Position Encoder

Erich Styger

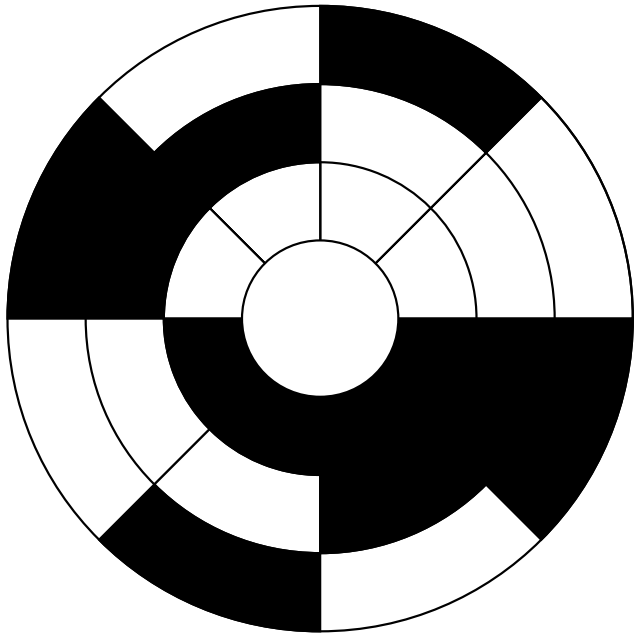
erich.styger@hslu.ch
Lucerne University of Applied Sciences and Arts

Introduction

- Need to know the position of robot
- Gyroscope, accelerometer, GPS, ...
- Wheel/shaft sensors: incremental or absolute sensors
- Will use position information for speed estimation

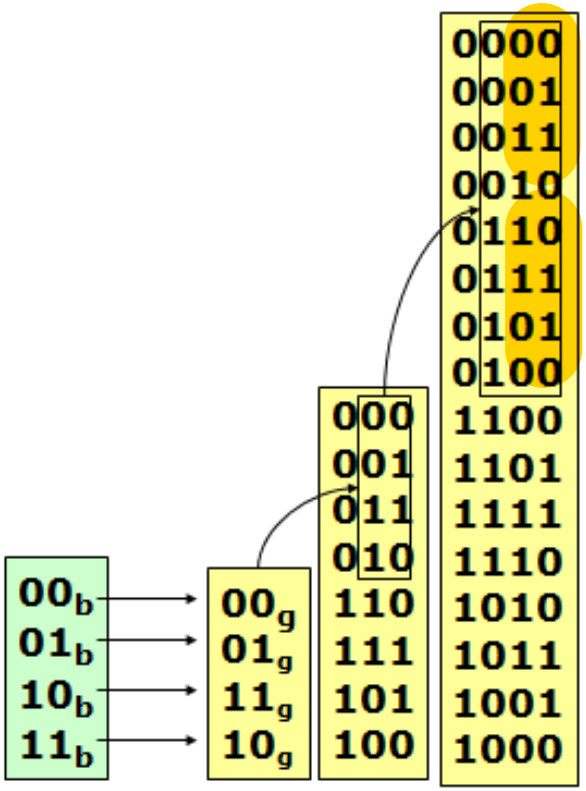
Absolute Digital Angle Encoder

- Encoder with sectors attached to shaft/wheel
- 2^n sensors for sector identification
- Mechanical issue: multiple bits changing



Binary-Reflected Gray Code

- Frank Gray, Bell Labs, patented 1947
- Hamming distance 1: two sucessive codes differ by one bit
- Same number of bits for Gray and Binary code
- *Permutation*: every code is present only once
- *Cyclic*: last code rolls over to first code
- *Recursive*: G_n is embedded in G_{n+1}



