EECS192 Lecture 11
Apr. 3, 2018

Notes:
1. Progress Report due Tues 4/3 at beginning class
2. Check off 4/6: practice course, 5 min
3. Mon. 4/9: (6-7 pm) round 1
   1. 6.5 makes first turn
   2. 7 half track in < 5 minutes
   3. 9 track in less than 2 minutes
4. CalDay Sat. April 21 @ UCB, 10am -12 pm
5. Quiz 5 on 4/10 on steering control

Topics
• Backup brushed driver
• Feedforward steering control
• Step response and P.I.D. intuition
• Notes from HW2
• Discrete time control recap
• MCUXpresso and FreeRTOS notes
Better is the enemy of the good.

Optimism:
- underestimate complexity
- overestimate ability
Previous Round 2 Results

- **2015**: 36.11, 36.29, 37.19, 38.83, 40.77, 44.44, 44.95, 47.95, 48.3, 49.77, 50.49, DNF
- **2016**: 28.45, 34.97, 37.99, 44.71, 58.87, 64.58, 76.6, DNF, DNF, DNF
Motor Drive Quick and Dirty w/ Op Amp

Driving MOSFETs and motor

Checklist:
1) Emergency stop
2) Reset Protection
3) Snubbing
Bicycle Steering Control - recap

Proportional control:
\[ r = 0 \quad \text{(to be on straight track)} \]
\[ \delta = u = kp \times e \]

Proportional+derivative

\[ P+I+D \]
Proportional + Integral

On board Anti-windup
Bicycle Steering Model

Proportional control: \[ \delta(t) = k_p y_a(t) \]

Eigenvalues: \[ \lambda_{1,2} = \frac{V}{2} \left( -k_p \pm \sqrt{k_p^2 - \frac{4k_p}{L}} \right) \]

Critical damping: \( \lambda_1 = \lambda_2 \Rightarrow \)
\[ k_p^2 = 4 \frac{k_p}{L} \quad \text{or} \quad k_p = \frac{4}{L} = \frac{4}{0.2} \text{ m} = 20 \text{ rad/m} = 1150 \text{ deg/m} \]
At 3 m/s, doesn’t work well- servo saturates, also simulation dynamics…

0.7 m/s
\[ k_p = 1500 \text{ deg/m} \]
\[ K_d = 0 \]
Steering Control - PD

Example under-damped steering:

<table>
<thead>
<tr>
<th>time (sec)</th>
<th>lateral error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-0.2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>0.1</td>
</tr>
<tr>
<td>7</td>
<td>-0.1</td>
</tr>
<tr>
<td>8</td>
<td>-0.2</td>
</tr>
</tbody>
</table>
Step: 15 cm
Choose step response 1m = 300 ms
Then lateral velocity = 15 cm/300 ms = 0.5 m/sec
$\delta = kp \times 7.5 \text{ cm} + kd \times 0.5 \text{ m/sec} \Rightarrow kd \sim [0.15 \text{ sec}] kp$
Step: 15 cm
Choose step response 1m = 300 ms
Then lateral velocity = 15 cm/300 ms = 0.5 m/sec
Proportional + derivative control.
$K_p = 40 \text{ deg/cm, 70 rad/m}$
$K_d = 1000 \text{ deg/(m/sec)}$
$V = 3 \text{ m/s}$
NOTE: = bang-bang!
Proportional + derivative control.

\[ K_p = 200 \text{ deg/m}, \]
\[ K_d = 30 \text{ deg/(m/sec)} \]

\[ V = 3 \text{ m/s} \]

NOTE: = not bang-bang
Proportional + derivative control.
Kp = 200 deg/m, Kd = 30 deg/(m/sec)
V=3 m/s  NOTE: NO STEERING DELAY, no deadband

def set_steering_fast(self, angle_cmd, dt):
    self.steering_state = angle_cmd  # update state
    self.vr.simxSetFloatSignal('steerAngle',
        angle_cmd*(math.pi/180.0), vrep.simx_opmode_oneshot)
    return(angle_cmd)
Kp = 200 deg/m, Kd = 30 deg/(m/sec). V=3 m/s

Slew 60 deg/16 ms

Slew 60 deg/160 ms
2 ms/s, boom 0.1 m, kp=1500 deg/m

Step response example

2 ms/s, boom 0.1 m, kp=500 deg/m
Simulation notes

• What are other line tracking errors in addition to 128 pixel quantization?

