گزارش پایانی آز سخت‌افزار

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

آرین احدی نیا

مصطفی اوجاقی

امیرسپهر پورفناد

استاد درس: جناب آقای دکتر اجلالی

دستیاران آموزشی: جناب آقای فصحتی، سرکار خانم رضازاد

پاییز ۱۴۰۱

فهرست مطالب

مقدمه

امروزه صنایع و امور بسیاری در جهان از جمله حمل و نقل هوایی و دریایی، خودروهای خودران، صنایع نظامی-دفاعی و موارد بسیاری از این دست وابسته به مکان‌یابی‌های دقیق هستند. امروزه در جهان در بسیاری از کاربردها این مهم به وسیله سامانه موقعیت‌یابی جهانی[۱] که تحت عنوان سیستم‌های آ-جی‌پی‌اس[۲] ، گلوناس[۳] ، گالیله[۴] و بایدو[۵] به ترتیب توسط ایالات متحده آمریکا، فدراسیون روسیه [۶] ، اتحادیه اروپا و جمهوری خلق چین توسعه داده شده‌اند انجام می‌شود.

این مکان‌یابی علاوه بر دقت[۷] ، باید قابلیت اطمینان[۸] بالایی داشته باشد به این صورت که بتواند دائما مکان شی متحرک را رهگیری کند. در بسیاری از کاربردها مانند صنایع هوایی، عدم رهگیری درست مکان حتی می‌تواند به فاجعه منتهی شود به عنوان مثال پرواز شماره ۷ هواپیمایی کره[۹] در تاریخ اول سپتامبر ۱۹۸۳ با مشکل در سیستم مکان‌یابی و اشتباه خلبان به مناطق پرواز ممنوع شبه جزیره ساخالین تحت حاکمیت اتحاد جماهیر شوروی هدایت شد و با شلیک نیروی هوایی شوروی این بوئینگ ۷۴۷ در آب‌های دریای ژاپن سرنگون شد و تمام ۲۶۹ مسافر و خدمه آن کشته شدند. افزونگی[۱۰] در بسیاری از مسائل راهکاری برای افزایش قابلیت اطمینان است. در این مساله نیز می‌توانیم به استفاده از چندین سیستم قابلیت اطمینان سیستم را افزایش دهیم اما همچنان مشکلاتی مانند عدم آنتن‌دهی جی‌پی‌اس در مکان‌های بسته باقی خواهد ماند.

برای حل این مشکل از باید از روش‌های ترکیبی استفاده کرد. در این سیستم ما با استفاده از ژیروسکوپ سعی می‌کنم در شرایط عدم آنتن‌دهی مکان‌یابی را ادامه دهیم. این روش علاوه بر حل مشکل آنتن‌دهی، می‌تواند به عنوان یک روش تقویتی برای دقت جی‌پی‌اس نیز مورد استفاده قرار گیرد.

در ادامه این گزارش ابتدا به بررسی اجمالی تحقیق انجام شده می‌پردازیم و معماری آن را تشریح می‌کنیم. سپس به نحوه اتصال به سیستم جهت استفاده از آن می‌پردازیم و در نهایت در مورد ابعاد فنی پروژه صحبت می‌کنیم. در نهایت نیز با بررسی قیمت محصول، جمع‌بندی را انجام می‌دهیم.

---

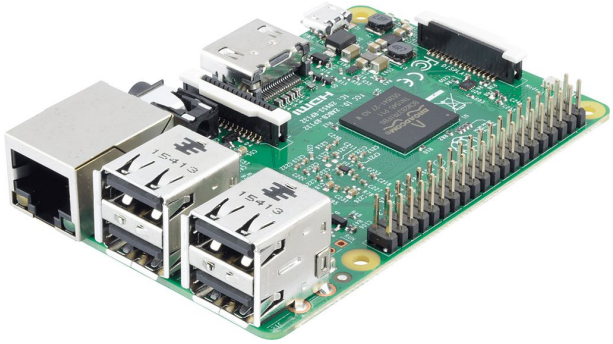
[۱] Global Positioning System (GPS)
[۲] A-GPS
[۳] GLONASS
[۴] Galileo
[۵] Baidu
[۶] سابقا اتحاد جماهیر شوروی
[۷] Accuracy
[۸] Reliability
[۹] KAL 007
[۱۰] Redundancy

کاتالوگ

مشخصه	توضیحات
دمای عملیاتی	صفر تا ۸۵ درجه سانتی‌گراد
برق مورد نیاز	۵ ولت، ۲٫۵ آمپر
برد اصلی	Raspberry Pi 3b
مکان‌یاب	NEO-6M
ژیروسکوپ	MPU9250

معماری محصول

این محصول بر پایه Raspberry Pi است که تصویر آن را در شکل پایین مشاهده می‌فرمایید.



