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.

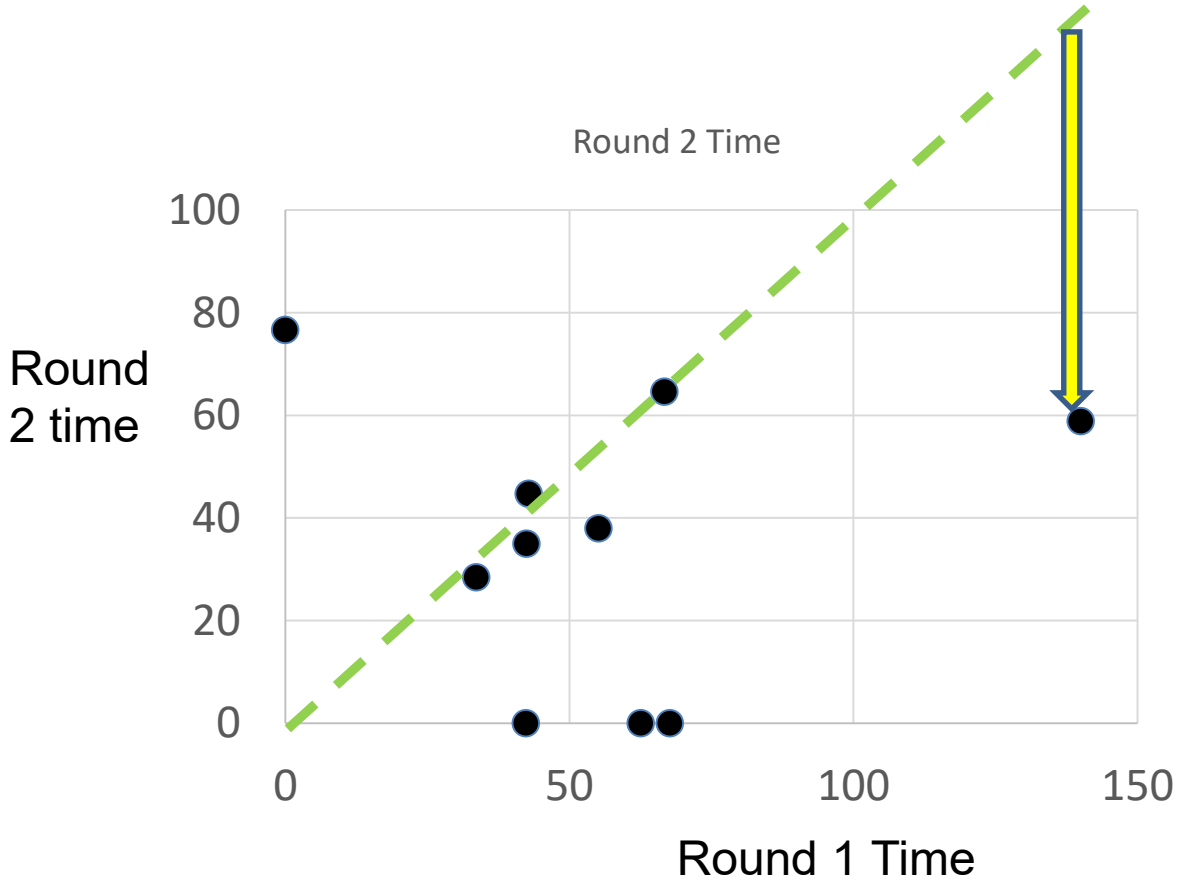
Optimism:

underestimate complexity+

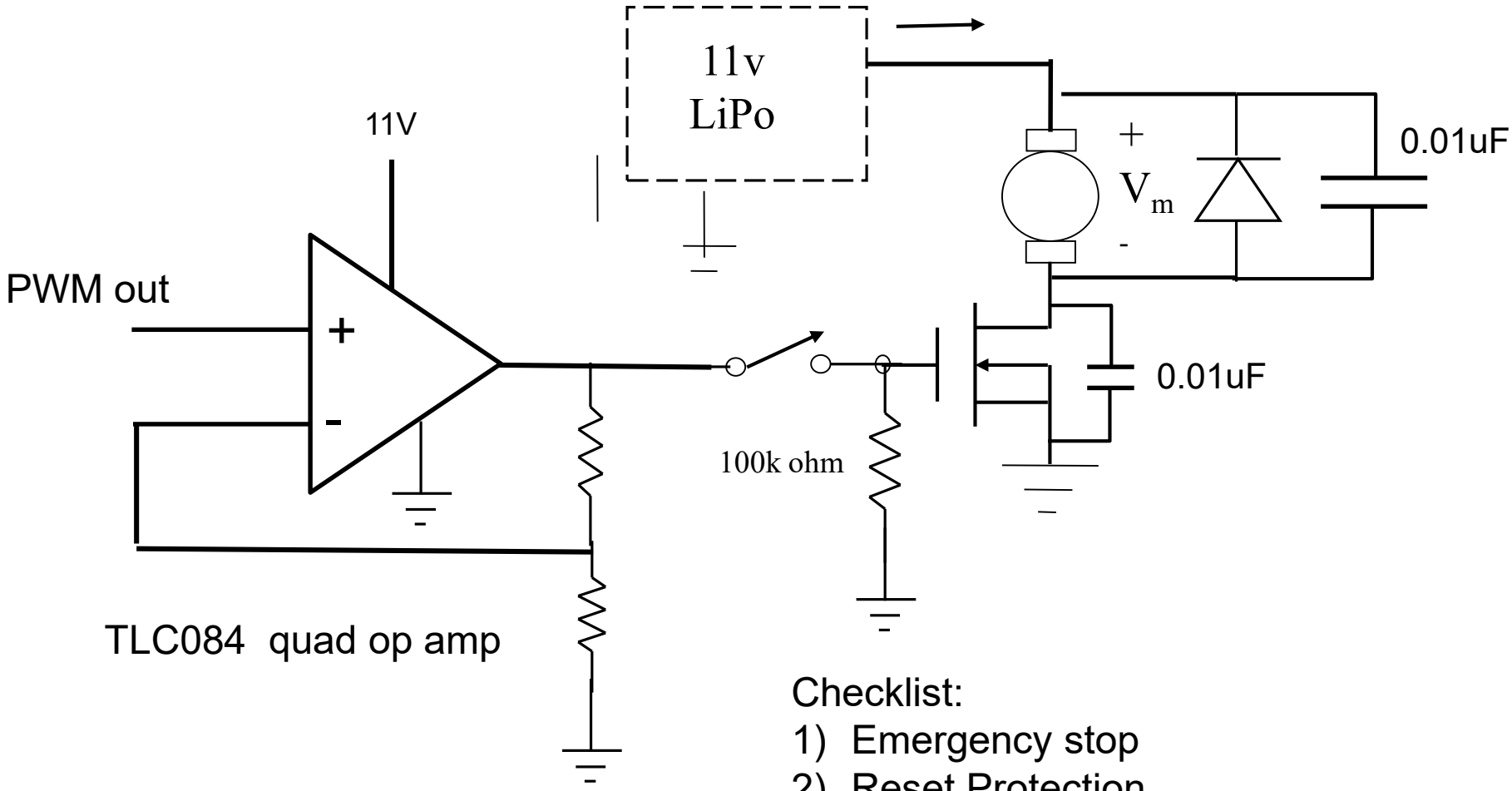
overestimate ability

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



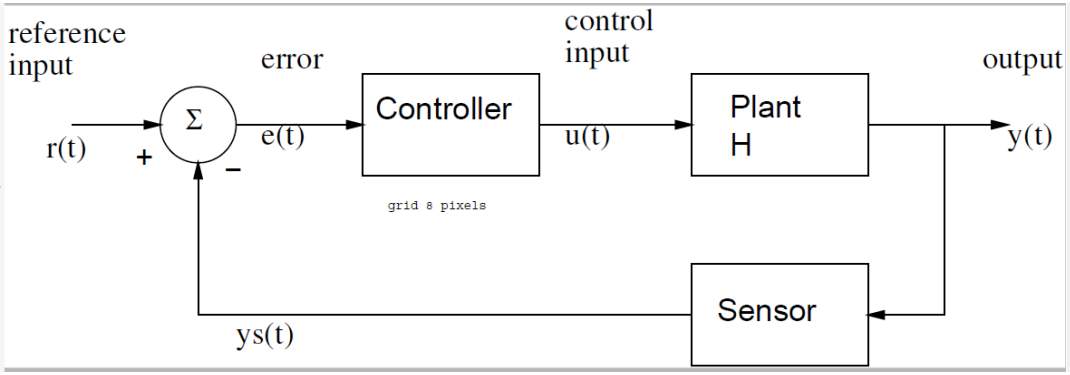
Motor Drive Quick and Dirty w/ Op Amp



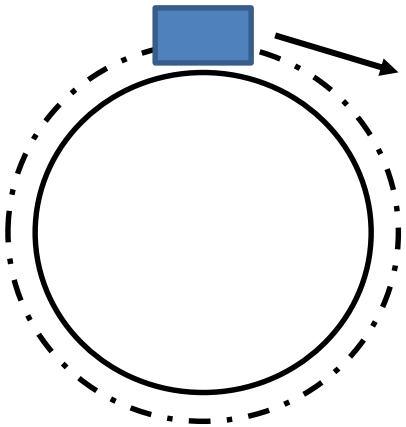
Checklist:

