EECS 473 Midterm Exam

Fall 2026

Name: _____ unique name: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

NOTES:

1. Closed book and Closed notes other than the provided handout.
2. There are **13** pages total for the exam as well as handouts which you will need for the last question.
3. Calculators are allowed, but no PDAs, Portables, Cell phones, etc. Using a calculator to store notes is not allowed nor is a calculator with any type of wireless capability.
4. You have about 120 minutes for the exam.
5. Though the last question is worth significantly less than half the points, we expect it will take at least half of the exam time.
6. If you use the last (blank) page for any answers you must *clearly* indicate that on the page where your answer is expected.

Be sure to show work and explain what you've done when asked to do so. That may be very significant in the grading of this exam.

1) **<u>Fill in</u> the square in front of all the true statements.**
[10 points, -2 per wrong response, minimum 0]

☐ An LDO is a type of switching regulator. **[FALSE]**

☐ A fully-charged 2000mAh LiPo battery can be expected to last 60 minutes or longer when drained at a rate of 4A. **[FALSE]**

☐ On a PCB, a 10 mil-wide trace has more resistance than a 30 mil-wide trace of twice the length. **[TRUE]**

☐ An LKM implementing a character device may return less than the number of bytes requested by a read() call. **[TRUE]**

☐ In the context of Linux device drivers done in class, the return value of a "read" function indicates how many characters were copied by the driver into the caller's buffer. **[TRUE]**

☐ The primary reason for using a Mutex is signaling. For example, Task A (or an ISR) might properly use a Mutex to tell Task B that an event has happened. **[FALSE]**

☐ If an LCD library hides the details of the LCD's registers from the programmer, the library cannot be using bit banging.
[FALSE]

☐ A function in an LKM can call other functions in that LKM using the normal C function-call mechanism.
[TRUE]

☐ An ideal linear regulator with a 6V, 2A input and 4V output will likely waste more power than it will supply. **[FALSE]**

☐ FreeRTOS uses round-robin scheduling if tasks have the same priority. **[TRUE*]**

2) **Scheduling** [12 points]

Say you have the following groups of tasks. For each group find the CPU utilization and identify which groups are RM and which are EDF schedulable. Indicate if you needed to do the critical instant analysis. *If needed, **clearly** show that analysis.* The following equation may prove useful. $\sum_{i=1}^{n} U \leq n(2^{1/n}-1)$

Group	T1 Execution Time	T1 Period	T2 Execution Time	T2 Period	T3 Execution Time	T3 Period	% Utilization
A	1	2	2	6	2	7	111.9%
B	1	4	2	7	1	3	86.9%
C	2	6	2	8	3	10	88.3%
D	2	4	2	7			78.6%

Group	EDF Schedulable?	RM Schedulable?	Did you need to examine the critical instance?
A	No	No	No
B	Yes	Yes	Yes
C	Yes	No	Yes
D	Yes	Yes	No

B: Longest period task finishes on time and nothing has failed until then, so it works.

0-1	1-2	2-3	3-4	4-5	5-6	6-7
T3	T1	T2	T3	T1	T2←met	

C: T3 hasn't run 3 times by time 10.

0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	~~10-11~~
T1	T1	T2	T2	T3	T3	T1	T1	T2	T2	~~T3~~

3) **On Schedule** [6 points]
Under **non-preemptive** rate-monotonic scheduling, is it possible for a task set with arbitrarily low CPU utilization to miss a deadline? Clearly justify your answer. **[6]**

YES. Let a high-priority task have runtime=ε, period=1, and a low-priority task have runtime=2, period=M. At t=0 the high-priority job completes, then the low-priority job starts and cannot be preempted. The next high-priority release at t=1 misses its deadline at t=2.

Utilization ε + 2/M can be made arbitrarily small.

4) **Power to the Processors!** [8 points]
Say we have a 8V 200mAh LIPO battery. We are using it through an LDO which outputs 5V and has a quiescent current of 1mA. You determine that the load draws a constant current and that the battery lasts 5 hours under that constant load.

