

# EECS192 Lecture 4


## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

- 2/16 Quiz 1 (10 minutes) line camera timing issues

### Topics

- 
- Checkpoints 2,3,4,5
  - Line Camera- line finding
  - HW1 Line finding (Python)
  - Position Encoder/ Velocity Sensing
  - ESP32 pulse count and interrupt
  - Velocity control (intro)
  - Battery safety

# CP2- PWM for driving steering servo and ESC

Write code which performs the following sequence of functions:

C2.1: Start wheels turning, and ramp up to full speed in 5 seconds and down to zero speed in another 5 seconds.

C2.2: Set steering angle approximately half full-left and hold for 5 seconds. (For example, if full steering range is  $\pm 20$  degrees, set steering angle to  $+10$  degrees.)

C2.3: Set steering angle straight and hold for 5 seconds.

C2.4: Set steering angle approximately half full-right, and hold for 5 seconds.

C2.5: Set steering angle back to approximately straight.

C2.6: Show steering changing and wheels turning at the same time

C2.7: Report Data RAM and Instruction RAM usage. How much of each is left?  
[pio run -v in terminal window]

C2.8: All members must fill out the checkpoint survey before the checkoff close.  
Completion is individually graded

# Huzzah32/ESP32 WROOM memory

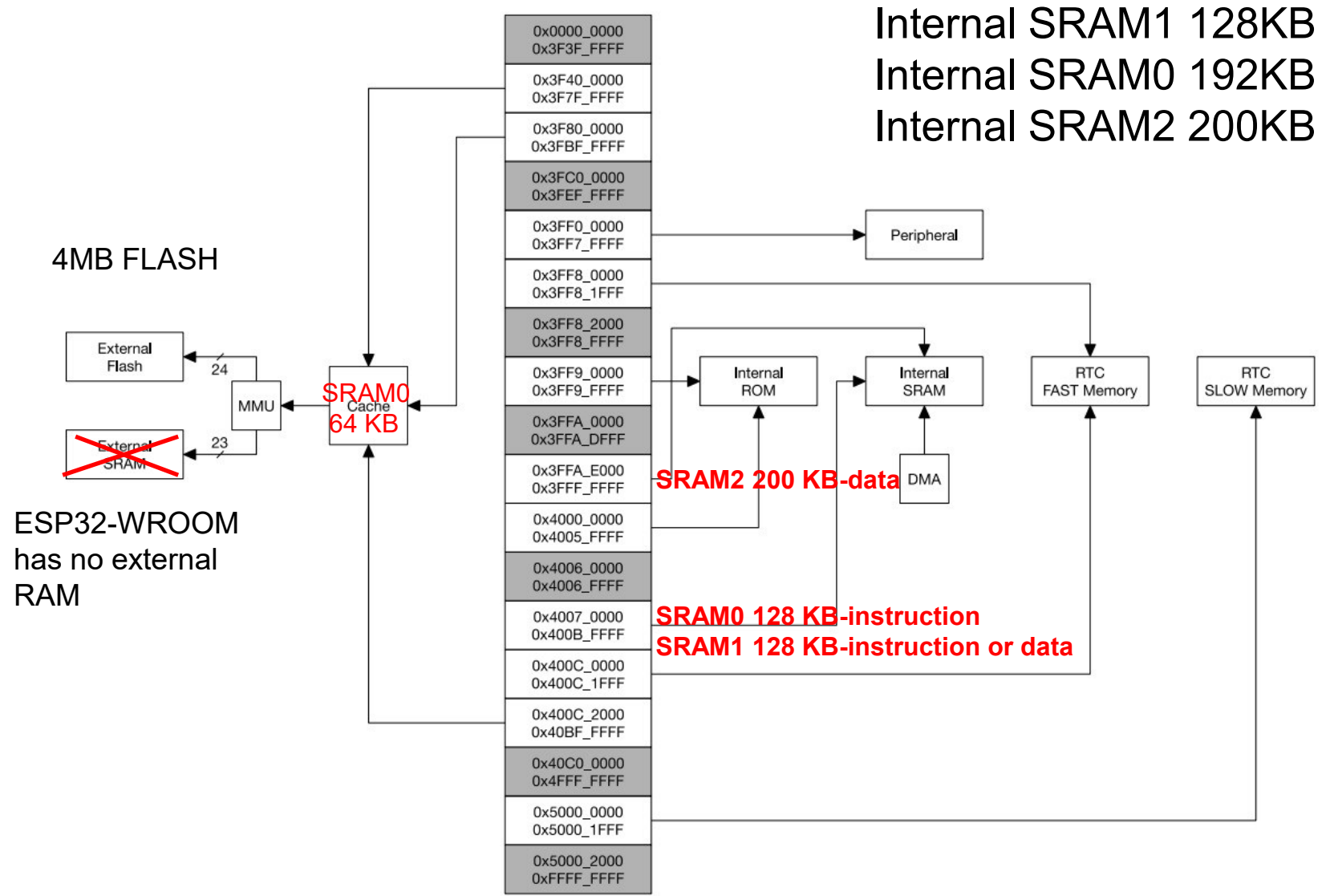


Figure 2: System Address Mapping

Some timing critical code may be placed into IRAM to reduce the penalty associated with loading the code from flash. ESP32 reads code and data from flash via a 32 kB cache. In some cases, placing a function into IRAM may reduce delays caused by a cache miss.

# CP3- remote control with UDP

(Intended to be easy since project proposal due Tues 2/9, and CP4 harder)

The car should be upside down, or lifted off the ground so it does not move

CAUTION: setting PWM to 1.0 ms or 2.0 ms can drive servo to the end of its range and cause it to burn out.

C3.1 From the remote client keyboard, send a speed command for the drive motor. Pick a range such as 0-100, and show that you can specify a range of values. The motor should remain turning until the next command.

C3.2: Find the minimum ESC PWM value which causes the wheels to turn.

C3.3: With motors turning, send an “Emergency Stop” command to the car (the motor should be turned off or braked, if available).

C3.4: From the remote client, send a command to set the steering angle.

C3.5: Find the range of steering angles for your car. (These values should be used for range checking in your code for the rest of the semester.)

C3.6: Show that motor speed and steering can be set independently from the remote client keyboard.

C3.7: Report Data RAM and Instruction RAM usage. How much of each is left?

```
From VS terminal > pio run -v
```

C3.8: All members must fill out the checkpoint survey before the checkoff close. Completion is individually graded.

# CP4- Open Loop Figure 8 and Line Camera

C4.1 Show car driving an open-loop topological figure 8, at any speed with at least one full CW and at least one full CCW circle. Car should be going as slow as practical.

C4.2 Show ability to stop car in middle of figure 8 using “Emergency stop command”

C4.3 Show that you are able to read the line camera data and discriminate the line. Possible ways to do this include printing data to the serial console or using the UDP logging framework (preferred). You must be able to explain the output format quickly during the checkpoint.

C4.4 Use a dark surface and a white stripe approximately 2.5 cm wide. When the camera is moved to the left or right of the track, show that the steering servo/car front wheels will respond appropriately. You are not expected to have a nice sensing algorithm (findmax is sufficient) or well tuned steering control loop for this checkpoint.

C4.5 All members must fill out the checkpoint survey before the checkoff close. Completion is individually graded.

# CP5- velocity sensing

The car should be upside down, or lifted off the ground so it does not move

C5.1 Demonstrate that you have velocity sensing working and the output in terms of some physical units (m/s, mm/s, etc). Turning the wheels by hand should show a low velocity.

C5.2 Set a low constant motor PWM. Show that the estimated velocity is relatively constant.

C5.3 Show that with the constant PWM from C5.2 that the velocity sensor estimated velocity drops if the wheels are loaded or stopped.

C5.4 Show velocity control. The recommended target setpoint is 3 m/s, which should provide enough encoder counts for a somewhat stable control loop. It's fine if the applied PWM is noticeably jittering or if the actual velocity is inaccurate. However, if you load the wheels (with, say, a book), the controller should compensate by applying a higher drive strength. (Print PWM and sensed velocity as load is applied to wheel.)

C5.5 Show velocity control working with the basic line sensing from C4.4. (Printing PWM, sensed velocity, and line center is sufficient, as load is applied to wheel and car is positioned by hand)

C5.6 All members must fill out the checkpoint survey before the checkoff close. Completion is individually graded.

# EECS192 Lecture 4

## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

- 2/16 Quiz 1 (10 minutes) line camera timing issues

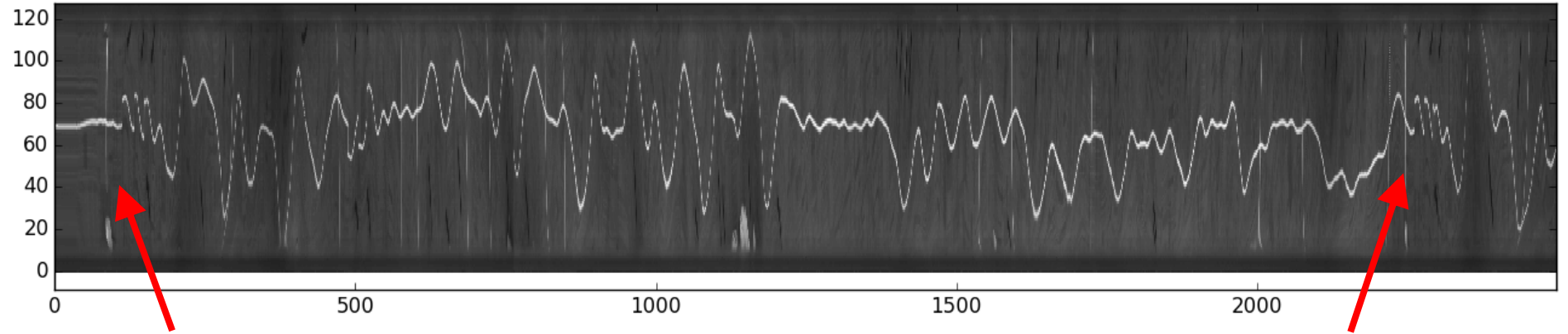
### Topics

- Checkpoints 2,3,4,5
- Line Camera- line finding
- HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- ESP32 pulse count and interrupt
- Velocity control (intro)
- Battery safety



# Example Line camera data

natcar2016\_team1.csv

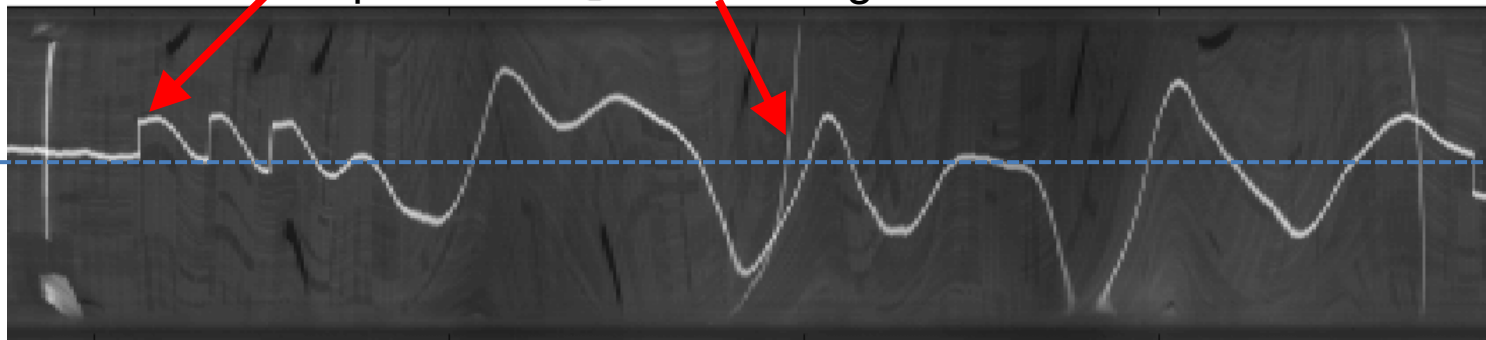


Start line

Steps

crossing

Old style finish line



100

200

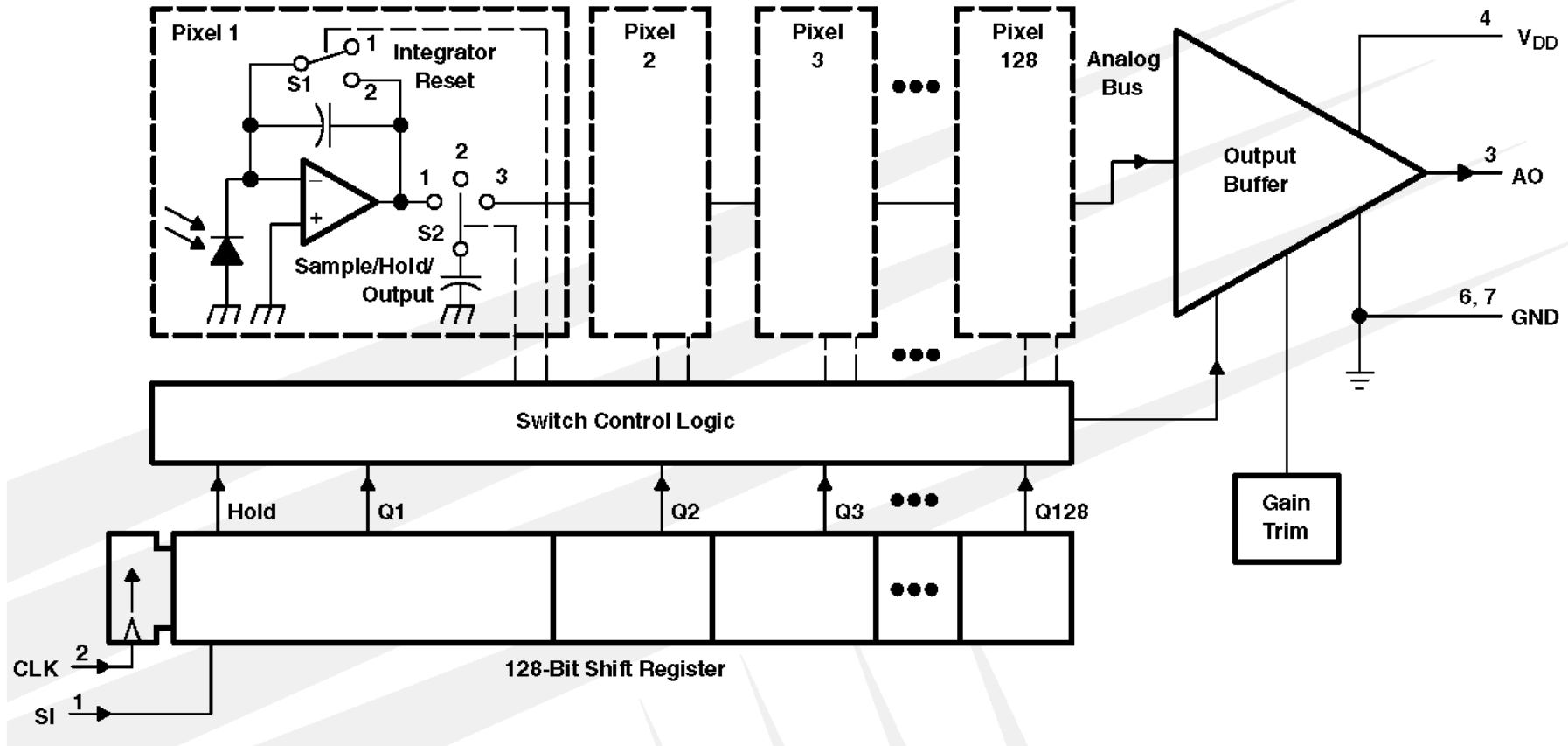
300

400



# TSL 1401 line sensor

## Functional Block Diagram



SI: serial input. SI defines the start of the data-out sequence.  
AO: analog output  
CLK: clock