1) Emergency stop
2) Reset Protection
3) Snubbing

Bicycle Steering Control- recap



Note steady state error: car follows larger radius

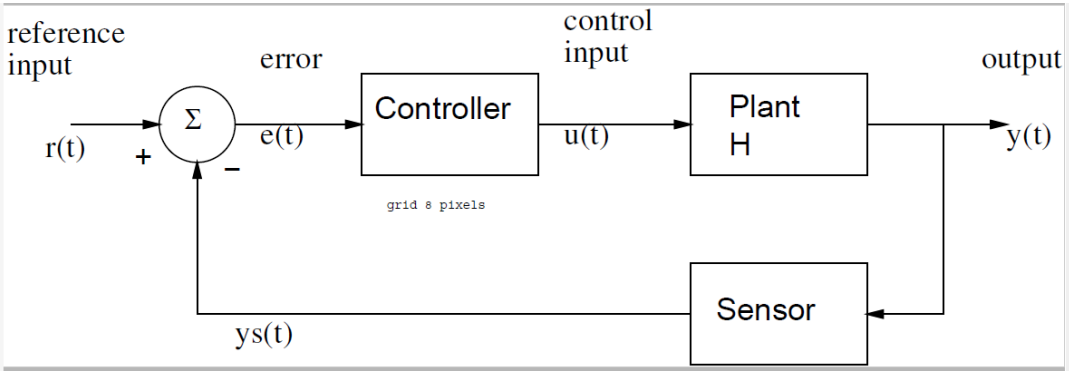


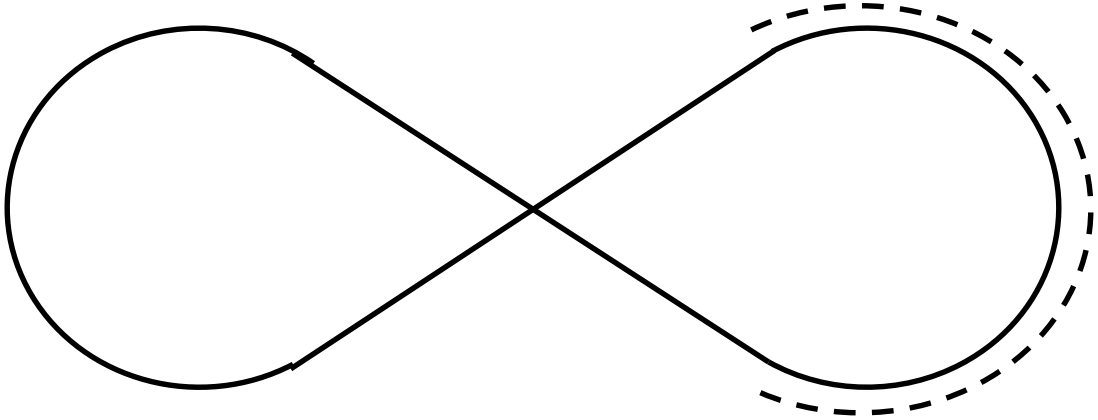
Proportional control:
r = 0 (to be on straight track)
δ=u = kp*e

Proportional+derivative

P+I+D

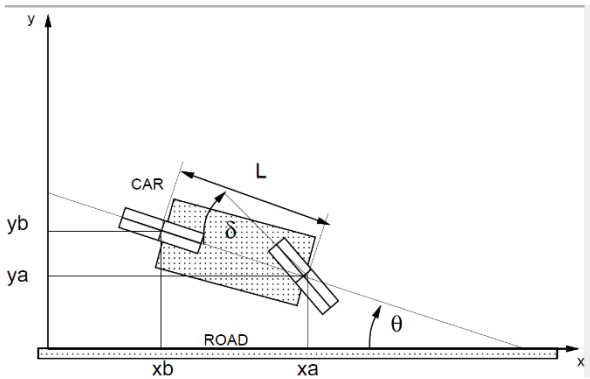
Proportional + Integral





On board Anti-windup

Bicycle Steering Model



Proportional control: $\delta(t) = k_p y_a(t)$

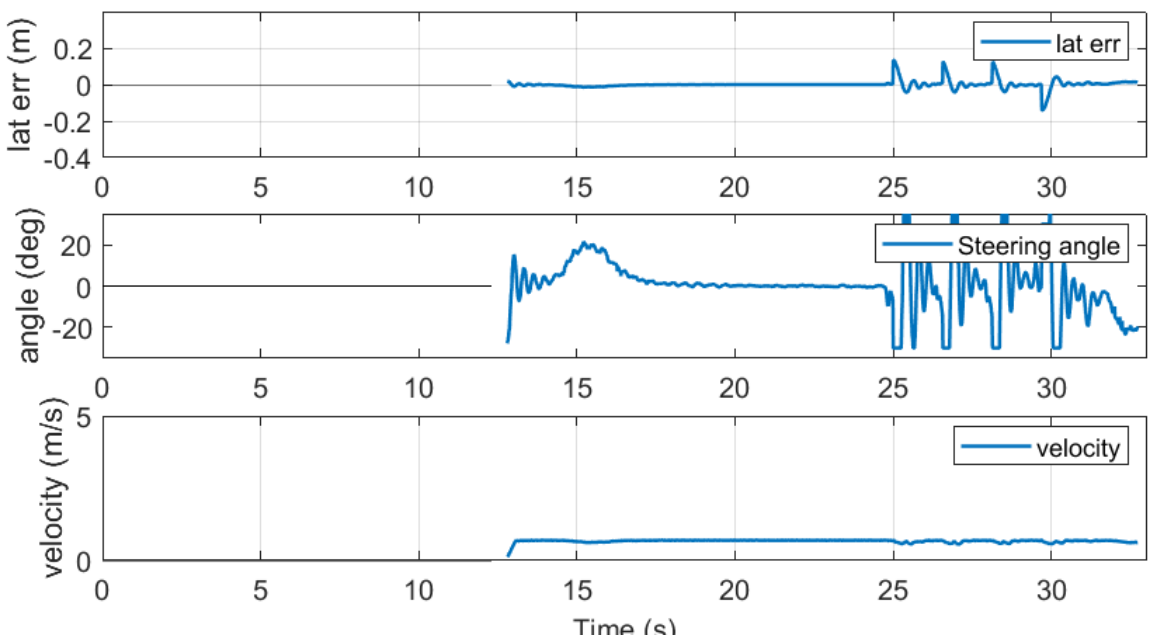
$$\ddot{y}_a + V k_p \dot{y}_a(t) + \frac{V^2}{L} k_p y_a(t) = 0.$$

