MICROPYTHON PROGRAMMING REFERENCE FOR THE FISCHERTECHNIK RX CONTROLLER

Importing libraries

These libraries are required for programming models with the RX Controller.

1. **import asyncio:**
   - **What it is:** asyncio is a Python library that provides infrastructure for writing parallel executable code.
   - **What is it good for:** The library enables the simultaneous and efficient execution of multiple tasks (such as reading sensors or activating actuators). It is useful in scenarios where it is necessary to wait for different input/output operations (such as the response of a sensor) without blocking the execution of other tasks.
2. **import fischertechnik.factories as rx_factory:**
   - **What is this?:** This line imports the factories module from the fischertechnik library and assigns it to the alias rx_factory to facilitate referencing.
   - **What is this good for:** The fischertechnik.factories library provides the necessary functions to create and manage the different types of sensors and actuators compatible with the RX Controller**.**
3. **from fischertechnik.logging import log as print:**
   - **What is this?:** This line imports the log function from the logging module of fischertechnik and assigns it to the alias print.
   - **What is this good for:** This allows print to be used to log debugging or information messages, similar to the print() function in Python, but specifically for the Fischertechnik library with additional logging functions.

Initialization of the components

After the imports, the components are initialized using various functions. These initializations are crucial to configure the system correctly and to ensure that each sensor and actuator is ready for operation. Without these initializations, the program would not be able to communicate with the hardware components and operations such as reading sensors or activating actuators would not be possible.

1. **Initialization of the factories:**
    - In order to be able to use sensors and actuators, the corresponding factories must first be initialized, which prepare the system to create objects that represent the hardware components.

```
rx_factory.init_controller_factory()
rx_factory.init_input_factory()
rx_factory.init_output_factory()
rx_factory.init_motor_factory()
rx_factory.init_i2c_factory()
```

    - **rx_factory.init_controller_factory():** Initializes the controller required to manage communication with the control electronics.
    - **rx_factory.init_input_factory():** Initialization that enables the creation and management of sensors (inputs).
    - **rx_factory.init_output_factory():** Initialization of the outputs, which enables the creation and management of actuators (outputs).
    - **rx_factory.init_motor_factory():** Initializes the motors and specializes in their management.
    - **rx_factory.init_i2c_factory():** Initializes I2C devices that are used to manage sensors or actuators that are connected via the I2C protocol.

2. **Initialization of sensors and components**

The fischertechnik RX Controller enables the connection and disconnection of various sensors and actuators to different ports and thus offers great flexibility in the design and construction of projects. The following describes the types of sensors that can be connected, how they are configured and initialized, and what they are used for.

- **photoresistor**
    - **Input port:** Each digital input port**.**
    - **Initialization:**

```
RX_I5_photo_resistor = rx_factory.input_factory.create_photo_resistor(controller, 5)
```

  - **Configuration:** This sensor measures the light intensity. Its resistance varies depending on the amount of light; more light, less resistance and less light, more resistance.
  - **Use:** Can be used to detect changes in lighting conditions, e.g. to automatically switch on a lamp when it gets dark.

- **Ultrasonic sensor**
  - **Input port:** Each digital input port**.**
  - **Initialization:**

    ```
    RX_I7_ultrasonic_distance_meter = rx_factory.input_factory.create_ultrasonic_distance_meter(controller, 7)
    ```
  - **Configuration:** This sensor uses sound waves to measure the distance to an object. The distance is determined by measuring the time it takes for the sound wave to reach the object and return.
  - **Usage:** Used in applications such as collision avoidance for robots or to detect the presence of objects in an area.

- **Color sensor**
  - **Input port:** Each digital input port**.**
  - **Initialization:**

    ```
    RX_I6_color_sensor = rx_factory.input_factory.create_color_sensor(controller, 6)
    ```
  - **Configuration:** This sensor detects the color of an object in front of it by measuring the intensity of the reflected light in the RGB color channels (red, green, blue).
  - **Application:** Useful in applications that require color detection, such as sorting processes or line detection in robotics.