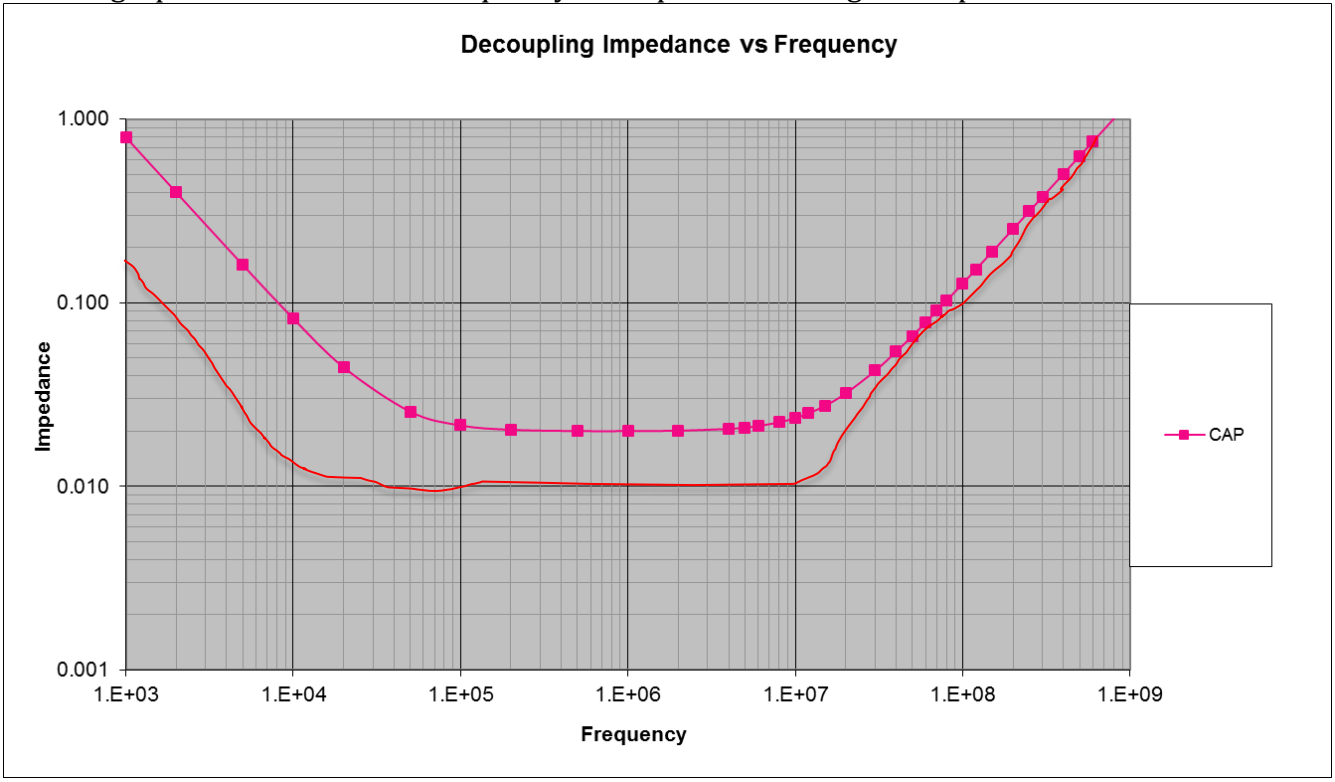
a. What is the effective resistance of load? Show your work. **[4]**
I_bat = 200 mAh / 5 h = 40 mA; I_load = 40 - 1 = 39 mA.
R = V/I = 5/0.039 = 128.2 ohms.

b. What percent of the battery's capacity is not supplied to the load (so wasted by the LDO in some way)? You may assume that the battery keeps its voltage constant at 8V the whole time. Show your work. **[4]**

Input energy: (8 V)(0.040 A)(5 h) = 1.60 Wh. Load receives (5 V)(0.039 A)(5 h) = 0.975 Wh.
Not delivered to load: (1.60 - 0.975)/1.60 ~= 39%.

5) **Decoupling capacitors** [7 points]

The graph below shows the frequency vs. impedance for a given capacitor. Redraw the



graph showing the same information but if instead we used **10** new capacitors (in parallel), *each* of which had half the capacitance, but 5 times the resistance and 10 times the inductance as the original.

