4D-Joystick – Safety features

1. Introduction

The 4D-Joystick has implemented some safety features, which provide safe operation. The main purpose is to avoid software and communication failures. This document gives an overview which and how safety features are implemented.

2. Communication

A SPI-interface is used to exchange data between the joystickunit (master) and the remoteunit (slave). The transfer speed is quite high, the cable is long and the remoteunit is a quite strong wireless transmitter which can also interfere with the SPI interface and the long cable. This all can lead to transfer failures.
Because a lot of packages get lost at high SPI frequencies (~20% of packages are lost at 2 MHz) the transfer frequency was reduced dramatically (40kHz). This led to a stable communication, where no packages get lost (at least as tested so far).

However, to further improve communication an additional safety step was implemented. Each package contains a CRC value, which can be checked at the receiver. If the CRC is wrong, the package is ignored and the last valid package is used for the next cycle. Because the communication cycle is quite short (~4ms) the user will not even notice if a single package gets lost.

If a lot of package get lost (25 consecutive packages) the remoteunit will switch to a failsafe mode. It will update the output with the inputs of the remote, like it does in teacher mode or without a joystickunit, at a reduced rate (once every 100ms). The remoteunit will try to start communication with the joystickunit again and switches to normal operation as soon as a valid package is received.

3. Remoteunit

The firmware of the remoteunit is a quite simple software, which is running in a loop. Every loop is one full communication cycle with the joystickunit (read inputs, send inputs to joystickunit, receive outputs from joystickunit, set outputs).
However, to avoid that the software is stuck somewhere in this loop a watchdog is used. The watchdog is a hardware peripheral of the controller which resets the device if it not reaches certain checkpoints in a fixed period.

In case of the remoteunit this checkpoint is set at the end of the loop, after the outputs are set. Normally this checkpoint should be reached every ~4ms. However, especially during first run this can take longer, because initialization is not finished and the first communication cycle can be missed. For this reason, the watchdog is configured to reset the controller when loop execution takes longer than 15ms.

4. Joystickunit

The joystickunit also uses a watchdog to ensure that software didn't stuck somewhere, but the firmware is more complex because it used a RTOS. Therefore, the software does not run in one simple loop, but is divided in task which run in loops. There are five tasks:

- **system-task:** is called every ~250ms with middle priority and handles general system work (debug-led, voltage monitoring, …).
- **remoteunit-task:** is called every ~4ms with high priority and handles communication with joystickunit and processing of the related data
- **buttons-task:** is called every ~2ms with middle priority and handles all button inputs (buddybuttons and rotary encoder).
- **userinterface-task:** is called every ~100ms with low priority and handles the display and interpretation of the rotary encoder.
- **CLI-task:** is called every ~10ms with low priority and handles command line inputs.

For a save operation it is mandatory that the remoteunit-task, the system-task and the buttons-task is executed periodically. The other tasks are not mandatory for main operation (only for configuration).

However, only one task can be monitored by the watchdog. Due to design decisions the system-task is monitored. The watchdog of this controller is more advanced than the remoteunit-watchdog and has a window mode. For this reason, the watchdog not only checks if the checkpoint was reached within a fixed time, but also check if the time was to short to reach the checkpoint (f.e. because the checkpoint itself is stuck in a loop).

Normally the system task should be executed every ~250ms. The watchdog will reset the controller if time between execution of task is lower than ~150ms or if task is not executed for more than 300ms.

To monitor the other tasks a software-watchdog is implemented in the system-task. Each task must notify the system-task after each execution. The system-task counts the notifications for each task and knows a minimum value of execution during one system-task-cycle. If this minimum number is not reached, the system-task will reset the controller.

Minimum amount of execution during one system-task-cycle (~250ms):

- **remoteunit-task:** 30 (should normally be ~60)
- **buttons-task:** 60 (should normally be ~120)
- **userinterface-task:** 1 (should normally be 2 or 3)

However, the CLI-task is not monitored. It is blocked during command-execution and user inputs and therefore cycle time has high variances and is depended on the user.
It is not mandatory that CLI-task is working correctly for main operation. The user will notice if this task is stuck and can restart the joystick on his own.

Therefore, the system-task is monitored by the watchdog and every other task (expect CLI-task) is monitored by the system-task. This leads to a safe system (at least in relation to stuck of software).