شکل ۱: تصویر Raspberry Pi 3b که در پروژه استفاده شده است.

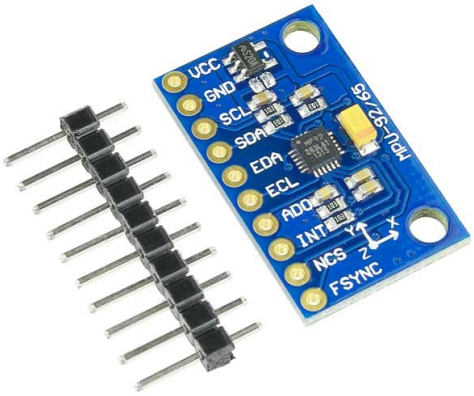
برای سنجش حرکت در این پروژه از ژیروسکوپ ۹ محوره MPU9250 استفاده کرده‌ایم که تصویر آن را در ادامه مشاهده می‌فرمایید.

همچنین در این محصول از ماژول مکان‌یاب NEO-6M GPS استفاده شده است که در تصویر آن را در شکل پایین مشاهده می‌فرمایید.

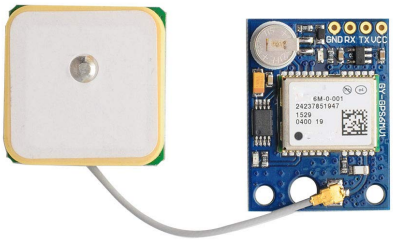
در نهایت ماژول‌های فوق به Raspberry متصل می‌شوند و از طریق یک نرم‌افزار اطلاعات آنها خوانده و به داده مفید تبدیل می‌شوند.

راه‌اندازی و راهنمای کاربری

برای اتصال به رزبری‌پای و دریافت دادگان مربوطه دو روش وجود دارد که در ادامه هر یک را توضیح داده‌ایم.



شکل ۲: تصویر ماژول MPU9250 Gyroscope که در پروژه استفاده شده است.



شکل ۳: تصویر ماژول NEO-6M GPS که در پروژه استفاده شده است.

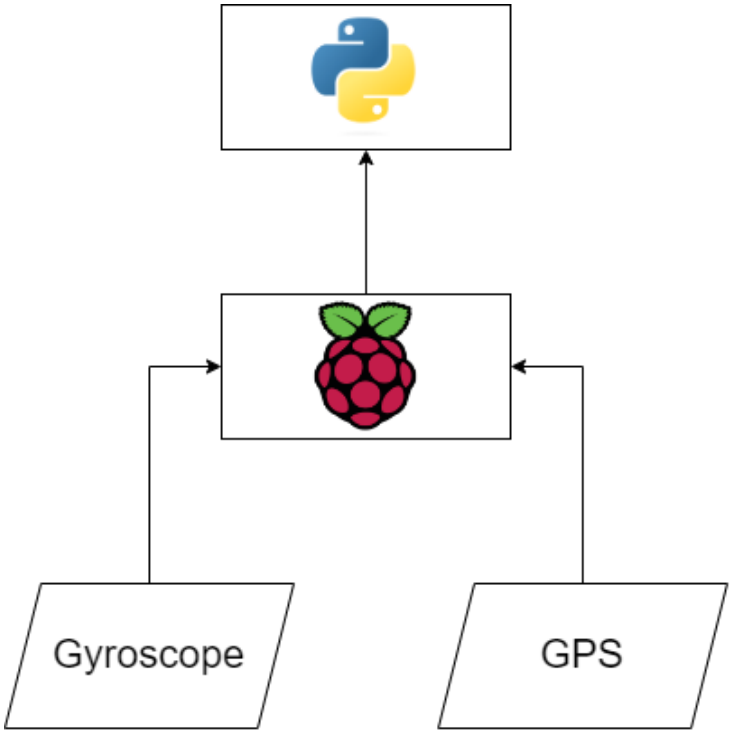
۱. استفاده از Monitor و اتصال مستقیم به محیط رزبری

در این حالت می‌توانیم با استفاده از یک کابل HDMI به رزبری وصل شویم و مستقیما از طریق محیط سیستم عامل رزبری کار را پیش بگیریم.