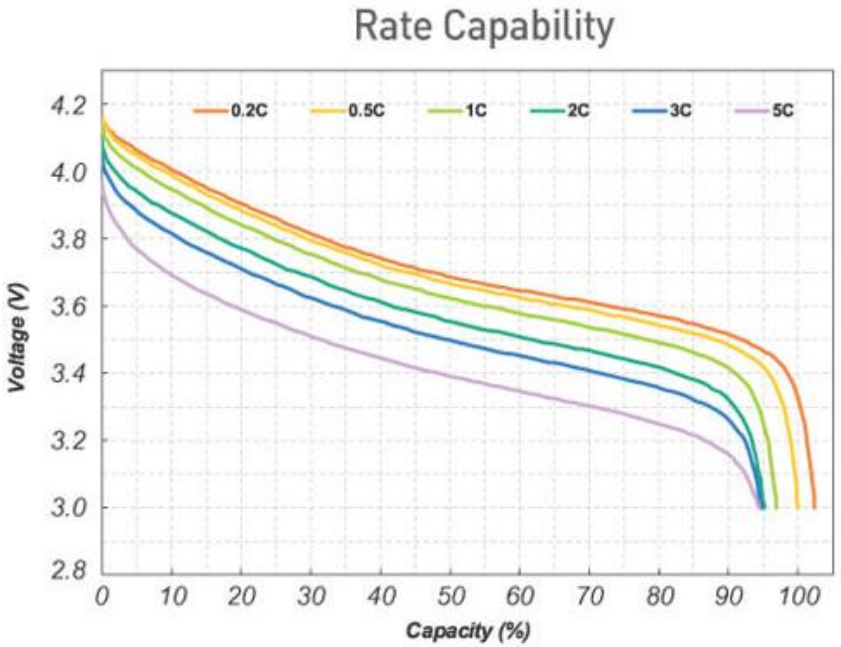
Comments:

Ceq = 10(C/2) = 5C; ESReq = (5R)/10 = 0.5R; ESLeq = (10L)/10 = L.

Red curve: should be 1/5 of the old line where C dominates, ½ where R dominates and the same where the inductance (L) dominates. Line is really rough…

6) **Batteries** [11 points]

Answer the following questions about a 2000mAh battery which has the discharge characteristics seen below. You only need to briefly show your work unless otherwise directed.



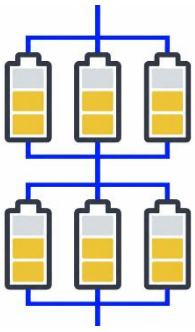
a) *About* how long would the battery be able to maintain at least 3.5V output if the load was using 4A? **[3]**

4 A = 2C. At 3.5 V, about 60% of capacity is used: (0.60)(2 Ah)/(4 A) = 0.30 h ~ 18 min.

b) As above, but if the load was using 8A? **[3]**

8 A = 4C (between 3C and 5C). About 40% at 3.5 V: (0.40)(2 Ah)/(8 A) = 0.10 h ~ 6 min.

c) Say you need 7.2V with the load using 12 A. You have six of the batteries in a 2S3P scheme (2 series, 3 parallel, as shown to the right). How long will the battery last? Clearly explain your answer. **[5]**



Six cells in 2S3P: 12 A / 3 = 4 A per cell (2C); 7.2 V pack requires each cell >= 3.6 V. At 2C, the graph reaches 3.6 V after *roughly* 42% discharged, so runtime ~ (0.42*(6 Ah)/(12 A) ~= 12 min. Two series cells double voltage but do not double Ah per string.

7) **Reading in an LKM** [10 points]

The following is the memory_read function from Lab 4 (partC.c) after some error checking and comments were removed.

