

# reWRITE

EE149/249A Project Report, Fall 2015

Reia Cho, Christian (CJ) Geering, Nathaniel Mailoa, Rachel Zhang



## I. INTRODUCTION

## II. BILL OF MATERIALS

## III. SYSTEM

*A. LightBlue Bean*

*B. BNO055 Absolute Orientation Sensor Break-out Board*

*C. Li-Ion Battery*

*D. Force Sensor*

*E. Button and LED Sequin*

## IV. CASING

## V. POSITION RECONSTRUCTION

*A. IMU Sensor Data*

*B. Filtering and Thresholding*

*C. Transformation to Fixed Reference Frame*

*D. Velocity Adjustment*

*E. Tip Position Reconstruction*

*F. Plotting*

## VI. QUANTITATIVE ANALYSIS AND SCHEDULING

## VII. MODE OF OPERATION

## VIII. ACKNOWLEDGEMENTS

We would like to acknowledge the following individuals for their support and contribution in this project.

- Trung Tran, *National Instruments*
- Prof. Sanjit Seshia, *UC Berkeley*
- Matthew Weber, *UC Berkeley*
- Eric Kim, *UC Berkeley*
- Casey Rogers, *UC Berkeley 3D Modeling Club*

## IX. APPENDIX 1: LIGHTBLUE BEAN CODE

```

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imu.h>

#define LED (0)
#define FORCE_SENSOR (1)
#define BUTTON (3)

Adafruit_BNO055 bno = Adafruit_BNO055();

uint8_t sys, gyroCal, accelCal, magCal = 0;
long time = 0;
boolean button = 0;
boolean sync = 0;
int do_loop = 0;
boolean debug = 0;
byte count = 0;
byte cur_count = 0;
char buff[60] = {0};
int reading_no = 0;
int ax = 0;
int ay = 0;
int az = 0;
int ex = 0;
int ey = 0;
int ez = 0;
boolean force = 0;

imu::Vector<3> accel;
imu::Vector<3> euler;

// Timer ISR to trigger reading
ISR(TIMER1_COMPA_vect){
    do_loop = 1;
    count++;
}

void setup(void){
    // Set to red for calibration
    Bean.setLed(255,0,0);

    sync = 0;
    do_loop = 0;
    reading_no = 0;
    count = 0;

    pinMode(LED, OUTPUT);
    pinMode(BUTTON, INPUT);
    pinMode(FORCE_SENSOR, INPUT);

    cli();          // disable global interrupts

```

```

TCCR1A = 0;      // set entire TCCR1A register to 0
TCCR1B = 0;      // same for TCCR1B

// set compare match register to desired timer count:
OCR1A = 117;     // 15ms period

// turn on CTC mode:
TCCR1B |= (1 << WGM12);

// Set CS10 and CS12 bits for 1024 prescaler:
TCCR1B |= (1 << CS10);
TCCR1B |= (1 << CS12);

// enable timer compare interrupt:
TIMSK1 |= (1 << OCIE1A);
sei();           // enable global interrupts

// Start serial communication
Serial.begin(57600);
Serial.println("Orientation_Sensor_Raw_Data_Test"); Serial.println("");

/* Initialise the sensor */
if(!bno.begin())
{
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while(1);
}

delay(1000);

bno.setExtCrystalUse(true);
}

void loop(void){
    if (do_loop){
        // Get data from IMU, force sensor and button
        cur_count = count;
        bno.getCalibration(&sys, &gyroCal, &accelCal, &magCal);
        accel = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
        euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
        force = digitalRead(FORCE_SENSOR);
        button = digitalRead(BUTTON) | (force << 1);

        // If just got synced
        if (sync == 0 && gyroCal == 3 && accelCal == 3 && magCal == 3){
            buff[0] = char(1 << 7);
            buff[1] = char(sys << 6 | accelCal << 4 | gyroCal << 2 | magCal);
            buff[11] = char(cur_count);

            Serial.write((const unsigned char*)buff, 12);
            Serial.flush();
            Bean.setLed(246, 255, 0);
            reading_no = 0;
        }
    }
}

