Design

Architectures

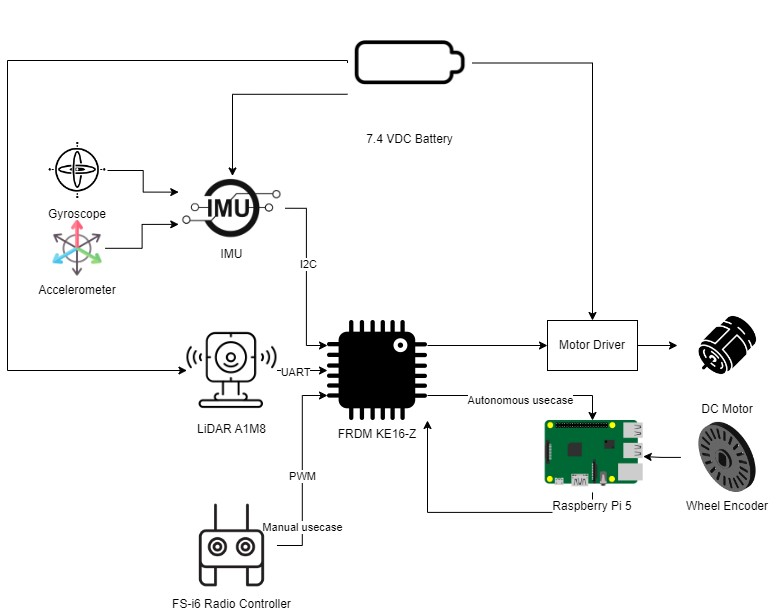


Figure 1-System Architecture

The entire system is powered using a 2S LiPo battery, limited using a Step-Down Buck Converter, chosen due to the specifications of the Raspberry Pi 5 which requires a 5V and 5A input.

The inputs of the FRDM KE-16Z development board are the IMU, comprised of a Gyroscope and an Accelerometer, the Lidar and the radio controller. The data from the radio controller is processed and given to the motor driver, hence obtaining the driving and steering control of the application. The acquired data from the IMU and the Lidar is sent to the Raspberry Pi 5.

The Raspberry Pi 5 receives the Lidar and IMU data from the NXP development board and acquires data from the wheel encoder. This data is further processed using ROS and a SLAM framework.

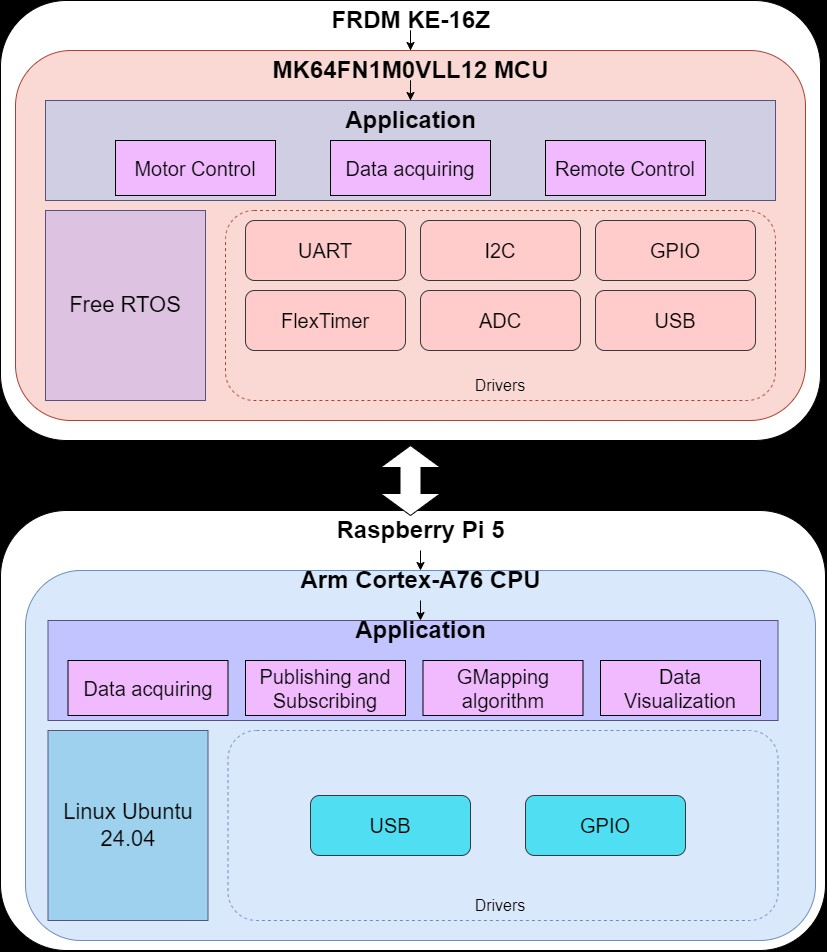


Figure 2-Software Design

The two boards used in this application are the NXP FRDM KE-16Z development board and the Raspberry Pi 5. They are transmitting data between them serially.

