~/Desktop/Fall 2023/ME 405/Lab0x05/imu_driver.py

```
'''
    @name                   imu_driver.py
    @brief                  driver class for the BNO055 IMU sensor from adafruit
    @author                 tanner, noah
    @date                   november, 2023
'''
from pyb import I2C
import controls

def combine_bytes(msb, lsb):
    combined_value = ( msb << 7 ) | lsb
    return combined_value

def remap(imu):
    imu.change_mode('CONFIG')
    axis_config_reg     = 0x41
    axis_sign_reg       = 0x42
    axis_remap_config   = 0x21
    axis_remap_sign     = 0x02

    imu.controller.mem_write(axis_remap_config, 0x28, axis_config_reg)
    imu.controller.mem_write(axis_remap_sign, 0x28, axis_sign_reg)
    imu.change_mode('IMU')

    print('axis remapped, mode changed to imu')

class bno055:

    def __init__(self, controller: I2C):
        self.controller         = controller
        self.imu_address        = 0x28
        # registers
        self.mode_reg           = 0x3D
        self.cal_reg            = 0x35

        self.acc_off_x_l        = 0x55
        self.acc_off_x_m        = 0x56
        self.acc_off_y_l        = 0x57
        self.acc_off_y_m        = 0x58
        self.acc_off_z_l        = 0x59
        self.acc_off_z_m        = 0x5A
        self.acc_offs_list      = [ self.acc_off_x_l,
                                    self.acc_off_x_m,
                                    self.acc_off_y_l,
                                    self.acc_off_y_m,
                                    self.acc_off_z_l,
                                    self.acc_off_z_m ]

        self.mag_off_x_l        = 0x5B
        self.mag_off_x_m        = 0x56C
        self.mag_off_y_l        = 0x5D
        self.mag_off_y_m        = 0x5E
        self.mag_off_z_l        = 0x5F
```

```
        self.mag_off_z_m        = 0x60
        self.mag_offs_list      = [ self.mag_off_x_l,
                                    self.mag_off_x_m,
                                    self.mag_off_y_l,
                                    self.mag_off_y_m,
                                    self.mag_off_z_l,
                                    self.mag_off_z_m ]

        self.gyr_off_x_l        = 0x61
        self.gyr_off_x_m        = 0x62
        self.gyr_off_y_l        = 0x63
        self.gyr_off_y_m        = 0x64
        self.gyr_off_z_l        = 0x65
        self.gyr_off_z_m        = 0x66
        self.gyr_offs_list      = [ self.gyr_off_x_l,
                                    self.gyr_off_x_m,
                                    self.gyr_off_y_l,
                                    self.gyr_off_y_m,
                                    self.gyr_off_z_l,
                                    self.gyr_off_z_m]

        self.acc_rad_l          = 0x67
        self.acc_rad_m          = 0x68
        self.mag_rad_l          = 0x69
        self.mag_rad_m          = 0x6A

        self.eul_pitch_l        = 0x1A
        self.eul_roll_l         = 0x1C
        self.eul_head_l         = 0x1E
        self.euler_meas_list    = [ self.eul_pitch_l,
                                    self.eul_roll_l,
                                    self.eul_head_l ]

        self.gyr_x_l            = 0x14
        self.gyr_x_m            = 0x15
        self.gyr_y_l            = 0x16
        self.gyr_y_m            = 0x17
        self.gyr_z_l            = 0x18
        self.gyr_z_m            = 0x19
        self.gyr_list           = [ self.gyr_x_l,
                                    self.gyr_y_l,
                                    self.gyr_z_l ]

        self.acc_rad_l          = 0x67
        self.acc_rad_m          = 0x68
        self.mag_rad_l          = 0x69
        self.mag_rad_m          = 0x6A
        self.rad_offs_list      = [ self.acc_rad_m,
                                    self.acc_rad_l,
                                    self.mag_rad_m,
                                    self.mag_rad_l, ]

    def change_mode(self, mode: str):
        self.mode = mode
        '''
            @name           change_mode
```

```
            @brief          method to change the operating mode of the IMU to one of the following fusion modes: X, Y, Z
        '''
        if self.mode == 'IMU':
            reg_value   = 0b1000
            self.controller.mem_write(reg_value, self.imu_address, self.mode_reg , timeout = 1000)
            print('Mode changed to IMU')
        elif self.mode == 'COMPASS':
            reg_value   = 0b1001
            self.controller.mem_write(reg_value, self.imu_address, self.mode_reg , timeout = 1000 )
            print('Mode changed to COMPASS')
        elif self.mode == 'M4G':
            reg_value   = 0b1010
            self.controller.mem_write(reg_value, self.imu_address, self.mode_reg , timeout = 1000 )
            print('Mode changed to M4G')
        elif self.mode == 'NDOF_FMC_OFF':
            reg_value   = 0b1011
            self.controller.mem_write(reg_value, self.imu_address, self.mode_reg , timeout = 1000 )
            print('Mode changed to NDOF_FMC_OFF')
        elif self.mode == 'NDOF':
            reg_value   = 0b1100
            self.controller.mem_write(reg_value, self.imu_address, self.mode_reg , timeout = 1000 )
            print('Mode changed to NDOF')
        elif self.mode == 'CONFIG':
            reg_value   = 0b0000
            self.controller.mem_write(reg_value, self.imu_address, self.mode_reg , timeout = 1000 )
            print('Mode changed to CONFIG')
        else:
            print('Invalid mode')

    def cal_status(self):

