Announcements
1. 3/3 Quiz 3: TSL1401 line camera
2. CalDay Sat. April 18 10 am @ UCB
3. HW 1 due 3/3
Topics

• Upcoming checkpoints
• Q2 Solution
• Velocity Control (recap)
• HW1 Track Detection (recap)
• Steering control (intro)
• Telemetry logging
Upcoming Checkpoints

2/21 C4: easy, work ahead!
C4.2: Line camera image capture with exposure control.
C4.4.4 Line camera calibration: measure track lateral displacement in mm
HW 1 line detection (due 3/3)

2/28 C5: may be harder, mounting, prototyping velocity sensor, writing control code
C5.3: BBBL, motor driver, velocity sensor mounted to car
C5.4: Basic track detection and wheels turn toward track (benchtop)
C5.5: basic velocity sensor, estimation and benchtop control: 3 m/s.

3/6 C6.3: The vehicle must complete the figure-8 course completely autonomously in under 3 minutes.
C6.3.4: running with velocity control
Topics

• Upcoming checkpoints
• Q2 Solution
• Velocity Control (recap)
• HW1 Track Detection (recap)
• Steering control (intro)
• Telemetry logging
Topics

- Upcoming checkpoints
- Q2 Solution
- Velocity Control (recap)
- HW1 Track Detection (recap)
- Steering control (intro)
- Telemetry logging
Quadrature Encoder

![Quadrature Encoder Diagram](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.

https://codeforfree.weebly.com/uploads/1/0/1/6/10160088/5858962_orig.png
Sharp GPS260

Fig. 9 Test Circuit for Response Time

- Choose current 4 mA in LED
- Vcc = 3.3 V
- May want regulated/clean voltage for Vcc

Fig. 13 Detecting Position Characteristics (2)

100 us response time
Beagle Bone Blue Quad Encoder

int rc_encoder_read (int ch)

Returns
   The current position (signed 32-bit integer)
   or -1 and prints an error message is there is a problem.
Ch 1-3 are available
Examples:
   rc_test_encoders.c.
Velocity Sensing

- On board: estimating $\Delta x/\Delta T$

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

On board…
Proportional control:
\[ U = kp \cdot e = kp \cdot (r - y); \]
Here: \( r \) is desired velocity, \( U \) is PWM %

Proportional + integral control
\[ U = kp \cdot e + ki \cdot e_{\text{sum}}; \]
\[ e_{\text{sum}} = e_{\text{sum}} + e; \]
Topics

• Upcoming checkpoints
• Q2 Solution
• Velocity Control (recap)
• HW1 Track Detection (recap)
• Steering control (intro)
• Telemetry logging
Python Template for Track Finding

# 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 \{10,100\}.
    Represents whether or not each frame contains a detected line.

# cross_found_list - A length n array of Booleans \{10,100\}.
    Represents whether or not each frame contains a crossing.
Possible algorithms for line detection

e.g. scipy.signal.filter. Many options. Here 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
Topics

• Upcoming checkpoints
• Q2 Solution
• Velocity Control (recap)
• HW1 Track Detection (recap)
• Steering control (intro)
• Telemetry logging
Steering Control overview

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

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

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

Proportional+derivative

P+I+D

Note steady state error:
car follows larger radius
Steering Control - PD

Example under-damped steering:
Bicycle Steering Model

Equations On board
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:

$$\lambda_{1,2} = \frac{V}{2} \left( -k_p \pm \sqrt{k_p^2 - \frac{4k_p}{L}} \right)$$
**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: \( \lambda_1 = \lambda_2 \rightarrow \)

\( k_p^2 = 4 \frac{k_p}{L} \) or \( k_p = 4/L = 4/0.3 \text{ m} = 13 \text{ rad/m} = 760 \text{ deg/m} \)

At 2 m/s, doesn’t work well- servo saturates, also simulation dynamics…

- 2 m/s \( k_p = 800 \text{ deg/m} \) \( \text{Kd} = 0 \)

- 1 m/s \( k_p = 800 \text{ deg/m} \) \( \text{Kd} = 0 \)
Step: 15 cm
Choose step response 1m = 300 ms
Then lateral velocity = 15 cm/300 ms = 0.5 m/sec
At mid point:
δ = 0 = kp 7.5 cm + kd 0.5 m/sec ➔ kd ~ [0.15 sec] kp
Step: 15 cm
Choose step response 1m = 300 ms
Then lateral velocity = 15 cm/300 ms = 0.5 m/sec
Proportional + Integral

On board Anti-windup
Feedforward

On board
Topics

- Upcoming checkpoints
- Q2 Solution
- Velocity Control (recap)
- HW1 Track Detection (recap)
- Steering control (intro)
- Telemetry logging
while(rc_get_state()! =EXITING)
{
    // just data for csv format
    current_time = rc_nanos_since_boot() - start_time;
    old_tick = ticks;
    fprintf(logfile, "%ld, ", old_tick); // pass value which not changing by other process
    current_time_f = ((double) current_time) / 1e6; // milliseconds
    run_time_f = ((double) run_time) / 1000.0; // us

    fprintf(logfile, "%8.3lf, %8.3lf, ", current_time_f, run_time_f);
    fprintf(logfile, "%" PRIu64 "", current_time);
    fprintf(logfile, "%" PRIu64 "\n", run_time);

    end_time = rc_nanos_since_boot() - start_time;
    run_time = end_time - current_time;
    while(old_tick == ticks)
    {
        rc_usleep(100); // sleep 100 us
    }
}

run_time: min 20 us, typical 30-50 us, max 6600 us
Software Notes

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

Threads are asynchronous wrt interrupt!

rc_pthread_set_processniceness() ?