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

Critical damping: $\lambda_1 = \lambda_2$ ➔

$k_p^2 = 4\, k_p/L$ or $k_p = 4/L = 4/0.2$ m = 20 rad/m = 1150 deg/m

At 3 m/s, doesn't work well- servo saturates, also simulation dynamics…

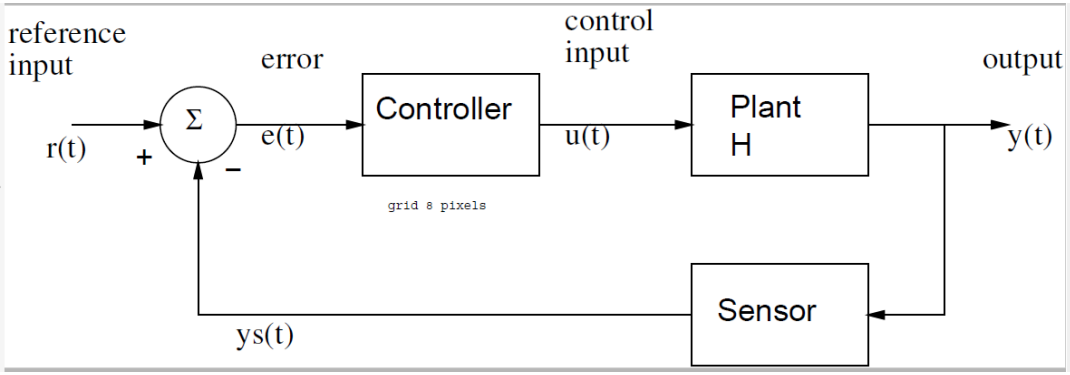


0.7 m/s

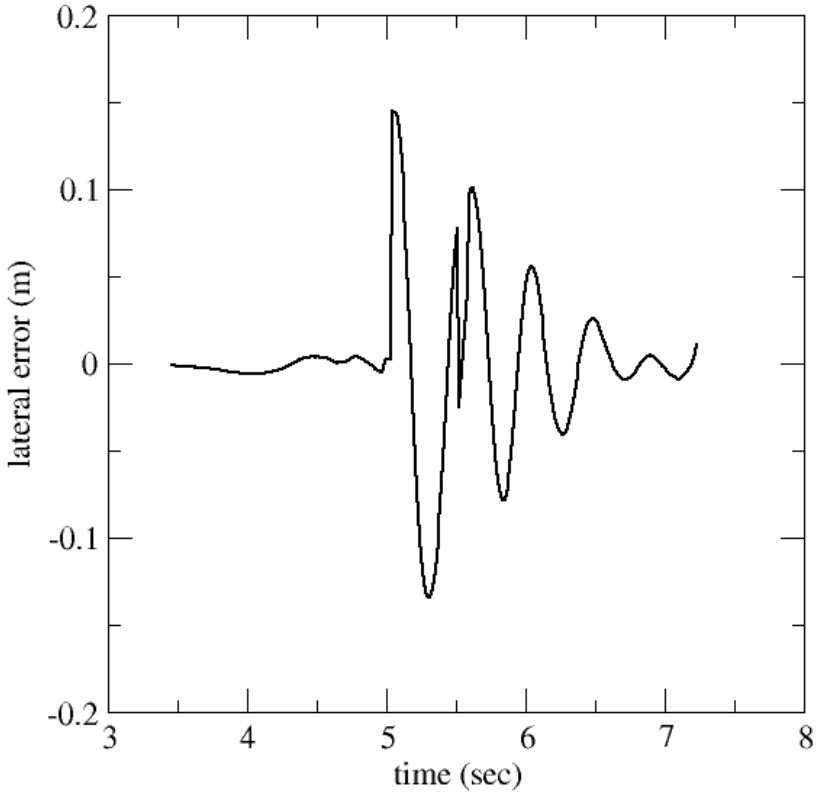
k_p =1500 deg/m

Kd = 0

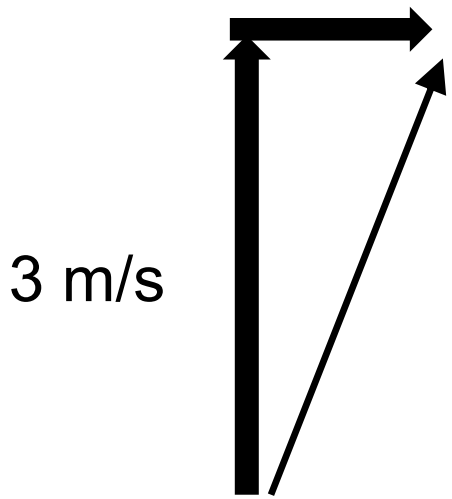
Steering Control- PD

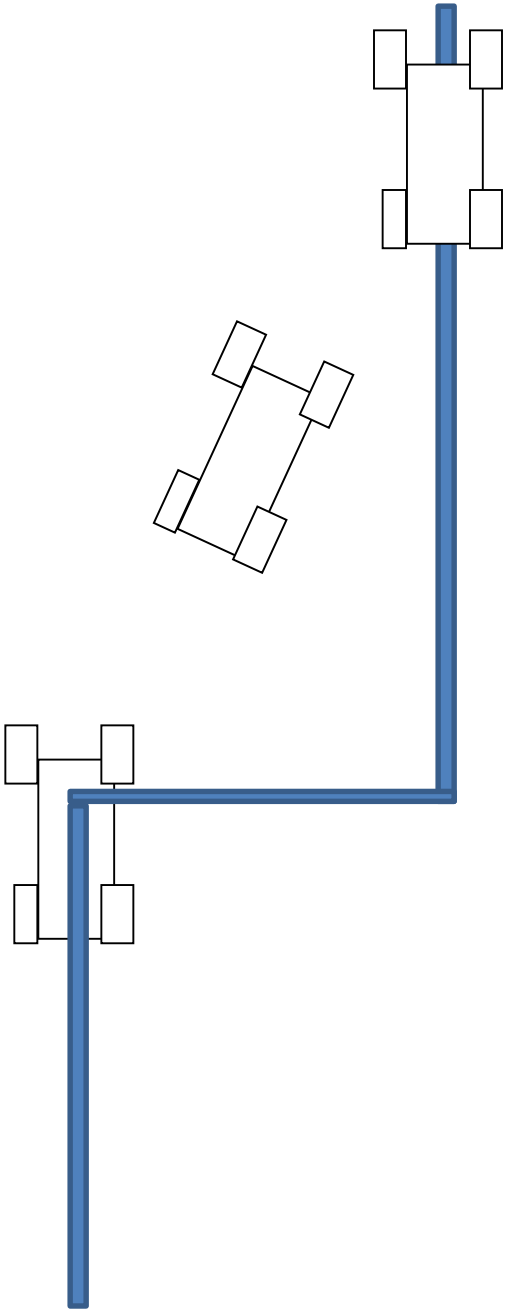


Example under-damped steering:



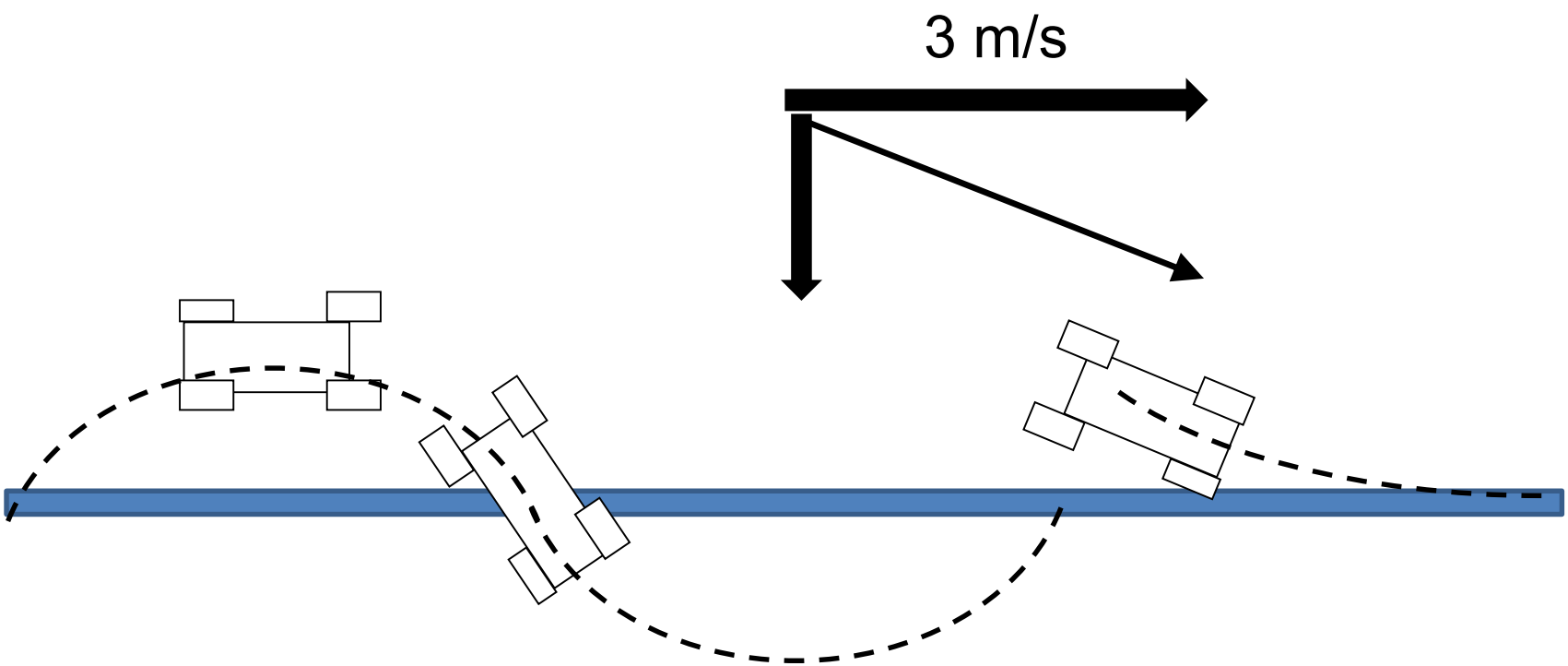
PD parameters





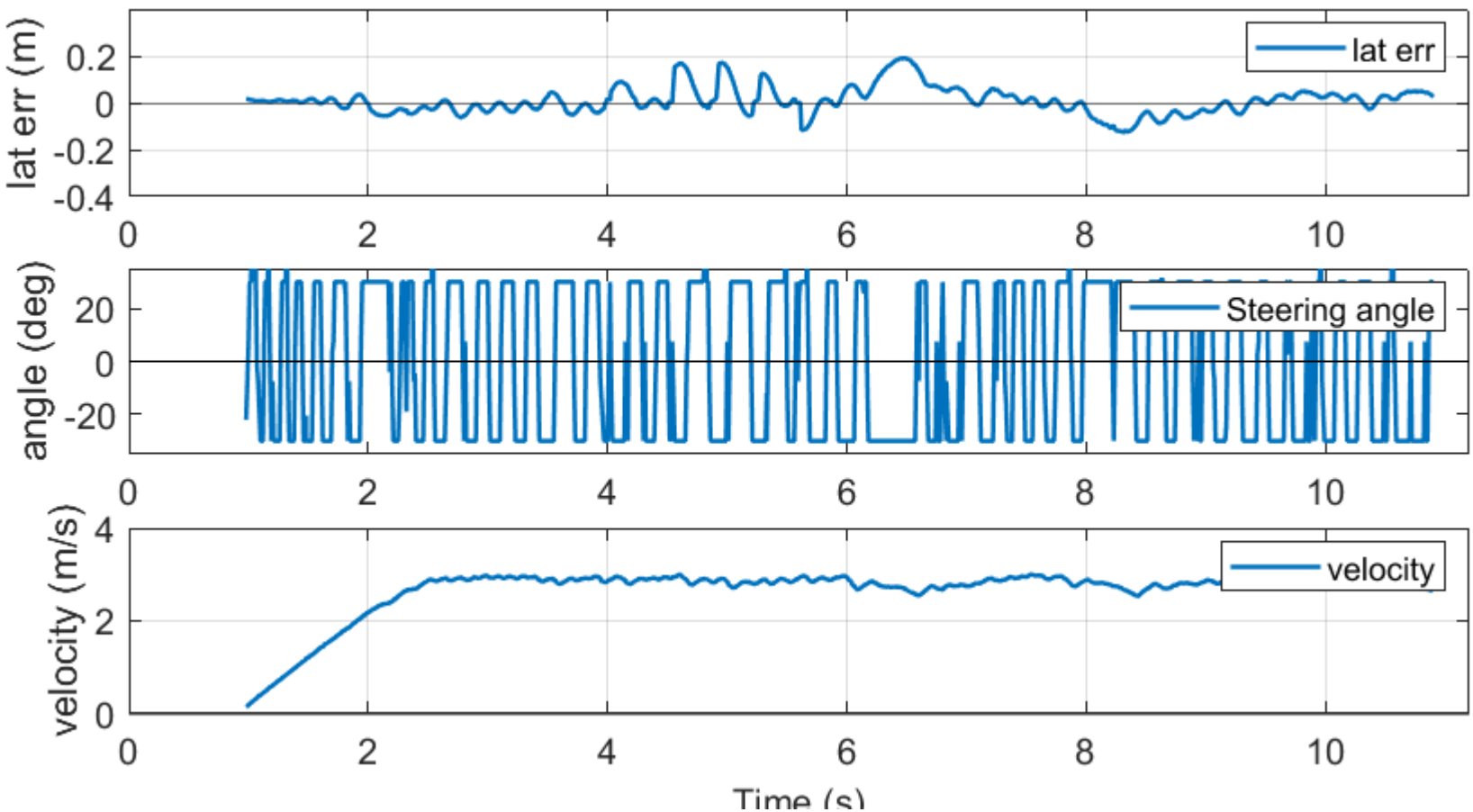
Step: 15 cm
Choose step response 1m = 300 ms
Then lateral velocity = 15 cm/300 ms = 0.5 m/sec
δ = kp 7.5 cm + kd 0.5 m/sec ➔ kd ~ [0.15 sec] kp

