Link to bootcamp repo: `https://github.com/UTSARobotics/ros1_bootcamp`

1 Introduction

Today's activity will demonstrate hardware integration and visualization in a ROS environment. We will use the ROS Serial Arduino Package to send IMU data from an MPU6050 and ESP32S to a PC, and then visualize that data in RViz. All required libraries will be included in `src/mpu6050_esp32/` package.

1.1 Materials

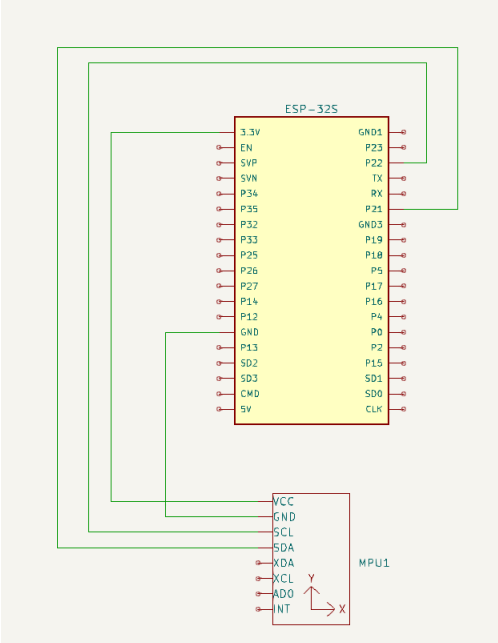
In this activity, you will need the following materials:

- ESP32S development board
- MPU6050 IMU sensor
- Breadboard
- Jumper wires
- USB cable (for programming and power supply)

2 Testing devices with Adafruit MPU6050 example over serial

To start off we're going to verify all components are working as they should. This'll also give us some example code to work with.

- Connect all your components as such:



- Get example code from here or look at solution branch of our repo
- Compile and upload code. While uploading hold down "boot" button, release when you see code begin to upload.
- View data in serial monitor

3 Programming ROS Serial Arduino

In this part, we will program the ESP32 using the ROS Serial Arduino package. This process involves several steps:

- Create udev rules: We first create udev rules for the ESP32 to ensure that it always gets the same port name when connected to the PC. This makes it easier to communicate with the ESP32 from ROS.
- Modify the the original sketch: Modify the original sketch to read data from the MPU6050 and publish it as a sensor_msgs/Imu message. This message consists of linear acceleration, angular velocity, and orientation data, all of which are represented as float64 data types. This involves initializing the MPU6050 in the setup() function and reading acceleration and gyroscope data in the loop() function.
- Start ROS master: Before running the rosserial python client to establish a connection between the ESP32 and ROS, we need to start a ROS master by running `roscore`.
- Run the rosserial python client: Once we have a ROS master running, we can start the rosserial python client, which establishes a connection between the ESP32 and ROS. This allows the ESP32 to publish messages to ROS topics.
  ```
  rosrun rosserial_python serial_node.py _port:=/dev/esp32 _baud:=57600
  ```
- Echo and visualize data: With the connection established, we can echo the IMU data by running `rostopic echo /imu` and visualize the IMU data in RViz.

4 Filter Setup and Usage

The raw readings from the IMU can be noisy and may not provide accurate orientation data. Therefore, we use a filter to estimate the orientation from the acceleration and gyroscope data. The Madgwick filter is a popular choice for this purpose.

The Madgwick filter uses a quaternion representation for the orientation. Quaternions are a mathematical construct that extends the concept of rotation in three dimensions. They consist of four parts: a scalar part and a vector part. The scalar part represents the amount of rotation, and the vector part represents the axis about which the rotation occurs.

Here are the steps we follow to incorporate the filter:

- Modify the sketch: We modify our sketch to compute the orientation using the Madgwick filter. This involves initializing the filter in the setup() function and updating the filter with the acceleration and gyroscope data in each loop iteration.
- Add orientation to the IMU message: After updating the filter, we add the computed orientation to the sensor_msgs/Imu message. This involves converting the orientation from the filter's internal quaternion representation to a geometry_msgs/Quaternion message, which can be included in the sensor_msgs/Imu message.

- Publish the message: Finally, we publish the sensor_msgs/Imu message with the computed orientation. This message can be echoed and visualized just like the raw IMU data.