- **NTC resistor**
  - **Input port:** Each digital input port**.**
  - **Initialization:**

    ```
    RX_I1_ntc_resistor = rx_factory.input_factory.create_ntc_resistor(controller, 1)
    ```
  - **Configuration:** This is a temperature sensor that uses an NTC (Negative Temperature Coefficient) thermistor whose resistance decreases as the temperature rises.
  - **Use:** Used to measure the ambient temperature or the temperature of objects.

- **IR track sensor**

- **Input port:** Each digital input port**.**
- **Initialization:**

```
RX_I3_trail_follower = rx_factory.input_factory.create_trail_follower(controller, 3)
RX_I4_trail_follower = rx_factory.input_factory.create_trail_follower(controller, 4)
```

- **Configuration:** Used to follow a line on a surface, often using infrared light to detect the contrast between the line and the background. Note that a single sensor must be connected to two inputs.
- **Application:** Ideal for line-following robots that have to follow a predefined path.

- **Phototransistor**
  - **Input port:** Each digital input port**.**
  - **Initialization:**

    ```
    RX_I4_photo_transistor = rx_factory.input_factory.create_photo_transistor(controller, 4)
    ```

  - **Configuration:** This sensor is light-sensitive and enables the detection of light intensity or the presence of light in a specific area.
  - **Use:** Can be used to detect changes in ambient lighting or the presence of light in safety applications.

- **Mini push-button**
  - **Input port:** Each digital input port**.**
  - **Initialization:**

    ```
    RX_I8_mini_switch = rx_factory.input_factory.create_mini_switch(controller, 8)
    ```

  - **Configuration:** This is a simple switch that can detect whether it is open or closed.
  - **Application:** Can be used as a limit switch to detect the presence or absence of an object or as a control button.

- **Reed contact**
  - **Input port:** Each digital input port**.**
  - **Initialization:**

    ```
    RX_I2_reed_switch = rx_factory.input_factory.create_mini_switch(controller, 2)
    ```

  - **Configuration:** A magnetic sensor that is activated when a magnetic field is present and indicates whether the field is present or absent.
  - **Application:** Often used in security systems or to detect the position of moving objects.

- **I2C sensors**: Sensors connected via the I2C protocol require special initialization, as this protocol enables communication with multiple devices via a single data bus.
  - **RGB gas sensor**
    - **I2C port:** Can be connected to any available I2C port.

      ```
      RX_I2C_1_gesture_sensor = rx_factory.i2c_factory.create_gesture_sensor(controller, 1)
      ```
    - **Configuration:** This sensor can detect movements such as waving or hand movements. It is also able to measure distances, light intensity and its composition.
    - **Application:** Used for contactless control of devices, such as in user interfaces or robot controls.
  - **Environmental sensor**
    - **I2C port:** Can be connected to any available I2C port.
    - **Initialization:**

      ```
      RX_I2C_2_environment_sensor = rx_factory.i2c_factory.create_environment_sensor(controller, 2)
      ```
    - **Configuration:** Measures environmental parameters such as temperature, humidity, air quality and air pressure.
    - **Application:** Ideal for monitoring the ambient conditions in a specific area.
  - **Combination sensor 10-pole**
    - **I2C port**: Can be connected to any available I2C port.
    - **Initialization:**

      ```
      RX_I2C_3_combined_sensor = rx_factory.i2c_factory.create_combined_sensor(controller, 3)
      ```
    - **Configuration:** Includes several sensors such as accelerometer, gyroscope and magnetometer, which enable the detection of movement, orientation and magnetic field.
    - **Usage:** Used in applications for motion detection or stabilization.

Summary: For each sensor type, the correct port must be selected based on the application configuration and the sensor initialized using the factories provided. This initial configuration is essential to ensure that the sensors function correctly and provide accurate data. In the next part, we will cover how to use these sensors and interpret their data.