# TSL 1401 line sensor

## PARAMETER MEASUREMENT INFORMATION

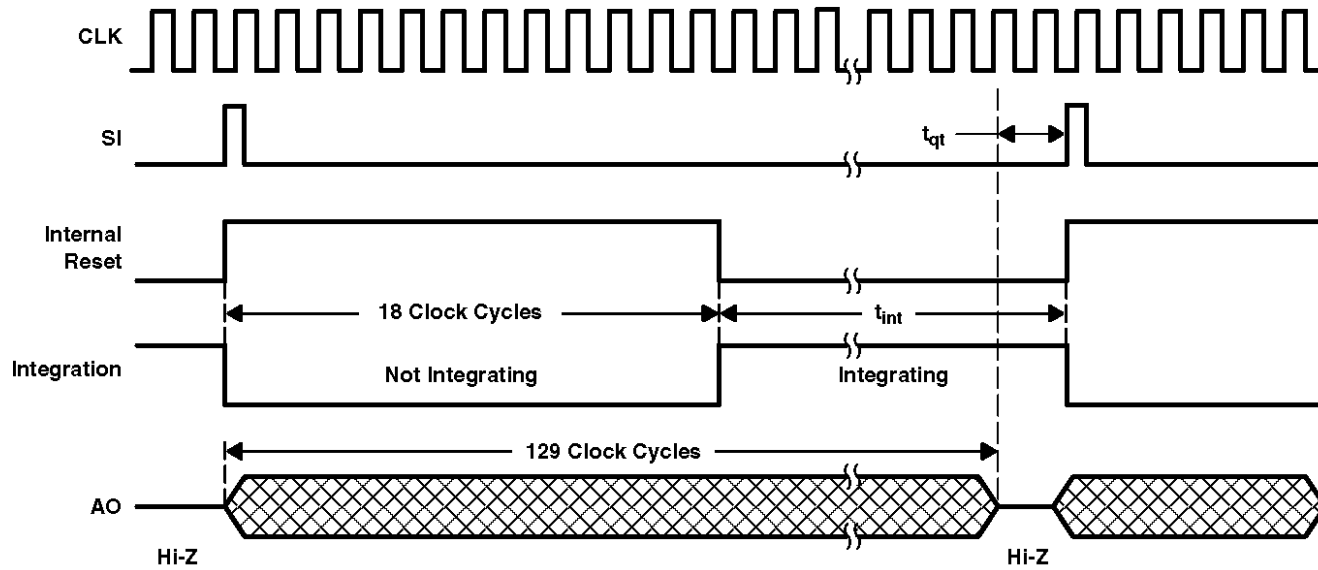


Figure 1. Timing Waveforms

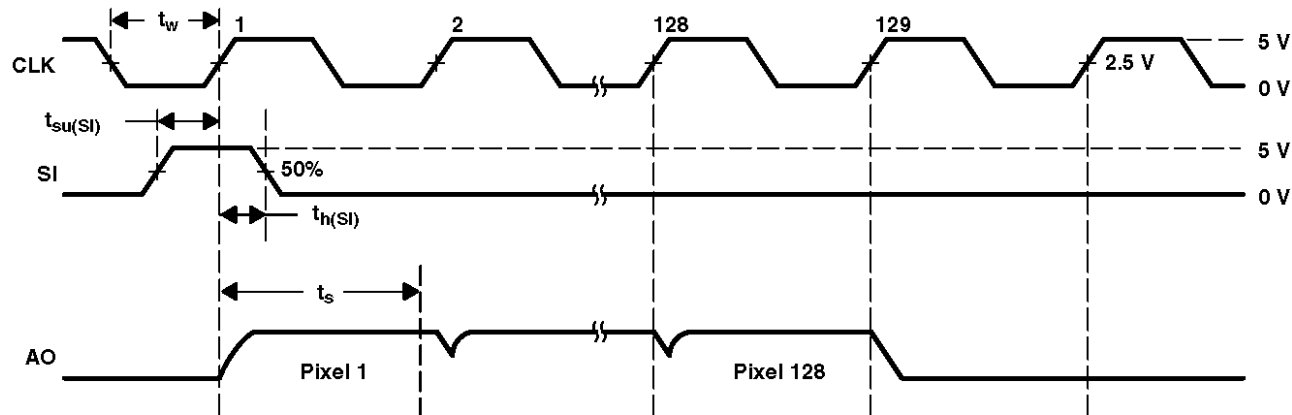
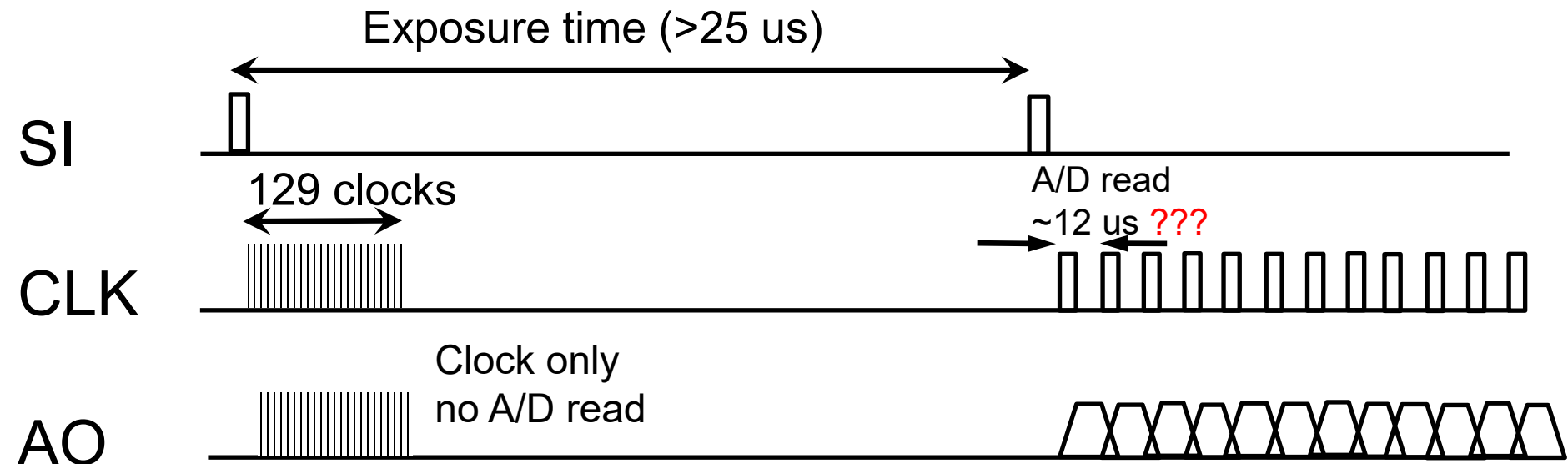


Figure 2. Operational Waveforms

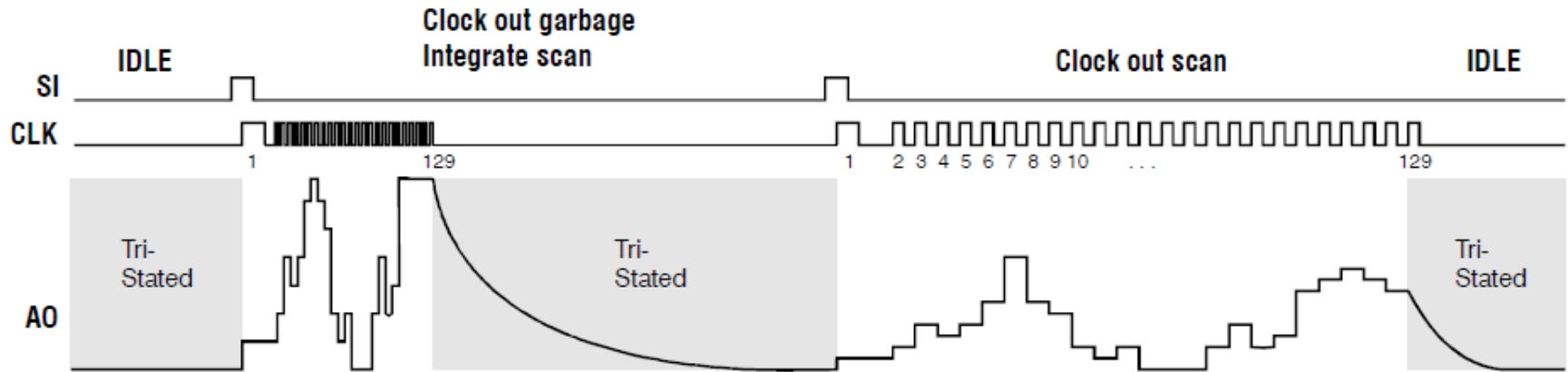
# TSL 1401 line sensor- option exposure control



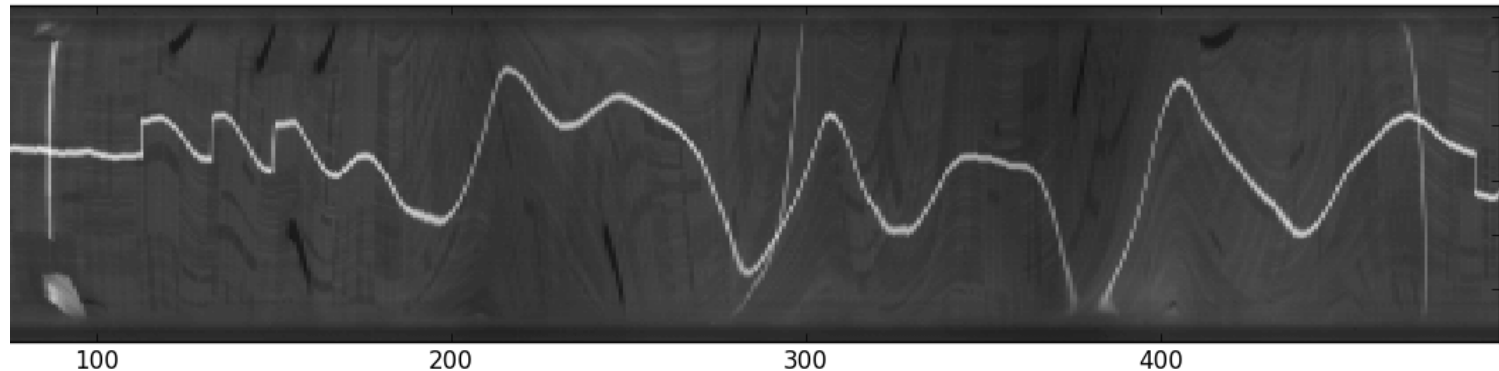
## Timing options:

- PWM channel for clock
- Manual clock generation (`gpio_set_level()`, `read_atod`, etc)
- `vTaskDelay(pdMS_TO_TICKS(3000));` [1 ms resolution]
- Busy wait (may trigger watch dog for long delays)  
`while(task_counter_value < end_value)`  
`{ timer_get_counter_value(TIMER_GROUP_0, TIMER_0, &task_counter_value); }`
- Interrupt alarm [usec resolution] (see `timer-group-example.c` in `SkeletonHuzzah32`)  
[docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/timer.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/timer.html)

# Automatic Gain Control



In all the discussion that follows, we will be using one-shot imaging.



- Choose exposure time based on average illumination
- Keep frame rate constant e.g. read sensor twice 1+4 → 4 +1 ms
- (Constant time is important for control- will see later)

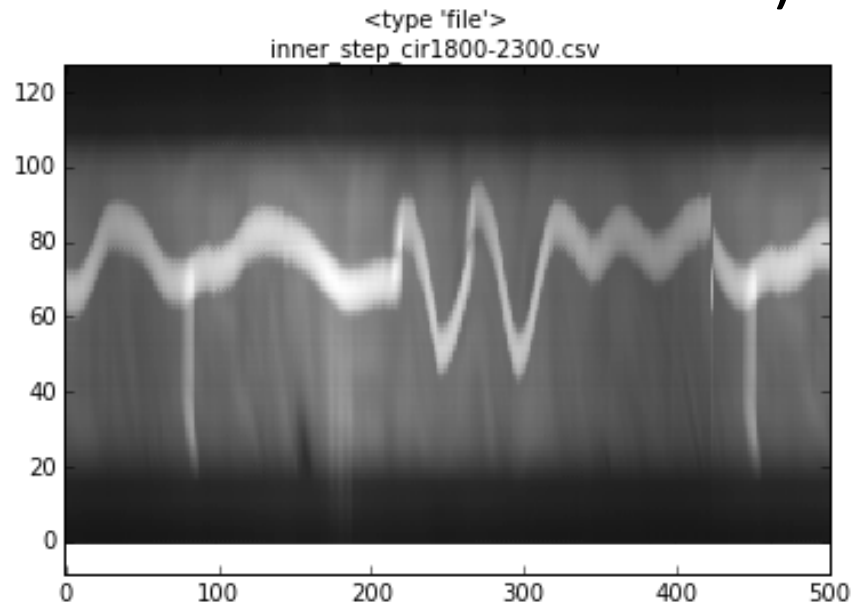
# Possible algorithms for line detection

e.g. prototype with `scipy.signal.filter`. Many options.