۲. اتصال به وسیله SSH به رزبری

در این حالت می‌توانیم با استفاده از آدرسی که در بخش مستندات فنی به رزبری نسبت می‌دهیم، با دستور SSH به ترمینال رزبری متصل شویم.

برای تحوه اتصال رزبری به ماژول‌های مورد استفاده به بخش مستندات فنی مراجعه فرمایید.



شکل ۴: معماری پروژه و ماژول‌های مورد استفاده در آن

زمان‌بندی انجام پروژه

زمان	دستور جلسه	توضیحات
۴ آبان	ارائه پروپوزال و تصویب آن	
۱۸ آبان	ارائه گزارش میانی اول	راه اندازی ژیروسکوپ و برقراری ارتباط آن با رزبری‌پای
۲ آذر	ارائه گزارش میانی دوم	راه اندازی GPS و برقراری ارتباط آن با رزبری‌پای
۱۶ آذر	ارائه گزارش میانی سوم	نوشتن دستورات لازم برای محاسبه مکان از روی ژیروسکوپ
۳۰ آذر	تحویل اولیه پروژه	آزمایش سامانه
۷ دی	تحویل نهایی پروژه	رفع خطا

مستندات فنی

نصب OS

برای نصب OS ابتدا فایل مربوطه را از سایت Raspberry دانلود می‌کنیم. سپس آن را روی microSD با استفاده از دستور

```
sudo dd if=<image_path> of=<microSD_dev_path> status=progress
```

می‌نویسیم. سپس microSD را در رزبری قرار داده و دستگاه را بوت می‌کنیم.

راه‌اندازی SSH

برای راه‌اندازی SSH، در boot/ ابتدا یک فایل به نام ssh ایجاد می‌کنیم. این کار باعث می‌شود تا تنظیمات SSH روی رزبری فعال شود. در گام بعد یک فایل به نام userconf ایجاد کرده که به واسطه آن یک یوزر جدید برای استفاده از SSH ایجاد می‌شود. در این فایل نام‌کاربری و Hash پسورد نوشته می‌شود. با استفاده از دستور زیر Hash را بدست می‌آوریم.

```
echo ‘raspberry’ | openssl passwd -6 -stdin
```

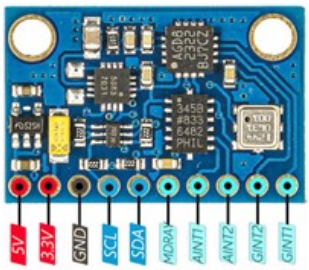
در نهایت محتویات زیر را در فایل userconf قرار می‌دهیم.

```
pi:<password_hash>
```

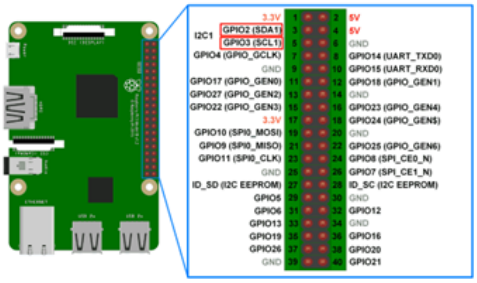
راه‌اندازی واسط‌های مربوطه

برای این بخش ما دو واسط Wi-Fi و I2C را با استفاده از دستور raspi-config فعال کردیم. این دستور یک محیط شبه گرافیکی پویا در اختیار قرار می‌دهد که به راحتی می‌توان در قسمت interface واسط‌های مذکور را فعال کرد.