• What are some practical limits on steering control?
Decent PD control

Too aggressive PD control
Discrete Time Control

\[ u[n] = kp*(r[n]-y[n]) \]

Time Series Plot: unnamed

On board
10 Questions to Consider when Reviewing Code

Jacob Beningo
Embedded Systems Conference -2017


1. Does the program build without warnings?
2. Are there any blocking functions?
3. Are there any potential infinite loops?
4. Should this function parameter be const?
6. Has extern been limited with a liberal use of static?
7. Do all if … else if … conditionals end with an else?
8. Are assertions and/or input/output checks present?
9. Are header guards present? The guard prevents double inclusion of the #include directives.
10. Is floating point mathematics being used?
FreeRTOS+MCUXpresso

Demo:
• Queue
• Task List
• Heap

– Log task: use for non-blocking printf for debugging
26. Execution pattern highlighting task prioritization and pre-emption in a hypothetical application in which each task has been assigned a unique priority.
Software Notes - Camera AGC

Line Find $\rightarrow$ steer & vel $\rightarrow$ idle

SI

$T_{\text{exposure}}$

CLK

129 clocks

Dummy read

129 clocks

Read video line

Why read camera this way?
Idle Task

/* configUSE_IDLE_HOOK must be set to 1 in FreeRTOSConfig.h for the idle hook function to get called. */

extern void vApplicationIdleHook( void );

https://github.com/ucb-ee192/SkeletonMCUX/tree/master/frdmk64f_skeleton/main.c
Idle Task

/* Declare a variable incremented by the hook function. */

volatile uint32_t ulIdleCycleCount = 0UL;

/* Idle hook functions MUST be called
vApplicationIdleHook(), take no parameters, and return
void. */

void vApplicationIdleHook( void )
{
    char log[MAX_LOG_LENGTH + 1]; // this uses stack...
    /* This hook function only increments a counter. */
    ulIdleCycleCount++;
    // every 1 million idle cycles
    if ((ulIdleCycleCount % 1000000) == 0)
    {
        sprintf(log, "Idle = %ld \n\r",
                (long) ulIdleCycleCount);
        log_add(log);
    }
}

https://github.com/ucb-ee192/
    SkeletonMCUX/tree/master/frdmk64f_skeleton/idletask.c
Task 1

/* Application API */
extern void write_task_1(void *pvParameters);

Inside main:
if (xTaskCreate(write_task_1, "WRITE_TASK_1", configMINIMAL_STACK_SIZE + 166, NULL, tskIDLE_PRIORITY + 2, NULL)
!= pdPASS)
{
    PRINTF("Task creation failed!.
\r\n");
    while(1); // hang indefinitely
}

https://github.com/ucb-ee192/SkeletonMCUX/tree/master/frdmk64f_skeleton/main.c
Queue in FreeRTOS


A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.
Queue in FreeRTOS

Task A

```
int x;
x = 20;
```

Task B

```
int y;
```

The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).

Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

---

Figure 31. An example sequence of writes to, and reads from a queue
```c
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);
    ...
```
Extra Slides
Proportional control.
$K_p = 20 \text{ deg/cm, } 35 \text{ rad/m}$

$V = 2.2 \text{ m/s}$

NOTE: = bang-bang!
Proportional + derivative control.

$K_p = 20 \ \text{deg/cm}, \ 35 \ \text{rad/m}$

$K_d = 300 \ \text{deg/(m/sec)}$

$V = 2.2 \ \text{m/s}$

NOTE: = bang-bang!