Here are 3 suggestions:

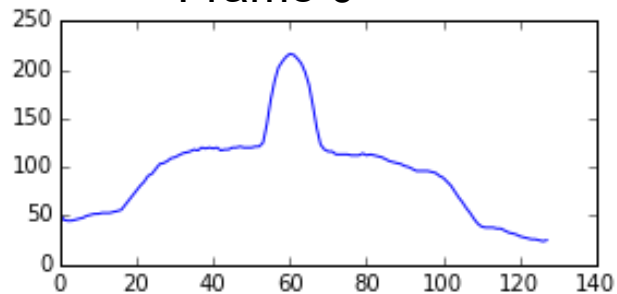
- Subtraction- to find left and right edge of line (ok if not noisy, somewhat lighting invariant)
- Difference of Gaussians (idea is to smooth then differentiate)
- Correlation (best match position for known features)
  - `scipy.signal.correlate`

Try out algorithms in HW1

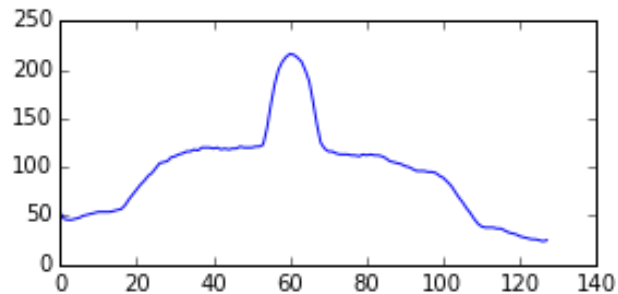


# TSL 1401 line sensor NATCAR 8 bit

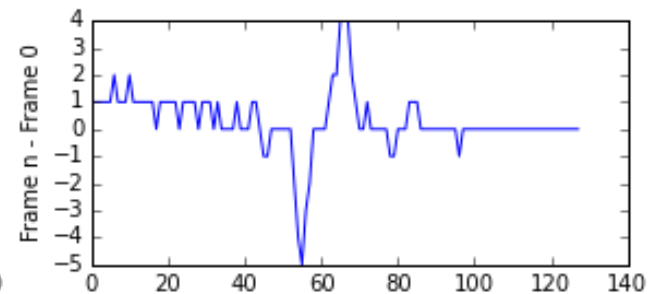
Frame 0



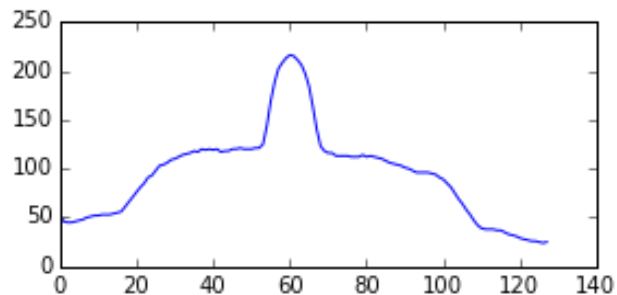
Frame 1



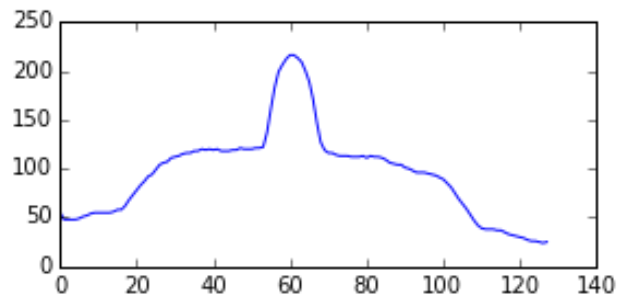
Frame 1-Frame 0



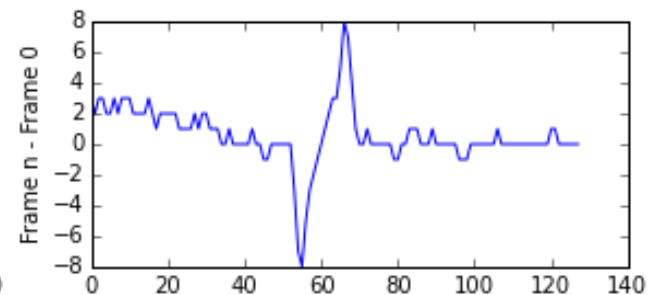
Frame 0



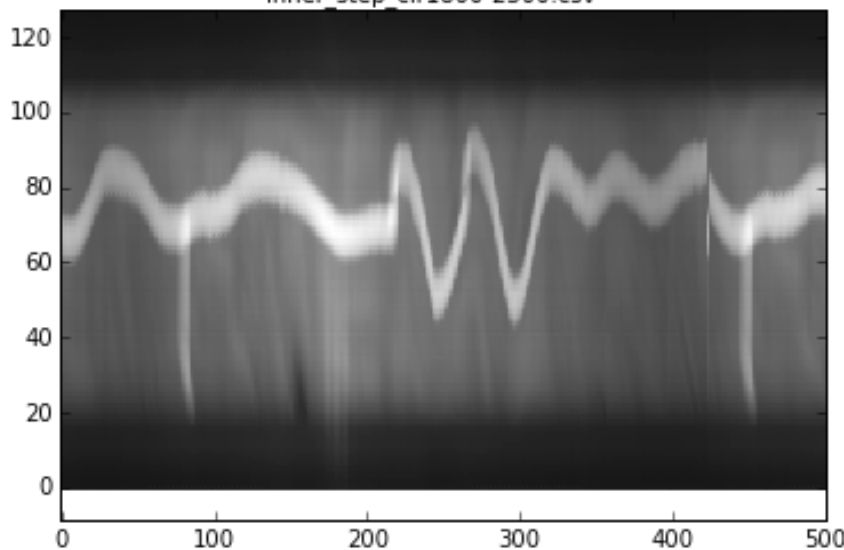
Frame 2



Frame 2-Frame 0



<type 'file'>  
inner step cir1800-2300.csv



# Alternative #2 Difference of Gaussians

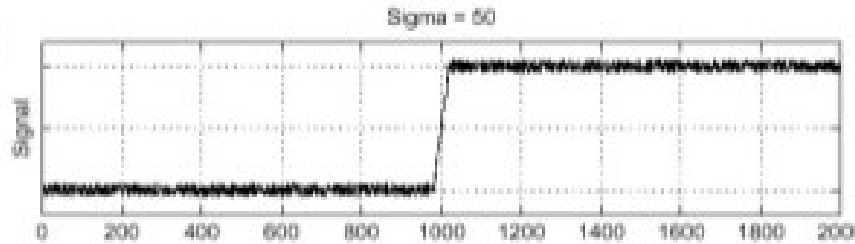
Laplacian of Gaussian smoothing

$$\Delta[G_\sigma(x, y) * f(x, y)] = [\Delta G_\sigma(x, y)] * f(x, y) = LoG * f(x, y)$$

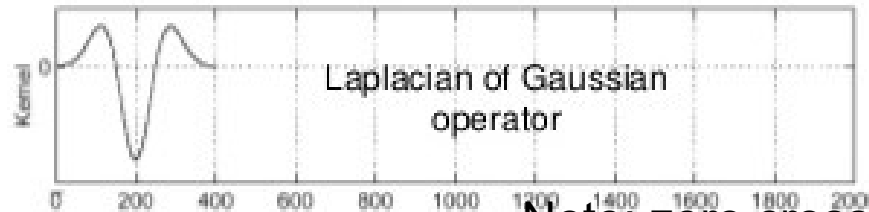
Convolve with Difference of Gaussians kernel (approx. to LoG)

$$\Gamma_{\sigma_1, \sigma_2}(x) = I * \frac{1}{\sigma_1 \sqrt{2\pi}} e^{-(x^2)/(2\sigma_1^2)} - I * \frac{1}{\sigma_2 \sqrt{2\pi}} e^{-(x^2)/(2\sigma_2^2)}.$$

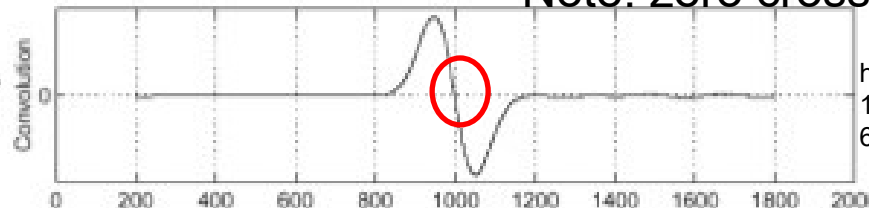
Consider  $\frac{\partial^2}{\partial x^2}(h * f)$   
 $f$



$\frac{\partial^2}{\partial x^2}h$



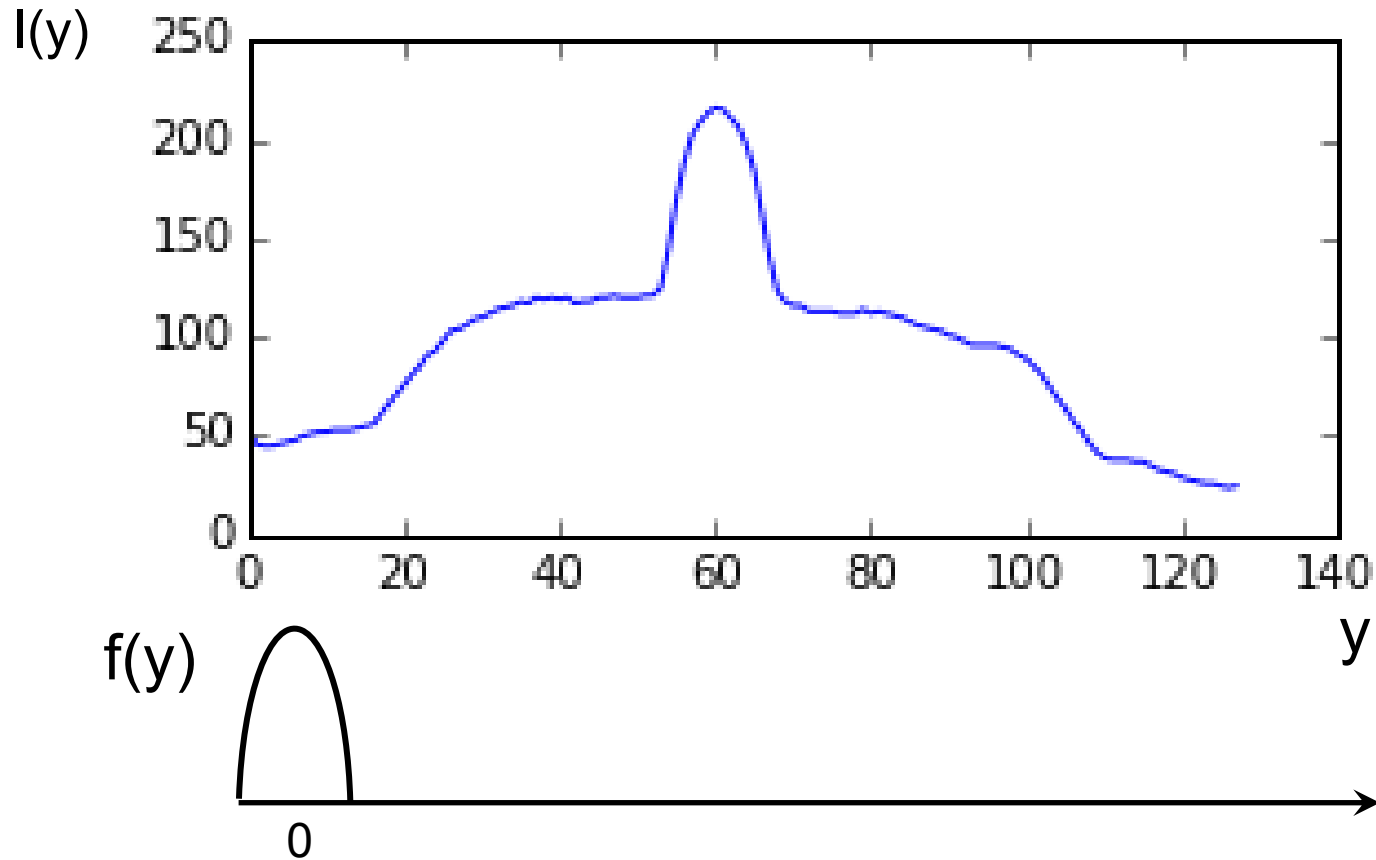
$(\frac{\partial^2}{\partial x^2}h) * f$



Note: zero crossing is edge location

<https://image.slidesharecdn.com/cbirfeatures-150705141111-lva1-app6892/95/cbir-features-47-638.jpg?cb=1436105787>

# Alternative #3 Correlation



$$\arg \min_{\Delta y} \| I(y) - f(y - \Delta y) \|_2$$

Notes: normalize, find by least squares or search. Can use  $\Delta y(n-1)$  to initialize



# EECS192 Lecture 4


## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

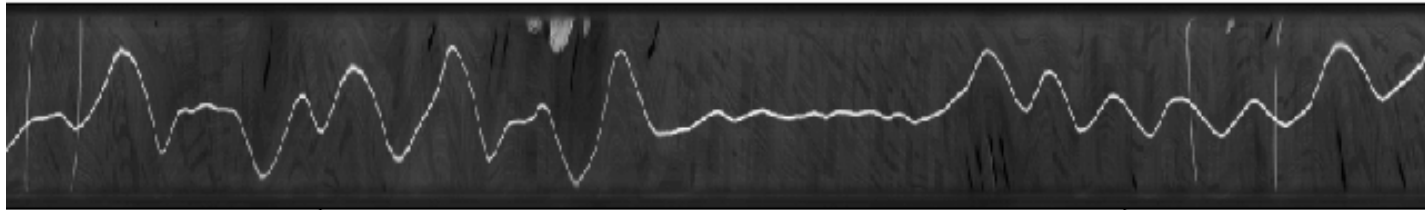
- 2/16 Quiz 1 (10 minutes) line camera timing issues

### Topics

- Checkpoints 2,3,4,5
- Line Camera- line finding
-  HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- ESP32 pulse count and interrupt
- Velocity control (intro)
- Battery safety