وصل کردن ژیروسکوپ به رزبری



شکل ۵: تصویر پین‌های ماژول ژیروسکوپ



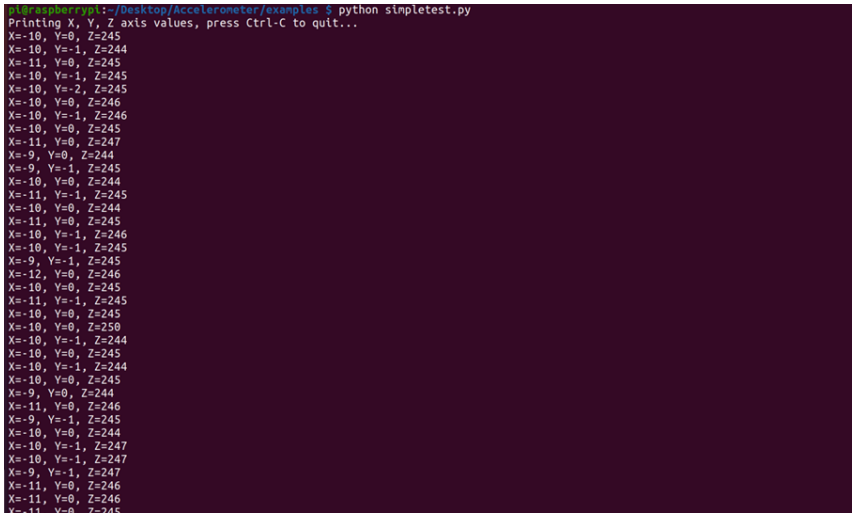
شکل ۶: تصویر پین‌های ماژول رزبری

مطابق شکل فوق، دو پین (SDA, SCL) پروتکل I2C را در دو دستگاه رزبری و ژیروسکوپ به هم متصل کردیم. همچنین برای برق دستگاه از پین 3.3V و GND استفاده می کنیم. این اتصالات از طریق یک بردبورد انجام می‌شود.

راه‌اندازی شتاب‌سنج

با استفاده از مخزن زیر در گیت‌هاب، کدهای مربوط به شتاب‌سنج را دریافت و آنها را با تغییر مناسب اجرا می‌کنیم. توجه فرمایید برای این کدهای نیاز به نصب پکیج smbus است.

https://github.com/adafruit/Adafruit_Python_ADXL345.git



شکل ۷: نمونه‌ای از اجرای کد شتاب‌سنج

```
# Minimal constants carried over from Arduino library
ADXL345_ADDRESS          = 0x53
ADXL345_REG_DEVID        = 0x00 # Device ID
ADXL345_REG_DATAX0       = 0x32 # X-axis data 0 (6 bytes for X/Y/Z)
ADXL345_REG_POWER_CTL    = 0x2D # Power-saving features control
ADXL345_REG_DATA_FORMAT  = 0x31
```

```
ADXL345_REG_BW_RATE      = 0x2C
ADXL345_DATARATE_0_10_HZ = 0x00
ADXL345_DATARATE_0_20_HZ = 0x01
ADXL345_DATARATE_0_39_HZ = 0x02
ADXL345_DATARATE_0_78_HZ = 0x03
ADXL345_DATARATE_1_56_HZ = 0x04
ADXL345_DATARATE_3_13_HZ = 0x05
ADXL345_DATARATE_6_25HZ  = 0x06
ADXL345_DATARATE_12_5_HZ = 0x07
ADXL345_DATARATE_25_HZ   = 0x08
ADXL345_DATARATE_50_HZ   = 0x09
ADXL345_DATARATE_100_HZ  = 0x0A # (default)
ADXL345_DATARATE_200_HZ  = 0x0B
ADXL345_DATARATE_400_HZ  = 0x0C
ADXL345_DATARATE_800_HZ  = 0x0D
ADXL345_DATARATE_1600_HZ = 0x0E
ADXL345_DATARATE_3200_HZ = 0x0F
ADXL345_RANGE_2_G        = 0x00 # +/-  2g (default)
ADXL345_RANGE_4_G        = 0x01 # +/-  4g
ADXL345_RANGE_8_G        = 0x02 # +/-  8g
ADXL345_RANGE_16_G       = 0x03 # +/- 16g

def read(self):
    """Read the current value of the accelerometer and return it as a tuple
    of signed 16-bit X, Y, Z axis values.
    """
    raw = self._device.readList(ADXL345_REG_DATAX0, 6)
    return struct.unpack('<hhh', raw)
```

در کد فوق، ابتدا مقادیر از دیتاشیت Set می‌شوند و سپس با استفاده از پیاده‌سازی کتاب‌خانه داده از روی سریال خوانده می‌شود.