PD parameters

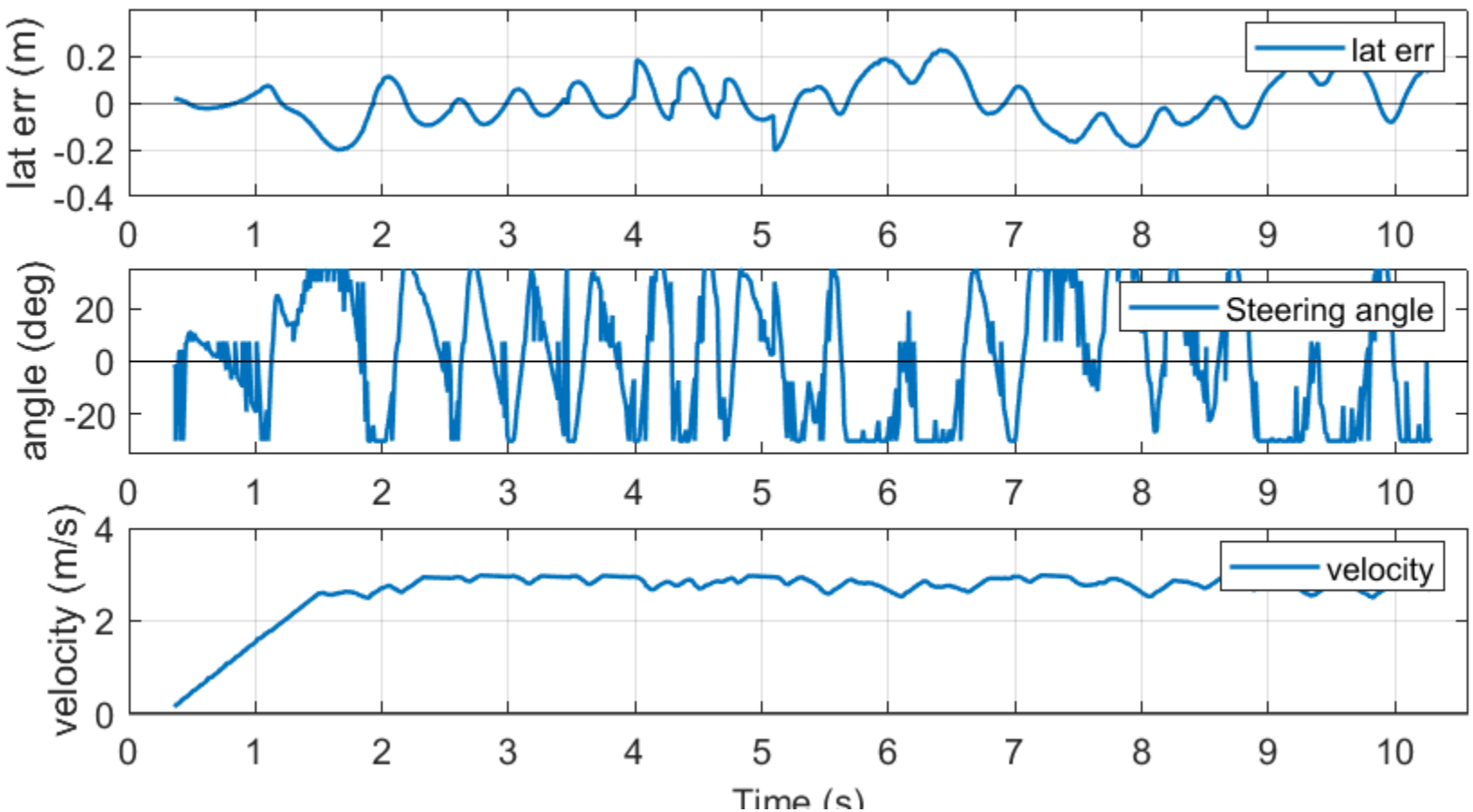


Step: 15 cm
Choose step response 1m = 300 ms
Then lateral velocity = 15 cm/300 ms = 0.5 m/sec

Proportional + derivative control.
Kp = 40 deg/cm, 70 rad/m
Kd = 1000 deg/(m/sec)
V=3 m/s
NOTE: = bang-bang!

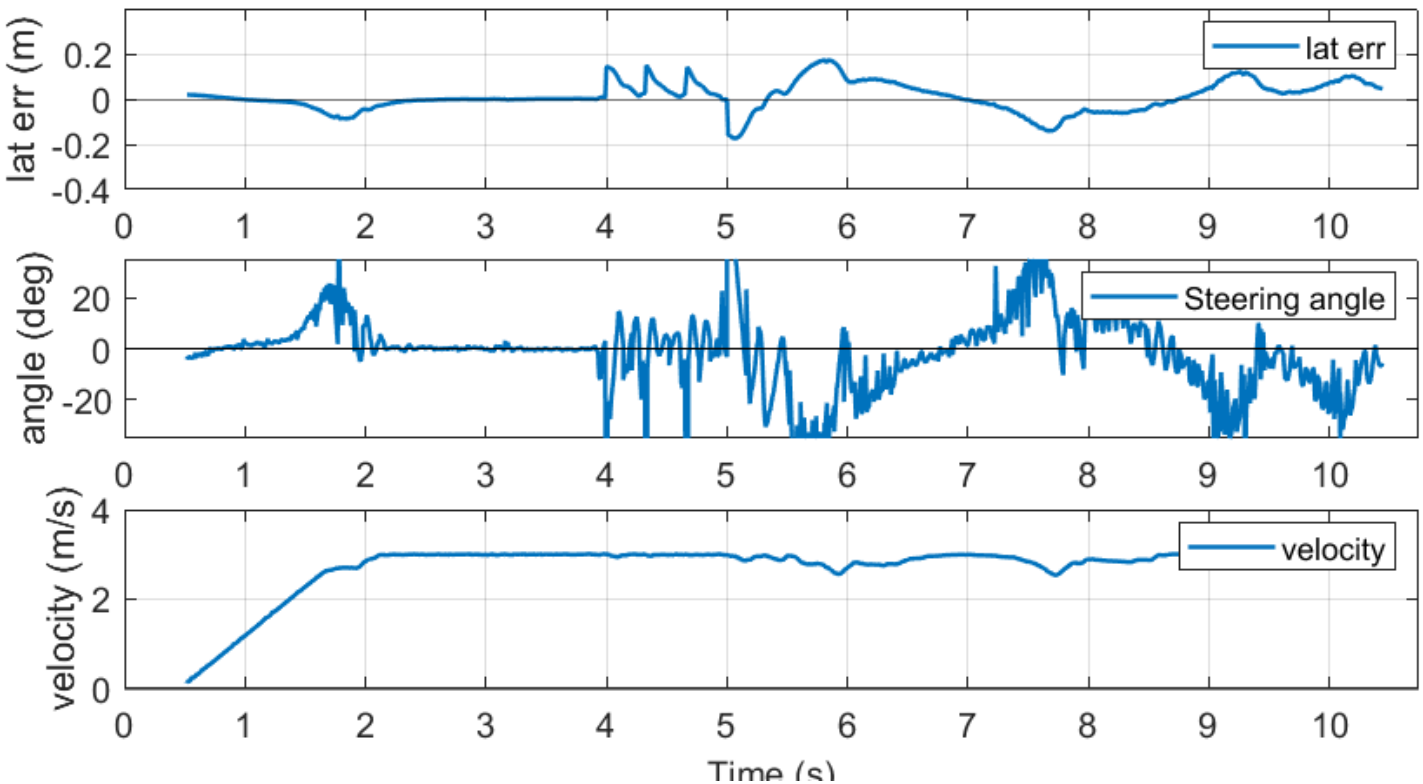


Proportional + derivative control.
Kp = 200 deg/m,
Kd = 30 deg/(m/sec)
V=3 m/s
NOTE: = not bang-bang

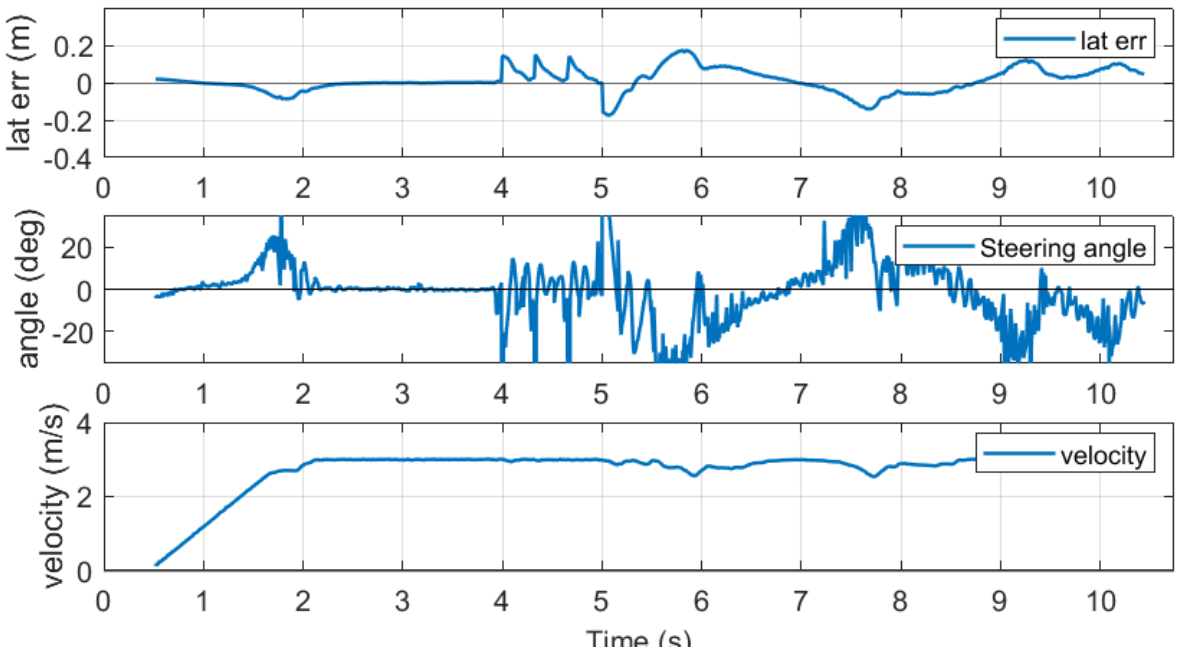


Proportional + derivative control.
Kp = 200 deg/m, Kd = 30 deg/(m/sec)
V=3 m/s NOTE: NO STEERING DELAY, no deadband