        '''
            @name           calibration_status
            @brief          retrieves calibration status from the imu and parse into individual statuses
        '''
        try:
            cal_status  = self.controller.mem_read(1, self.imu_address, self.cal_reg, addr_size=8)                                      # read cal_status, return a bytes object
            bits        = [ (cal_status[0] >> i ) & 1 for i in range(7, -1, -1) ]
            return bits

        except OSError as e:
            print(f'Error: {e}')

    def get_cal_coeff(self):
        '''
            @name           get_cal_coeff
            @brief          retrieves calibration coefficients from IMU as an array of packed binary data once cal status checks out
        '''
```

```
158         acc_off = [ 0, 0, 0, 0, 0, 0 ]
159         mag_off = [ 0, 0, 0, 0, 0, 0 ]
160         gyr_off = [ 0, 0, 0, 0, 0, 0 ]
161         rad_off = [ 0, 0, 0, 0 ]
162 
163         idx = 0
164         for reg in self.acc_offs_list:
165             acc_off[idx] = self.controller.mem_read(1, self.imu_address, reg)
166             idx += 1
167 
168         idx = 0
169         for reg in self.mag_offs_list:
170             mag_off[idx] = self.controller.mem_read(1, self.imu_address, reg)
171             idx += 1
172 
173         idx = 0
174         for reg in self.gyr_offs_list:
175             gyr_off[idx] = self.controller.mem_read(1, self.imu_address, reg)
176             idx += 1
177 
178         idx = 0
179         for reg in self.rad_offs_list:
180             rad_off[idx] = self.controller.mem_read(1, self.imu_address, reg)
181             idx += 1
182 
183 
184         return acc_off, mag_off, gyr_off, rad_off
185 
186     def write_cal_coeff(self, cal_vals):
187         '''
188             @name               write_cal_coeff
189             @brief              method to write calibration coefficients back to the
    IMU from pre-recorded packed binary data
190         '''
191         imu_obj.change_mode('CONFIG')
192         idx = 0
193         for reg in self.acc_offs_list:
194             val     = int(cal_vals[idx], 16)
195             buf = bytearray([val])
196             self.controller.mem_write(buf, self.imu_address, reg, timeout = 1000)
197             idx += 1
198 
199         for reg in self.mag_offs_list:
200             val     = int(cal_vals[idx], 16)
201             buf = bytearray([val])
202             self.controller.mem_write(val, self.imu_address, reg, timeout = 1000)
203             idx += 1
204 
205         for reg in self.gyr_offs_list:
206             val     = int(cal_vals[idx], 16)
207             buf = bytearray([val])
208             self.controller.mem_write(val, self.imu_address, reg, timeout = 1000)
209             idx += 1
210 
211         for reg in self.rad_offs_list:
212             val     = int(cal_vals[idx], 16)
```

```
            buf = bytearray([val])
            self.controller.mem_write(val, self.imu_address, reg, timeout = 1000)
            idx += 1

        imu_obj.change_mode('IMU')
        print('Calibration data written, mode changed to imu')

    def euler(self):
        '''
            @name           euler
            @brief          reads euler angles from IMU to use as measurements for feedback
        '''
        eul_meas_bytes = [ 0, 0, 0 ]

        idx = 0
        for reg in self.euler_meas_list:
            byte = self.controller.mem_read(2, self.imu_address, reg)
            eul_meas_bytes[idx] = ((byte[1] << 8) | byte[0]) / 9
            idx += 1

        print(f'Yaw Rates [ Pitch: {eul_meas_bytes[0]}, Roll: {eul_meas_bytes[1]}, Head: {eul_meas_bytes[2]} ]')
        return eul_meas_bytes

    def ang_vel(self):
        '''
            @name           angular velocity
            @brief          reads angular velocity from the IMU to use as measurements for feedback
        '''
        gyr_meas_bytes = [ 0, 0, 0 ] # msb, lsb

        idx = 0
        for reg in self.gyr_list:
            byte = self.controller.mem_read(2, self.imu_address, reg)
            gyr_meas_bytes[idx] = (byte[1] << 8) | byte[0]
            idx += 1

        print(f'Angular Velocities [ X: {gyr_meas_bytes[0]}, Y: {gyr_meas_bytes[1]}, Z {gyr_meas_bytes[2]} ]')

def face_north(imu):
    controls.stop()
    current_angle = imu.euler()
    controls.pivot_left(15)
    while not (0 <= current_angle[0] <= 7):
        current_angle = imu.euler()
    print('stop')
    controls.stop()

def calibrate(thing: bno055):
    bit = thing.cal_status()

    thing.change_mode('NDOF')
    print('Calibration Beginning')

```

```
    while sum(bit) != 8:
        bit = thing.cal_status()
        print(bit)

    print('Calibration finished')
    thing.change_mode('CONFIG')
    print('Calibration finished.')

def save_calibration(imu, filename):

    val = imu.get_cal_coeff()

    with open(filename, 'w') as file:
        for sublist in val:
            for item in sublist:
                integer_val = int.from_bytes(item, 'big')
                hex_val = hex(integer_val)
                file.write(hex_val + ',')
    print(f'Calibration results saved to: {filename}')

# file = 'cal_coeff.txt'
# try:
#     with open(file, 'r') as file:
#         cal_vals = file.readlines()
#         cal_vals = cal_vals[0].split(',')
#         print(cal_vals)
# except:
#     print('no calibration coefficient file found')
# create controller
i2c = I2C(1, I2C.CONTROLLER)
i2c.init(I2C.CONTROLLER, baudrate=400_000)

# create bno055 objects
imu_obj = bno055(i2c)
```