The NXP FRDM KE-16Z development board uses FreeRTOS for task scheduling.

It uses the UART, I2C, GPIO, ADC and FlexTimer drivers for communicating with the sensors and the remote controllers, as well as controlling the ESC and the servo motor (therefore the driving and steering of the robot).

The applications developed on this board are Motor Control, Data Acquiring from the sensors and Remote Control for processing the commands received from the controller.

The Raspberry Pi 5 uses Linux Ubuntu 24.04LTS as its Operating System, chosen due to its compatibility with the latest ROS version, ROS2 Jazzy Jalisco.

The driver used in the application is GPIO, used for acquiring data from the wheel encoder.

The applications developed and run on this development board are Data Acquiring from the encoder, Publishing and Subscribing, the GMapping SLAM algorithm and Data Visualization of the acquired map in Rviz.

Hardware Design

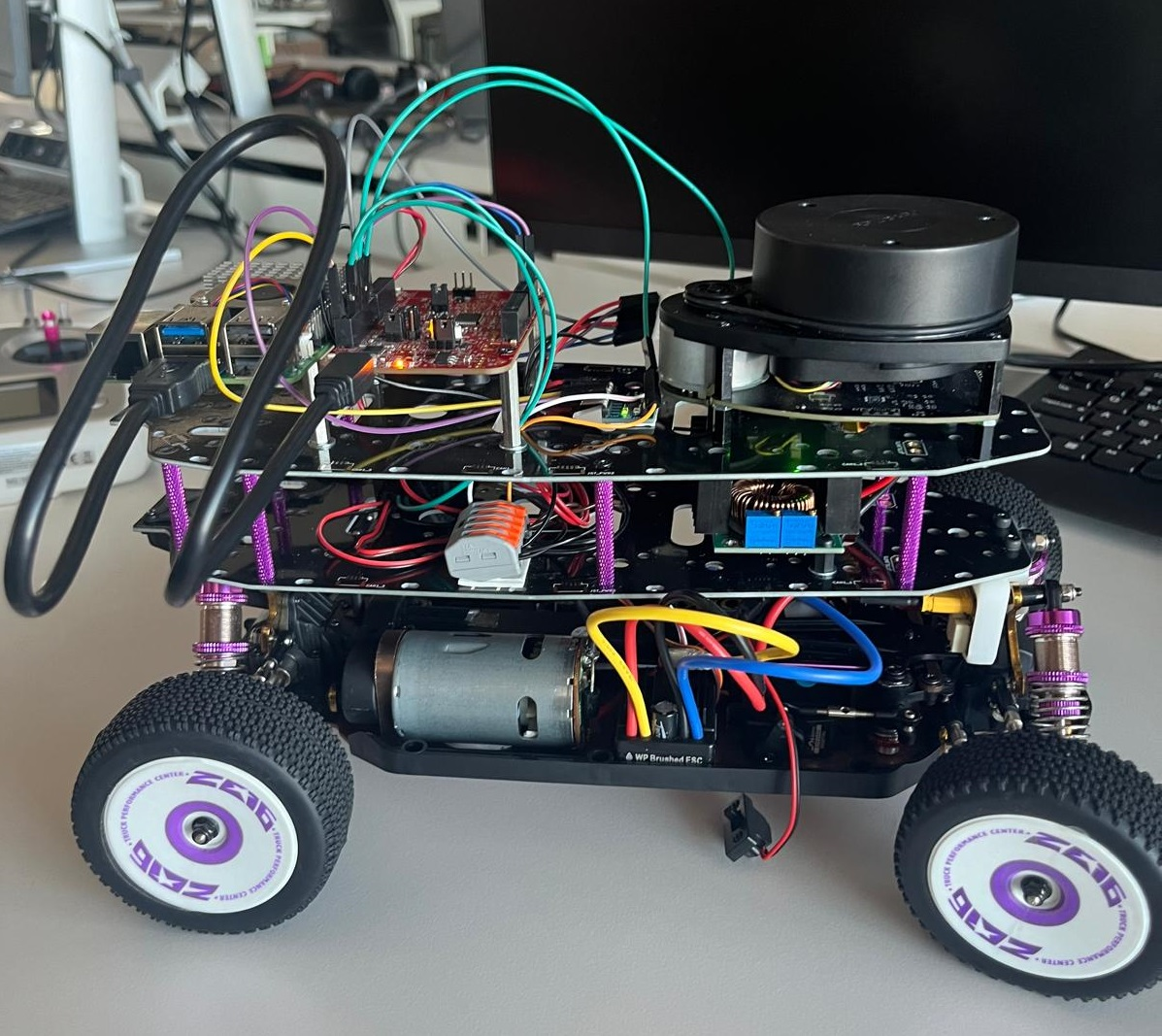


Figure 3-The developed robot

The elements of the robot are arranged on three levels.

On the first level, the motor, ESC, servomotor and battery can be found, as well as the Power Distribution Board. The wheel encoder is attached to the axis of the robot.