Detailed description of the information that can be obtained from digital and analog sensors in the Fischertechnik RX Controller

The sensors connected to the fischertechnik RX Controller can be divided into digital and analog sensors depending on the type of signal generated and the information provided. The following describes what information can be obtained from these sensors and how this data can be used.

Digital sensors: Digital sensors provide discrete values, usually "on" or "off" (1 or 0), "open" or "closed" or some other form of binary value.

- **Mini push-button**
  - **Information gained:**
    - **is_open():** Returns True if the switch is open, False if it is closed.
    - **is_closed():** Returns True if the switch is closed, False if it is open.
    - **get_state():** Returns the current state of the switch (open or closed).
  - **Use:** Can be used to detect the presence of an object or as a control button in a system, e.g. to start or stop a process.
- **Reed contact**
  - Information **gained**:
    - **is_open():** Returns True if the switch is open, False if it is closed.
    - **is_closed():** Returns True if the switch is closed, False if it is open.
    - **get_state():** Returns the current state of the switch (open or closed).
  - **Uses:** Commonly used to detect the presence of a magnetic field, useful in security applications or for positioning.
- **IR track sensor**
  - **Information gained:**
    - **get_state():** Returns a value indicating whether the sensor detects the line (0 for not detected, 1 for detected).
  - **Application:** Important for line-following robots that have to follow a predefined line on a surface.

- **Photo Transistor**
    - **Information gained:**
        - **is_bright():** Returns True if the detected light is high.
        - **is_dark():** Returns True if the detected light is low.
        - **get_state():** Returns the current state based on the detected light (bright or dark).
    - **Use:** Can be used to detect the presence of light or darkness in an area, useful in day/night detection or object detection.

Analog sensors: Analog sensors provide continuous values that can vary within a certain range and represent physical quantities such as light, temperature, distance, etc.

- **photoresistor**
    - **Information gained:**
        - **get_resistance():** Returns the resistance value, which is inversely proportional to the light intensity (more light, less resistance).
    - **Application:** Used to measure ambient light intensity, e.g. for the automatic control of lighting systems.
- **Ultrasonic sensor**
    - **Information gained:**
        - **get_distance():** Returns the distance to an object in units from 0 to 1024, where 0 is the minimum distance and 1024 is the maximum distance.
    - **Uses:** Used to measure distance to objects, useful in applications such as obstacle avoidance for robots.
- **Color sensor**
    - **Information gained:**
        - **get_voltage():** Returns the voltage value that correlates with the intensity of the detected RGB color components.
    - **Use:** Enables the color recognition of objects, useful for sorting processes or the recognition of colored lines.

- **NTC resistor**
  - **Information gained:**
    - **get_resistance():** Returns the resistance of the thermistor, which varies inversely to the temperature (more temperature, less resistance).
  - **Application:** Used to measure the ambient temperature or objects, suitable for thermal monitoring applications.

Summary: Digital and analog sensors provide a variety of data that can be used to interact with the environment, control automated systems or collect data for analysis. Digital sensors provide binary values that are useful for simple control decisions, while analog sensors provide continuous values that enable precise measurement of physical quantities. The correct interpretation of this data is crucial for the efficient functioning of any system based on the fischertechnik RX Controller.

RGB gas sensor

The RGB gesture sensor is an advanced device that combines several functions in a single chip. It can recognize gestures and measure ambient light, RGB light (red, green, blue) and proximity. Below you will find a detailed description of the sensor's capabilities, the available functions and the values that can be retrieved.

Initialization of the sensor: In order to use the RGB gas sensor, it must first be initialized correctly. This is done by creating an instance of the sensor via the I2C factory.

```
RX_I2C_1_gesture_sensor = rx_factory.i2c_factory.create_gesture_sensor(controller, 1)
```

This code creates the object RX_I2C_1_gesture_sensor, which is assigned to I2C port 1. After initialization, you can start configuring the various functions and retrieve data from the sensor.

