UART Communication Protocol for our Autonomous Vehicle

Team 10 - PREN2

March 5, 2025

1 Introduction

This document describes a 64-bit fixed-size UART protocol designed for our autonomous vehicle. Key requirements are:

- Low overhead,
- Robustness with CRC error detection,
- Fixed frame size (2 read cycles for each frame),
- Ability to carry up to 50 bits of parameter/reserved data.

Each 64-bit frame is laid out as follows:

$$\underbrace{\text{Addr}}_{\text{2 bits}} \parallel \underbrace{\text{Command}}_{\text{4 bits}} \parallel \underbrace{\text{Parameter/Reserved}}_{\text{50 bits}} \parallel \underbrace{\text{CRC-8}}_{\text{8 bits}}$$

2 Frame Definition

A single UART frame is 64 bits. The fields are:

- **Address (2 bits)**:

  $$\text{Addr} \in \{0b00,\ 0b01,\ 0b11\}$$

  Example address mapping:

  $$0b00 \to \text{Raspberry Hat},$$
  $$0b01 \to \text{Motion Controller},$$
  $$0b11 \to \text{Grip Controller}.$$

- **Command (4 bits)**:

  $$\text{Command} \in \{0x0, \dots, 0xF\}$$

  Commands currently in use are documented in Table 1.

- **Parameter/Reserved (50 bits)**:

  $$\underbrace{p_{49}\, p_{48}\ \cdots\ p_1\, p_0}_{\text{50 bits}}$$

  This field can encode various data types, including those listed in Table 2 (e.g. `int16`, `float16`, flags, etc.) depending on the command. If fewer than 50 bits are needed, the unused bits can be reserved or zeroed.

- **CRC-8 (8 bits)**:

  $$\underbrace{c_7\, c_6\ \cdots\ c_1\, c_0}_{\text{8 bits}}$$

  This is a standard 8-bit CRC computed over the top 56 bits.

3 Command Reference

Hex	Binary	Command / Parameters
`0x0`	`0b0000`	Move (int16: distance in cm, 0/unused for infinite)
`0x1`	`0b0001`	Reverse (int16: distance in cm, 0/unused for infinite)
`0x2`	`0b0010`	Turn (int16: degrees, +right / −left)
`0x3`	`0b0011`	Rotate (int16: degrees, default 180 if 0/unused)
`0x4`	`0b0100`	Stop Normal (open)
`0x5`	`0b0101`	Stop Emergency (open)
`0x6`	`0b0110`	Info (open)
`0x7`	`0b0111`	Ping (open)
`0x8`	`0b1000`	Error Check (open)

Table 1: Command Codes, in Hex and Binary, with Parameter Definitions

4 Type Reference

Type	**Range / Description**
`int16`	-32768 to 32767
`int8`	-128 to 127
`uint16`	0 to 65535
`uint8`	0 to 255
`fp16 / float16`	half precision
`fp8 / float8`	mini precision

Table 2: Possible Data Types for Parameters

5 CRC-8 Computation

5.1 Polynomial Definition

The polynomial we'll use for `CRC-8` is:

$$x^8 + x^2 + x^1 + x^0,$$

often represented in hexadecimal as `0x07`.

5.2 Why This Polynomial?

The polynomial

$$P(x) \;=\; x^8 + x^2 + x + 1$$

is known to be *primitive* over the finite field GF(2). This implies that the linear feedback shift register (LFSR) based on $P(x)$ can generate any nonzero 8-bit sequence in a maximal cycle (of length $2^8 - 1 = 255$). In terms of CRC properties, using a primitive polynomial of degree 8 provides:

- **Guaranteed detection of any single-bit error**: A one-bit error corresponds to the polynomial x^k for some k. Because $P(x)$ is primitive, x^k is never divisible by $P(x)$ for $0 \le k < 255$.
- **Guaranteed detection of many double-bit errors**: If two bits are in error, the error polynomial is $x^m + x^n$. Since $P(x)$ is primitive, it cannot divide $x^m + x^n$ unless $m = n$. Thus, distinct double-bit errors (i.e. $m \neq n$) are almost always detected.[1]
- **Detection of bursts up to 8 bits long**: In general, an n-bit CRC can reliably detect bursts of up to n bits in length. Here, that means any burst error of up to 8 consecutive bits will *not* yield a zero remainder when dividing by $P(x)$.

Short Frame Advantage: Because our frames are only 64 bits, the chance of an undetected multi-bit error is further reduced. For random error patterns, the probability of an undetected error is at most $1/2^8 = 1/256$. Actual practical detection is typically even better than the pure random guess model, especially for correlated or bursty errors, due to how the polynomial aligns with short data lengths.

[1] Strictly speaking, *no* CRC polynomial can detect *all* double-bit errors at arbitrary distances, but a primitive polynomial has a higher minimum distance for typical codeword lengths and is exceptionally good for short frames like ours.

5.3 Computation Steps (Bitwise)

1. Initialize the CRC register:
$$\mathrm{crc} \leftarrow 0x00.$$
2. Process the first 7 bytes (56 bits) in the frame (i.e. `Addr`, `Command`, and `Parameter/Reserved`).
3. For each byte `b` among those 7 bytes:
$$\mathrm{crc} \leftarrow \mathrm{crc} \oplus \texttt{b}.$$
Then, for each of the 8 bits in that byte:
$$\begin{cases} \text{if } (\mathrm{crc}\,\&\,0x80) \neq 0: & \mathrm{crc} \leftarrow (\mathrm{crc} \ll 1) \oplus 0x07 \\ \text{else}: & \mathrm{crc} \leftarrow (\mathrm{crc} \ll 1) \end{cases}$$
(Mask `crc` with `0xFF` after each shift to keep it at 8 bits.)
4. At the end of this 7-byte process, `crc` is the final CRC-8. Store it into the last byte of the 64-bit frame.

6 Transmit & Receive Flow

6.1 Transmit Flow

1. Construct the 56 bits of data, for example:
$$\text{Frame\_Hi} = (\text{Addr} \ll 62) \mid (\text{Command} \ll 58) \mid (\text{Param}_{50} \ll 8).$$
2. Compute CRC-8 across these 56 bits, producing one byte.
3. Append this `crc_byte` as the final 8 bits.
4. Send the resulting 64-bit frame via UART.

6.2 Receive Flow

1. Read the full 64 bits from UART (fixed-size).
2. Separate the top 56 bits (`Addr + Command + Param/Reserved`) and the last 8 bits (the received CRC).
3. Compute a new CRC-8 over the 56 bits.
4. Compare the computed CRC with the received CRC:
$$\text{if computedCRC} \neq \text{receivedCRC} \implies \text{reject packet (error).}$$
5. If the CRCs match, accept the frame and parse the 56 bits according to the protocol's definitions.