```

```

// Wait for confirmation from PC
while (Serial.read() != '1');

// Wait for button press
Bean.setLed(0,0,255);
while (digitalRead(BUTTON) != 1);
while (digitalRead(BUTTON) != 0);
Bean.setLed(0,255,0);

sync = 1;
}
else {
// If not synced yet
if (sync == 0){
// Set calibration packet in buffer
buff[12*reading_no] = char(1 << 7);
buff[12*reading_no+1] = char(sys << 6 | accelCal << 4 | gyroCal << 2 |
magCal);
buff[12*reading_no+11] = char(cur_count);

// Turn on LED based on calibration state
if (gyroCal == 3 && magCal == 3){
if (accelCal == 0) Bean.setLed(0,0,0);
if (accelCal == 1) Bean.setLed(73,243,243);
if (accelCal == 2) Bean.setLed(242,189,73);
}
}
else {
// Format data for packeting
ax = int(accel.x()*100) & 0xfff;
ay = int(accel.y()*100) & 0xfff;
az = int(accel.z()*100) & 0xfff;

ex = int(euler.x()*100);
ey = int(euler.y()*100);
ez = int(euler.z()*100);

// Set data packet in buffer
buff[12*reading_no] = char(1 << 7 | button << 4 | ax >> 8);
buff[12*reading_no+1] = char(ax & 0xff);
buff[12*reading_no+2] = char(ay >> 4);
buff[12*reading_no+3] = char((ay & 0xf) << 4 | (az >> 8) & 0xf);
buff[12*reading_no+4] = char(az & 0xff);
buff[12*reading_no+5] = char(ex >> 8);
buff[12*reading_no+6] = char(ex & 0xff);
buff[12*reading_no+7] = char(ey >> 8);
buff[12*reading_no+8] = char(ey & 0xff);
buff[12*reading_no+9] = char(ez >> 8);
buff[12*reading_no+10] = char(ez & 0xff);
buff[12*reading_no+11] = char(cur_count);

// User feedback for force sensor on LED Sequin
if(force) digitalWrite(LED, HIGH);
else digitalWrite(LED, LOW);
}
}

```

```

    }

    reading_no++;
    // If buffer filled with 5 readings , send to PC
    if (reading_no > 4) {
        Serial.write((const unsigned char*)buff, 60);
        reading_no = 0;
    }
}
// Reset ISR signal
do_loop = 0;
}
}

```

## X. APPENDIX 2: PYTHON CODE

```

import numpy as np
import matplotlib.pyplot as plt
import math
import sys
import serial
import glob
import time
import re
import os

# Data arrays
ax = np.array([])
ay = np.array([])
az = np.array([])
vx = np.array([])
vy = np.array([])
vz = np.array([])
x = np.array([])
y = np.array([])
z = np.array([])
t = np.array([])
tipx = np.array([])
tipy = np.array([])
forceSensor = np.array([])

# Constant parameters
len_pen = 0.1
accel_thres = 0.15
accel_time_lim = 0.1
scale_factor = [10/9.5, 10/7.5]

# Number of readings per processing loop
CHUNKS = 20

def serial_ports():
    """Lists serial ports
    __Raises:
    __EnvironmentError:
    """

```

```

#####On unsupported or unknown platforms
Returns:
#####A list of available serial ports
"""
    if sys.platform.startswith('win'):
        ports = ['COM' + str(i + 1) for i in range(256)]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this is to exclude your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')
    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result

def run():
    global ax, ay, az, vx, vy, vz, x, y, z, t, v_anchor, tipx, tipy, forceSensor
    print("reWRITE Position Reconstruction")

    # Connect to serial port
    ports = serial_ports()
    if ports:
        print("Available serial ports:")
        for (i,p) in enumerate(ports):
            print("%d) %s"%(i+1,p))
    else:
        print("No ports available. Check serial connection and try again.")
        print("Exiting ...")
        return
    portNo = input("Select the port to use: ")
    ser = serial.Serial(ports[int(portNo)-1])
    ser.baudrate=57600
    ser.timeout=10

    # Reset variables
    cur_idx = 0
    mean_x = 0
    mean_y = 0
    mean_z = 0
    last_ax = 0
    last_ay = 0
    last_az = 0
    base_ex = 0
    base_ey = 0
    base_ez = 0
    base_time = 0

```

```

# Calibration constants
last_zero = [1,1,1]
first_nonzero = [0,0,0]
last_forceOn = -1
lpf_alpha = 0.9
hpf_alpha = 0.99
cal_pow = 2
failcount = 0

ser.flush()

# Calibration stage
calibrated = 0
if (sys.argv[2] == 'c'):
    calibrated = 1

while(calibrated != 1):
    try:
        line = (ser.read(12))
        c = line[1]
        # Print calibration status
        print(str((c & 0xc0) >> 6) + str((c & 0x30) >> 4) + str((c & 0x0c) >> 2) +
              str(c & 0x03))
        if ((c & 0x3f) == 0x3f): calibrated = 1
    except:
        pass

print("Recording_in_3...")
time.sleep(1)
print("2...")
time.sleep(1)
print("1...")
time.sleep(1)
print("Start")
ser.write('1'.encode())
ser.flushInput()
print(ser.inWaiting())

# Start plot
plt.ion()
fig = plt.figure()
figA = fig.add_subplot(111)
figA.set_xlabel('x')
figA.set_ylabel('y')

# Throw away first 20 data points
for i in range(20):
    line = ser.read(12)

# Main loop
while(True):
    print(cur_idx)
    ex = np.zeros(CHUNKS)
    ey = np.zeros(CHUNKS)
    ez = np.zeros(CHUNKS)