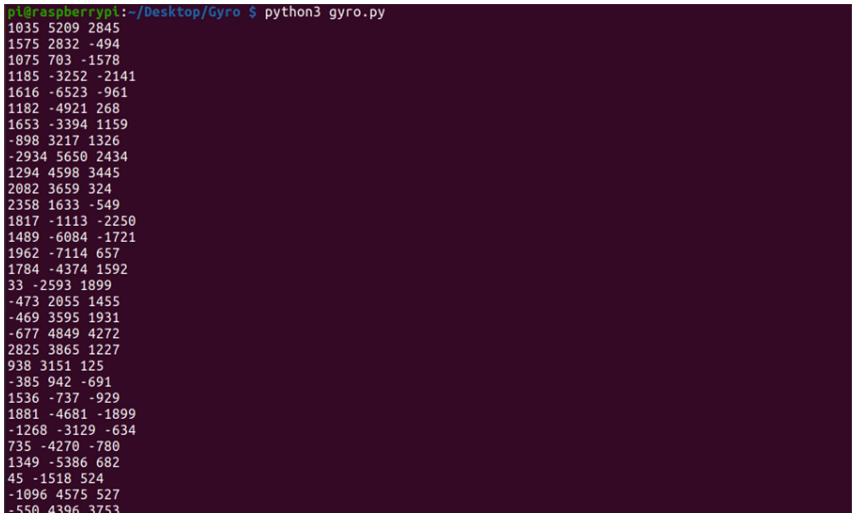
راه‌اندازی ژیروسکوپ

با استفاده از مخزن زیر در گیت‌هاب، کدهای مربوط به شتاب‌سنج را دریافت و آنها را با تغییر مناسب اجرا می‌کنیم. توجه فرمایید برای این کدهای نیاز به نصب پکیج smbus است.

https://github.com/bashardawood/L3G4200D-Python.git

در تصویر ۵ نیز نمونه خروجی را می‌توانید مشاهده کنید. برای این خروجی در حال چرخاندن سنسور بودیم و تغییرات نیز در خروجی مشهود است. هر ۰٫۰۸ ثانیه یک بار داده از سنسور دریافت و در خروجی نمایش داده می‌شود تا بتوان تاثیرات چرخاندن را در خروجی مشاهده کرد.

دقت بفرمایید که در کد به صورت دستی اورفلو بیش از ۱۶ بیت گرفته شده است.



شکل ۸: نمونه‌ای از اجرای کد ژیروسکوپ

```python
import smbus
import time

bus = smbus.SMBus(1)

bus.write_byte_data(0x68, 0x20, 0x0F)
bus.write_byte_data(0x68, 0x23, 0x30)

time.sleep(0.5)

data0 = bus.read_byte_data(0x68, 0x28)
data1 = bus.read_byte_data(0x68, 0x29)

xGyro = data1 * 256 + data0
if xGyro > 32767 :
xGyro -= 65536

data0 = bus.read_byte_data(0x68, 0x2A)
data1 = bus.read_byte_data(0x68, 0x2B)

yGyro = data1 * 256 + data0
if yGyro > 32767 :
yGyro -= 65536

data0 = bus.read_byte_data(0x68, 0x2C)
data1 = bus.read_byte_data(0x68, 0x2D)

zGyro = data1 * 256 + data0
if zGyro > 32767 :
zGyro -= 65536

print("Rotation in X-Axis : %d" %xGyro)
print("Rotation in Y-Axis : %d" %yGyro)
print("Rotation in Z-Axis : %d" %zGyro)
```

راه‌اندازی فشارسنج و دماسنج

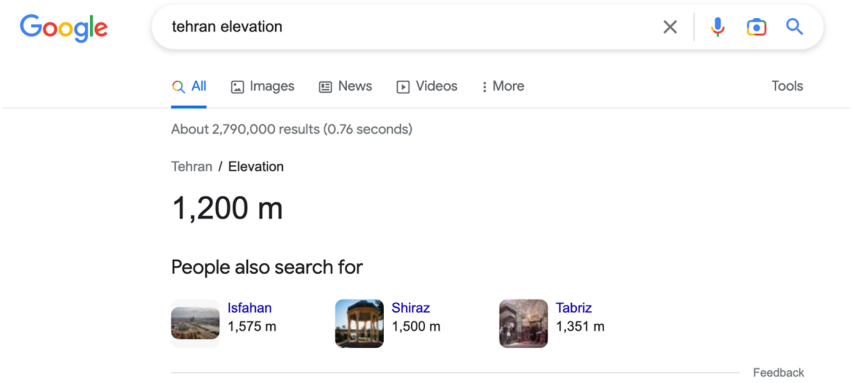
با استفاده از مخزن زیر در گیت‌هاب، کدهای مربوط به شتاب‌سنج را دریافت و آنها را با تغییر مناسب اجرا می‌کنیم. توجه فرمایید برای این کدهای نیاز به نصب پکیج smbus است.

https://github.com/adafruit/Adafruit_Python_BMP.git

با استفاده از این ماژول می‌توانیم ارتفاع از سطح دریا را محاسبه کنیم. همانطور که ملاحظه می‌فرمایید عدد خروجی با ارتفاع حدودی شهر تهران هم‌خوانی دارد. علاوه بر این دما و فشار نیز خروجی داده می‌شود.

```
pi@raspberrypi:~/Desktop/Pressure_sensor/examples $ python3 simpletest.py
Temp = 26.60 *C
Pressure = 88483.00 Pa
Altitude = 1127.97 m
Sealevel Pressure = 88483.00 Pa
```