Gesture recognition: The RGB gesture sensor can recognize various hand movements such as up, down, left and right. This function is useful for controlling devices without physical contact.

- **Activation of gesture recognition:**

  ```
  RX_I2C_1_gesture_sensor.enable_gesture()
  ```

- **Recall recognized gestures:**

  ```
  gesture = RX_I2C_1_gesture_sensor.get_gesture()
  print(gesture)
  ```

- **Deactivation of gesture recognition:**

```
RX_I2C_1_gesture_sensor.disable_gesture()
```

Measurement of RGB light: The sensor can detect the intensity of the light in the RGB color components and thus provide an accurate representation of the ambient light color.

- **Activation of the light sensor:**

  ```
  RX_I2C_1_gesture_sensor.enable_light()
  ```

- **Retrieve RGB data:**

  ```
  rgb_values = RX_I2C_1_gesture_sensor.get_rgb()
  print(f "Red: {rgb_values[0]}, Green: {rgb_values[1]}, Blue: {rgb_values[2]}")
  ```

  - **get_rgb():** Returns a tuple containing the intensity values for the red, green and blue components. Each value can normally vary from 0 to 255, depending on the sensor setting and lighting conditions.

- **Retrieve HEX values:**

  ```
  hex_value = RX_I2C_1_gesture_sensor.get_hex()
  print(f "Color in HEX format: {hex_value}")
  ```

  - **get_hex()**: Returns a HEX representation of the detected color, useful for applications that require a color identifier.

- **Retrieve HSV values:**

  ```
  hsv_values = RX_I2C_1_gesture_sensor.get_hsv()
  print(f "Hue: {hsv_values[0]}, saturation: {hsv_values[1]}, value: {hsv_values[2]}")
  ```

  - **get_hsv():** Returns the values for hue, saturation and value (brightness), which are useful for color description in graphical and design applications.

- **Deactivation of the light sensor:**

  ```
  RX_I2C_1_gesture_sensor.disable_light()
  ```

Ambient light measurement: In addition to RGB measurements, the sensor can provide an ambient light measurement, which is useful for determining the general lighting conditions.

- **Calling up the ambient light:**

  ```
  ambient_light = RX_I2C_1_gesture_sensor.get_ambient() print(f
  "Ambient light: {ambient_light}")
  ```

  - **get_ambient():** Returns a value that represents the intensity of the ambient light. This value can be used to automatically adjust the brightness of displays or lighting.

Proximity detection: The RGB gas sensor can detect the proximity of objects based on the amount of reflected infrared light.

- **Activation of proximity detection:**

  ```
  RX_I2C_1_gesture_sensor.enable_proximity()
  ```

- **Retrieve approximate data:**

  ```
  proximity = RX_I2C_1_gesture_sensor.get_proximity() print(f
  "Proximity: {proximity}")
  ```

- **get_proximity():** Returns a value between 0 and 255, where 0 means that there is no object nearby and 255 means that the object is very close to the sensor.

- **Deactivation of proximity detection:**

  ```
  RX_I2C_1_gesture_sensor.disable_proximity()
  ```

Summary: The RGB gesture sensor is a versatile device that combines gesture detection, RGB light and ambient light measurement and proximity detection in a single chip. This makes it ideal for applications with advanced user interfaces, automatic light adaptation and security systems. The sensor provides a rich interface to interact with the environment through detailed and precise data.

Environmental sensor

The BME680 is an integrated environmental sensor that can measure various parameters such as temperature, relative humidity, air pressure and air quality. These features make it ideal for environmental monitoring applications, HVAC systems, weather stations and other IoT (Internet of Things) devices. The capabilities of the sensor, its initialization and the acquisition of the various data are described below.

Initializing the environmental sensor: In order to use the environmental sensor, it must first be initialized correctly. This is done by creating an instance of the sensor via the I2C factory.

```
RX_I2C_2_environment_sensor = rx_factory.i2c_factory.create_environment_sensor(controller, 2)
```

This code creates the object RX_I2C_2_environment_sensor, which is assigned to I2C port 2. After initialization, the environmental data provided by the sensor can be retrieved.