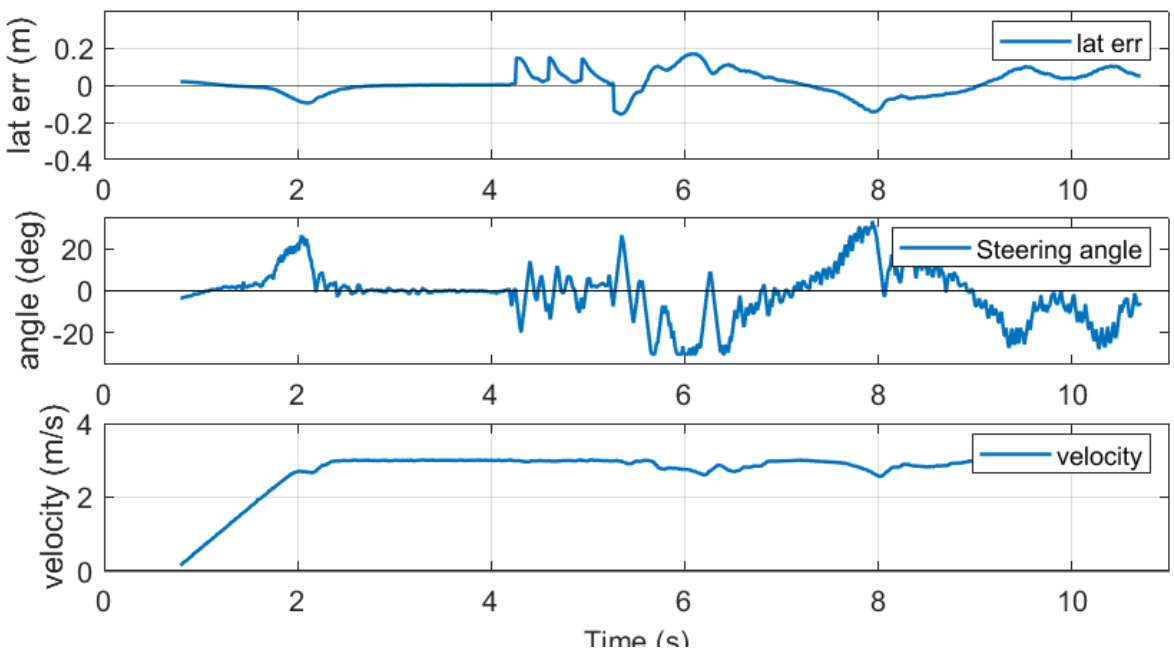
```
def set_steering_fast(self, angle_cmd, dt):
    self.steering_state = angle_cmd # update state
    self.vr.simxSetFloatSignal('steerAngle',
        angle_cmd*(math.pi/180.0), vrep.simx_opmode_oneshot)
    return(angle_cmd)
```



Kp = 200 deg/m, Kd = 30 deg/(m/sec). V=3 m/s



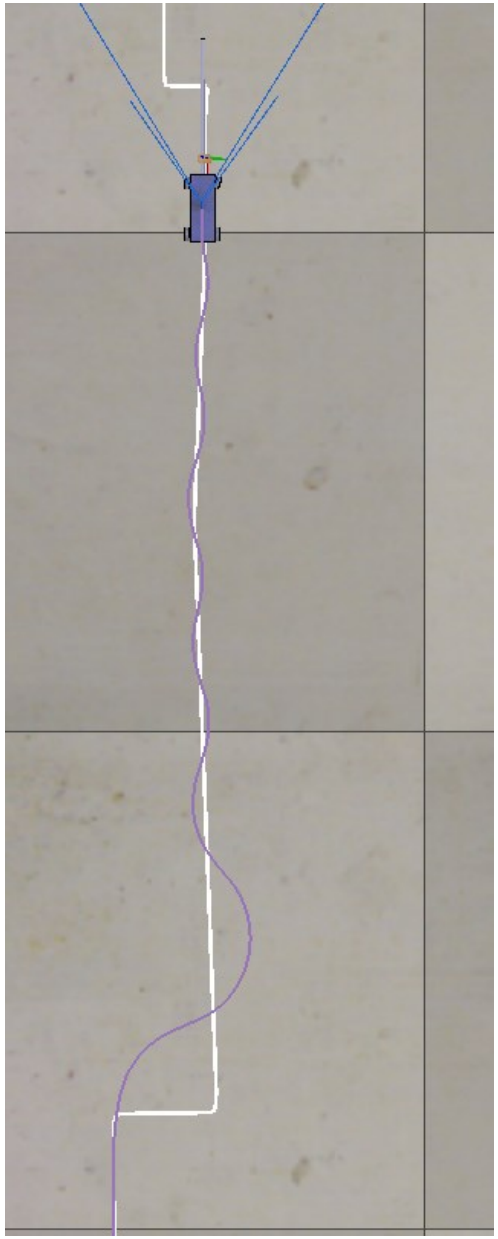
Slew 60 deg/16 ms



Slew 60 deg/160 ms

Step response example

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

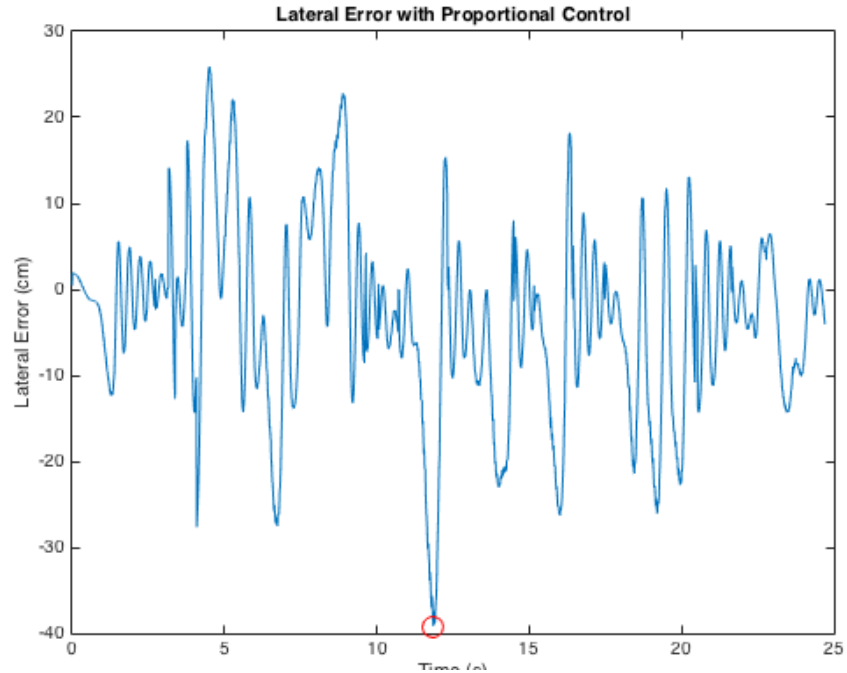


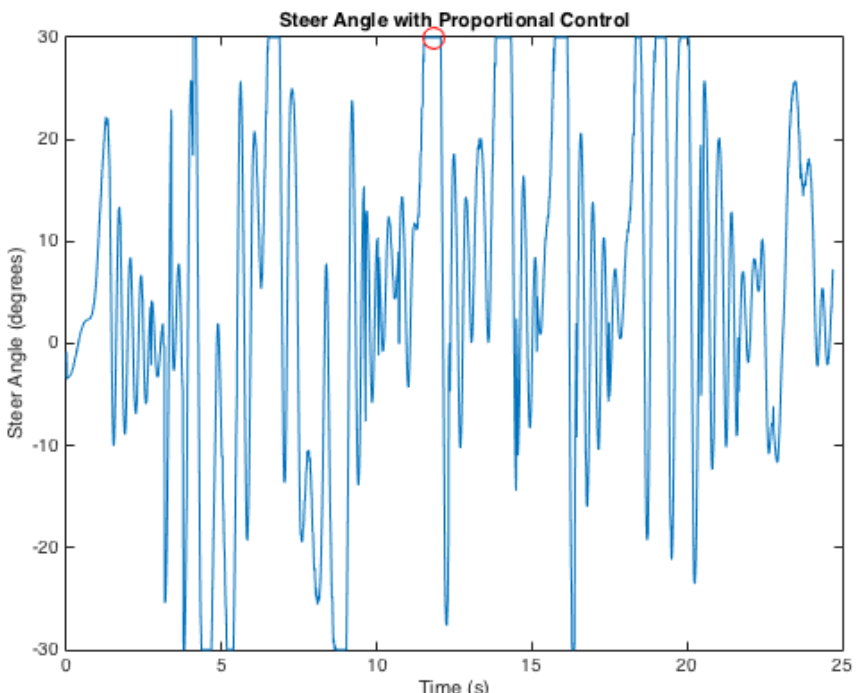
2 ms/s, boom 0.1 m, kp=500 deg/m

Simulation notes

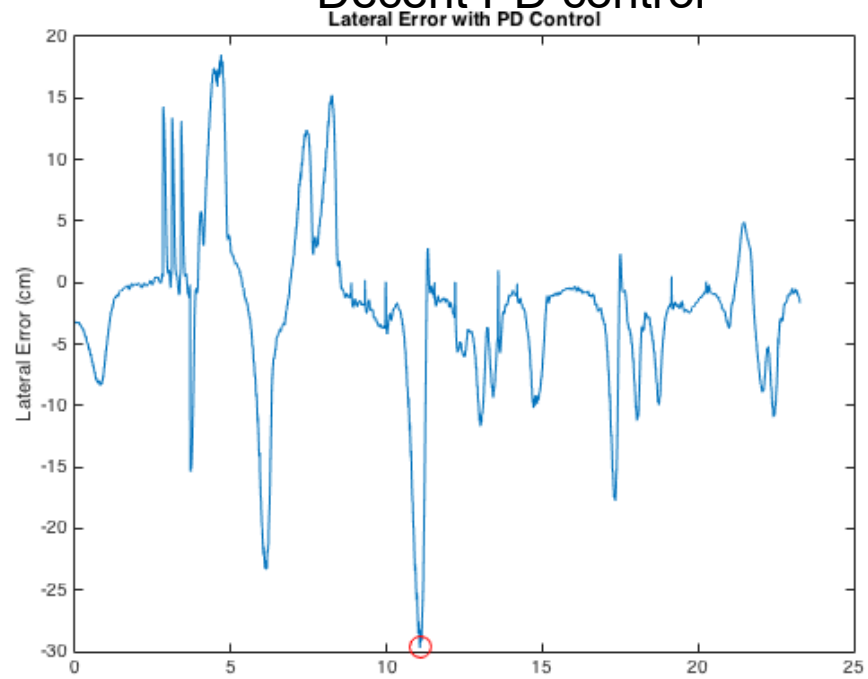
- What are other line tracking errors in addition to 128 pixel quantization?
- What are some practical limits on steering control?