شکل ۹: نمونه‌ای از اجرای کد ارتفاع و فشار



شکل ۱۰: ارتفاع شهر تهران از دریا که با خروجی داده شده همخوانی دارد.

```python
def read_pressure(self):
    UT = self.read_raw_temp()
    UP = self.read_raw_pressure()
    # Datasheet values for debugging:
    # UT = 27898
    # UP = 23843
    # Calculations below are taken straight from section 3.5 of the
    datasheet.
    # Calculate true temperature coefficient B5.
    X1 = ((UT - self.cal_AC6) * self.cal_AC5) >> 15
    X2 = (self.cal_MC << 11) // (X1 + self.cal_MD)
    B5 = X1 + X2
    self._logger.debug('B5 = {0}'.format(B5))
    # Pressure Calculations
    B6 = B5 - 4000
    self._logger.debug('B6 = {0}'.format(B6))
```

```
    X1 = (self.cal_B2 * (B6 * B6) >> 12) >> 11
    X2 = (self.cal_AC2 * B6) >> 11
    X3 = X1 + X2
    B3 = (((self.cal_AC1 * 4 + X3) << self._mode) + 2) // 4
    self._logger.debug('B3 = {0}'.format(B3))
    X1 = (self.cal_AC3 * B6) >> 13
    X2 = (self.cal_B1 * ((B6 * B6) >> 12)) >> 16
    X3 = ((X1 + X2) + 2) >> 2
    B4 = (self.cal_AC4 * (X3 + 32768)) >> 15
    self._logger.debug('B4 = {0}'.format(B4))
    B7 = (UP - B3) * (50000 >> self._mode)
    self._logger.debug('B7 = {0}'.format(B7))
    if B7 < 0x80000000:
    p = (B7 * 2) // B4
    else:
    p = (B7 // B4) * 2
    X1 = (p >> 8) * (p >> 8)
    X1 = (X1 * 3038) >> 16
    X2 = (-7357 * p) >> 16
    p = p + ((X1 + X2 + 3791) >> 4)
    self._logger.debug('Pressure {0} Pa'.format(p))
    return p

    def read_altitude(self, sealevel_pa=101325.0):
    """Calculates the altitude in meters."""
    # Calculation taken straight from section 3.6 of the datasheet.
    pressure = float(self.read_pressure())
    altitude = 44330.0 * (1.0 - pow(pressure / sealevel_pa, (1.0/5.255)))
    self._logger.debug('Altitude {0} m'.format(altitude))
    return altitude

```

جمع‌بندی ماژول GY801

در قطعه کد زیر سعی کردیم تا قسمت‌های مختلف ماژول GY801 را در کنار یکدیگر قرار دهیم تا بتوان با استفاده از یک واسط مناسب، توابع دسترسی به ماژول را فراخوانی کرد و داده‌های موردنیاز را از آن خواند.

```
#!/usr/bin/python3

import smbus
import time
from math import *

bus = smbus.SMBus(1);           # 0 for R-Pi Rev. 1, 1 for Rev. 2

# General constants
EARTH_GRAVITY_MS2   = 9.80665 # m/s2
STANDARD_PRESSURE   = 1013.25 # hPa

# ADXL345 (accelerometer) constants
ADXL345_ADDRESS =   0x53