Functions of the sensor

- **Temperature measurement**
  - **Retrieve the temperature:**

    ```
    temperature = RX_I2C_2_environment_sensor.get_temperature() print(f
    "Temperature: {temperature} °C")
    ```

    - **get_temperature():** Returns the ambient temperature in degrees Celsius (°C). This information is useful for monitoring the indoor or outdoor climate and for applications that require precise temperature control.

- **Measurement of relative humidity**
  - **Retrieve the relative humidity:**

```
humidity = RX_I2C_2_environment_sensor.get_humidity() print(f "Relative
humidity: {humidity} %")
```

        - **get_humidity():** Returns the relative humidity in percent (%). This measurement is important for controlling air quality, avoiding excessive dryness or high humidity and for comfort and health applications.

- **Air pressure measurement**
    - **Retrieve the air pressure:**

      ```
      pressure = RX_I2C_2_environment_sensor.get_pressure()
      print(f "Air pressure: {pressure} hPa")
      ```

        - **get_pressure():** Returns the air pressure in hectopascals (hPa). This data is essential in meteorological applications as it helps to predict weather changes such as climate and storms.

- **Air quality measurement (IAQ):** The environmental sensor can measure indoor air quality (IAQ) by detecting volatile organic compounds (VOCs) in the air.
    - **Retrieve the air quality as a number:**

      ```
      iaq_number = RX_I2C_2_environment_sensor.get_indoor_air_quality_as_number() print(f "Air
      quality (IAQ): {iaq_number}")
      ```

        - **get_indoor_air_quality_as_number():** Returns a number representing the indoor air quality. This value can be interpreted to determine the overall air quality and take action if improvement is required.
    - **Retrieve the air quality as text:**

      ```
      iaq_text = RX_I2C_2_environment_sensor.get_indoor_air_quality_as_text() print(f "Air
      quality: {iaq_text}")
      ```

        - **get_indoor_air_quality_as_text():** Provides a textual description of the air quality (e.g. "Good", "Moderate", "Poor", etc.), which makes it easier for end users to interpret the results.

Calibration of the sensor: The environmental sensor may require a calibration phase to ensure the accuracy of the air quality measurements.

- **Calibration of the sensor:**

  ```
  RX_I2C_2_environment_sensor.calibrate()
  ```

    - **calibrate():** This command starts the calibration of the sensor to ensure that the air quality measurements are accurate.

- **Check the need for calibration:**

  ```
  needs_calibration = RX_I2C_2_environment_sensor.needs_calibration()
  if needs_calibration:
      print("The sensor requires calibration.")
  ```

- **needs_calibration():** Returns a boolean value indicating whether the sensor needs calibration.

Summary: The environmental sensor provides a comprehensive range of environmental measurements, including temperature, humidity, air pressure and air quality. This data is critical for a variety of applications, from environmental monitoring to smart HVAC systems. The ability to measure air quality is particularly important in environments where indoor air quality can affect health and well-being.

Combination sensor

The combination sensor, which includes an accelerometer, a gyroscope and a magnetometer. This device provides measurements of motion, orientation and magnetic field and is ideal for applications in robotics, motion detection, device stabilization and navigation. Below you will find a detailed description of the functions, initialization and data acquisition of this sensor.