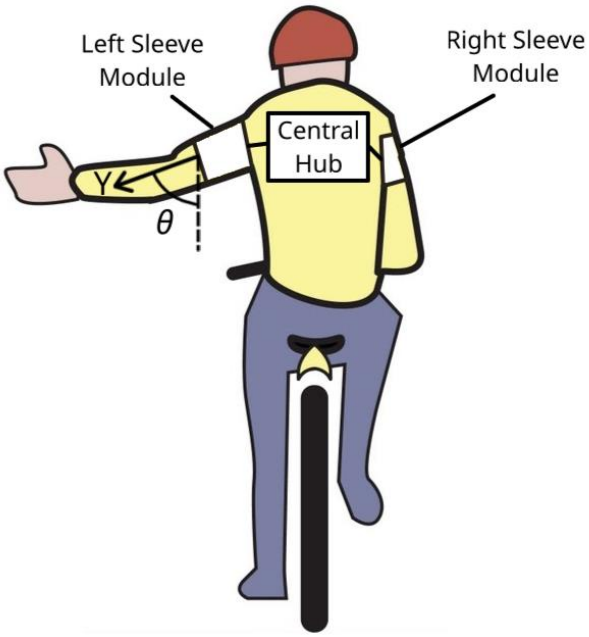
```
ssize_t memory_read (struct file * filp, char *buf, size_t count,
                     loff_t * f_pos)
{
  if (*f_pos == 0)
    {
      copy_to_user (buf, memory_buffer, 1);
      *f_pos += 1;
      return 1;
    }
  else
      return 0;
 }
```

Complete a read function (named "fixed read") where the user can read the first 12 characters from the global string "s" contiguously across reads. Respect count and return 0 only at EOF once the 12 characters have been read.

```
const char s[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
ssize_t fixed_read (struct file * filp, char *buf, size_t count,
             loff_t * f_pos)
{
  if (*f_pos >= 12) return 0; // EOF
  size_t n = min(count,(12 - *f_pos));
  copy_to_user(buf, s + *f_pos, n))
  *f_pos += n;
  return n;
}
```

8) Bike Blinkers [36 points]

It is 8:54 AM on a foggy day. You are cycling downhill on Murfin, ready to turn left onto Hayward when you spot a BMW at the stop sign. You know you have the right of way, so you stick out your left arm to signal a turn. However, the BMW fails to notice and pulls out anyway, narrowly missing you. Enraged, you begin working on blinkers for cyclists that you can activate by lifting an arm. You will design the **central hub** mounted on your jacket, and **two sleeve modules**. The central hub has two **connectors** with wires going to the sleeve modules.



You have the following parts for the central hub:

- 1 Arduino Uno (supply voltage 7–12V)
- 1 12V battery pack
- 1 Logic level shifter
- Any passives you may need

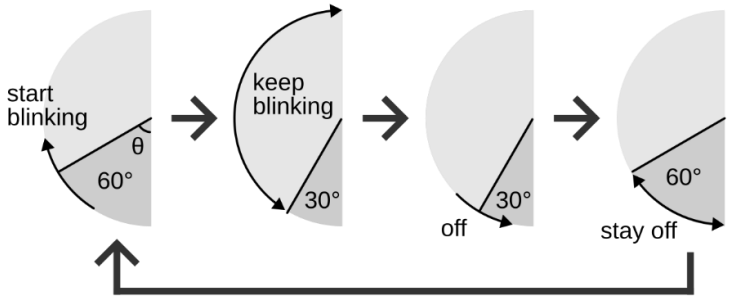
And the following for **each** sleeve module:

- 1 BMA400 IMU
- 1 LZ4-00A108 ultra-bright amber LED module
- 1 N-channel MOSFET (assume ideal)
- Any passives you may need

The MCU should poll both IMUs on the same I^2C bus and measure the angle between gravity and the nominal +Y axis of the IMU. Call this angle θ (theta). You can calculate θ by taking a sample (x, y, z) from the IMU, normalizing it to a unit vector, and then taking the arccosine of the normalized y value.

Once $\theta \geq 60°$, the LED on that side should start blinking by toggling about every 500 ms. Blinking should continue as long as $\theta \geq 30°$, to allow the arm to relax a bit. When $\theta < 30°$, blinking should stop, and should stay off until $\theta \geq 60°$ again. If the LED is on, you are free to decide whether to shut off the LED immediately, or to let it finish its 500 ms.