# HW1 Line Sensing

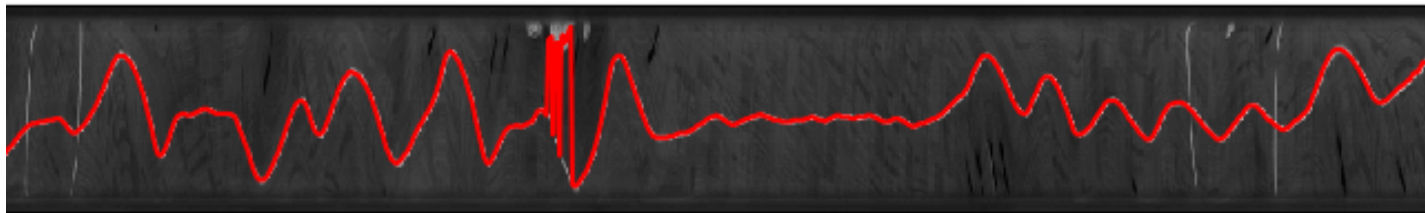
Track Section



1000

1500

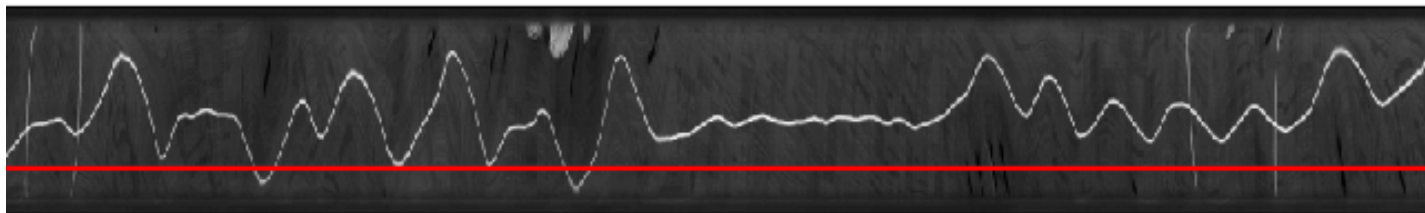
Track Center



1000

1500

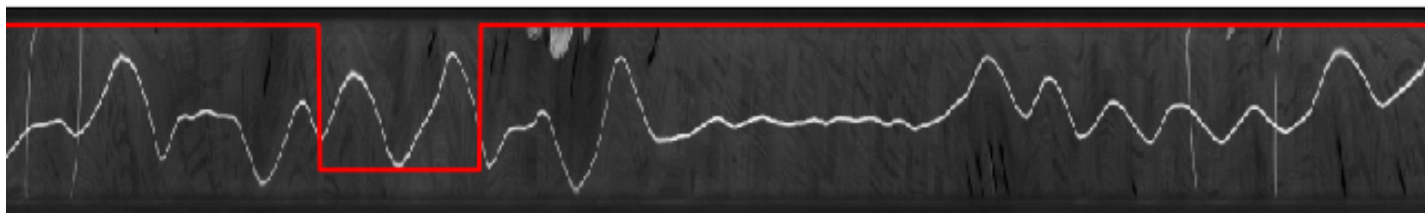
Track Found



1000

1500

Cross Found



1000

1500

Time

# EECS192 Lecture 4

## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

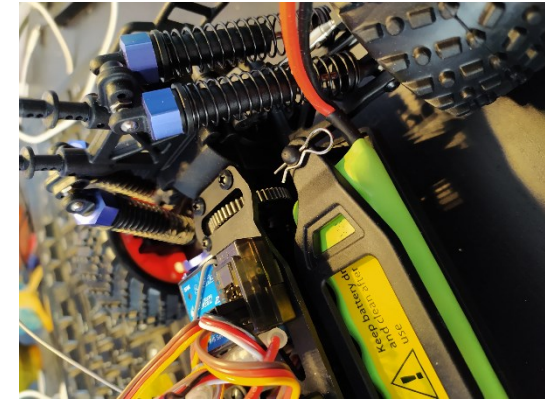
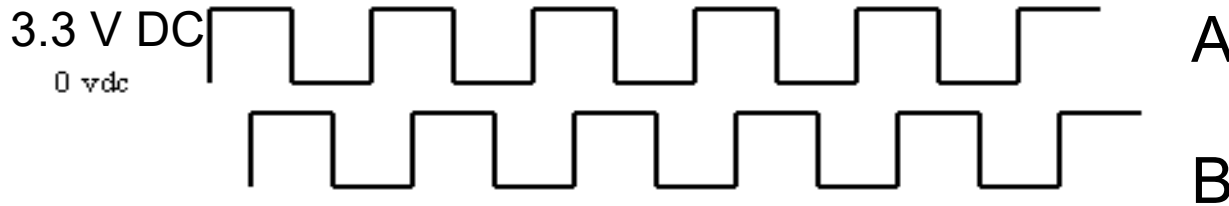
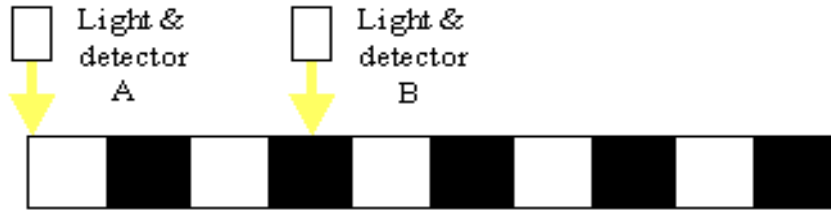
- 2/16 Quiz 1 (10 minutes) line camera timing issues
- Survey for Cory Courtyard

### Topics

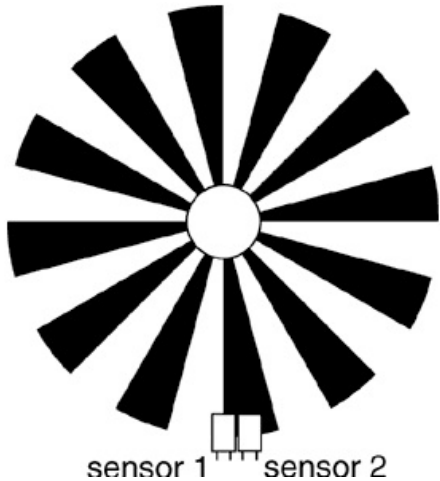
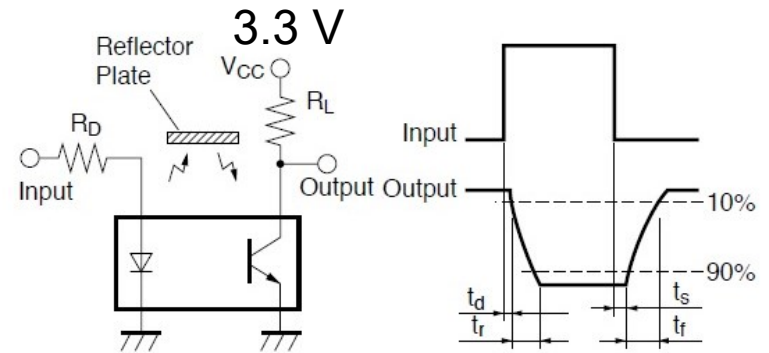
- Checkpoints 2,3,4,5
- Line Camera- line finding
- HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- ESP32 pulse count and interrupt
- Velocity control (intro)
- Battery safety



# Velocity sensor mounting (preview- week 4)



<https://www.sinotech.com/wp-content/uploads/quadrature-encoder.gif>



100 us  
response time

(? Analog) or  
pulse count input or  
GPIO interrupt?

# Sharp GPS260

Fig.9 Test Circuit for Response Time

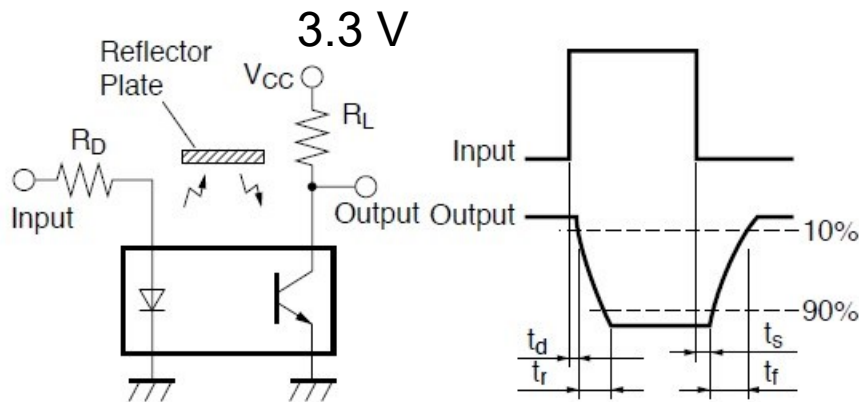
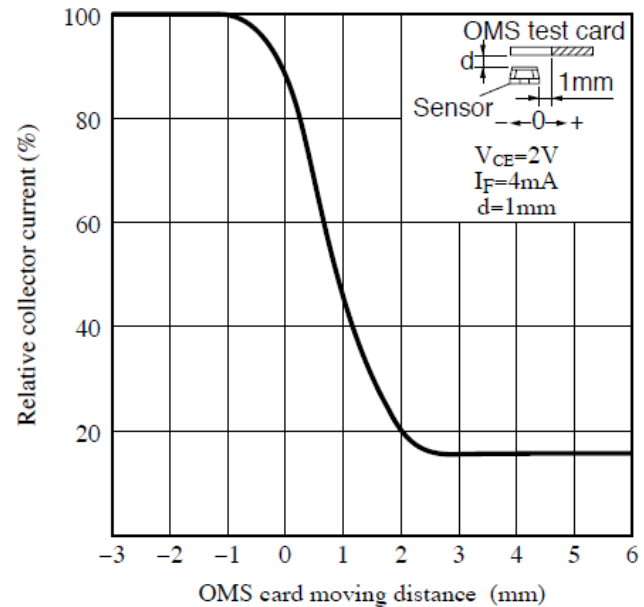


Fig.13 Detecting Position Characteristics (2)

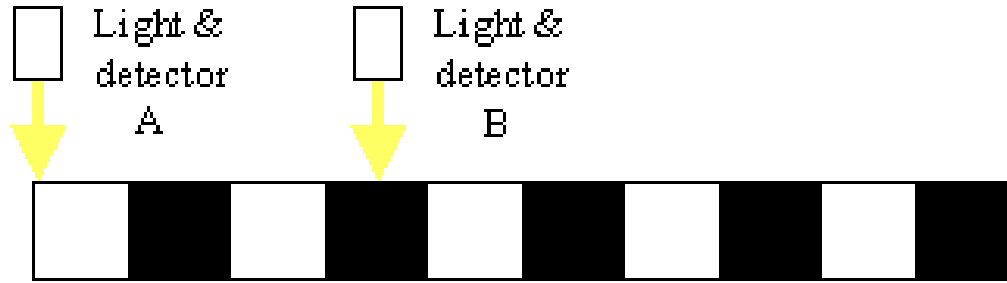


- Choose current 4 mA in LED
- $V_{cc} = 3.3 \text{ V}$
- May want regulated/clean voltage for  $V_{cc}$

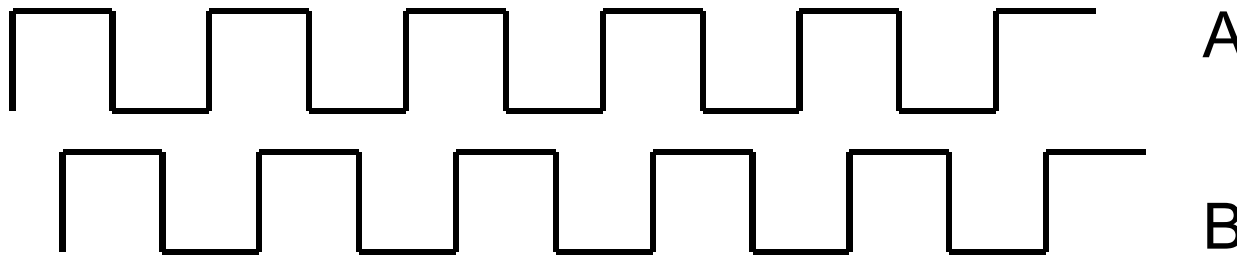


100 us  
response  
time

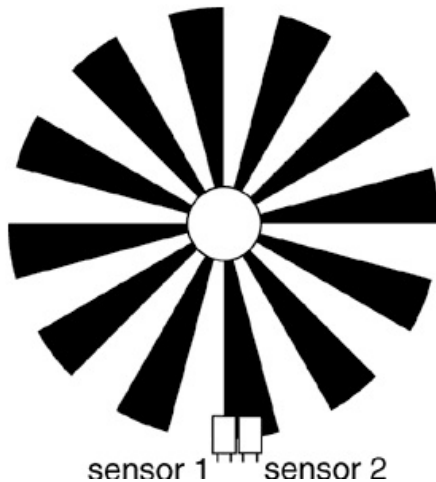
# Quadrature Encoder



3.3 V DC  
0 vdc

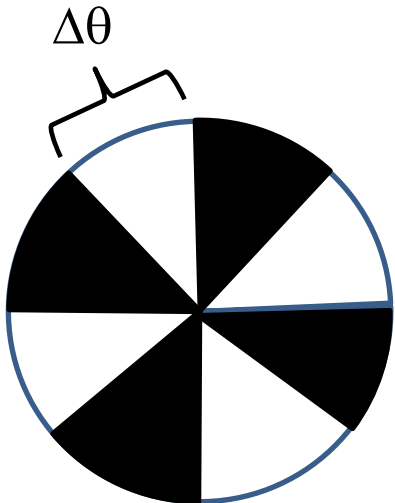
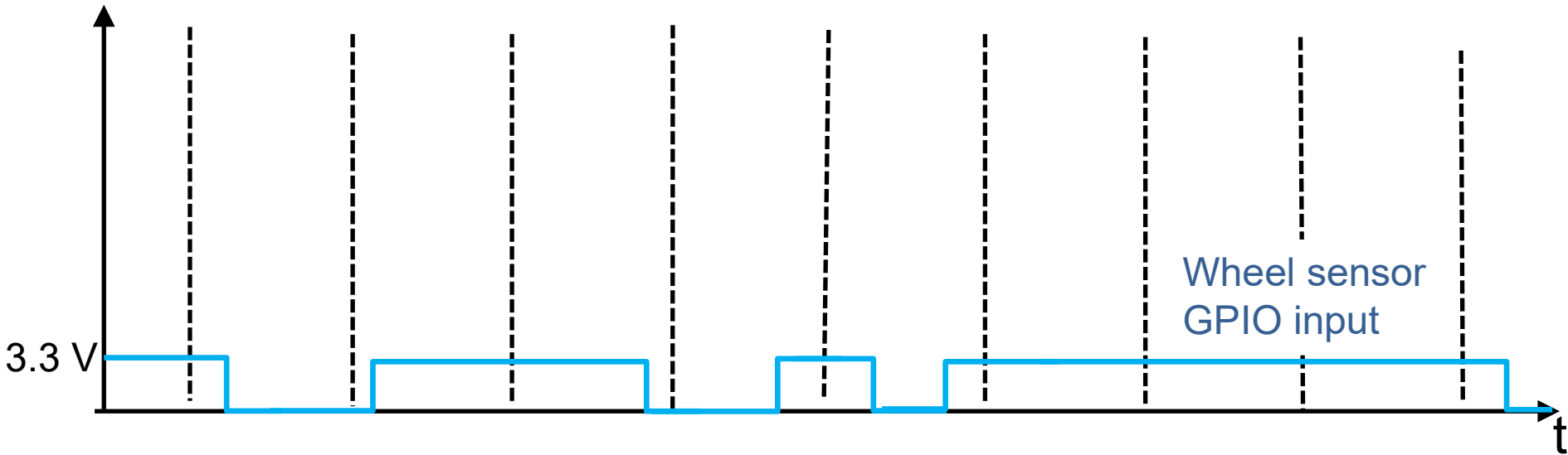


<https://www.sinotech.com/wp-content/uploads/quadrature-encoder.gif>



Fab suggestion: aluminum foil covered with black paper with slots. 4 slots probably enough. Note: sensors can be placed where convenient—don't need to look at same slot.