On the second level, we can find the Power Management system. Here we can find the Step-Down Buck Converter that connects to two Wago connectors, one for the ground and one for the power supply. These are used for distributing power for the entirety of the system using jumpers. The receiver of the FlySky FS-i6 radio controller is also placed here.

On the third and highest level, we can see the NXP FRDM KE-16Z development board and the Raspberry Pi 5. The IMU and Lidar are also found on this level and connected to the NXP dev board accordingly.

Software Design

There are two main parts of this project, robot\_control and slam\_processing, each of them developed on a different board.

On the NXP dev board, the following modules can be found:

* rbwsFreeRTOS
* rbwsMotorControl
* rbwsPWT
* rbwsInertialMeasurementUnit
* rbwsKalmanFilter
* rbwsLidarA1M8

On the Raspberry Pi 5, a workspace was created for ROS2 and multiple packages were created. In the end application, we can run the data acquiring nodes from these packages if we were to only acquire data or run the mapping node, where the data acquiring, processing and visualization in Rviz2 can all be found in the same place. However, if we were to list all the available packages developed here, in a more succinct manner, these are:

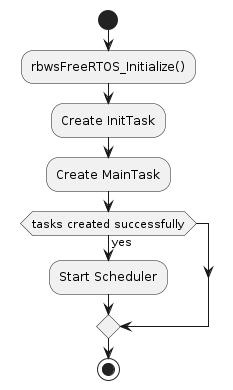
* imu\_data
* lidar\_data
* encoder\_data
* common\_serial\_service
* my\_gmapping\_package

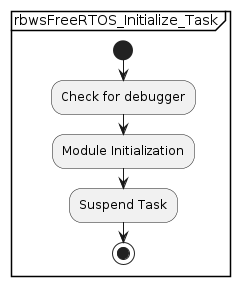
The modules developed on the NXP dev board are written in the C programming language, meanwhile the modules developed on the Raspberry Pi 5 are written in Python.

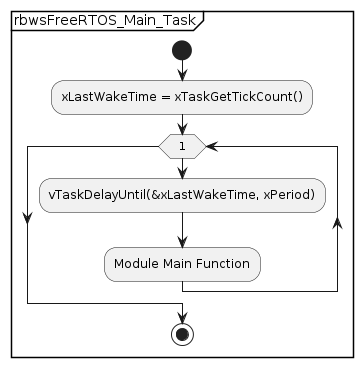
All these modules are described using Activity Diagrams created in PlantUML.

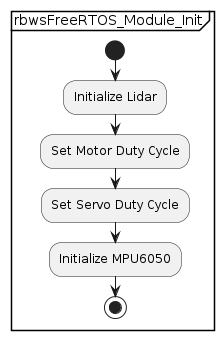
All the modules and what they do are presented in detail in the final documentation of my bachelor’s thesis and the images are uploaded to the repository containing its implementation.