```

```
ADXL345_DEVID           =   0x00
ADXL345_THRESH_TAP      =   0x1D
ADXL345_OFSX            =   0x1E
ADXL345_OFSY            =   0x1F
ADXL345_OFSZ            =   0x20
# Components Offset
#The OFSX, OFSY, and OFSZ registers are each eight bits and
#offer user-set offset adjustments in twos complement format
#with a scale factor of 15.6 mg/LSB (that is, 0x7F = +2 g).
# Real Offset : OFS_ x 15.625
ADXL345_DUR             =   0x21
ADXL345_Latent          =   0x22
ADXL345_Window          =   0x23
ADXL345_THRESH_ACT      =   0x24
ADXL345_THRESH_INACT    =   0x25
ADXL345_TIME_INACT      =   0x26
ADXL345_ACT_INACT_CTL   =   0x27
ADXL345_THRESH_FF       =   0x28
ADXL345_TIME_FF         =   0x29
ADXL345_TAP_AXES        =   0x2A
ADXL345_ACT_TAP_STATUS  =   0x2B
ADXL345_BW_RATE         =   0x2C
ADXL345_POWER_CTL       =   0x2D
ADXL345_INT_ENABLE      =   0x2E
ADXL345_INT_MAP         =   0x2F
ADXL345_INT_SOURCE      =   0x30
ADXL345_DATA_FORMAT     =   0x31
# Register 0x31 - Data Format - Read/Write
# D7: SELF_TEST D6: SPI D5: INT_INVERT  D4: 0   D3: FULL_RES    D2:Justify
     D1-D0: Range
# D1-D0 = 00: +/-2G D1-D0 = 01: +/-4G   D1-D0 = 10: +/-8G   D1-D0 = 11:
   +/-16G
# Range:
#   FULL_RES=1: 3.9 mG/LSP (0.00390625 G/LSP)
#   +/-2G,10bit mode:  3.9 mG/LSP
#   +/-4G,10bit mode:  7.8 mG/LSP
#   +/-8G,10bit mode:  15.6 mG/LSP
#   +/-16G,10bit mode:  31.2 mG/LSP
ADXL345_DATAX0          =   0x32
ADXL345_DATAX1          =   0x33
ADXL345_DATAY0          =   0x34
ADXL345_DATAY1          =   0x35
ADXL345_DATAZ0          =   0x36
ADXL345_DATAZ1          =   0x37
ADXL345_FIFO_CTL        =   0x38
ADXL345_FIFO_STATUS     =   0x39

ADXL345_SCALE_MULTIPLIER    = 0.00390625    # G/LSP

ADXL345_BW_RATE_3200HZ  = 0x0F
ADXL345_BW_RATE_1600HZ  = 0x0E
ADXL345_BW_RATE_800HZ   = 0x0D
ADXL345_BW_RATE_400HZ   = 0x0C
ADXL345_BW_RATE_200HZ   = 0x0B
ADXL345_BW_RATE_100HZ   = 0x0A # (default)
ADXL345_BW_RATE_50HZ    = 0x09
```

```
ADXL345_BW_RATE_25HZ    = 0x08

ADXL345_RANGE_2G        = 0x00 # +/-  2g (default)
ADXL345_RANGE_4G        = 0x01 # +/-  4g
ADXL345_RANGE_8G        = 0x02 # +/-  8g
ADXL345_RANGE_16G       = 0x03 # +/- 16g

ADXL345_MEASURE         = 0x08

#L3G4200D (Gyroscope) constants
L3G4200D_ADDRESS        =   0x69

L3G4200D_WHO_AM_I       =   0x0F
L3G4200D_CTRL_REG1      =   0x20
L3G4200D_CTRL_REG2      =   0x21
L3G4200D_CTRL_REG3      =   0x22
L3G4200D_CTRL_REG4      =   0x23
# Register 0x23 - Control Register 4 - Read/Write
# D7: BDU   D6: BLE D5-D4: FS0-FS1  D3: 0   D2-D1:ST1-ST0   D0: SIM
# D5-D4 = 00: +/-2G
# Range D5-D4:
#   00 +/-250dps : 8.75mdps/LSP
#   01 +/-500dps : 17.50mdps/LSP
#   10 +/-1000dps : 35mdps/LSP
#   11 +/-2000dps : 70mdps/LSP
L3G4200D_CTRL_REG5      =   0x24
L3G4200D_REFERENCE      =   0x25
L3G4200D_OUT_TEMP       =   0x26
L3G4200D_STATUS_REG     =   0x27
L3G4200D_OUT_X_L        =   0x28
L3G4200D_OUT_X_H        =   0x29
L3G4200D_OUT_Y_L        =   0x2A
L3G4200D_OUT_Y_H        =   0x2B
L3G4200D_OUT_Z_L        =   0x2C
L3G4200D_OUT_Z_H        =   0x2D
L3G4200D_FIFO_CTRL_REG  =   0x2E
L3G4200D_FIFO_SRC_REG   =   0x2F
L3G4200D_INT1_CFG       =   0x30
L3G4200D_INT1_SRC       =   0x31
L3G4200D_INT1_TSH_XH    =   0x32
L3G4200D_INT1_TSH_XL    =   0x33
L3G4200D_INT1_TSH_YH    =   0x34
L3G4200D_INT1_TSH_YL    =   0x35
L3G4200D_INT1_TSH_ZH    =   0x36
L3G4200D_INT1_TSH_ZL    =   0x37
L3G4200D_INT1_DURATION  =   0x38

L3G4200D_RANGE_250      = 0x00  # +/-250dps : 8.75mdps/LSP
L3G4200D_RANGE_500      = 0x01  # +/-500dps : 17.50mdps/LSP
L3G4200D_RANGE_1000     = 0x02  # +/-1000dps : 35mdps/LSP