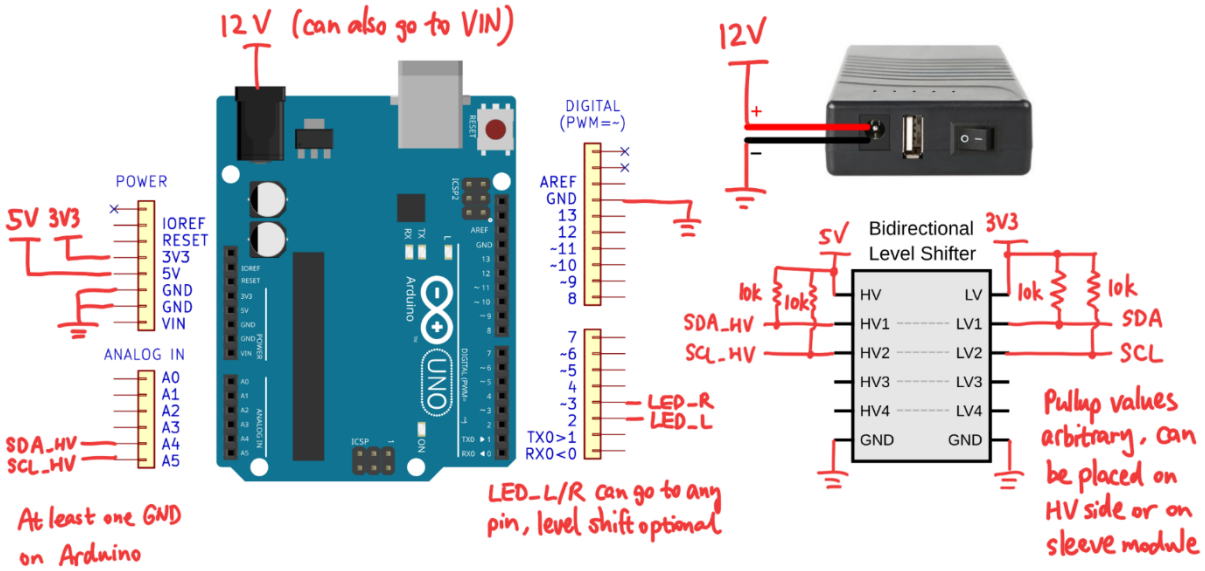
Note: For this problem, assume the BMA400 will accept a STOP between sending the register address and requesting the data. You do not need to use a repeated START. Further, assume both BMA400s are already configured to continuously measure acceleration using their default measurement ranges.



Part A: Wiring

First, please wire the central hub. **Wires that connect to the sleeve modules must be labeled.** Draw passives (resistors, capacitors, inductors) where needed (be sure to include values). Keep in mind that you have two sleeve modules to connect to!

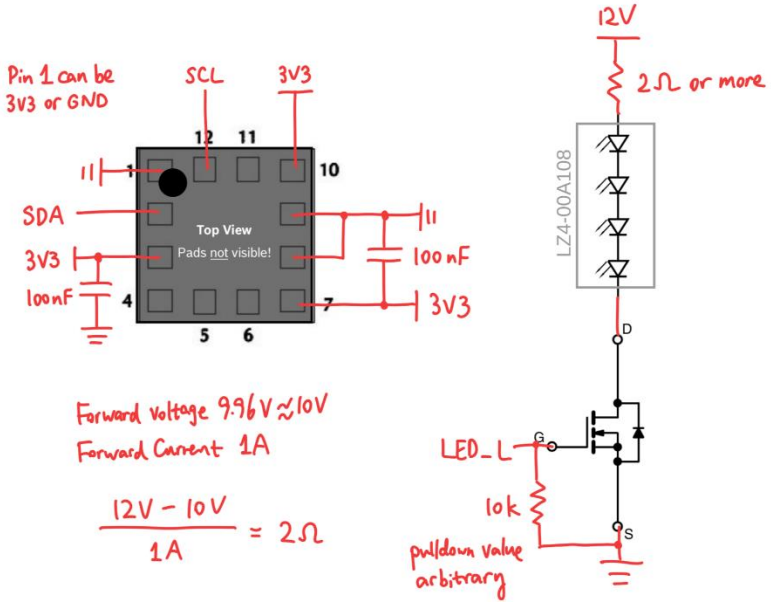
!



List out **all** labels that **need** to go between the central hub and the **left** sleeve module:

GND, SDA, SCL, 12V, 3V3, LED_L

Finally, please complete the **left** sleeve module, using the labels above where applicable.



Note: for LED resistor, absolute minimum is (12V – 11.6V) / 1A = 0.4Ω, typical is (12V – 9.96V) / 700mA ≈ 3Ω, max is (12V – 8.96V) / 700mA ≈ 4.3Ω

Part B: Short Answers [3 points]

1. How will the wiring for the right sleeve module be different from the left?
   Pin 1 (SDO) will be pulled high (or low, if left side pulled it high) to select another I2C address.