Constructing n-Bit Gray Code

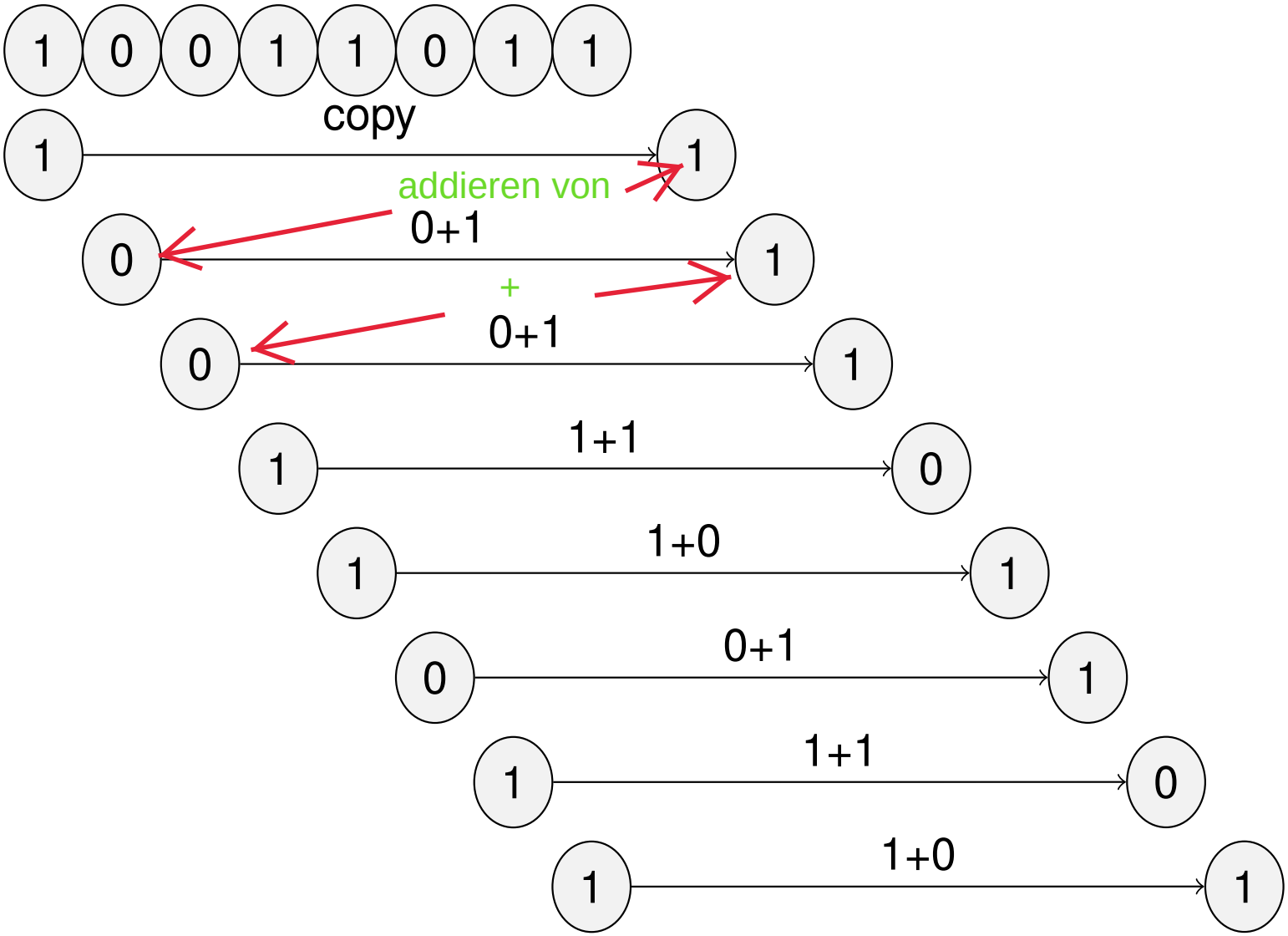
1. Reflection: Build reverse n-1 list
2. Concatenate the two lists
3. Prefix old entries with 0
4. Prefix new entries with 1