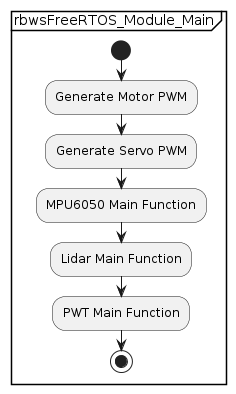
**FreeRTOS Activity Diagrams**

****

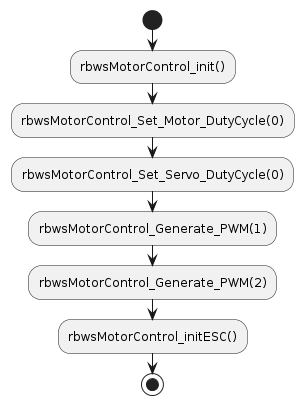
****

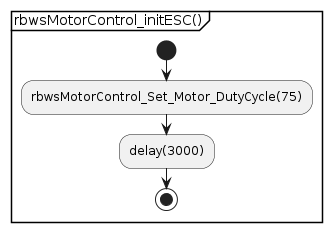
****

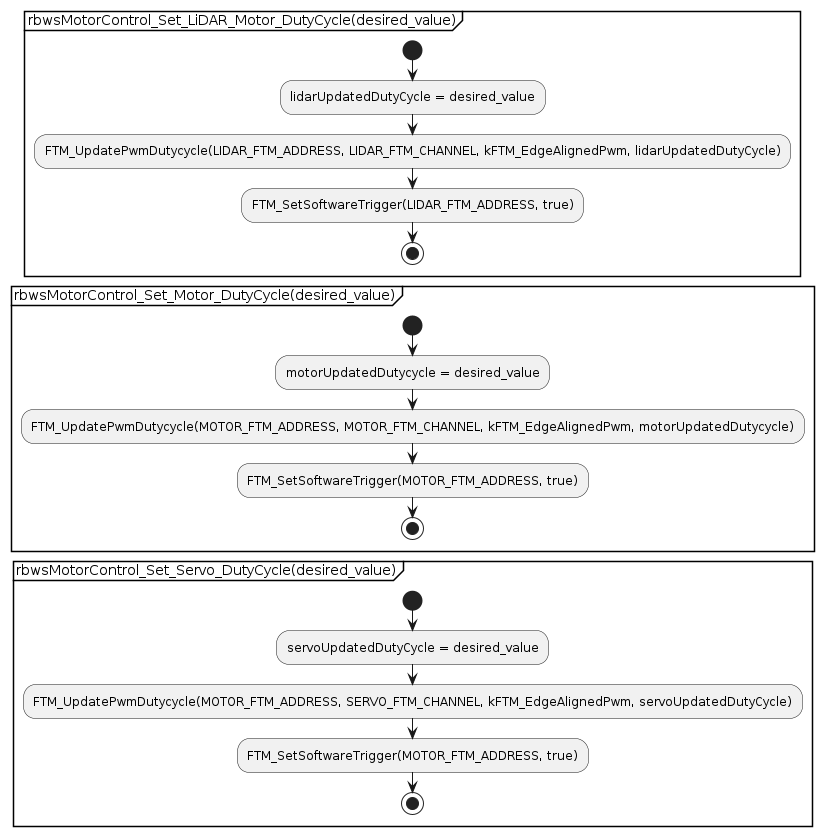
****

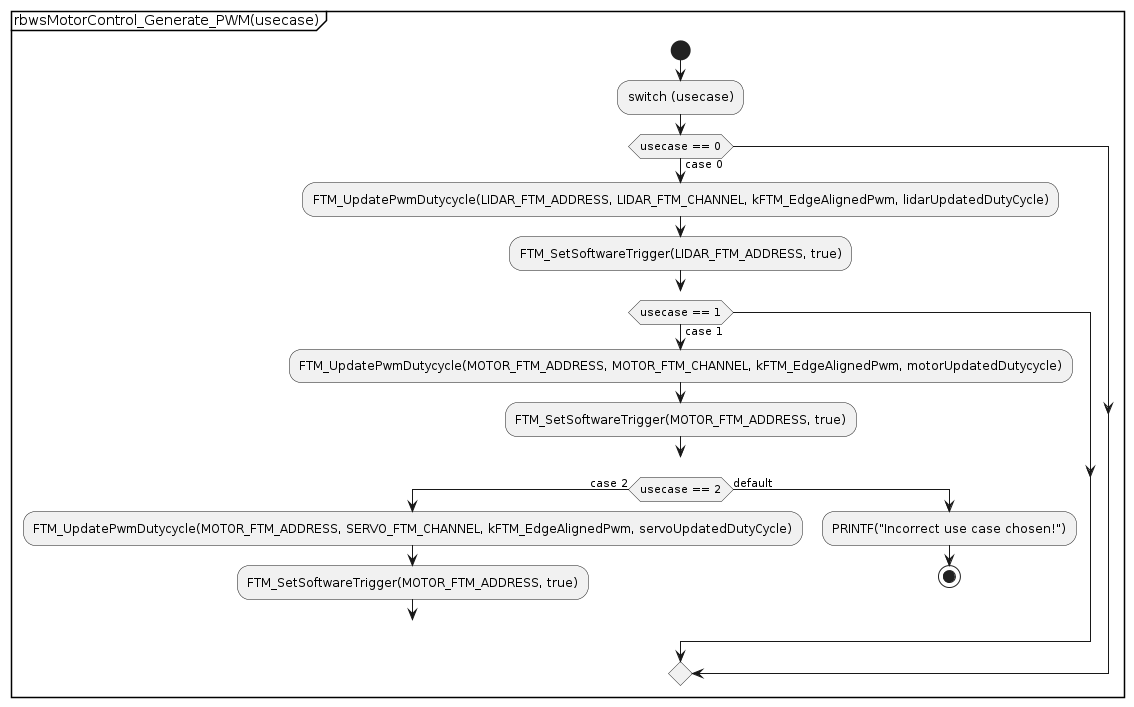
****

**Motor Control Activity Diagrams**

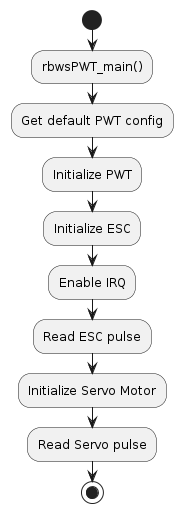
****

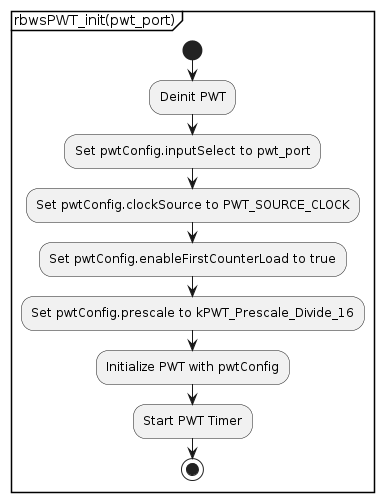
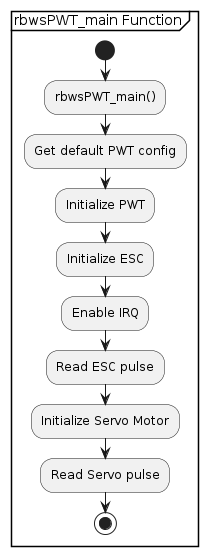
****