# Velocity sensing: encoder signal



$$\text{Car velocity} = (r_{\text{wheel}} \Delta\theta) / \Delta T$$

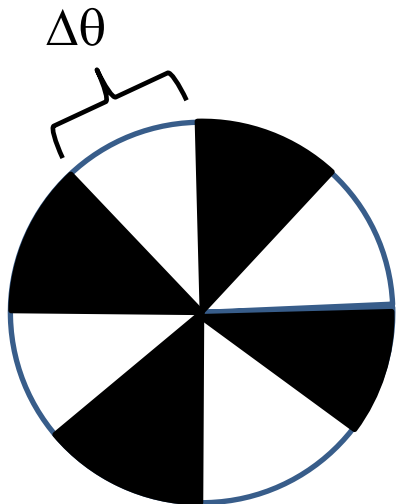
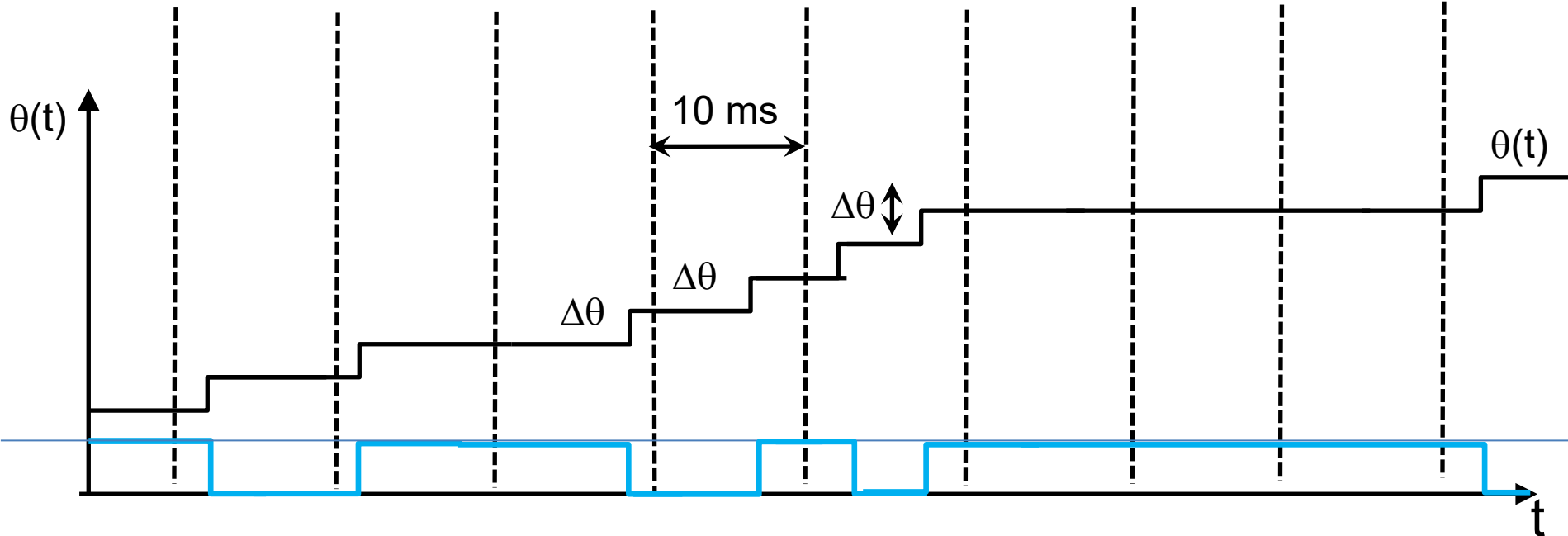
$$\text{If } r_{\text{wheel}} = 4 \text{ cm, } \Delta\theta = \pi/4$$

$$\Delta x = 3 \text{ cm} = r_{\text{wheel}} \Delta\theta$$

# Velocity sensing: method #1 uniform sampling

Number of edges in 10 ms:

1      1      1      1      2      0      0      0



$$V_{\text{uniform}} = \frac{N r_{\text{wheel}} \Delta\theta}{10 \text{ ms}}$$

Uniform  
sampling

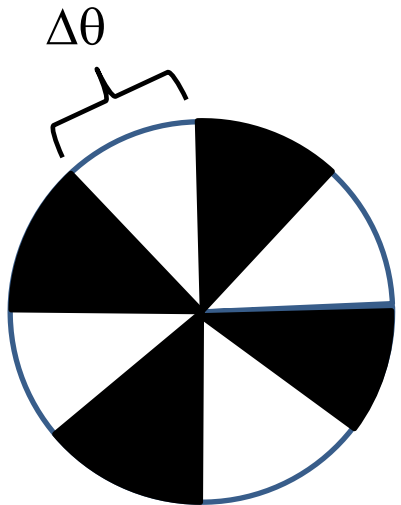
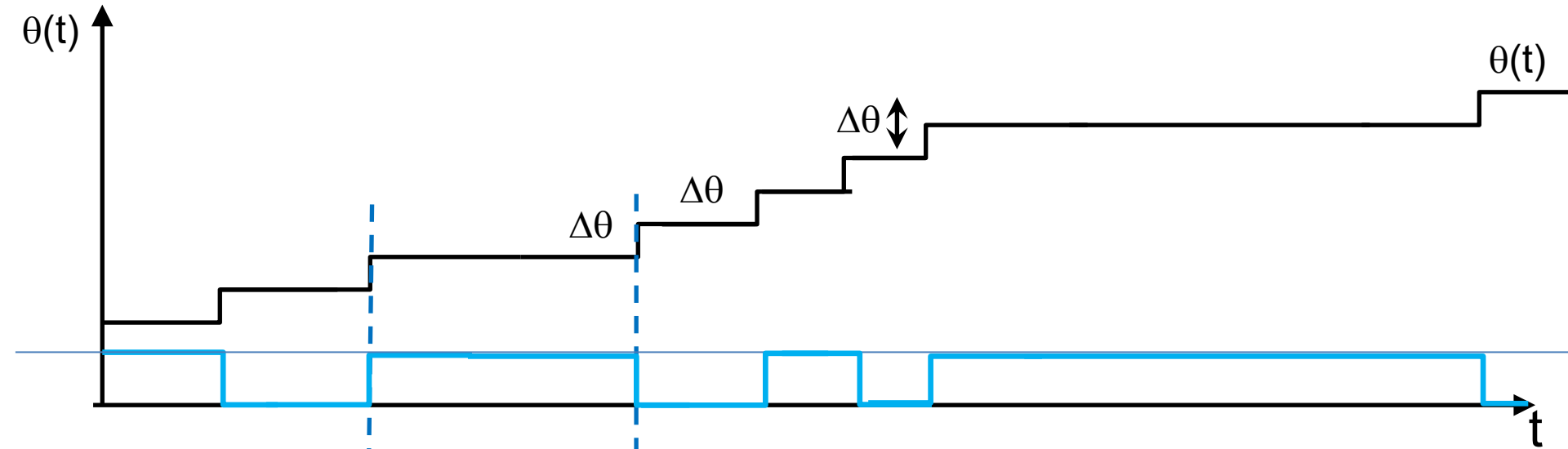
What is the problem with these parameters if

$$\Delta x = 3 \text{ cm} = r_{\text{wheel}} \Delta\theta$$

?



# Velocity sensing: method #2 edge timing



$$V_{\text{edge}} = \frac{R_{\text{wheel}} \Delta\theta}{\Delta T_1}$$

Edge  
timing

# Velocity sensing - options

$V \sim (\text{change in angle}) / (\text{change in time})$

Method #1: count number of edges in an interval (Pulse Counter Module)  
TRM: 18 Pulse Counting

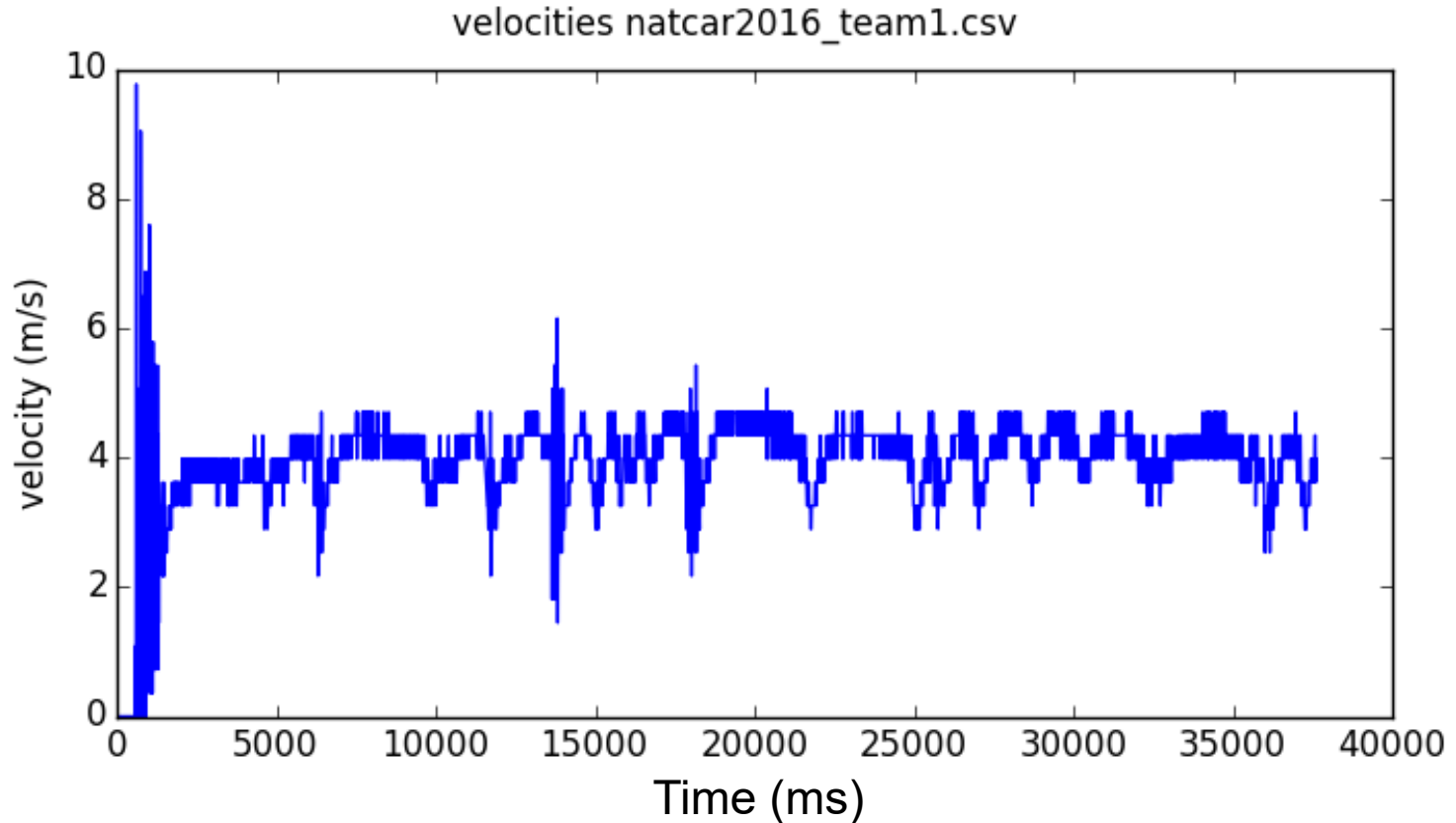
Method #2: count time between edges:  
interrupt on rising or falling edge, using GPIO interrupt  
measure time with

```
timer_get_counter_value(TIMER_GROUP_0,  
                        TIMER_0,  
                        &task_counter_value);
```

[docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html)

# Velocity Sensing

estimating  $\Delta x / \Delta T$



Note: care about velocity sensing usually at cruise speed (also stopping)

# EECS192 Lecture 4


## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

- 2/16 Quiz 1 (10 minutes) line camera timing issues

### Topics

- Checkpoints 2,3,4,5
- Line Camera- line finding
- HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- • ESP32 pulse count and interrupt
- Velocity control (intro)
- Battery safety

# Pulse Counting (Ch 18) (method #1)

[github.com/espressif/esp-idf/tree/release/v4.2/examples/peripherals/pcnt](https://github.com/espressif/esp-idf/tree/release/v4.2/examples/peripherals/pcnt)