Interrupt-
highest
priority (?)
ticks++;

thread telem_loop()
thread printf_loop()
thread setpoint_manager()

mpu/dmp interrupt
_balance_controller()

interrupt
_balance_controller()

Interrupt-
highest
priority (?)
ticks++;

thread telem_loop()
thread printf_loop()
When new data is ready in the buffer, the IMU sends an interrupt to the BeagleBone triggering the buffer read followed by the execution of a function of your choosing set with the `rc_mpu_set_dmp_callback()` function.

```c
// set up mpu configuration
rc_mpu_config_t mpu_config = rc_mpu_default_config();
mpu_config.dmp_sample_rate = SAMPLE_RATE_HZ;

// start mpu
if(rc_mpu_initialize_dmp(&mpu_data, mpu_config))

// this should be the last step in initialization
// to make sure other setup functions don't interfere
rc_mpu_set_dmp_callback(&__balance_controller);

// idle while sensing and control done elsewhere
while(rc_get_state()!=EXITING){
    rc_usleep(200000);
}
static void __balance_controller (void)
{
ticks++;
    end_time = rc_nanos_since_boot ();
    run_time = end_time - start_time;
    // time since previous interrupt

    /******************************************************************
    * STATE_ESTIMATION
    * read sensors and compute the state
    ******************************************************************/
    cstate.wheelAngleL =
        (rc_encoder_eqep_read(ENCODER_CHANNEL_L) * 2.0 * M_PI) /
        (ENCODER_POLARITY_L * GEARBOX * ENCODER_RES);

    /******************************************************************
    * Send signal to motors
    ******************************************************************/
    dutyL = cstate.d1_u - cstate.d3_u;
    rc_motor_set(MOTOR_CHANNEL_L, MOTOR_POLARITY_L * dutyL);
}

int main(int argc, char *argv[])
{
    int c;
    pthread_t setpoint_thread = 0;
    pthread_t printf_thread = 0;
    pthread_t telem_thread = 0;

    // print thread to print to screen without blocking main
    rc_pthread_create(&printf_thread, __printf_loop, (void*) NULL, SCHED_OTHER, 0);

    // start balance stack to control setpoints
    rc_pthread_create(&setpoint_thread, __setpoint_manager, (void*) NULL, SCHED_OTHER, 0);

    // telemetry thread to log to file
    rc_pthread_create(&telem_thread, telem_loop, (void*) NULL, SCHED_OTHER, 0);
    // telem loop could write to file
Example logging thread

```c
// telemetry thread to log to file
void* telem_loop(__attribute__((unused)) void* ptr)
{
    long old_tick=0; uint64_t initial_time, current_time;
    printf("telem thread\n"); fflush(stdout); // empty buffer
    initial_time = rc_nanos_since_boot();
    while(rc_get_state() != EXITING)
    {
        current_time = rc_nanos_since_boot();
        old_tick = ticks;
        fprintf(logfile, "%ld, ", old_tick); // pass value
        fprintf(logfile, "%10.3f, ",
                   (double)(current_time-initial_time)/1e6);
        fprintf(logfile,"%8.3f, ", cstate.yaw);
        fprintf(logfile,"%8.3f, %8.3f",
                   cstate.dutyL, cstate.dutyR);
        fprintf(logfile, "%8.3f\n", cstate.vBatt);
        while(old_tick == ticks)
        {
            rc_usleep(100); // sleep 100 us
        }
    }
    rc_usleep(1000000 / PRINTF_HZ);
    return NULL;
}
```
Debian Processes/Delay

```bash
htop
# systemctl disable avahi-daemon
# systemctl stop avahi-daemon

sudo kill -9 {avahi-daemon, rc_battery_monitor, apache2}.
```
Extra Slides
Automatic Gain Control

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

In all the discussion that follows, we will be using one-shot imaging.
Alternative #1 frame subtraction

TSL 1401 line sensor 8 bit

Frame 0

Frame 1

Frame 2

Frame 1-Frame 0

Notes: peak shows edge of track. Noisy, only 1 pixel resolution.
Alternative #2 Difference of Gaussians

Laplacian of Gaussian

\[ \Delta[G_\sigma(x, y) * f(x, y)] = [\Delta G_\sigma(x, y)] * f(x, y) = \text{LoG} * f(x, y) \]

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

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

Notes: zero crossing is edge location
Alternative #3 Correlation

\[
\text{arg min } \| I(y) - f(y - \Delta y) \|_2
\]

\[\Delta y\]

Notes: normalize, find by least squares or search. Can use \(\Delta y(n-1)\) to initialize.
Proportional + derivative control.
$K_p = 40 \text{ deg/cm}, 70 \text{ rad/m}$
$K_d = 1000 \text{ deg/(m/sec)}$
$V=3 \text{ m/s}, \text{slew rate} 600 \text{ deg/0.16 sec}$

NOTE: = bang-bang!
What is problem with bang bang?
Break servo, nonlinear (unstable)
Proportional + derivative control.
\( K_p = 200 \text{ deg/m}, \)
\( K_d = 30 \text{ deg/(m/sec)} = (0.15 \text{ sec}) K_p \)
\( V = 3 \text{ m/s}, \) slew rate 600 deg/0.16 sec

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)
$K_p = 200 \, \text{deg/m}, \; K_d = 30 \, \text{deg/(m/sec)}. \; V=3 \, \text{m/s}$

Slew 600 deg/160 ms

Slew 60 deg/160 ms