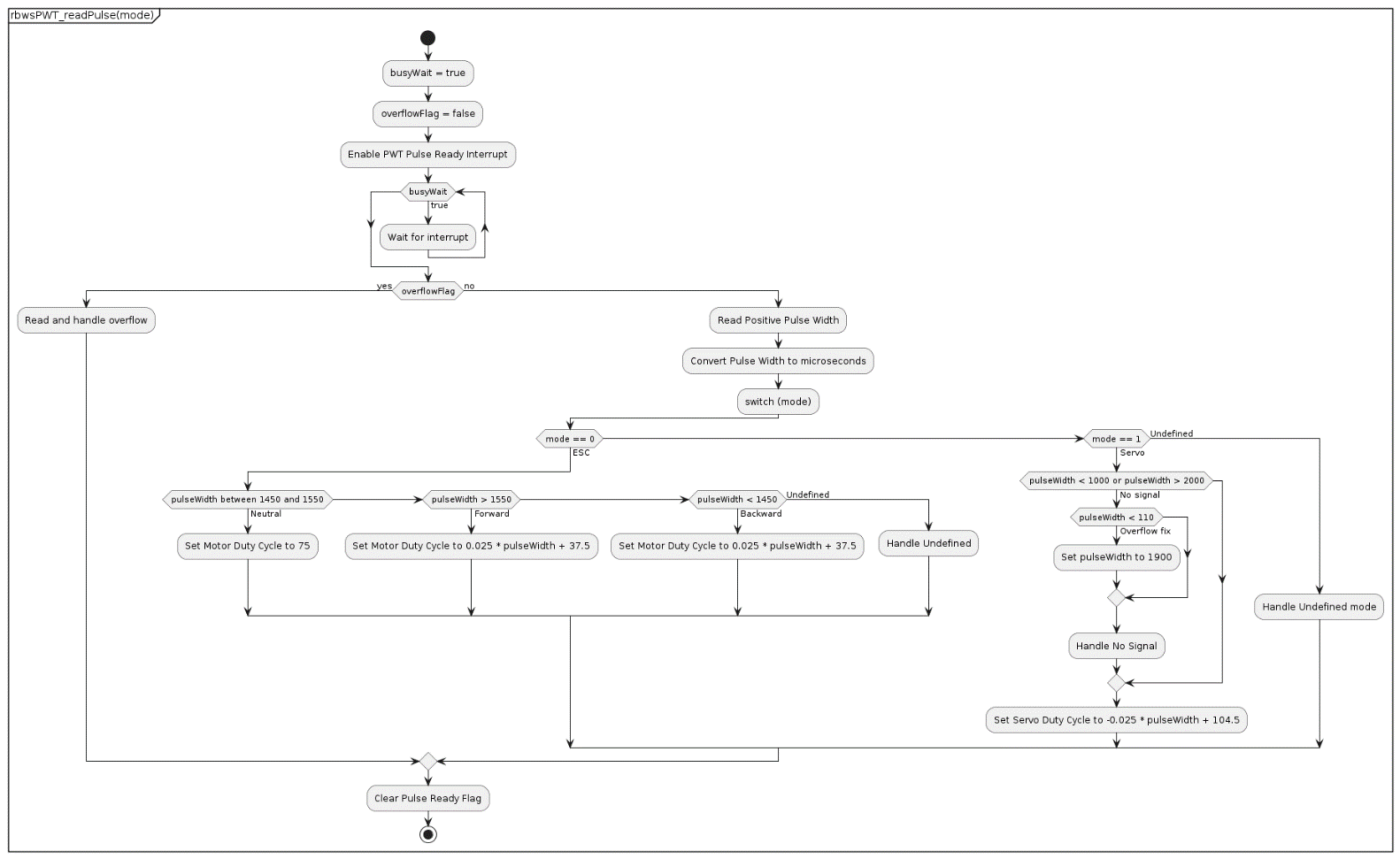
****

****

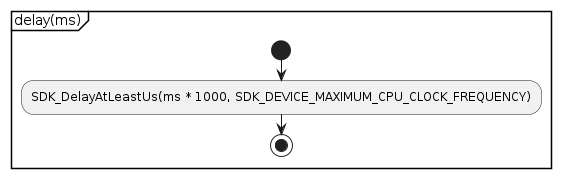
**Remote Control Activity Diagrams**

****

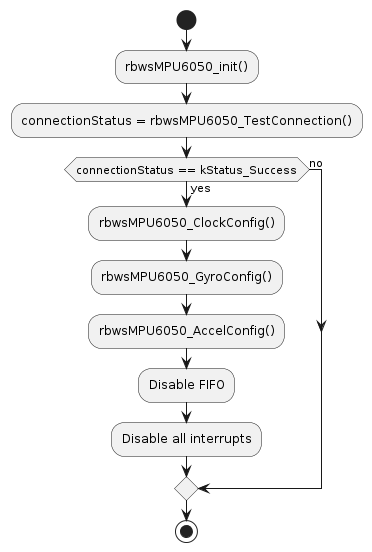
****

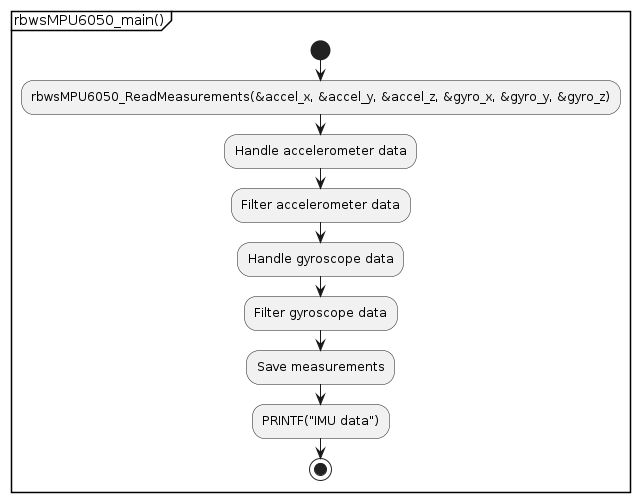
****

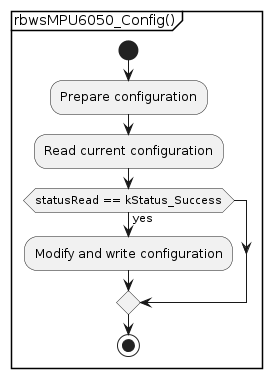
**General Purpose Diagrams (used by most, if not all modules)**

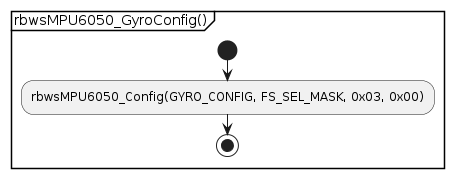