Initializing the combination **sensor** To use the combination sensor, all its components (accelerometer, magnetometer, gyroscope) must be initialized with the desired settings.

```
controller.RX_I2C_3_combined_sensor.init_accelerometer(2, 8, False)
controller.RX_I2C_3_combined_sensor.init_magnetometer(25)
controller.RX_I2C_3_combined_sensor.init_gyrometer(250, 12, False)
```

- **Accelerometer:**
    - **init_accelerometer(range, bandwidth, high_res):**
        - **range:** Defines the measuring range of the acceleration. In this case, 2 specifies a range of ±2g.
        - **bandwidth:** Defines the bandwidth of the output filter. Here 8 specifies the bandwidth in Hz.
        - **high_res:** A Boolean value that specifies whether the high-resolution mode is used.
- **Magnetometer:**
    - **init_magnetometer(data_rate):**
        - **data_rate**: Defines the data rate (in Hz) for the magnetometer. 25 specifies a data rate of 25 Hz.
- **Gyroscope:**
    - **init_gyrometer(range, bandwidth, high_res):**

- **range:** Defines the measuring range of the rotation speed. Here 250 specifies a range of ±250 °/s.
- **bandwidth:** Defines the bandwidth of the output filter. 12 specifies the bandwidth in Hz.
- **high_res:** A Boolean value that specifies whether the high-resolution mode is used.

Data acquisition of the combination sensor

1. **Accelerometer:** The accelerometer measures linear acceleration in the three directions (x, y, z), which is useful for determining relative position, speed and motion detection.
   - **Calling up the acceleration:**
     ```
     acc_x = controller.RX_I2C_3_combined_sensor.get_acceleration_x()
     acc_y = controller.RX_I2C_3_combined_sensor.get_acceleration_y()
     acc_z = controller.RX_I2C_3_combined_sensor.get_acceleration_z()
     print(f "Acceleration - X: {acc_x} g, Y: {acc_y} g, Z: {acc_z} g")
     ```
     - **get_acceleration_x():** Returns the acceleration on the x-axis in units of g.
     - **get_acceleration_y():** Returns the acceleration on the Y-axis in units of g.
     - **get_acceleration_z():** Returns the acceleration on the Z-axis in units of g.
2. **Magnetometer:** The magnetometer measures the strength of the magnetic field in the three directions and helps to determine the orientation in relation to the earth's magnetic field.
   - **Retrieve the magnetic field:**
     ```
     mag_x = controller.RX_I2C_3_combined_sensor.get_magnetic_field_x()
     mag_y = controller.RX_I2C_3_combined_sensor.get_magnetic_field_y()
     mag_z = controller.RX_I2C_3_combined_sensor.get_magnetic_field_z()
     print(f "Magnetic field - X: {mag_x} µT, Y: {mag_y} µT, Z: {mag_z} µT")
     ```
     - **get_magnetic_field_x():** Returns the strength of the magnetic field on the X-
       axis in microtesla (µT).
     - **get_magnetic_field_y():** Returns the strength of the magnetic field on the Y-
       axis in microtesla (µT).
     - **get_magnetic_field_z():** Returns the strength of the magnetic field on the Z-
       axis in microtesla (µT).
3. **Gyroscope:** The gyroscope measures the rotational speed in the three directions and provides essential data for controlling orientation and stabilization.

- **Retrieve the rotation speed:**

```
rot_x = controller.RX_I2C_3_combined_sensor.get_rotation_x()
rot_y = controller.RX_I2C_3_combined_sensor.get_rotation_y()
rot_z = controller.RX_I2C_3_combined_sensor.get_rotation_z()
print(f "Rotation speed - X: {rot_x} °/s, Y: {rot_y} °/s, Z: {red_z} °/s")
```

  - **get_rotation_x():** Returns the rotation speed on the X-axis in degrees per second (°/s).
  - **get_rotation_y():** Returns the rotation speed on the Y-axis in degrees per second (°/s).
  - **get_rotation_z():** Returns the rotation speed on the Z axis in degrees per second (°/s).

Summary: The combination sensor provides a comprehensive solution for measuring motion, orientation and magnetic field. With capabilities to measure acceleration, magnetic field and rotational speed, this sensor is extremely useful for applications in robotics, navigation, device stabilization and other motion control systems. Each of the measurements captured provides a detailed understanding of the connected device's state and motion, facilitating a wide range of applications from fall detection to position and orientation sensing.