EECS192 Lecture 8
Mar. 6, 2018

Notes:
1. Check off-
   1. 3/9/2018: C6 Benchtop line tracking
   2. 3/16/2018 C7 everything on car, drop-and-run, (velocity control), open loop fig 8
   3. 3/23/2018 C8: Closed loop figure 8 drop and run
2. Quiz 4 line sensor 3/13
3. Community Spirit: PCB peer review, Piazza, helping fellow students
4. CalDay Sat. April 21 @ UCB,
5. Line sensor processing HW1 due 3/13 – upload Python code to bcourses. (Will run on other data.)
6. Waterfall plot for line data
7. Lab safety/hygiene

Topics
• Line sensor conclusion
• Software: non blocking printf, timing, debugger
• Speed sensing/velocity control
• Velocity control detail
• Feedback control overview
• Bicycle steering model
# track_center_list - A length n array of integers from 0 to 127.
   Represents the predicted center of the line in each frame.

# track_found_list - A length n array of booleans.
   Represents whether or not each frame contains a detected line.

# cross_found_list - A length n array of booleans.
   Represents whether or not each frame contains a crossing.
Possible algorithms for line detection

e.g. scipy.signal.filter

• 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
Software Notes

Read sensors ➔ process ➔ output ..... Idle ....... Read sensors ➔ process ➔ output

idle

Interrupt-
highest priority

User IO
Blocking IO
Printf

idle

Interrupt-
highest priority
Non-blocking print

https://github.com/ucb-ee192/SkeletonMCUX/tree/master/frdmk64f_skeleton

```c
sprintf(log,"Idle. sum of cos = %d \n\r", (long)ZSum);

log_add(log);

void log_add(char *log)
{
xQueueSend(log_queue, log, 0);
    // send data to back of queue,
    // non-blocking, wait=0 ==> return immediately if the queue is already full.
}

static void log_task(void *pvParameters)
{
    ... 
    xQueueReceive(log_queue, log, portMAX_DELAY);
}
Timing from MCUXpresso Tools

- Global Systime (every 100 us)
- Periodic Interrupt Timer PIT-CVAL0 (count down)
- Internal module clock frequency (60 MHz)
Timing from MCUXpresso Tools

~38 clocks for sqrt (hardware floating point) ~0.6 us
~1689 clocks for cos() ~28 us
tick_start = xTaskGetTickCount();

/* Set timer period for channel 0 */
PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, USEC_TO_COUNT(100U, PIT_SOURCE_CLOCK));
// 100 us timing
    /* Enable timer interrupts for channel 0 */
PIT_EnableInterrupts(PIT, kPIT_Chnl_0, kPIT_TimerInterruptEnable);
    /* Enable at the NVIC */
EnableIRQ(PIT_IRQ_ID);

void PIT0_IRQHandler(void) \ 100 us
{
    systime++; /* hopefully atomic operation */
    PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
    /* Clear interrupt flag.*/
    pitIsrFlag = true;
}
Velocity sensing (recap)

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

On board…
Velocity sensing

\[ \theta(t) \]

\[ t \]

\[ q(t) \]

Uniform sampling

\[ V_{\text{edge}} = \frac{\Delta \theta}{\Delta T_1} \]

Edge timing

\[ V_{\text{uniform}} = \frac{N \Delta \theta}{10 \text{ ms}} \]

Uniform sampling
Velocity control overview

On board...
Proportional control:
\[ U = kp*e = kp*(r-y); \]

Proportional + integral control
\[ U = kp*e + ki*e_{\text{sum}}; \]
\[ e_{\text{sum}} = e_{\text{sum}} + e; \]
Bicycle Steering Model

Equations On board
Bicycle Steering Control

Proportional control:
$r = 0$ (to be on straight track)
$\delta = u = kp*e$

Proportional+derivative

$P+I+D$

On board

Note steady state error: car follows larger radius