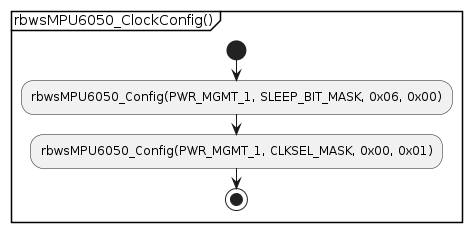
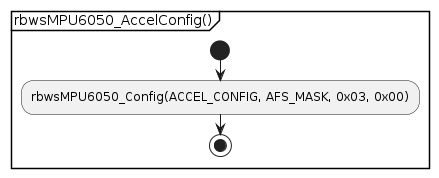
****

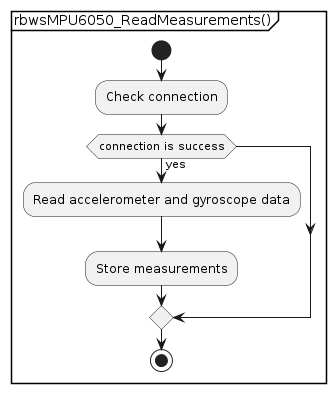
**Inertial Measurement Unit Activity Diagrams**

****

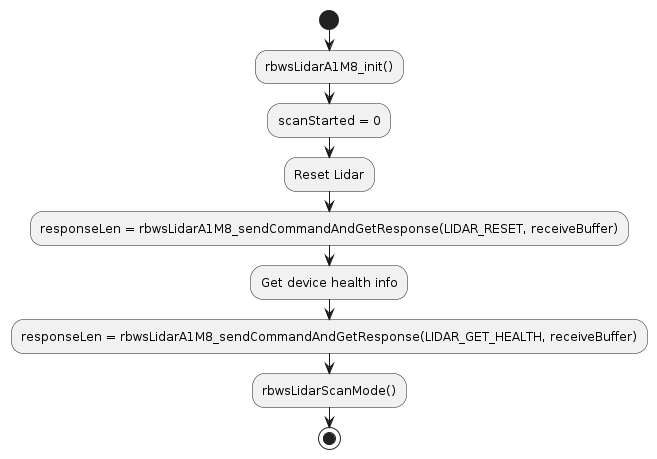
****

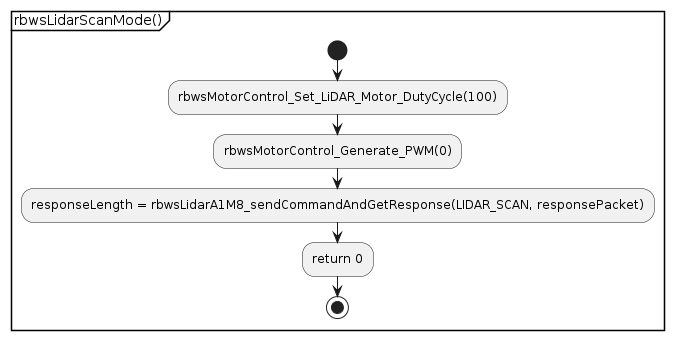
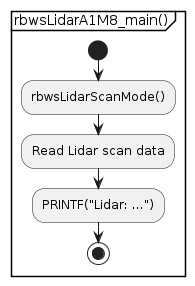
****