2. Pull out the BMA400 datasheet.
   a. Look at page 52, ACC_X_LSB/MSB. What is the minimum and maximum **signed** integer value you can get from the registers?
      –2048, 2047
   b. Look at page 64, ACC_CONFIG1. What is the default accelerometer measurement range in ±g's?
      ±4g

Part C: IMU, but better [9 points]

Please complete the following C++ class for the IMUs. **Notice this part spans two pages.**

Please complete the following C++ class for the IMUs.

```
#include <Arduino.h>
#include <math.h>
#include <Wire.h>

// globals, defines, etc.
#define ACC_BASE 0x04

class IMU {
private:
      uint8_t i2cAddr;  // 7-bit address

      // read `count` bytes into buf, no error check needed
      void readReg(uint8_t regAddr, uint8_t *buf, uint8_t count) {
            Wire.beginTransmission(i2cAddr);
            Wire.write(regAddr);
            Wire.endTransmission();
            Wire.requestFrom(i2cAddr, count);
            while (Wire.available()) {
                  *buf++ = Wire.read();
            }



      }
```

```
	// convert raw uint16 regVal into amount of g's (positive or negative)
	float uintToG(uint16_t regVal) {
		int16_t signed = (int16_t)regVal;
		if (signed > 2047) {
			signed -= 4096;
		}
		return (float)signed * 4.0 / 2048.0;
	}

	// scale x, y, z in place such that (x² + y² + z²) equals 1.0
	void normalize(float *x, float *y, float *z) {
		float magnitude = sqrtf((*x)*(*x) + (*y)*(*y) + (*z)*(*z));
		*x /= magnitude; *y /= magnitude; *z /= magnitude;
	}

public:
	// init is given to you
	void init(uint8_t addr) { i2cAddr = addr; }

	// return theta in degrees
	float readTheta() {
		uint16_t buf[6];
		readReg(ACC_BASE, buf, 6);
		uint16_t ux = buf[0] + buf[1] << 8,
			 uy = buf[2] + buf[3] << 8,
			 uz = buf[4] + buf[5] << 8;
		float x = uintToG(ux),
		      y = uintToG(uy),
		      z = uintToG(uz);
		normalize(&x, &y, &z);
		return acosf(y) / M_PI * 180;
	}
};
```

Part D: Setup your hub [7 points]

Write the setup function to initialize **all** your peripherals.

```
#include <Arduino.h>
#include “IMU.h”

// globals, defines, etc.
#define LED_L 2
#define LED_R 3

IMU imuL, imuR;
bool leftBlinking = false;
long leftLaskBlink = 0; // ms
bool leftLedOn = false;

void setup() {
      imuL.init(0b0010100);
      imuR.init(0b0010101); // LSb must be different
      pinMode(LED_L, OUTPUT);
      pinMode(LED_R, OUTPUT);
      digitalWrite(LED_L, LOW);
      digitalWrite(LED_R, LOW);
      Wire.begin();
}
```

Part E: Loopsided [8 points]

Because the two sides are identical in logic, we have taken apart the loop into separate functions. You will implement loopLeftSide (you can assume loopRightSide will be an exact copy, just with different variable names). You may find millis() helpful. Don't worry about wasting power. You still have access to all headers, globals, and defines from part D. You don't have to (but can) define variables for loopRightSide.

```
void loopLeftSide() {
	long now = millis();
	float theta = imuL.readTheta();

	if (!leftBlinking && theta >= 60) {
		leftBlinking = true;
		digitalWrite(LED_L, HIGH);
		leftLedOn = true;
		leftLastBlink = now;
	} else if (leftBlinking && theta < 30) {
		leftBlinking = false;
		digitalWrite(LED_L, LOW);
		leftLedOn = false;
	}

	if (leftBlinking && (now - leftLastBlink >= 500)) {
		leftLedOn = !leftLedOn;
		digitalWrite(LED_L, leftLedOn ? HIGH : LOW);
		leftLastBlink = now;
	}
}

void loopRightSide() {
	// copy and paste, find and replace
	// repetition is ok when you have space
}

void loop() {
	loopLeftSide();
	loopRightSide();
	delay(10);
}
```