```

```

tax = np.zeros(CHUNKS)
tay = np.zeros(CHUNKS)
taz = np.zeros(CHUNKS)
count = 0

# Populate 20 data points
while count < CHUNKS:
    if (failcount > 20):
        print("Failed more than 20 times")
        exit()
    try:
        line = ser.read(12)

        # Populate force sensor and button data
        force = (line[0] >> 5) & 0x1;
        button = (line[0] >> 4) & 0x1;

        # Populate acceleration data
        ax = (line[0] & 0xf) << 8 | line[1]
        if (ax & 0x800): ax = -1*int((ax ^ 0xffff) + 1)
        ax = ax/100

        ay = line[2] << 4 | (line[3] >> 8)
        if (ay & 0x800): ay = -1*int((ay ^ 0xffff) + 1)
        ay = ay/100

        az = (line[3] & 0xf) << 8 | line[4]
        if (az & 0x800): az = -1*int((az ^ 0xffff) + 1)
        az = az/100

        # Populate Euler angle data
        ex = line[5] << 8 | line[6]
        ex = int(ex)/100

        ey = line[7] << 8 | line[8]
        if (ey & 0x8000): ey = -1*int((ey ^ 0xffff) + 1)
        ey = int(ey)/100

        ez = line[9] << 8 | line[10]
        if (ez & 0x8000): ez = -1*int((ez ^ 0xffff) + 1)
        ez = int(ez)/100

        # Populate timestamp data
        timestemp = int(line[11])
        if t != [] and timestemp+base_time - t[-1] < 0:
            base_time = base_time + 256
        timestemp = timestemp + base_time

        # Acceleration high-pass, low-pass and thresholding
        temp = float(ax)
        if (cur_idx == 0 and count == 0):
            mean_x = temp
        mean_x = mean_x * hpf_alpha + temp * (1-hpf_alpha)
        temp = temp - mean_x
        last_ax = last_ax*lpf_alpha + temp*(1-lpf_alpha)
        tax[count] = last_ax if abs(last_ax) > accel_thres/3 else 0

```



```

temp = float(ayt)
if (cur_idx == 0 and count == 0):
    mean_y = temp
mean_y = mean_y * hpf_alpha + temp * (1-hpf_alpha)
temp = temp - mean_y
last_ay = last_ay*lpf_alpha + temp*(1-lpf_alpha)
tay[count] = last_ay if abs(last_ay) > accel_thres/3 else 0

temp = float(azt)
if (cur_idx == 0 and count == 0):
    mean_z = temp
mean_z = mean_z * hpf_alpha + temp * (1-hpf_alpha)
temp = temp - mean_z
last_az = last_az*lpf_alpha + temp*(1-lpf_alpha)
taz[count] = last_az if abs(last_az) > accel_thres/3 else 0

# Convert Euler angles to deltas in radians
if (cur_idx == 0 and count == 0):
    ex[count] = 0
    ey[count] = 0
    ez[count] = 0
    base_ez = ezt
    base_ey = eyt
    base_ex = ext
else:
    ez[count] = -(ezt - base_ez)/360*2*math.pi
    ey[count] = (eyt - base_ey)/360*2*math.pi
    ex[count] = (ext - base_ex)/360*2*math.pi

# Save timestamp and force sensor data
t = np.append(t, timetemp)
forceSensor = np.append(forceSensor, force)

# If button is pressed, clear figure
if(button):
    figA.cla()

count = count + 1

except:
    failcount = failcount + 1
    pass

# Convert IMU frame of reference to real coordinates
sex = np.sin(ex)
sey = np.sin(ey)
sez = np.sin(ez)
cex = np.cos(ex)
cey = np.cos(ey)
cez = np.cos(ez)

ax = np.append(ax, cey*cez*tax + (sex*sey*cez - cex*sez)*tay + (cex*sey*cez +
sex*sez)*taz)
ay = np.append(ay, cey*sez*tax + (sex*sey*sez + cex*cez)*tay + (cex*sey*sez -
sex*cez)*taz)