Converting Binary Code into Gray Code

1. Logical Shift right binary code
2. EXOR with binary code

$$Value_g = (Value_b >> 1) \otimes Value_b \quad (1)$$

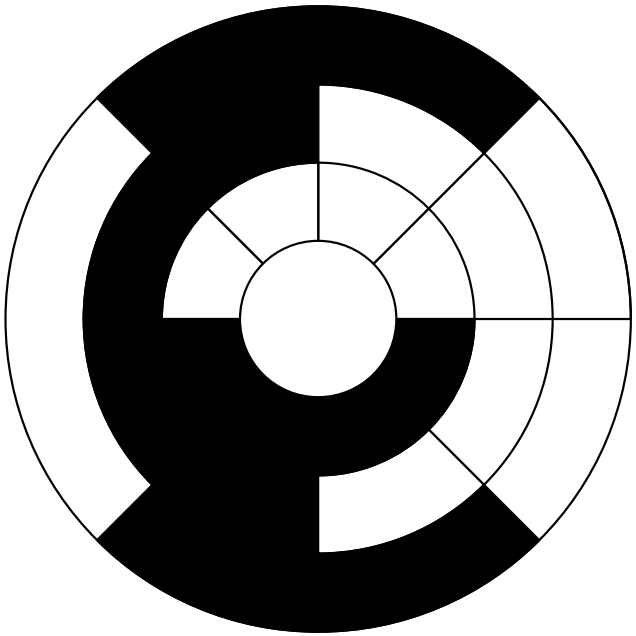
Converting Gray Code into Binary Code



Gray Absolute Angle Encoder

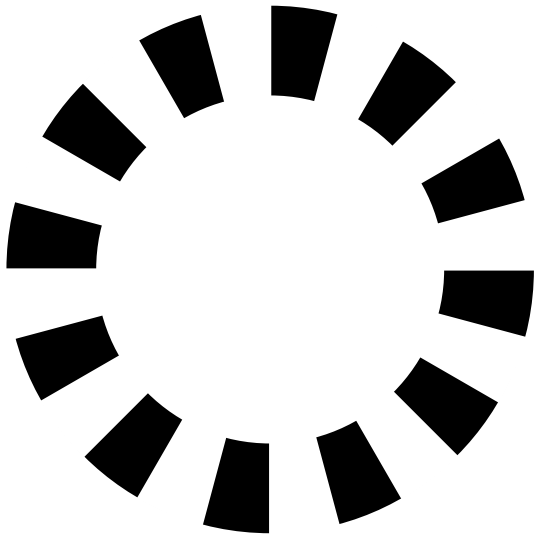
- Only one bit changes
- Need 2^n bits

nur immer ein bit ändert!