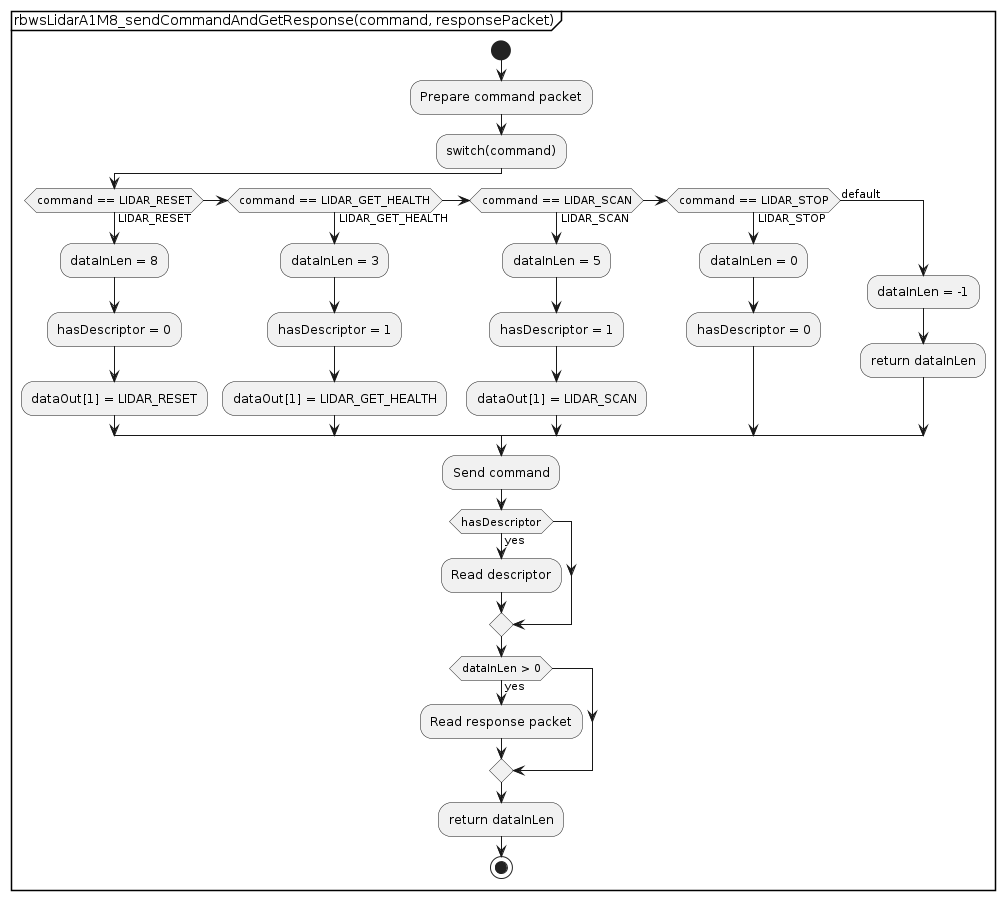
****

****

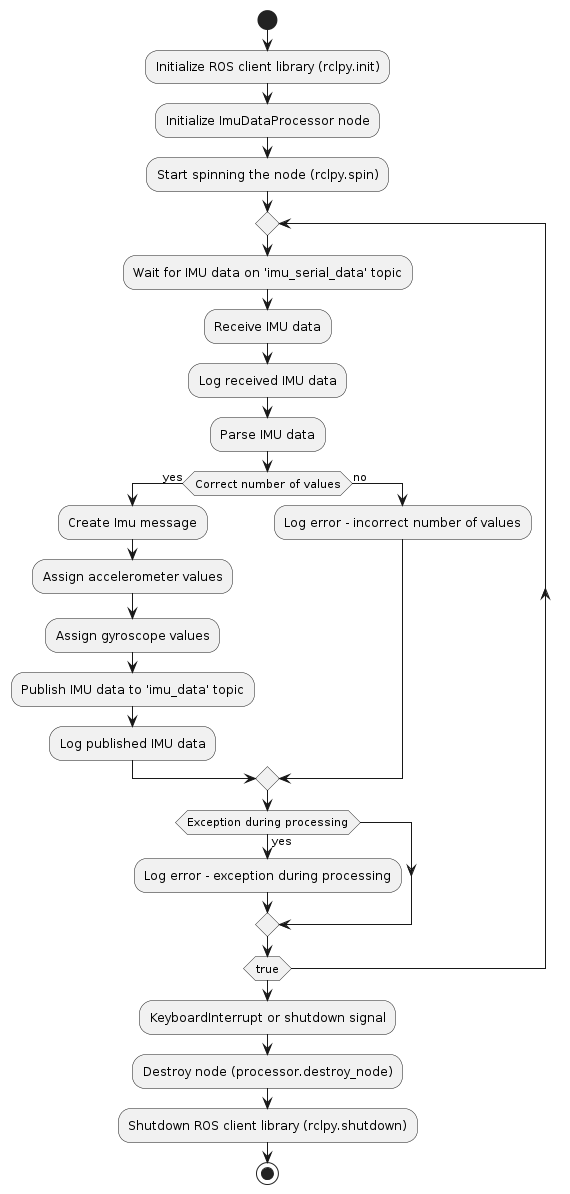
**Lidar Activity Diagrams**

****

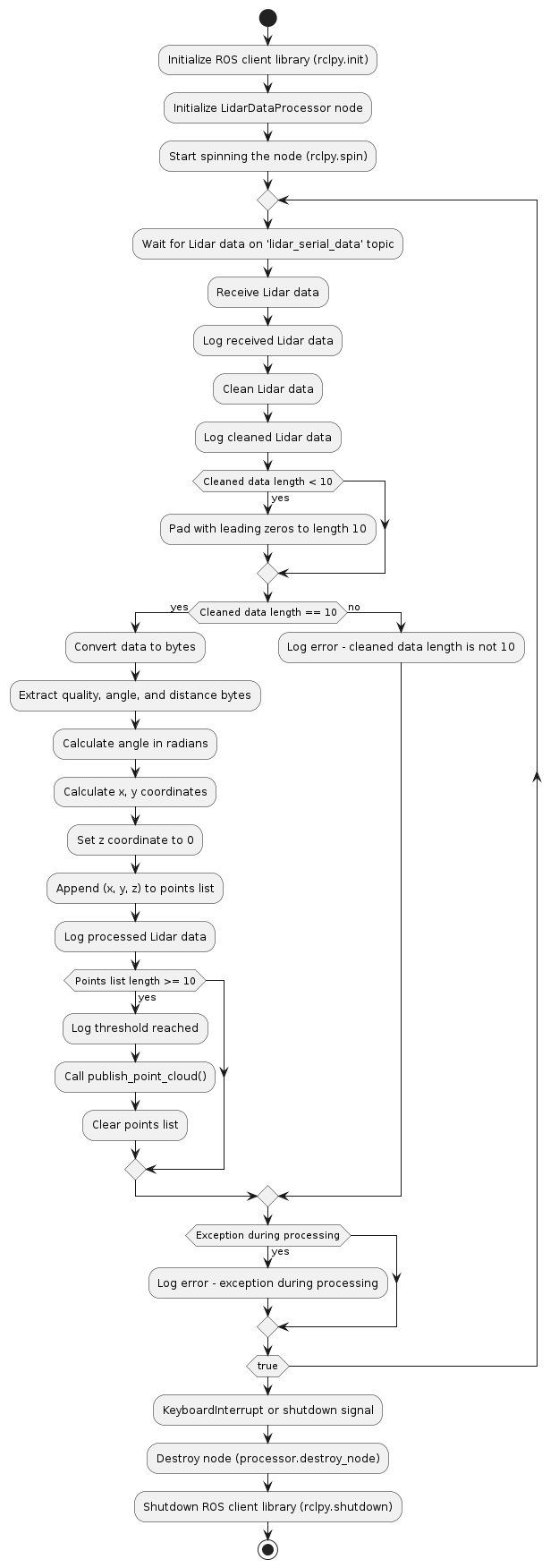