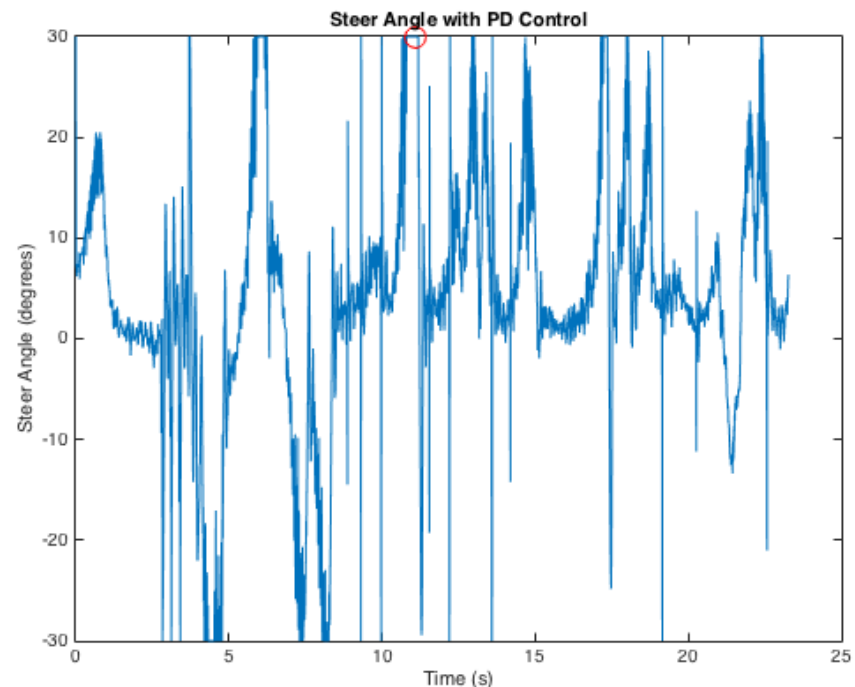
underdamped PD control



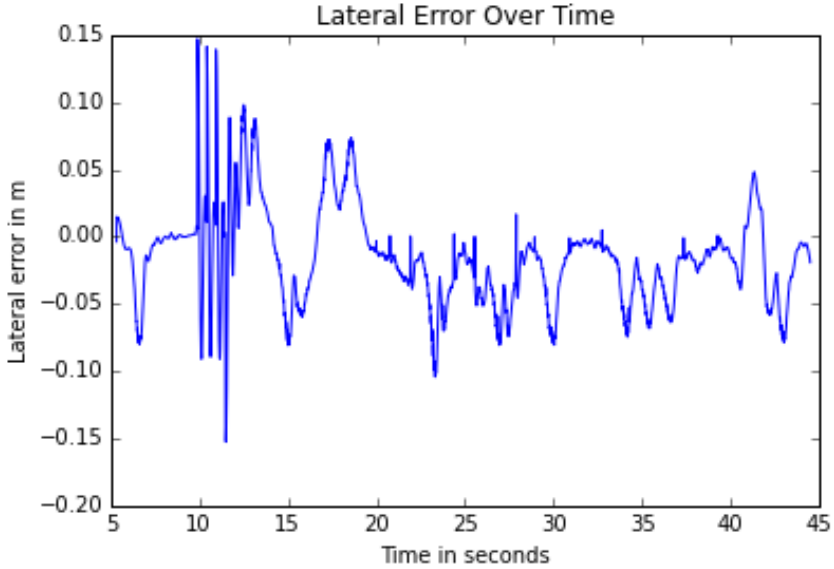


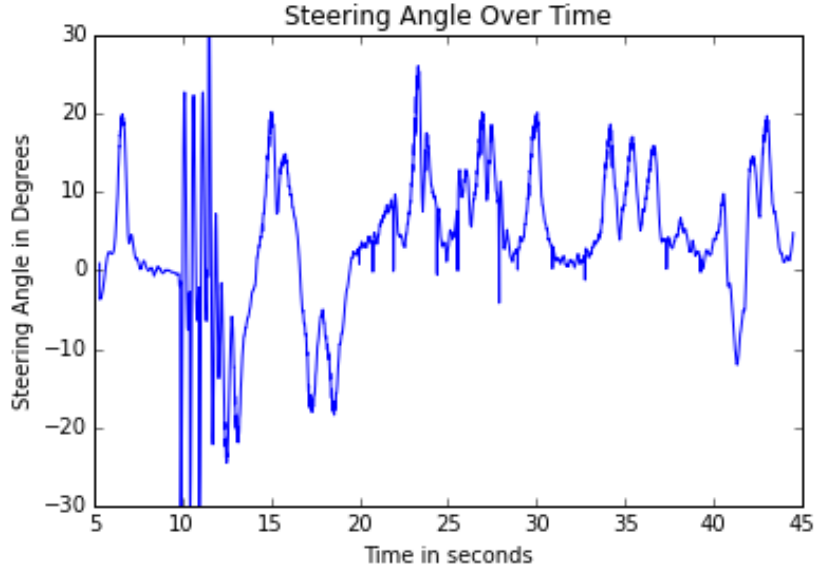
Decent PD control



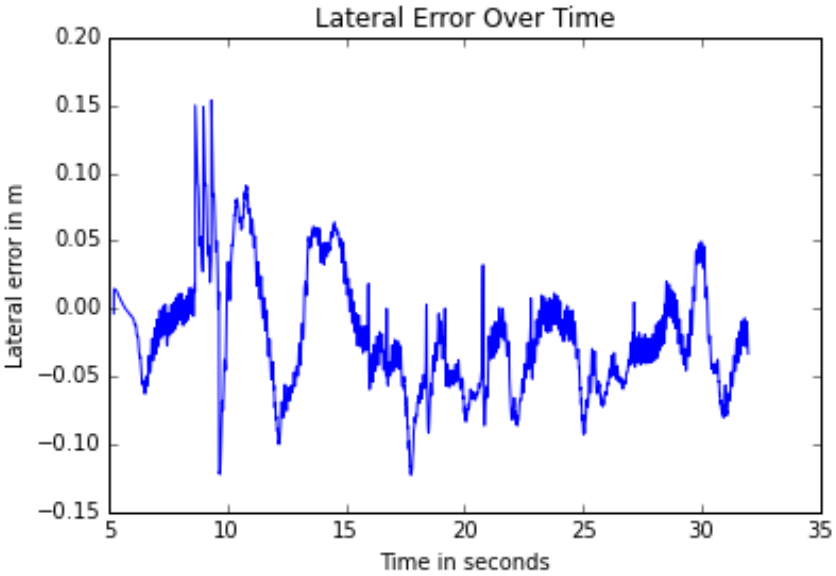


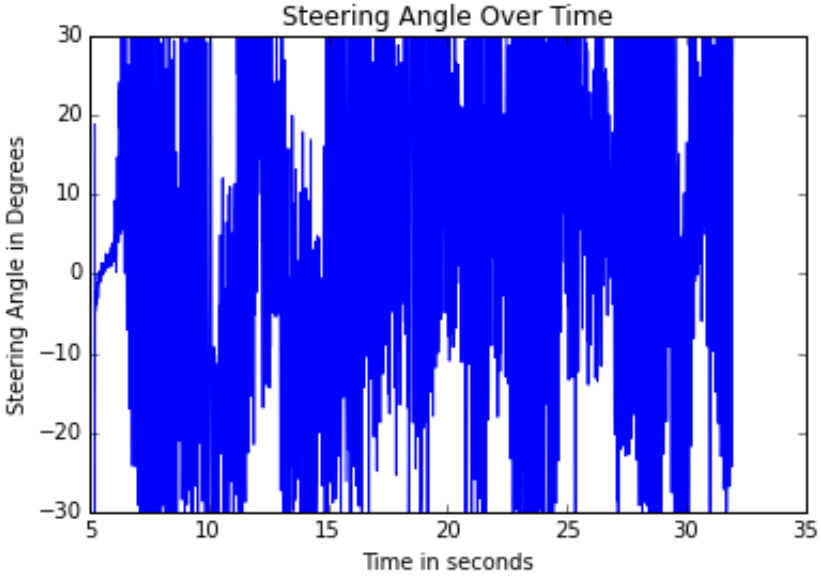
Decent PD control





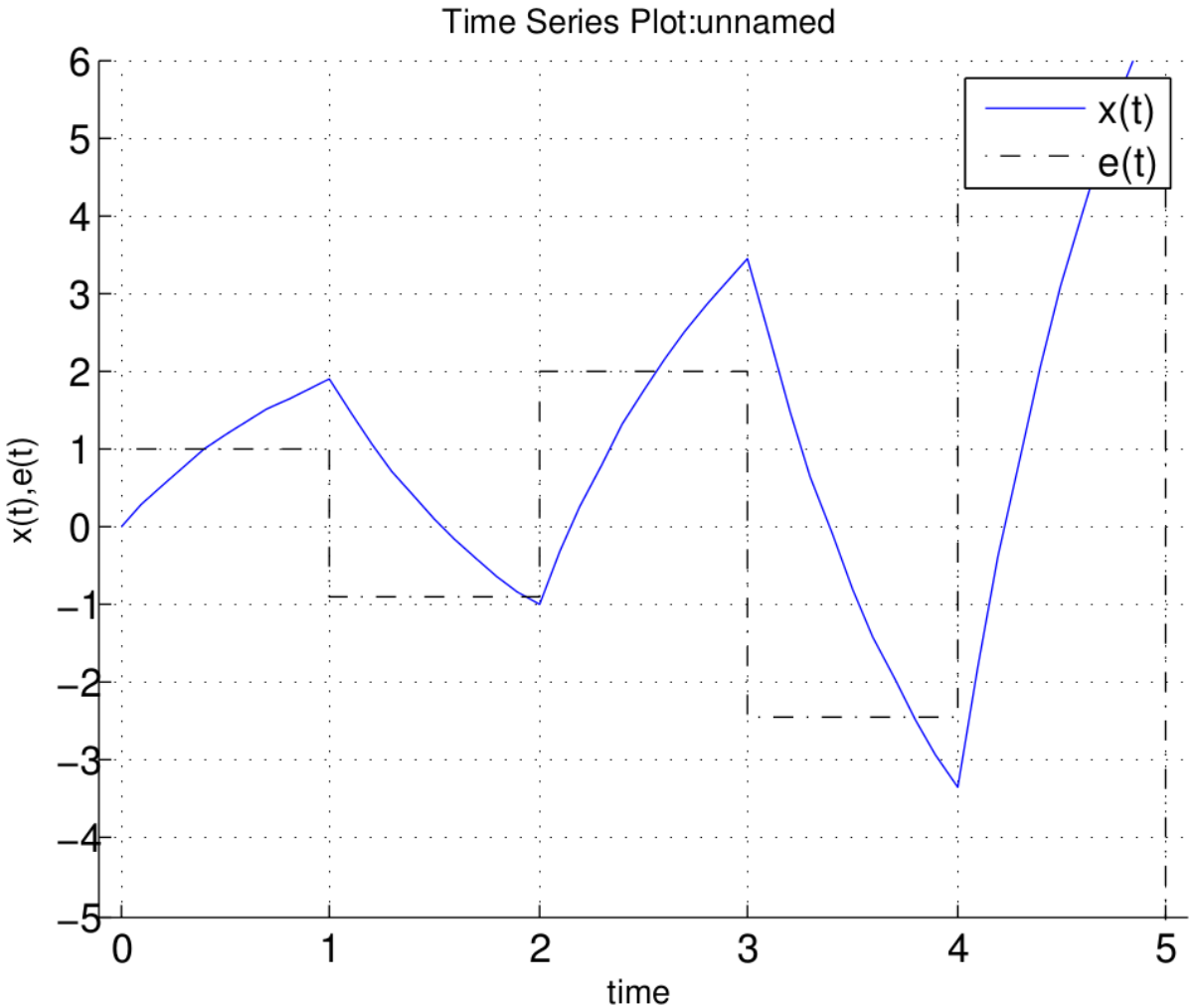
Too aggressive PD control





Discrete Time Control

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



On board

10 Questions to Consider when Reviewing Code

Jacob Beningo

Embedded Systems Conference -2017

https://www.designnews.com/electronics-test/10-questions-consider-when-reviewing-code/143583201956491?cid=nl.x.dn14.edt.aud.dn.20170329