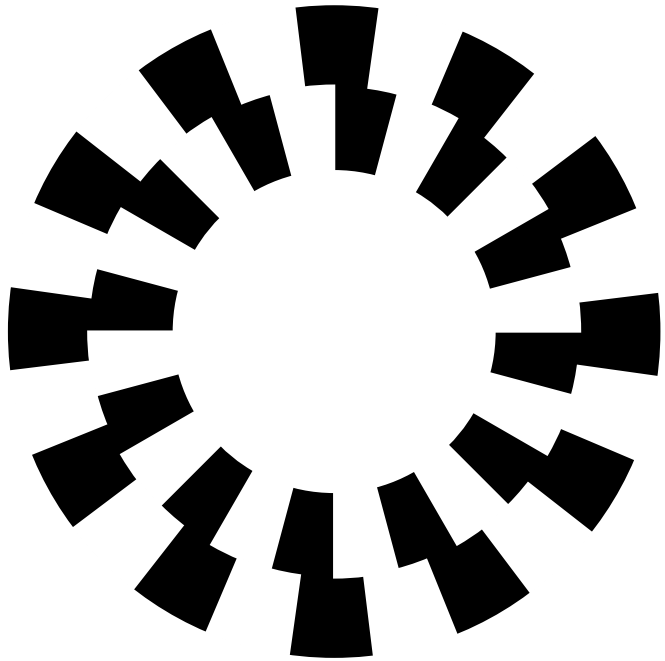
Simple Regular Encoder

- Incremental encoder
- Gray code: 0 1 0 1 ...
- No backward/forward information



Quadrature Encoder

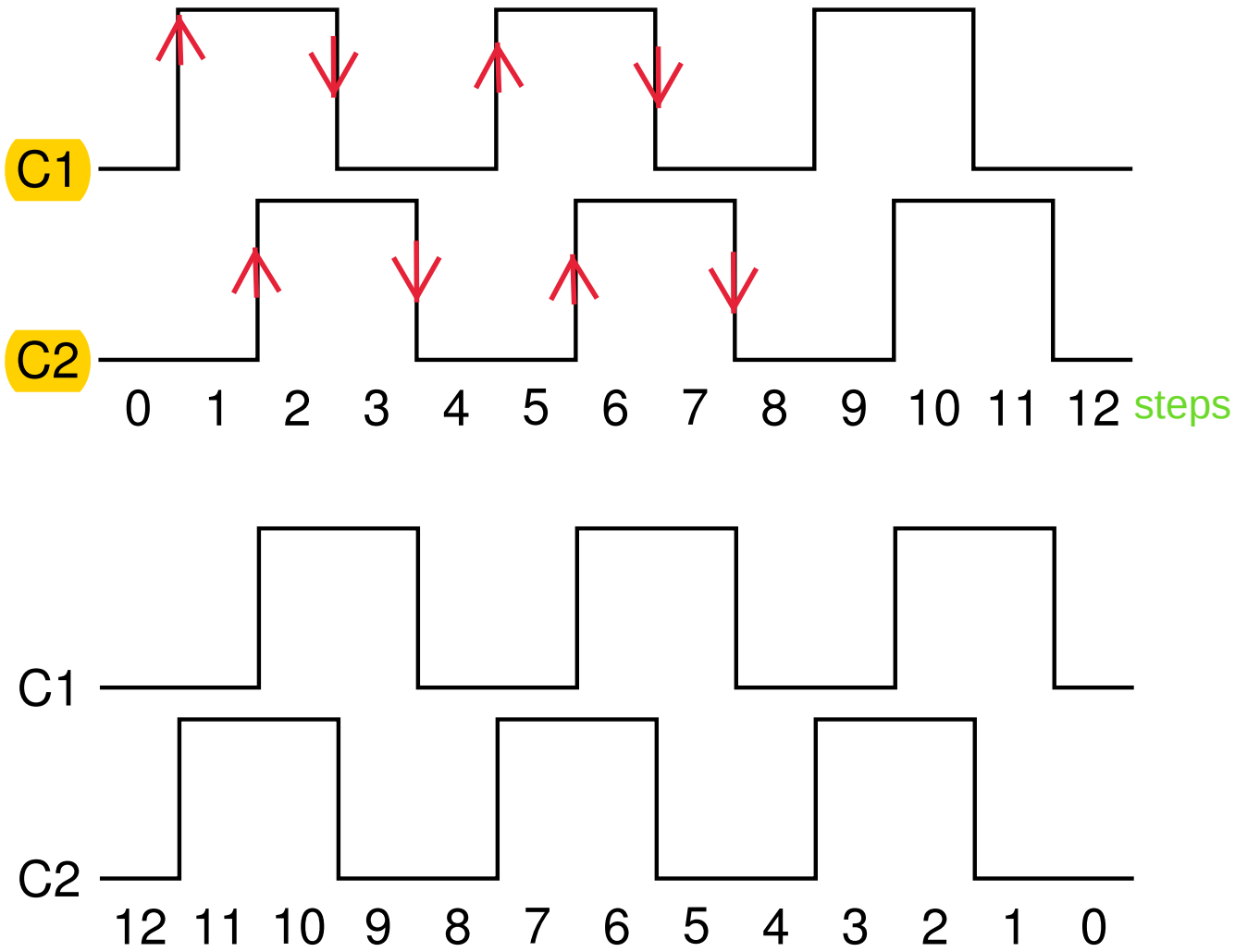
- Incremental encoder
- 2 binary signals
- Hardware implementation: one ring with 2 LED, Sensors with offset position



Quadrature Encoder

- 2 binary signals, Sinus-like
- Shifted by 90° (one quadrant)

Problem: to many interrupts



Realtime Aspects

- Data acquisition
  - Interrupts, Input Capture
  - Sampling
  - Hardware
- Sampling: Nyquist/Shannon

$$f_{sampling} > 4 * N * 2 * R \tag{2}$$

$$t_{sampling} < \frac{\frac{1s}{4*100*2}}{\frac{4000}{60}} = 18.75\mu s \tag{3}$$

Quadrature Decoding

- 2-dimensional array
- Previous and actual Gray Code as index
- Error correction
  - `pprev`, `prev` and `new`
  - In case of error, look at `pprev`

```
static const int8_t QUAD_Table[4][4] =
  {                  /* prev new */
    {                /* c12 c12 */
     0,              /* 00  00  no change */
     1,              /* 00  01 */
     -1,             /* 00  10 */
     QUAD_ERROR,     /* 00  11  error */
     },
    ...
  }
```

Quadrature Decoding

- Build index from quadrature bits
- Interrupt context
- Error correction: additional overhead

```
void QUAD_Decode(void) {
  int8_t new_step;
  uint8_t c12; /* value of the two sensor input */
  it should be fast as possible! is that fast? this section isn't reentrant
  c12 = (C1_GetVal()!=0?2:0)|(C2_GetVal()!=0?1:0);
  new_step = QUAD_Table[QUAD_last_quadrature_value][c12];
  if (new_step==QUAD_ERROR) { /* error case */
    QUAD_errors++;
  }
  QUAD_last_quadrature_value = c12;
  QUAD_currPos += new_step;
}
```

reentrant problem if: in case if it's interrupted and multiple tasks using it
-> make shure that only one is using that and then you have no problem

Summary

- Absolute digital encoder: Problematic with binary code
- Binary Reflected Gray Code: Only one bit changes from code to code
- Convert Codes: EXOR and iterative bit processing
- Signal Acquisition: Hardware, Interrupt and Sampling
- Signal Processing: Table with index
- Realtime aspects: Error correction, interrupt latency