Last week's progress/Current State of Project

Last week, we discussed and made changes to our parts list based on different mechanical and electrical. We ordered all of our parts, and some of them have arrived. Currently, we have the IMU, the battery/battery holder/charger, the boost converter modules, and the motor driver. Headers were soldered to our components where necessary.

We've been trying to get I2C working/read measurements from the IMU; we've written I2C functions but haven't gotten them fully working, and we are in the process of debugging. Code has been pushed to repo (IMU.c). Our main goal right now is to get this working. We've tested the battery with the boost converter modules and have successfully used them to power the ATMega, as well as the motor driver and motors. Since our motors haven't arrived, we instead used the 5V DC motors available in Detkin. We've successfully written code that allows us to control the direction/speed of both motors so that we can quickly/easily get everything working once our actual parts arrive. The code is in our repo.

Next week's plan

Resolve I2C issues and be able to read measurements from IMU.

Once the motors/brackets and wheels arrive, assemble them and try powering them with the battery and boost converter modules. Then, finalize wiring layout amd solder parts on.

Start working on PID control theory and code while other progress is being made.

Proof of Work
IMU Code Work:

```
// IMU Driver for ATmega328 using I2C - Adapted for LSM6DS0
// Author: COD
// Platform: AVR ATmega328

#include <avr/io.h>
#include "i2c.h"
#include "uart.h"

#define IMU_ADDR 0x6B // Default I2C address for LSM6DS0
#define IMU_REG_FIFO_CTRL4 0x0A
#define IMU_REG_CTRL1_XL 0x10
#define IMU_REG_CTRL2_G  0x11
#define IMU_REG_CTRL3_C 0x12
#define IMU_REG_CTRL4_C 0x13
#define IMU_REG_CTRL5_C 0x14
#define IMU_REG_CTRL6_C 0x15
```

```
#define IMU_REG_CTRL7_G 0x16
#define IMU_REG_CTRL8_XL 0x17
#define IMU_REG_CTRL9_XL 0x18

#define IMU_REG_STATUS_REG 0x1E //temp g xl [2:0]
#define IMU_REG_OUTX_G   0x22
#define IMU_REG_OUTY_G   0x24
#define IMU_REG_OUTZ_G   0x26

#define IMU_REG_OUTX_L_XL 0x28

volatile uint8_t imu_addr;

void IMU_init(uint8_t addr) {
    imu_addr = addr;

    I2C_writeRegister(imu_addr, 0x00, IMU_REG_FIFO_CTRL4);

    // Accelerometer: 119 Hz, ±2g, 50 Hz bandwidth
    I2C_writeRegister(imu_addr, 0x40, IMU_REG_CTRL1_XL); //0100 00 0 0
104Hz 2g low-res
    // Gyroscope: 119 Hz, 245 dps
    I2C_writeRegister(imu_addr, 0x40, IMU_REG_CTRL2_G); //0100 00 0 0
    I2C_writeRegister(imu_addr, 0x40, IMU_REG_CTRL3_C);
    I2C_writeRegister(imu_addr, 0x00, IMU_REG_CTRL4_C);
    I2C_writeRegister(imu_addr, 0x00, IMU_REG_CTRL5_C);
    I2C_writeRegister(imu_addr, 0x00, IMU_REG_CTRL6_C);
    I2C_writeRegister(imu_addr, 0x00, IMU_REG_CTRL7_G);

    I2C_writeRegister(imu_addr, 0x00, IMU_REG_CTRL8_XL);
    I2C_writeRegister(imu_addr, 0xE0, IMU_REG_CTRL9_XL); //I3C Disable

}

void IMU_getAll(int16_t *accel, int16_t *gyro, int16_t *temp) {
    uint8_t data[14];

    // Read Accel (6 bytes), Temp (2 bytes), Gyro (6 bytes)
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTX_L_XL, 14);
```

```
    for (int i = 0; i < 3; i++) {
        accel[i] = (int16_t)(data[i*2+1] << 8 | data[i*2]);
        gyro[i]  = (int16_t)(data[i*2+9] << 8 | data[i*2+8]);
    }

    *temp = (int16_t)(data[7] << 8 | data[6]);
}

int16_t IMU_getXAcc() {
    uint8_t data[2];
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTX_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}

int16_t IMU_getYAcc() {
    uint8_t data[2];
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTY_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}

int16_t IMU_getZAcc() {
    uint8_t data[2];
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTZ_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}

int16_t IMU_getXGyro() {
    uint8_t data[2];
    I2C_readRegister(imu_addr, data, IMU_REG_OUTX_G);
    I2C_readRegister(imu_addr, data + 1, IMU_REG_OUTX_G);

    //I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTX_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}

int16_t IMU_getYGyro() {
    uint8_t data[2];
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTY_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}
```

```
int16_t IMU_getZGyro() {
    uint8_t data[2];
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTZ_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}

int16_t IMU_getTemp() {
    uint8_t data[2];
    I2C_readCompleteStream(data, imu_addr, IMU_REG_OUTX_G, 2);
    return (int16_t)(data[1] << 8 | data[0]);
}

int IMU_checkNewData() {
    // For simplicity, return 1; LSM6DS0 supports status register at 0x1E
    return 1;
}
```