Missing function

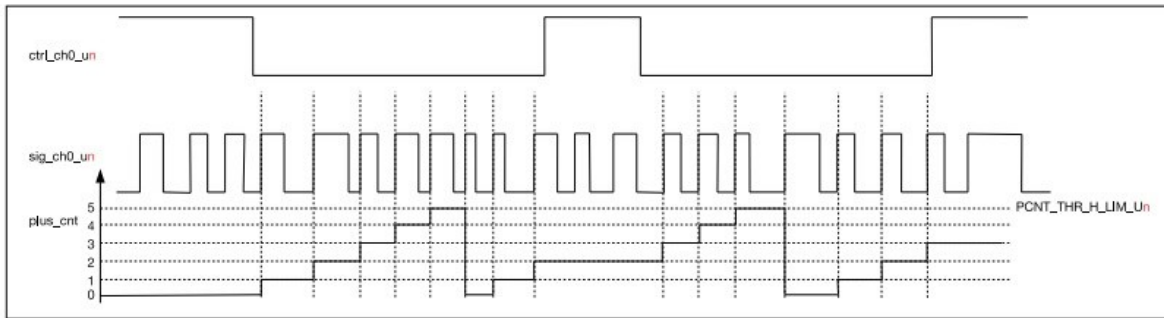


Figure 122: PULSE\_CNT Upcounting Diagram

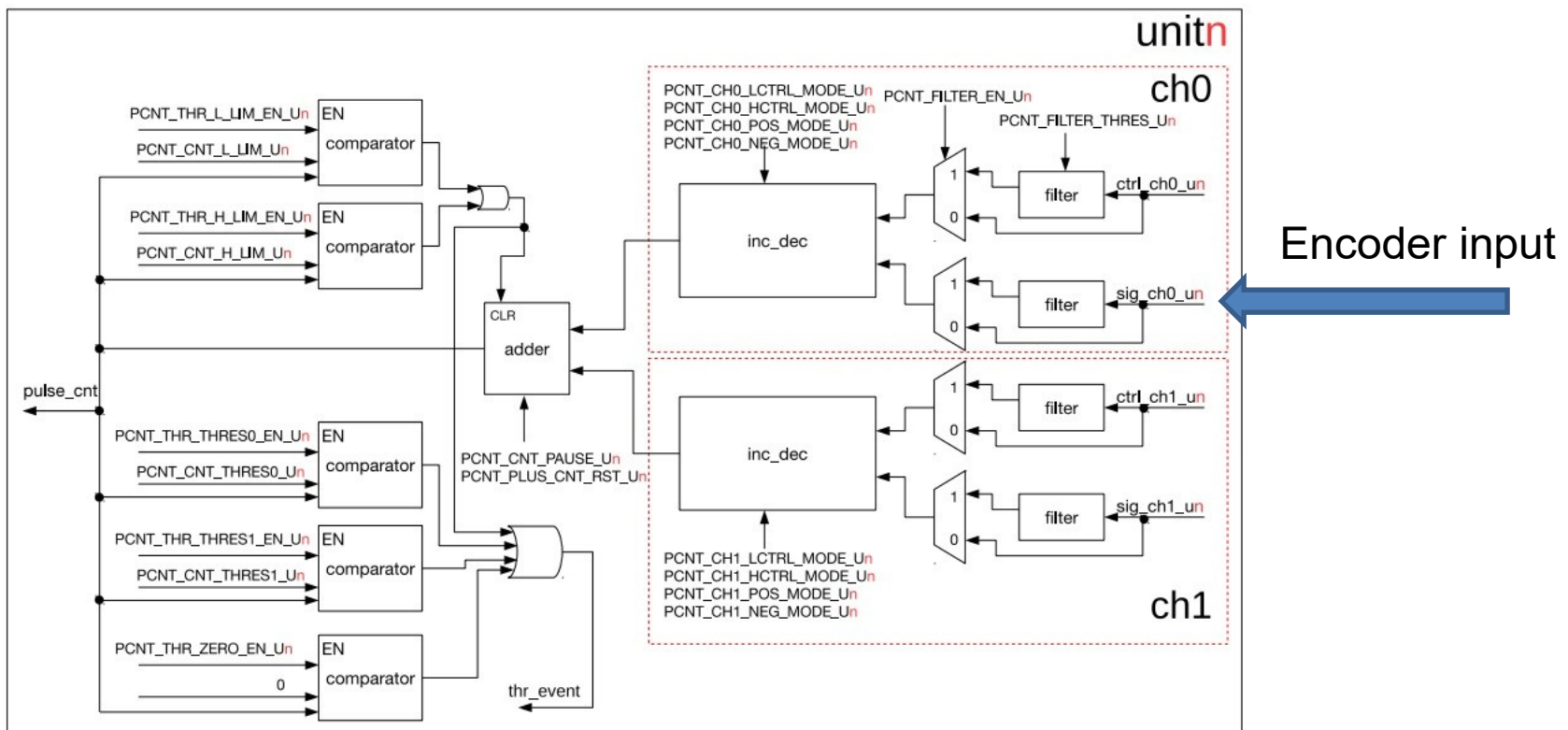


Figure 121: PULSE\_CNT Architecture

# Pulse Counter

## Reading # of pulses:

```
pcnt_get_counter_value(pcnt_unit_t pcnt_unit, int16_t *count)
```

Get pulse counter value. NOTE: signed int, so range is from 0 to + 32767  
(be careful when subtracting: count – old\_count)

## Initializing:

```
esp_err_t pcnt_counter_clear(pcnt_unit_t pcnt_unit)
```

Clear and reset PCNT counter value to zero.

```
esp_err_t pcnt_unit_config(const pcnt_config_t *pcnt_config)
```

Configure Pulse Counter unit. pcnt\_config: Pointer of Pulse Counter unit configure parameter

```
esp_err_t pcnt_set_pin(pcnt_unit_t unit, pcnt_channel_t channel,  
int pulse_io, int ctrl_io)
```

Configure PCNT pulse signal input pin and control input pin.

# pcnt config

```
pcnt_config_t pcnt_config = {
    // Set PCNT input signal and control GPIOs
    .pulse_gpio_num = PCNT_INPUT_SIG_IO,
    .ctrl_gpio_num = PCNT_PIN_NOT_USED, // use just signal input, one GPIO
    .channel = PCNT_CHANNEL_0,
    .unit = unit,

    // What to do on the positive / negative edge of pulse input?
    .pos_mode = PCNT_COUNT_INC, // Count up on the positive edge
    .neg_mode = PCNT_COUNT_INC, // Count up on negative edge

    // What to do when control input is low or high?
    .lctrl_mode = PCNT_MODE_KEEP, // Keep the primary counter mode if low
    .hctrl_mode = PCNT_MODE_KEEP, // Keep the primary counter mode if high

    // Set the maximum and minimum limit values to watch
    .counter_h_lim = PCNT_H_LIM_VAL,
    .counter_l_lim = PCNT_L_LIM_VAL,
};
/* Initialize PCNT unit */
pcnt_unit_config(&pcnt_config);
```

# EECS192 Lecture 4


## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

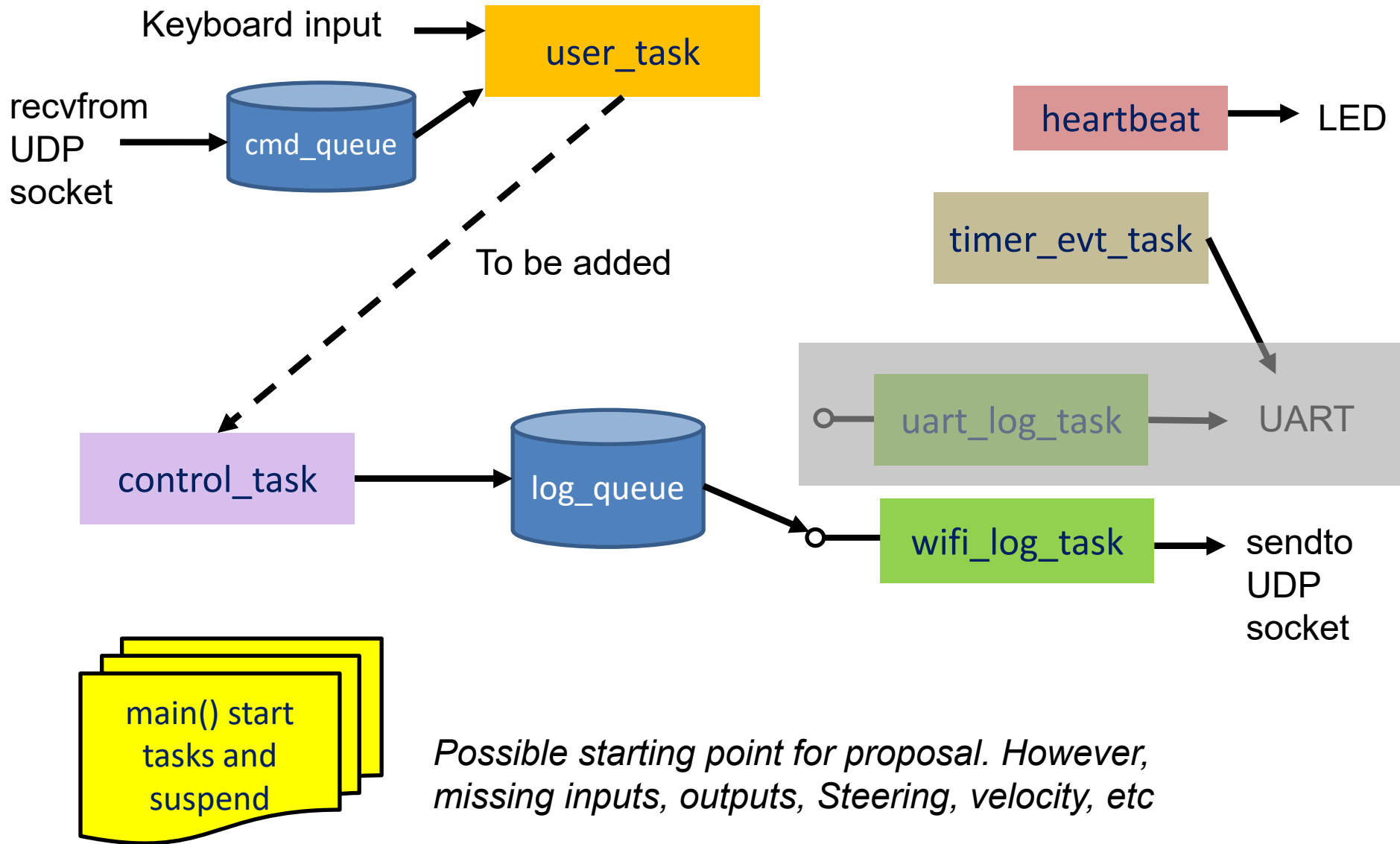
- 2/16 Quiz 1 (10 minutes) line camera timing issues

### Topics

- Checkpoints 2,3,4,5
- Line Camera- line finding
- HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- • ESP32 interrupt for time measurement
- Velocity control (intro)
- Battery safety



# SkeletonHuzzah32 Branch UDPCommd SW Block Diagram



Note conventions- data flow left to right

# Skeleton Tasks

control\_task  
(pri 5, delay 1000)

Heartbeat  
(pri 1, delay 1000)

Wifi\_log\_task  
(pri 1, delay 10)

User\_task  
(pri 0, 100)

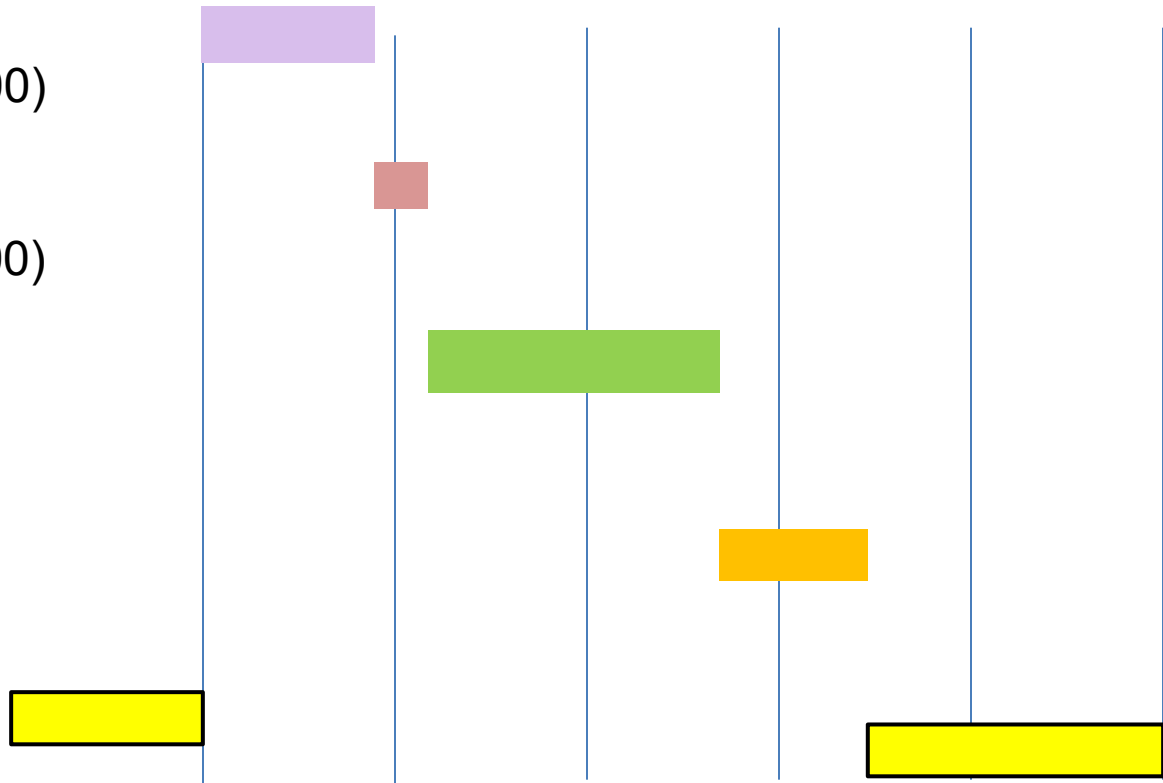
Idle  
(pri 0)

1000

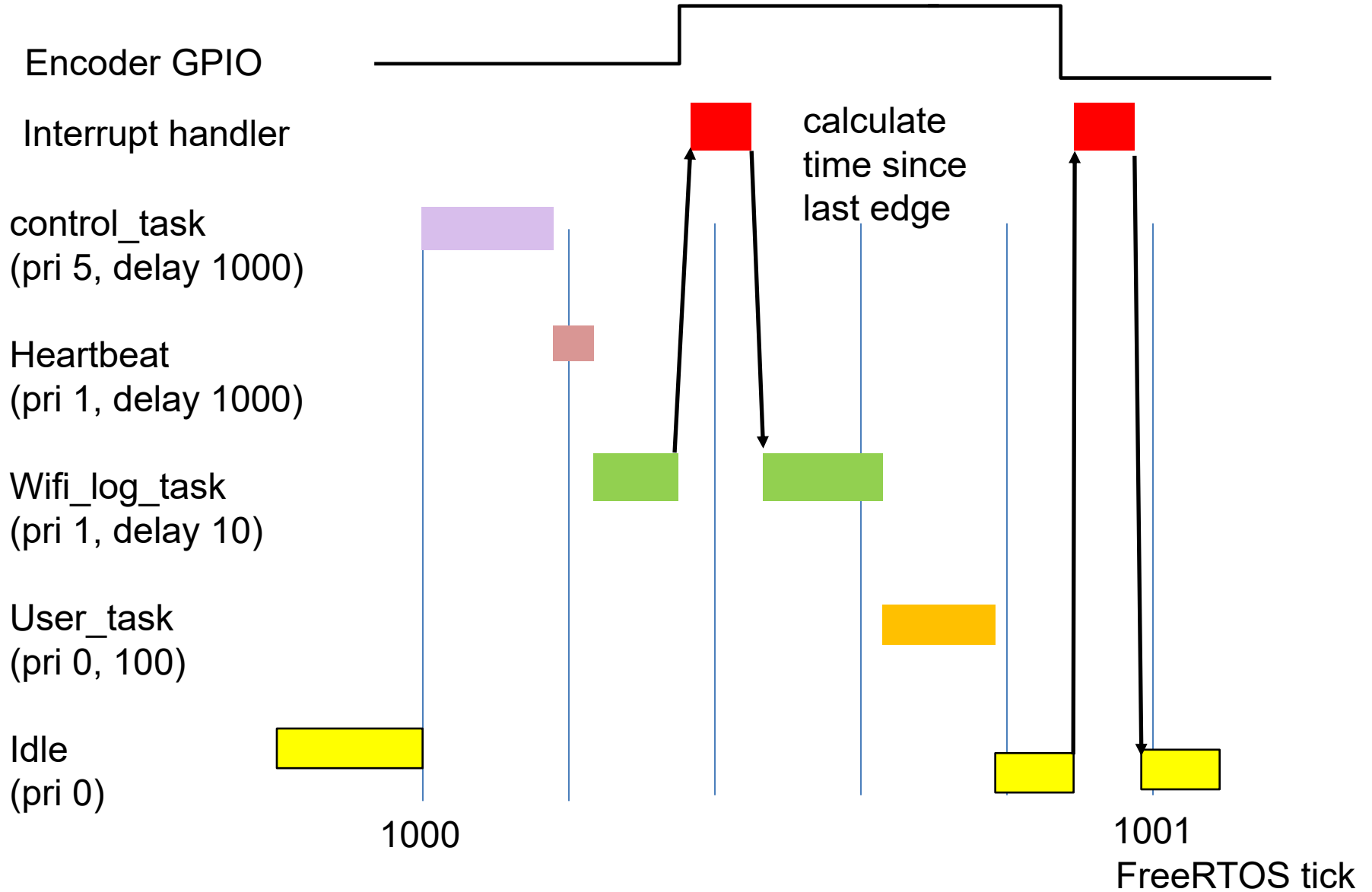
1001

FreeRTOS tick

`vTaskDelay(delay / portTICK_PERIOD_MS);`



# Skeleton Tasks with interrupt



`vTaskDelay(delay / portTICK_PERIOD_MS);`

# GPIO Interrupt example

## Interrupt handler function:

```
static void IRAM_ATTR gpio_isr_handler(void* arg)
{ read timer, calculate elapsed time and queue stuff here }
```