****

****

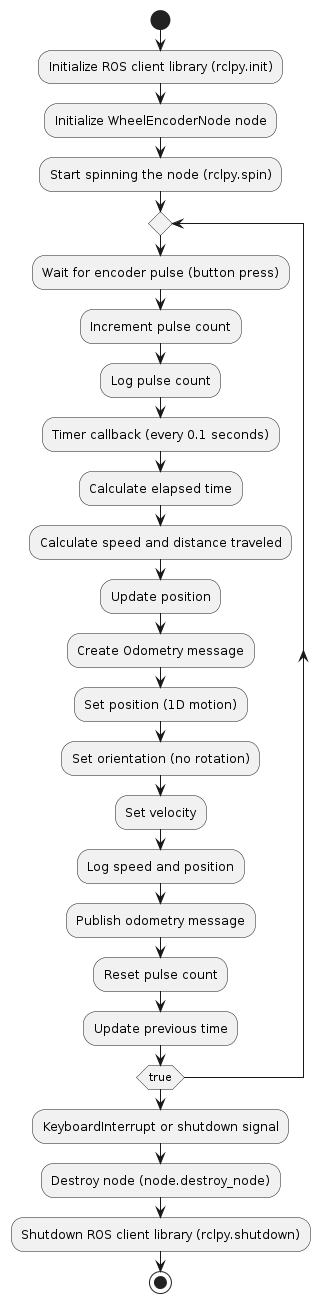
**Imu Node Activity Diagram**

****

**Lidar Node Activity Diagram**

****

**Wheel Encoder Node Activity Diagram**

****