Motor Test Code:

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "uart.h"

// Motor A Pins (Timer0)
#define L_MOTOR_DIR PD3 // High is forward, low is backward
#define L_MOTOR_PWM PD6 // OC0A
#define SLEEP PD4       // Sleep control (write high to enable motor
driver)

// Motor B Pins (Timer0)
#define R_MOTOR_DIR PD2
#define R_MOTOR_PWM PD5 // OC0B

// Bench joystock debugging
#define JOY PC3
```

```
void motor_init() {
    // Set PWM outputs and direction pins as outputs
    DDRD |= (1 << R_MOTOR_PWM) | (1 << L_MOTOR_PWM) | (1 << R_MOTOR_DIR) |
(1 << L_MOTOR_DIR) | (1 << SLEEP);

    // Wake up DRV8833
    PORTD |= (1 << SLEEP);

    // Setup Timer0 for Fast PWM
    TCCR0A |= (1 << COM0A1) | (1 << COM0B1); // Non-inverting for OC0A and
OC0B
    TCCR0A |= (1 << WGM01) | (1 << WGM00);   // Fast PWM mode
    TCCR0B |= (1 << CS01) | (1 << CS00);     // Prescaler 64

    // Enable Timer0 Compare Match A interrupt
    TIMSK0 |= (1 << OCIE0A);

    // NOT USED ON ACTUAL PROJECT JUST BENCH DEBUGGING
    // Joystick inputs as inputs with pull-ups
    DDRC &= ~(1 << JOY);
    ADMUX = (1 << REFS0) | (1 << MUX1) | (1 << MUX0); // Set Vref = AVcc,
Select ADC3 for PC3 (default)
    ADCSRA = (1 << ADEN)  | // Enable ADC
    (1 << ADATE) | // Auto-trigger (Free running mode)
    (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Set prescaler to 128
(16MHz / 128 = 125kHz)

     ADCSRB = 0; // Free running mode (default)

     ADCSRA |= (1 << ADSC); // Start first conversion

    sei(); // Enable global interrupts
}

void driveMotor(int16_t lSpeed, int16_t rSpeed) {
    // Clamp speed values to +/-255
    if (lSpeed > 255) lSpeed = 255;
    if (lSpeed < -255) lSpeed = -255;
    if (rSpeed > 255) rSpeed = 255;
    if (rSpeed < -255) rSpeed = -255;
```

```
    if (lSpeed >= 0) {
        PORTD &= ~(1 << L_MOTOR_DIR);
        OCR0A = lSpeed;
    } else {
        PORTD |= (1 << L_MOTOR_DIR);
        OCR0A = -lSpeed;
    }

    if (rSpeed >= 0) {
        PORTD &= ~(1 << R_MOTOR_DIR);
        OCR0B = rSpeed;
    } else {
        PORTD |= (1 << R_MOTOR_DIR);
        OCR0B = -rSpeed;
    }
}

uint16_t getJoy() {
    return ADC;
}

ISR(TIMER0_COMPA_vect) {
    uint16_t joy = getJoy();
    int16_t speed = 0;

    if (joy > 510) {
        // Map [500-900] ? [0-255]
        speed = (uint16_t)((joy - 500) * 255L / (1023 - 500));
    } else if (joy < 490) {
        // Map [180-500] ? [-255-0]
        speed = -((joy - 180) * 255L / (500 - 180));
    } else {
        speed = 0; // Dead zone
    }
    printf("JOY: %d | SPEED: %d\n", joy, speed);
    driveMotor(speed, -speed);
}

int main(void) {
```

```
    motor_init();
    uart_init();
    printf("Test");

    while (1) {}
}
```