```

```

az = np.append(az, -sey*tax + sex*cey*tay + cex*cey*taz)

# Process absolute acceleration data
for i in range(cur_idx, cur_idx+CHUNKS):
    if (i == 0):
        vx = np.array([0])
        vy = np.array([0])
        vz = np.array([0])
        x = np.array([0])
        y = np.array([0])
        z = np.array([0])
        v_anchor = np.array([1])
        continue

    timestep = (t[i] - t[i-1])*15/1000

    # Update velocity
    vx = np.append(vx, vx[i-1]+ax[i-1]*timestep)
    vy = np.append(vy, vy[i-1]+ay[i-1]*timestep)
    vz = np.append(vz, vz[i-1]+az[i-1]*timestep)

    # Update position
    x = np.append(x, x[i-1]+vx[i-1]*timestep)
    y = np.append(y, y[i-1]+vy[i-1]*timestep)
    z = np.append(z, z[i-1]+vz[i-1]*timestep)

    # Calibrate x velocity if more than accel_time_lim with no x accel
    if (abs(ax[i]) <= accel_thres):
        if first_nonzero[0] != -1 and last_zero[0] == -1:
            last_zero[0] = i
        else:
            last_zero[0] = -1
            if first_nonzero[0] == -1:
                first_nonzero[0] = i

    if (last_zero[0] != -1 and (t[i] - t[last_zero[0]]) > accel_time_lim*1000/15):
        :
        for j in range(first_nonzero[0], last_zero[0]+1):
            timestep = (t[j] - t[j-1])*15/1000
            vx[j] = vx[j] - ((j-first_nonzero[0])/(last_zero[0]-first_nonzero[0]))**
                cal_pow*vx[last_zero[0]]
            x[j] = x[j-1] + vx[j-1]*timestep
        for j in range(last_zero[0], i+1):
            vx[j] = 0
            x[j] = x[j-1]
        first_nonzero[0] = i-1
        last_zero[0] = -1

    # Calibrate y velocity if more than accel_time_lim with no y accel
    if (abs(ay[i]) <= accel_thres):
        if first_nonzero[1] != -1 and last_zero[1] == -1:
            last_zero[1] = i
        else:
            last_zero[1] = -1
            if first_nonzero[1] == -1:

```

```

        first_nonzero[1] = i

    if (last_zero[1] != -1 and (t[i] - t[last_zero[1]]) > accel_time_lim*1000/15):
        :
        for j in range(first_nonzero[1], last_zero[1]+1):
            timestep = (t[j] - t[j-1])*15/1000
            vy[j] = vy[j] - ((j-first_nonzero[1])/(last_zero[1]-first_nonzero[1]))**
                cal_pow*vy[last_zero[1]]
            y[j] = y[j-1] + vy[j-1]*timestep
        for j in range(last_zero[1], i+1):
            vy[j] = 0
            y[j] = y[j-1]
        first_nonzero[1] = i-1
        last_zero[1] = -1

# Calibrate z velocity if more than accel_time_lim with no z accel
if (abs(az[i]) <= accel_thres):
    if first_nonzero[2] != -1 and last_zero[2] == -1:
        last_zero[2] = i
    else:
        last_zero[2] = -1
    if first_nonzero[2] == -1:
        first_nonzero[2] = i

if (last_zero[2] != -1 and (t[i] - t[last_zero[2]]) > accel_time_lim*1000/15):
    :
    for j in range(first_nonzero[2], last_zero[2]+1):
        timestep = (t[j] - t[j-1])*15/1000
        vz[j] = vz[j] - ((j-first_nonzero[2])/(last_zero[2]-first_nonzero[2]))**
            cal_pow*vz[last_zero[2]]
        z[j] = z[j-1] + vz[j-1]*timestep
    for j in range(last_zero[2], i+1):
        vz[j] = 0
        z[j] = z[j-1]
    first_nonzero[2] = i-1
    last_zero[2] = -1

# Compute tip position from IMU position
tipx = np.append(tipx, x[cur_idx:]*scale_factor[0] - len_pen*sey)
tipy = np.append(tipy, y[cur_idx:]*scale_factor[1] - len_pen*sex)

# Plot if we go from force sensor on to off
if (last_forceOn != -1):
    first_zero_f = None
    for idx in range(cur_idx, cur_idx+CHUNKS):
        if forceSensor[idx] == 0:
            first_zero_f = idx
            break
    if (first_zero_f):
        figA.plot(tipx[last_forceOn:first_zero_f-5], tipy[last_forceOn:first_zero_f-5], 'blue')
        plt.axis('equal')
        last_forceOn = -1
        fig.canvas.draw()

```

```

if (last_forceOn == -1):
    first_zero_f = None
    for idx in range(cur_idx, cur_idx+CHUNKS):
        if forceSensor[idx] == 1:
            first_zero_f = idx
            break
    if (first_zero_f):
        last_forceOn = first_zero_f

cur_idx = cur_idx + CHUNKS

# Time limit of 10000 samples
if (cur_idx >= 10000):
    break

# Keep plot on
plt.show()

run()

```