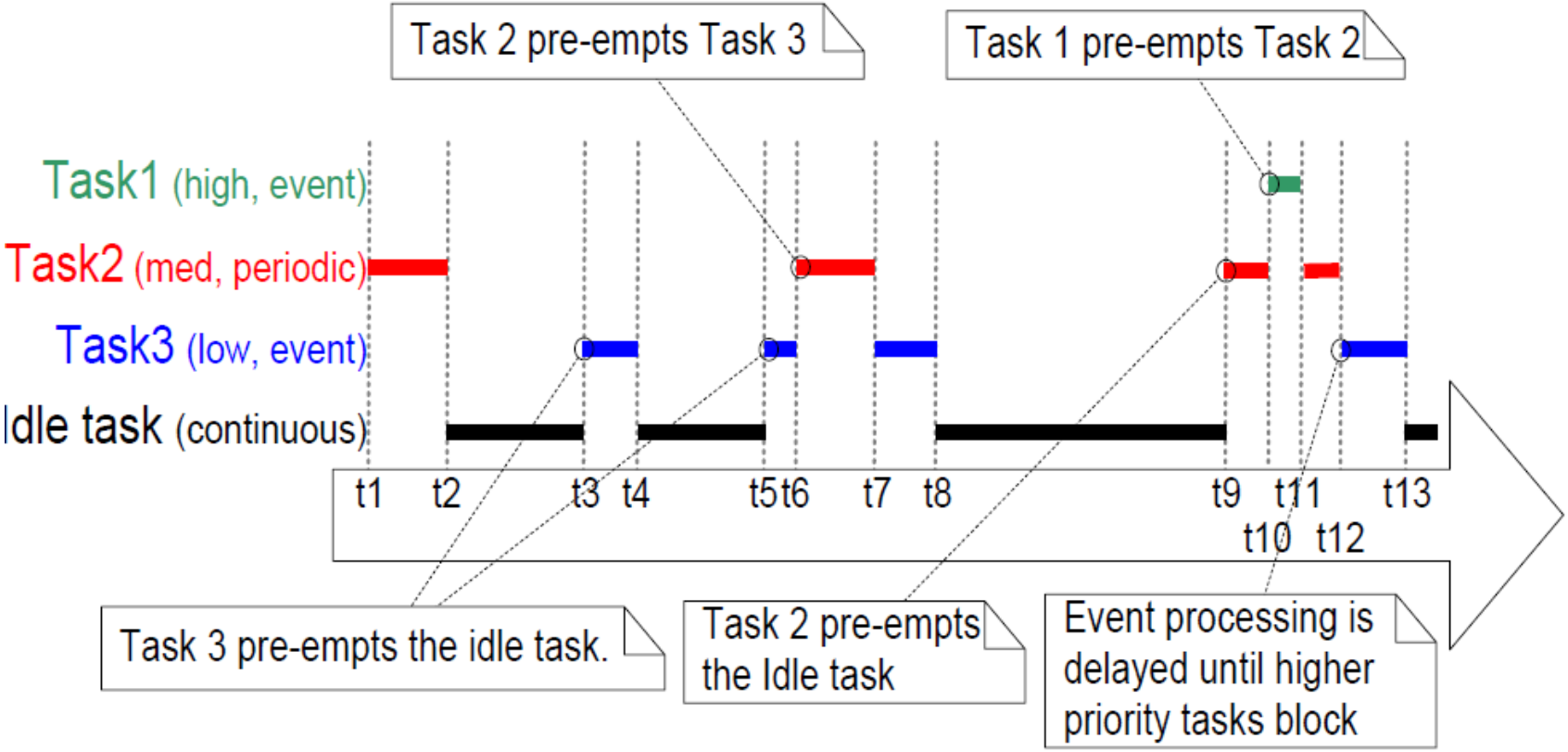
1. Does the program build without warnings?
2. Are there any blocking functions?
3. Are there any potential infinite loops?
4. Should this function parameter be const?
6. Has extern been limited with a liberal use of static?
7. Do all if … else if … conditionals end with an else?
8. Are assertions and/or 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:

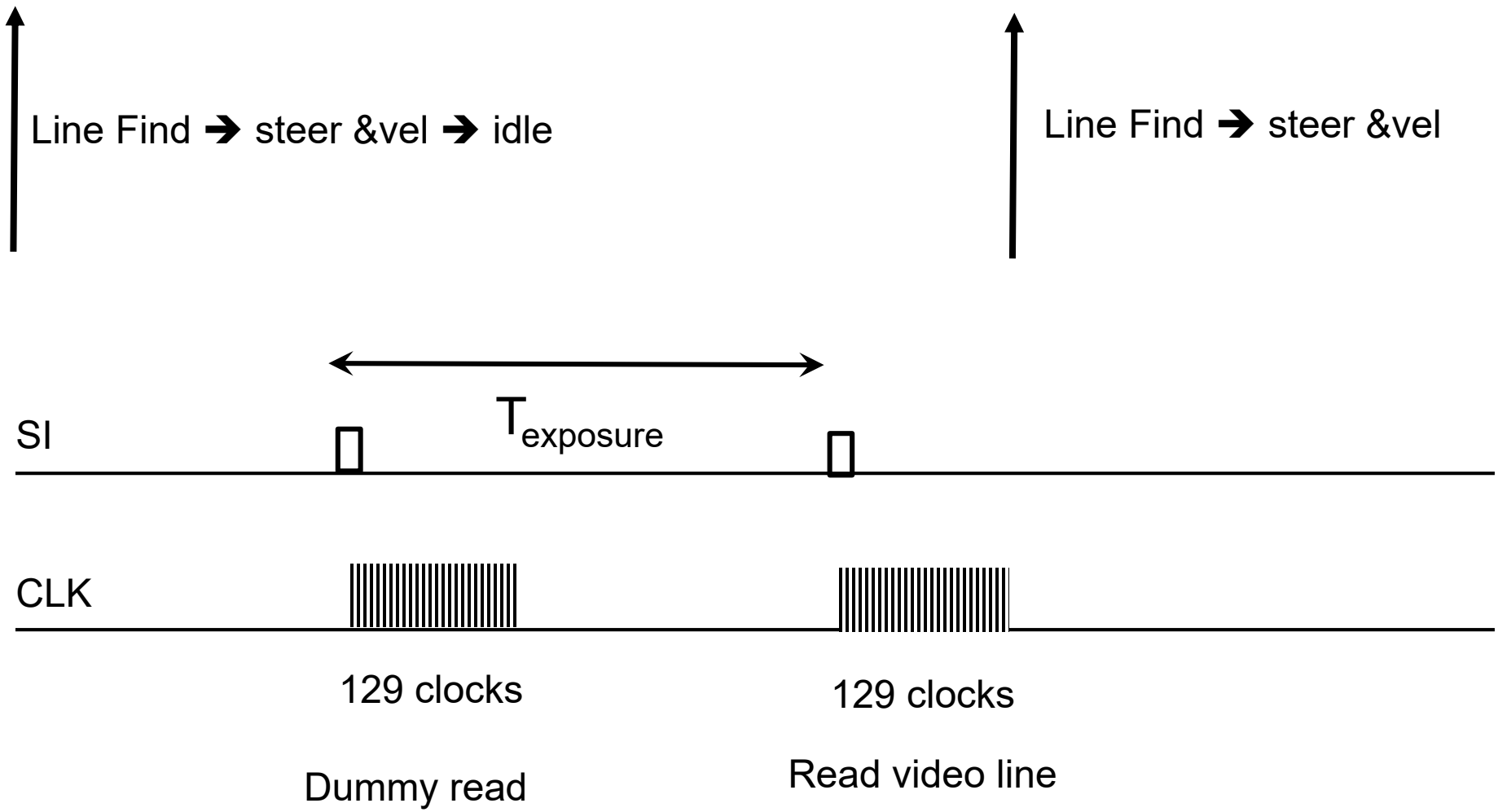
- Queue
- Task List
- Heap

  – 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



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);
```

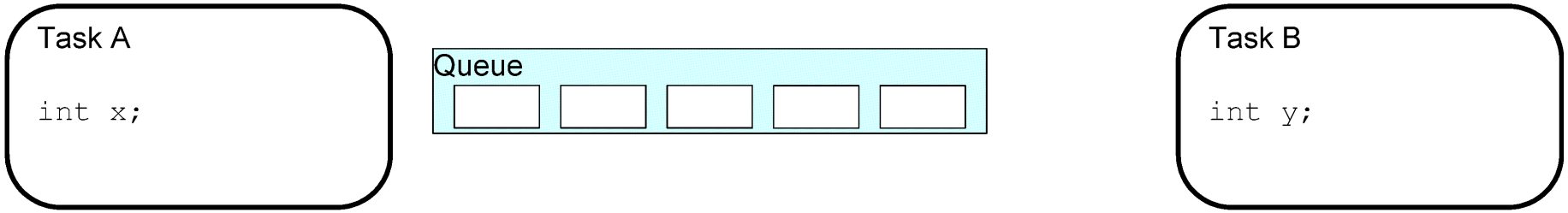
Inside main:

```
if(xTaskCreate(write_task_1,    C function
"WRITE_TASK_1",     Name for debugging purposes
configMINIMAL_STACK_SIZE + 166, Stack size for task
NULL,               Pointer to parameters to pass into task
tskIDLE_PRIORITY + 2, Task priority (0=lowest)
NULL)               Optional handle to created task
     != pdPASS)
    {   PRINTF("Task creation failed!.\r\n");
        while(1); // hang indefinitely
    }
```

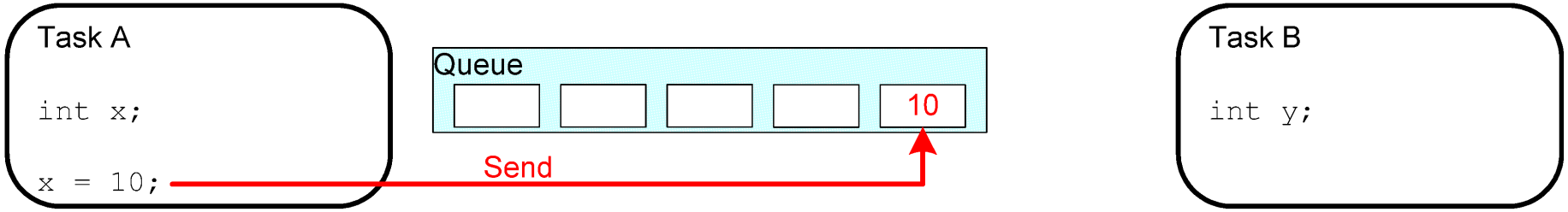
https://github.com/ucb-ee192/SkeletonMCUX/tree/master/frdmk64f_skeleton/main.c

Queue in FreeRTOS

161204 Pre-release for FreeRTOS V8.x.x. See http://www.FreeRTOS.org/FreeRTOS-V9.html for information about FreeRTOS V9.x.x. Use http://www.FreeRTOS.org/contact to provide feedback, corrections, and check for updates.



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

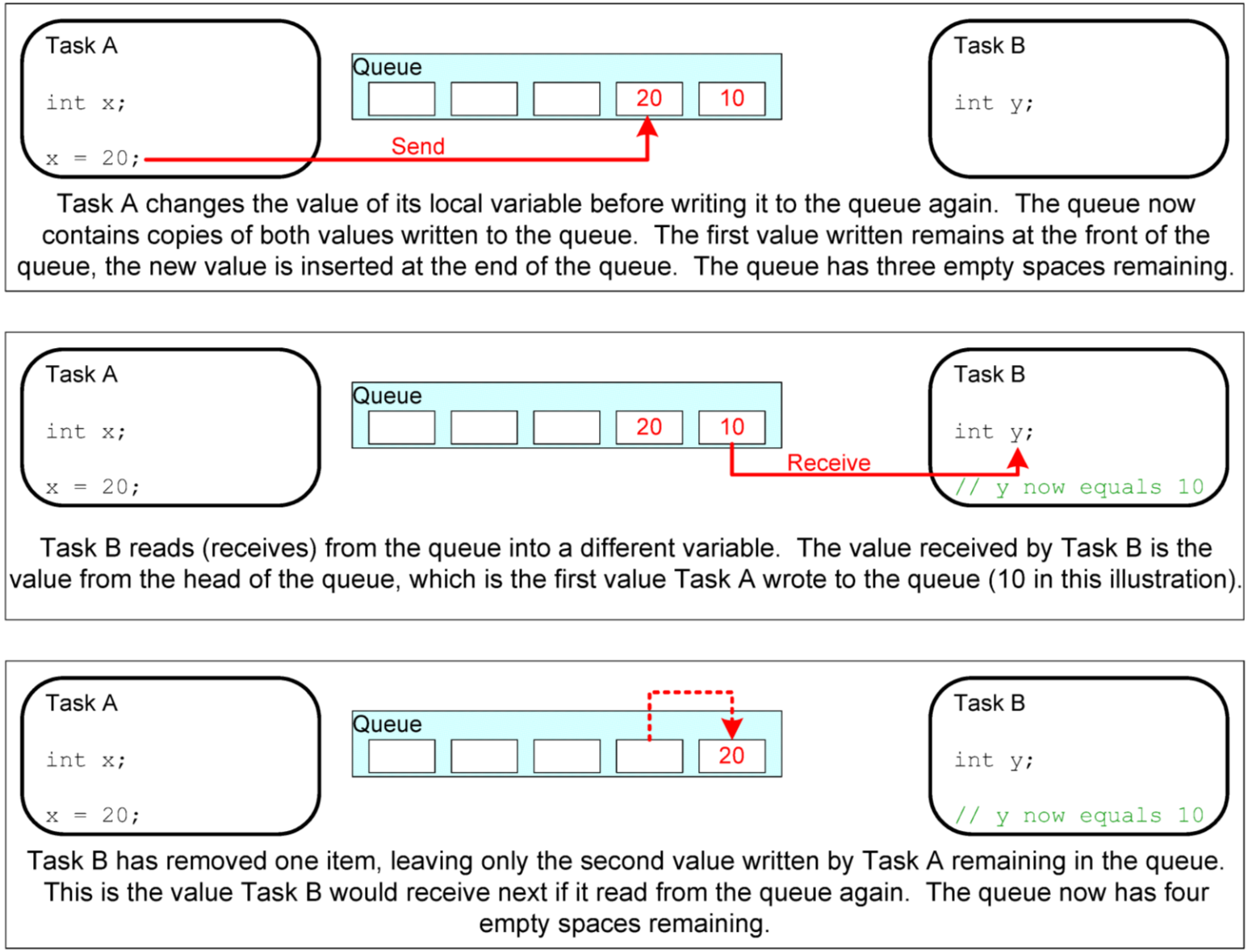


Figure 31. An example sequence of writes to, and reads from a queue

Non-blocking print

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

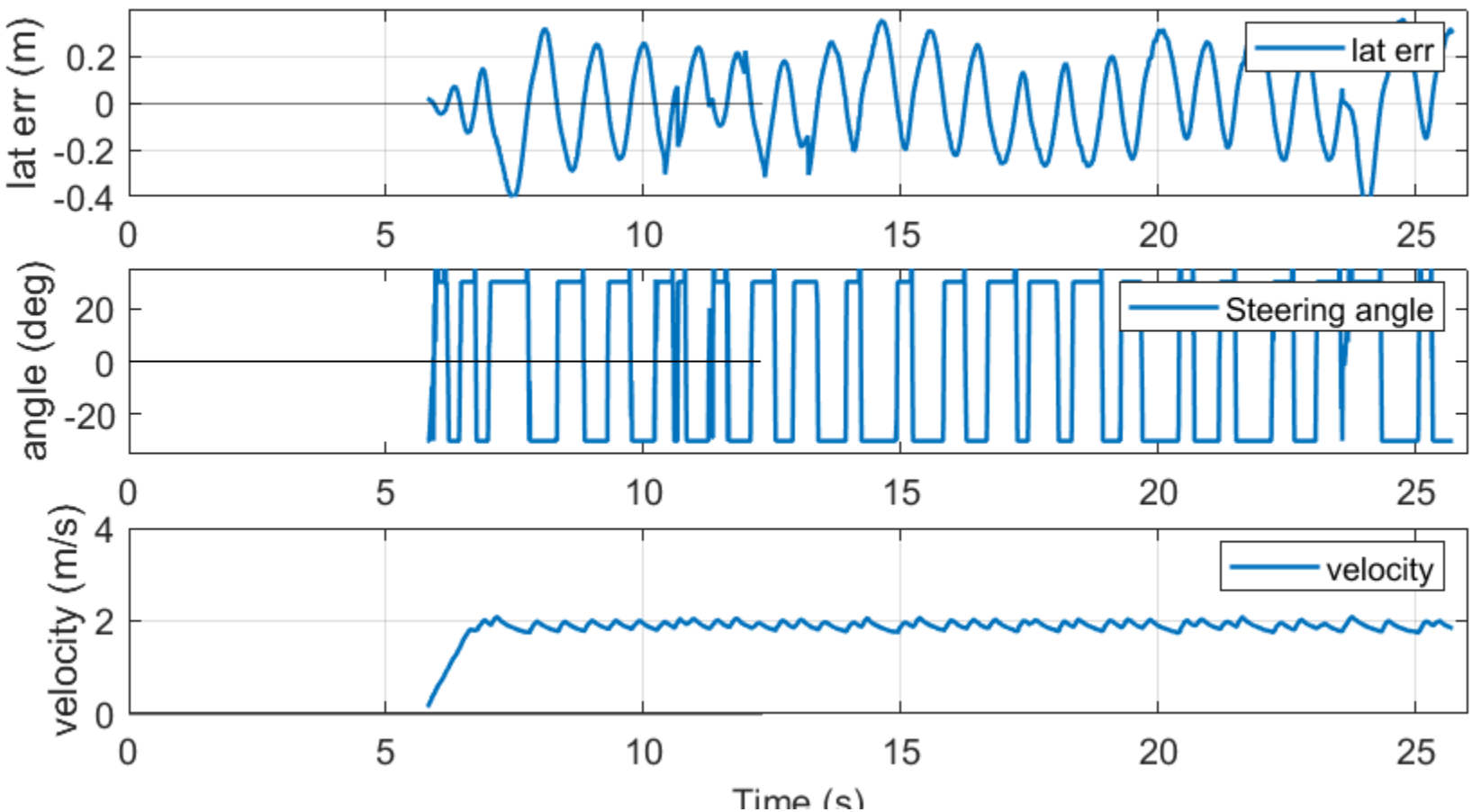
```
sprintf(log,"Idle. sum of cos = %d \n\r",
                                (long)ZSum);
log_add(log);

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);
```

Extra Slides

Proportional control.
Kp = 20 deg/cm, 35 rad/m
V=2.2 m/s
NOTE: = bang-bang!



Proportional + derivative control.
Kp = 20 deg/cm, 35 rad/m
Kd = 300 deg/(m/sec)
V=2.2 m/s
NOTE: = bang-bang!