Note timer resolution by default 0.2 us

## Setting up interrupt:

```
gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT); //install gpio isr service
```

```
//hook isr handler for specific gpio pin:
```

```
gpio_isr_handler_add(GPIO_INPUT_IO_0,
                    gpio_isr_handler,
                    (void*) GPIO_INPUT_IO_0);
```

Example code:

<https://github.com/espressif/esp-idf/tree/release/v4.2/examples/peripherals/gpio>

# GPIO rising edge interrupt setup

typedef struct

```
{
uint64_t pin_bit_mask; /* GPIO pin: set with bit mask, each bit maps to a GPIO */
gpio_mode_t mode; /*< GPIO mode: set input/output mode */
gpio_pullup_t pull_up_en; /*< GPIO pull-up */
gpio_pulldown_t pull_down_en; /*< GPIO pull-down */
gpio_int_type_t intr_type; /*< GPIO interrupt type */
} gpio_config_t;
```

Mode: {disable, input, output, etc}

Pullup\_t, pulldown\_t: {enable, disable}

typedef enum {

```
GPIO_INTR_DISABLE = 0, /*< Disable GPIO interrupt */
GPIO_INTR_POSEDGE = 1, /*< GPIO interrupt type : rising edge */
GPIO_INTR_NEGEDGE = 2, /*< GPIO interrupt type : falling edge */
GPIO_INTR_ANYEDGE = 3, /* GPIO interrupt : both rising and falling edge */
GPIO_INTR_LOW_LEVEL = 4, /*GPIO interrupt type : input low level trigger */
GPIO_INTR_HIGH_LEVEL = 5, /*GPIO interrupt type : input high level trigger */
GPIO_INTR_MAX,
} gpio_int_type_t;
```

# EECS192 Lecture 4

## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

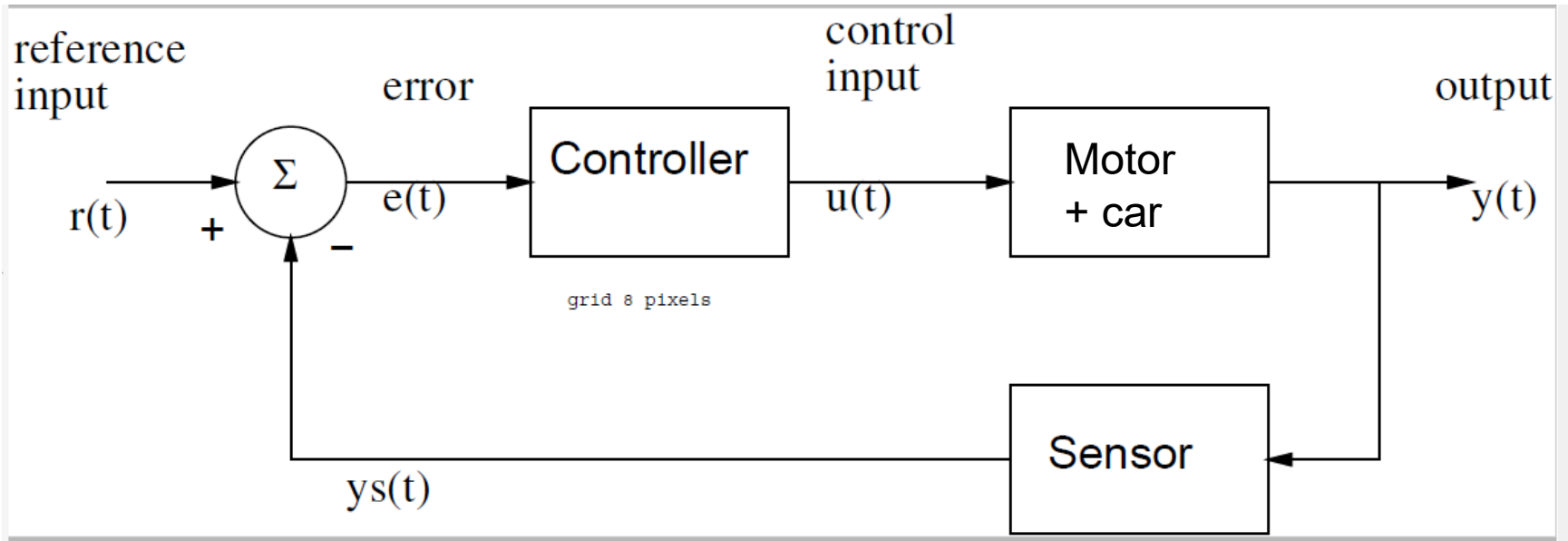
- 2/16 Quiz 1 (10 minutes) line camera timing issues

### Topics

- Checkpoints 2,3,4,5
- Line Camera- line finding
- HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- ESP32 pulse count and interrupt
- Velocity control (intro)
- Battery safety



# Velocity control overview



Proportional control:

$$u = k_p * e = k_p * (r - y);$$

Here:  $r$  is desired velocity,  $u$  is PWM % (1-2 ms)

Proportional + integral control

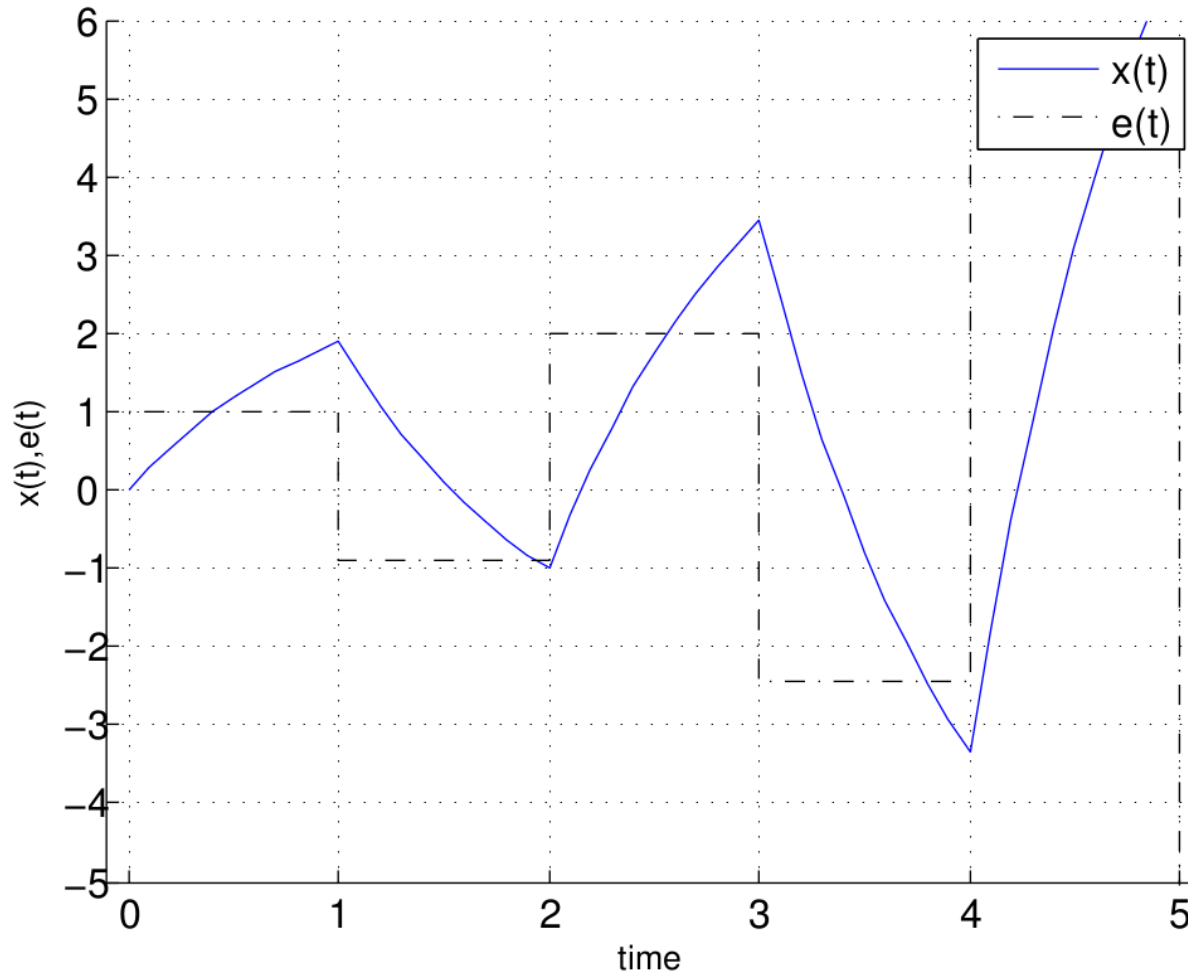
$$U = k_p * e + k_i * e\_sum;$$

$$e\_sum = e\_sum + e;$$

# Discrete Time Control

$$u[k] = k_p * (r[k] - x[k])$$

Time Series Plot:unnamed



Watch out for delay!



# EECS192 Lecture 4

## Line Sensor II and Velocity Sensing

Feb. 9, 2021

### Notes:

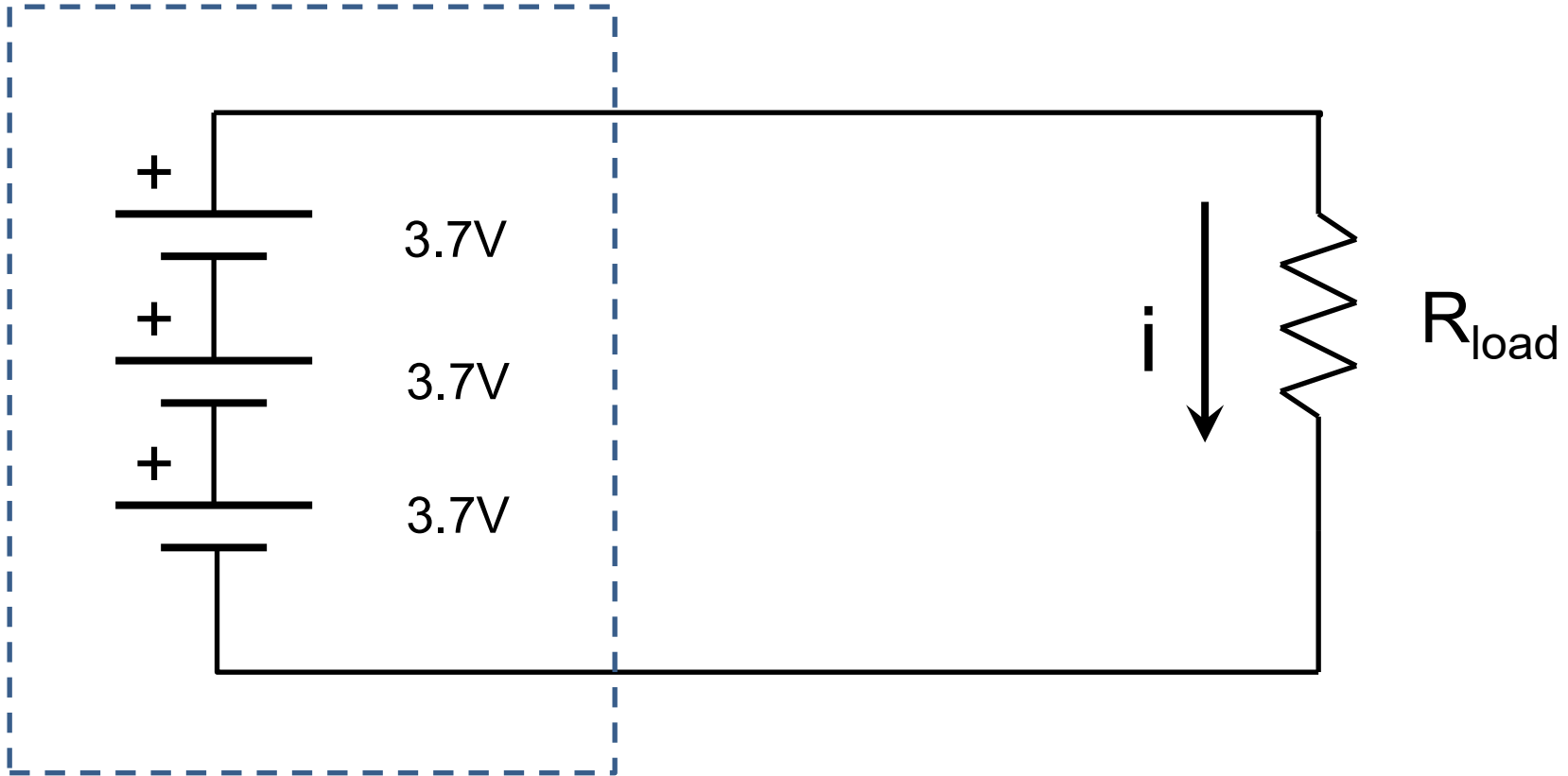
- 2/16 Quiz 1 (10 minutes) line camera timing issues

