization by interacting with CE, LUT, and RSC.

#### Key Features:

1. Query Valid Angles: Fetches valid beamforming angles.
2. Select Optimal Angle: Iterates through angles for maximum signal magnitude.
3. RIS Configuration: Posts the final optimal angle to RSC.

### **Table 5: Function Table GRC**

| **Function Name** | **Description** | **Inputs** | **Outputs** | **Example Usage** |
| --- | --- | --- | --- | --- |
| get\_magnitude | Fetches magnitude via API from CE.py. | None | Float (magnitude) | magnitude = get\_magnitude() |
| send\_to\_rsc | Sends the best beamforming angle to RSC.py. | Beamforming angle (integer) | JSON Response | send\_to\_rsc(90) |
| get\_valid\_angles | Fetches valid beamforming angles via API from RSC.py. | None | List of valid angles | angles = get\_valid\_angles() |
| find\_most\_frequent\_angle | Finds the most frequent beamforming angle from a list. | List of angles | Most frequent angle | best\_angle = find\_most\_frequent\_angle([90, 90, 45]) |

### **Table 6: API and Interface Table**

| **API Endpoint** | **Method** | **Description** | **Response Code** | **Response Body** | **Example Call** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/magnitude | GET | Fetches the signal magnitude from CE.py. | 200 (Success) | {"magnitude": 23.5} | curl -X GET http://localhost:5000/api/v1/magnitude |
| /api/v1/configuration\_write/capability\_query | GET | Fetches valid angles from RSC.py. | 200 (Success) | {"valid\_angles": [90]} | curl -X GET http://localhost:5003/api/v1/configuration\_write/capability\_query |
| /api/v1/configuration\_write | POST | Sends configuration parameters to RIS via RSC.py. | 200 (Success) | {"status": "Success"} | curl -X POST -d '{"beamforming\_angle": 90}' http://localhost:5003/api/v1/configuration\_write |

**Customization and Extension for Advanced Applications:**

A customized GRC implementation would invoke the above functions in order to interact with CE/ RSC modules to get the current channel estimation and to configure the RIS as per their adaptation algorithm.

The GRC.py code is a generic and modular framework designed for flexibility across different RIS implementations. Developers can modify the channel estimator module to align with the communication system in use, such as cellular or WiFi. Similarly, the RSC.py module can be replaced with one specific to the RIS system being used. The serial communication mode implemented here is also adaptable and can be configured based on the communication method required for the specific RIS setup.

## **4. 4 RSC.py**

Handles RIS hardware configuration using serial communication.

#### Key Features:

1. Capability Query: Provides a list of valid beamforming angles.
2. RIS Configuration: Sends beamforming patterns to RIS hardware.

### **Table 7: Function Table RSC**

| **Function Name** | **Description** | **Inputs** | **Outputs** | **Example Usage** |
| --- | --- | --- | --- | --- |
| capability\_query | Responds to the capability query from the GRC.  Extracts valid beamforming angles from LUT and provides them via an API. | None | List of valid angles | angles = capability\_query() |
| send\_to\_ris | Implements the proprietry method of communicating with the hardware for setting a configuration.  Sends the pattern corresponding to a beamforming angle to RIS. | Beamforming angle, Best pattern | None | send\_to\_ris(90, '!0XFFFF0000') |
| configure\_ris | Configures RIS using a beamforming angle received from GRC. | Beamforming angle | JSON Response | Accessed via /api/v1/configuration\_write API endpoint |

### **Table 8: API and Interface Table**

| **API Endpoint** | **Method** | **Description** | **Response Code** | **Response Body** | **Example Call** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/configuration\_write/capability\_query | GET | Fetches valid beamforming angles from LUT. | 200 (Success) | {"valid\_angles": [90]} | curl -X GET http://localhost:5003/api/v1/configuration\_write/capability\_query |
| /api/v1/configuration\_write | POST | Configures RIS with beamforming angle and pattern. | 200 (Success) | {"status": "Success"} | curl -X POST -d '{"beamforming\_angle": 90}' http://localhost:5003/api/v1/configuration\_write |

**Customization and Extension for Advanced Applications**

* + 1. The RSC.py module is designed to work seamlessly with the GRC.py framework, facilitating smooth interaction with any RIS board and its specific control systems. While the current implementation establishes serial communication with the RIS board via /dev/ttyUSB0, the module is adaptable to other hardware setups. Developers can modify the serial communication protocols and endpoints to align with the requirements of their specific RIS hardware. Similarly, the flexible design of GRC.py enables integration with any RIS-specific control program, making it straightforward to interface custom hardware with the existing generic controller architecture.

### **Table9: Endpoints Consumed Across Files**

| **File** | **Consumed Endpoints** | **Description** |
| --- | --- | --- |
| CE.py | None | Hosts its own endpoint (/api/v1/magnitude). |
| LUT.py | /api/v1/magnitude | Fetches magnitude from CE.py. |
| GRC.py | /api/v1/magnitude, /api/v1/configuration\_write/capability\_query, /api/v1/configuration\_write | Consumes endpoints from CE.py and RSC.py. |
| RSC.py | None | Hosts its own endpoints (/api/v1/configuration\_write, /capability\_query). |

### **5. Deployment and Execution**

* **Environment Setup:**

Hardware: USRP, RIS, Serial Connection.

Software: Python, Flask, GNU Radio.

* **Steps to Run:**

1. Start CE.py to provide magnitude values.
2. Execute LUT.py to generate the LUT.
3. Run GRC.py to determine beamforming angles.
4. Use RSC.py for configuring RIS based on angles.