### Topics

- Checkpoints 2,3,4,5
- Line Camera- line finding
- HW1 Line finding (Python)
- Position Encoder/ Velocity Sensing
- Peripheral interface: A/D, pulse count (intro)
- Velocity control (intro)
- Battery safety

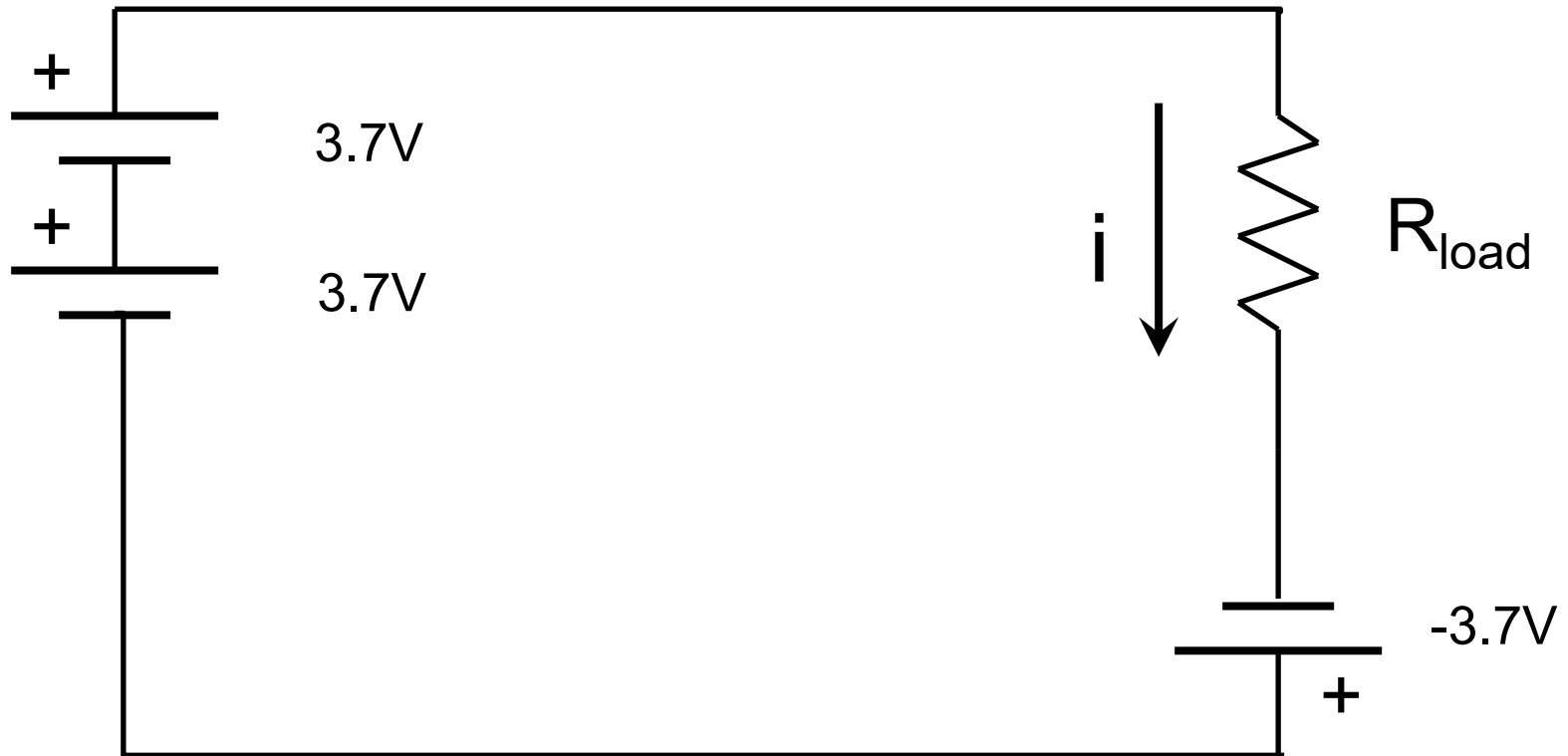


# Battery Model- 3S



# Battery Model- 3S

avoid weakly charged cell



# Latchup



Make sure Huzzah32 powered first,  
before ESC,  
before position encoder

## 1.4 Voltage and current operating ratings

Table 4. Voltage and current operating ratings

*LATCHUP!*

Symbol	Description	Min.	Max.	Unit
$V_{DD}$	Digital supply voltage	-0.3	3.8	V
$I_{DD}$	Digital supply current		140	mA
$V_{IO}$	IO pin input voltage	-0.3	$V_{DD} + 0.3$	V

Caution: input voltage from sensor may be greater than 0.3V when CPU is off  
 $V_{DD} = 0!$

Latchup phenomena:  
make sure  $V_{in}$  always less than  $V_{DD}$

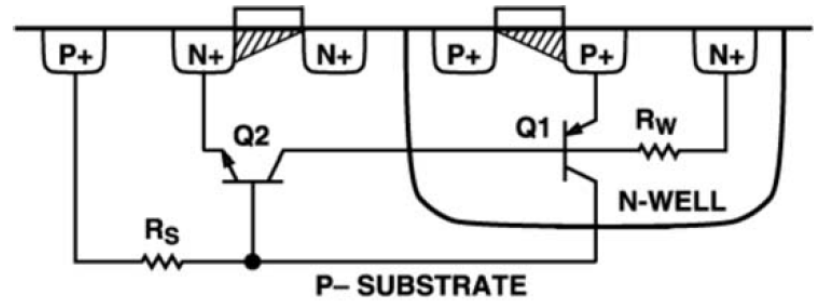
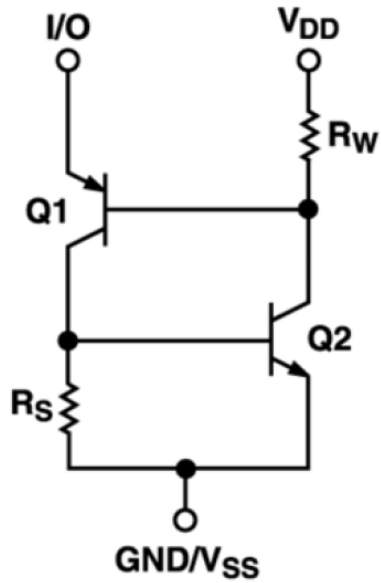
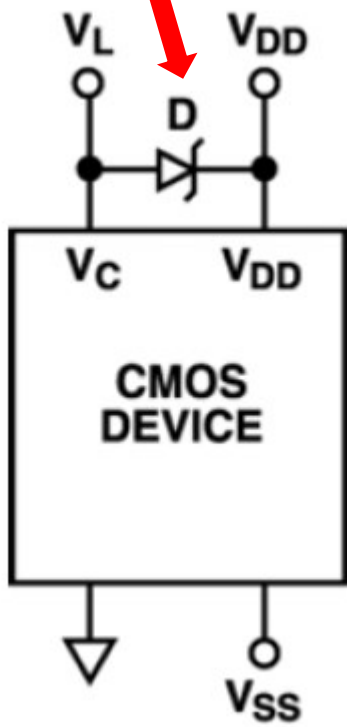


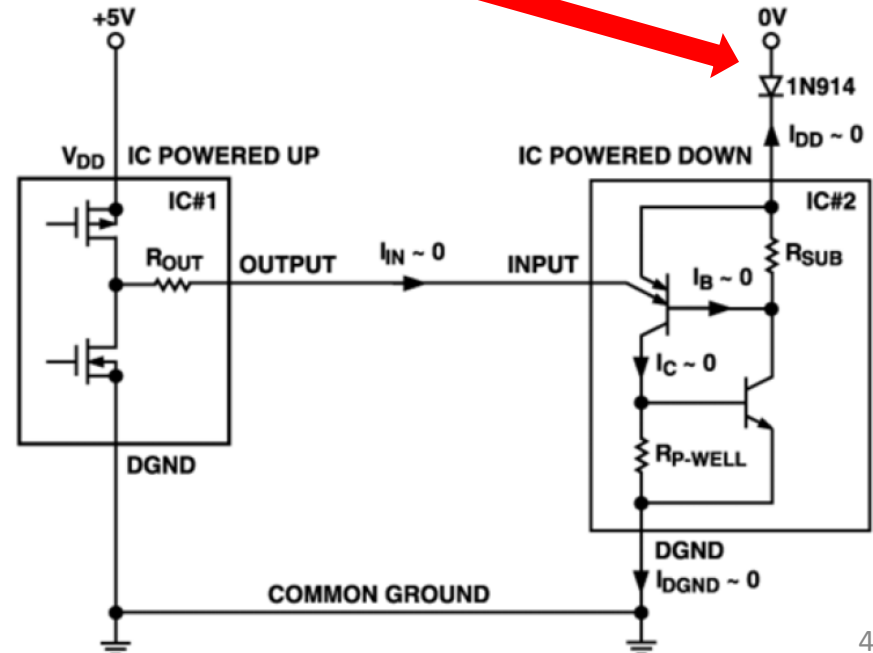
Figure 2. Cross-section of PMOS and NMOS devices, showing parasitic transistors Q1 and Q2.

Protection circuit

Protection circuit



Schottky diode



# Summary

Line Camera- line finding

HW1 Line finding (Python)

Position Encoder/ Velocity Sensing

Peripheral interface: pulse count, GPIO interrupt

Velocity control (intro)

Battery safety

# Extra Slides

# pcnt configuration

The configuration is provided separately per unit's channel using [pcnt\\_config\\_t](#) and covers:

- The unit and the channel number this configuration refers to.
- GPIO numbers of the pulse input and the pulse gate input.
- Two pairs of parameters: [pcnt\\_ctrl\\_mode\\_t](#) and [pcnt\\_count\\_mode\\_t](#) to define how the counter reacts depending on the the status of control signal and how counting is done positive / negative edge of the pulses.
- Two limit values (minimum / maximum) that are used to establish watchpoints and trigger interrupts when the pulse count is meeting particular limit. [also sets counter min/max range]

## Pulse counter configuration options

POS_MODE	LCTRL_MODE	HCTRL_MODE	sig l→h when ctrl=0	sig l→h when ctrl=1
1 (inc)	0 (-)	0 (-)	Inc ctr	Inc ctr
2 (dec)	0 (-)	0 (-)	Dec ctr	Dec ctr
0 (-)	x	x	No action	No action
1 (inc)	0 (-)	1 (inv)	Inc ctr	Dec ctr
1 (inc)	1 (inv)	0 (-)	Dec ctr	Inc ctr
2 (dec)	0 (-)	1 (inv)	Dec ctr	Inc ctr
1 (inc)	0 (-)	2 (dis)	Inc ctr	No action
1 (inc)	2 (dis)	0 (-)	No action	Inc ctr



# ADC Converter (Ch. 30)

```
adc1_config_width(ADC_WIDTH_BIT_12);  
adc1_config_channel_atten(ADC1_CHANNEL_0,ADC_ATTEN_DB_0);  
int val = adc1_get_raw(ADC1_CHANNEL_0);  
ATTEN_DB_0: 0-1 V, ATTEN_DB_11, 0-2.5V
```

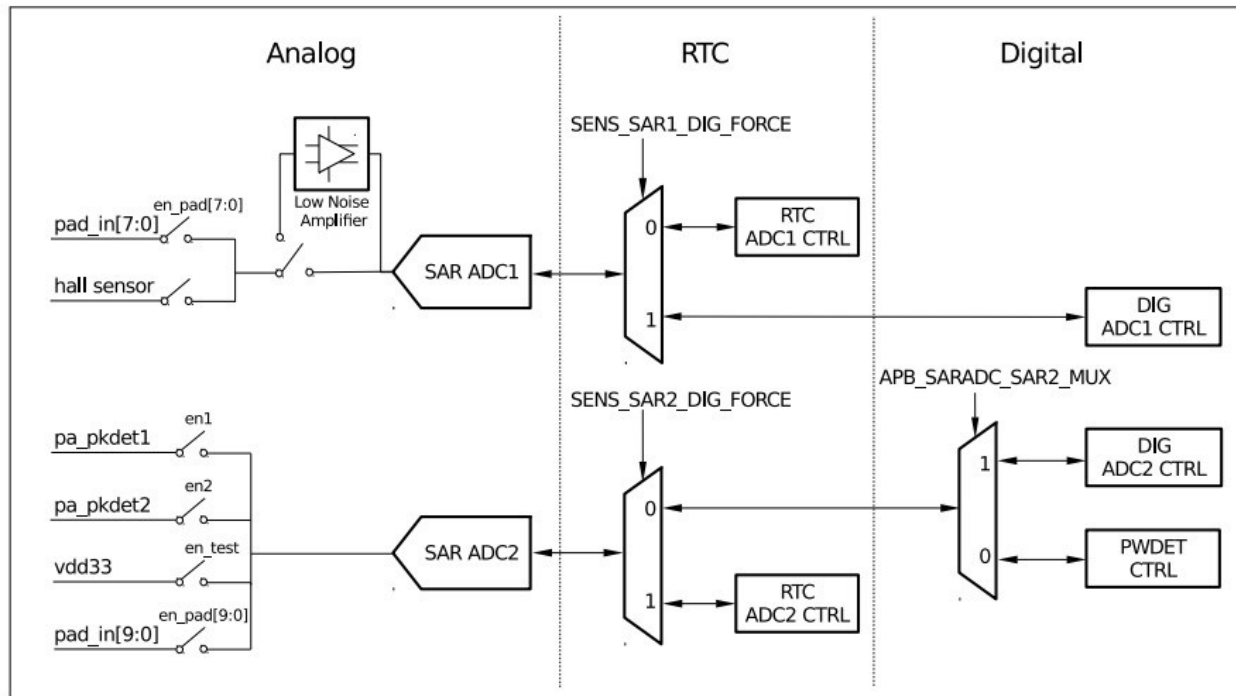


Figure 144: SAR ADC Outline of Function

<https://github.com/espressif/esp-idf/tree/release/v4.2/examples/peripherals/adc>

Claim: 2 M samples per second (Msps) = 0.5 us. **Actual 45 us for 12 bits.**

# Control Synopsis- Discrete Time

Control Law (P):

$$U(kT) = k_p [r(kT) - x(kT)]$$

New state equations:

$$x((k+1)T) = G(T)x(kT) + H(T)k_p(r(kT) - x(kT)) = [G - Hk_p]x(kT) + Hk_pr(kT) . \quad (24)$$

$$x((k+1)T) = [e^{aT} + \frac{k_p}{a}(1 - e^{aT})]x(kT) + Hk_pr(kT) = G'x(kT) + Hk_pr(kT) . \quad (25)$$

For stability:

$$|e^{aT} - \frac{k_p}{a}(e^{aT} - 1)| < 1. \quad (26)$$

Notes: stability depends on gain **and** T!