L3G4200D_RANGE_2000     = 0x03  # +/-2000dps : 70mdps/LSP

#HMC5883L (Magnetometer) constants
HMC5883L_ADDRESS        =   0x1E

HMC5883L_CRA            =   0x00
```

```
126 HMC5883L_CRB             =   0x01
127 HMC5883L_MR              =   0x02
128 HMC5883L_DO_X_H          =   0x03
129 HMC5883L_DO_X_L          =   0x04
130 HMC5883L_DO_Z_H          =   0x05
131 HMC5883L_DO_Z_L          =   0x06
132 HMC5883L_DO_Y_H          =   0x07
133 HMC5883L_DO_Y_L          =   0x08
134 HMC5883L_SR              =   0x09
135 HMC5883L_IR_A            =   0x0A
136 HMC5883L_IR_B            =   0x0B
137 HMC5883L_IR_C            =   0x0C
138
139 #BMP180 (Barometer) constants
140 BMP180_ADDRESS           = 0x77
141 BMP180_AC1               = 0xAA
142 BMP180_AC2               = 0xAC
143 BMP180_AC3               = 0xAE
144 BMP180_AC4               = 0xB0
145 BMP180_AC5               = 0xB2
146 BMP180_AC6               = 0xB4
147 BMP180_B1                = 0xB6
148 BMP180_B2                = 0xB8
149 BMP180_MB                = 0xBA
150 BMP180_MC                = 0xBC
151 BMP180_MD                = 0xBE
152
153
154 class IMU(object):
155
156     def write_byte(self,adr, value):
157         bus.write_byte_data(self.ADDRESS, adr, value)
158
159     def read_byte(self,adr):
160         return bus.read_byte_data(self.ADDRESS, adr)
161
162     def read_word(self,adr,rf=1):
163         # rf=1 Little Endian Format, rf=0 Big Endian Format
164         if (rf == 1):
165             low = self.read_byte(adr)
166             high = self.read_byte(adr+1)
167         else:
168             high = self.read_byte(adr)
169             low = self.read_byte(adr+1)
170         val = (high << 8) + low
171         return val
172
173     def read_word_2c(self,adr,rf=1):
174         val = self.read_word(adr,rf)
175         if(val & (1 << 16 - 1)):
176             return val - (1<<16)
177         else:
178             return val
179
180 class gy801(object):
181     def __init__(self) :
```

```
182         self.accel = ADXL345()
183         self.gyro = L3G4200D()
184         self.compass = HMC5883L()
185         self.baro = BMP180()
186 
187 
188 class ADXL345(IMU):
189 
190     ADDRESS = ADXL345_ADDRESS
191 
192     def __init__(self) :
193         #Class Properties
194         self.Xoffset = 0x00
195         self.Yoffset = 0x00
196         self.Zoffset = 0x00
197         self.Xraw = 0.0
198         self.Yraw = 0.0
199         self.Zraw = 0.0
200         self.Xg = 0.0
201         self.Yg = 0.0
202         self.Zg = 0.0
203         self.X = 0.0
204         self.Y = 0.0
205         self.Z = 0.0
206         self.df_value = 0b00001000  # Self test disabled, 4-wire interface
207                                     # Full resolution, Range = +/-2g
208         self.Xcalibr = ADXL345_SCALE_MULTIPLIER
209         self.Ycalibr = ADXL345_SCALE_MULTIPLIER
210         self.Zcalibr = ADXL345_SCALE_MULTIPLIER
211 
212         self.write_byte(ADXL345_BW_RATE, ADXL345_BW_RATE_100HZ) # Normal
    mode, Output data rate = 100 Hz
213         self.write_byte(ADXL345_POWER_CTL, ADXL345_MEASURE) # Auto Sleep
    disable
214         self.write_byte(ADXL345_DATA_FORMAT, self.df_value)
215 
216     # RAW readings in LPS
217     def getRawX(self) :
218         self.X_raw = self.read_word_2c(ADXL345_DATAX0)
219         return self.X_raw
220 
221     def getRawY(self) :
222         self.Yraw = self.read_word_2c(ADXL345_DATAY0)
223         return self.Yraw
224 
225     def getRawZ(self) :
226         self.Zraw = self.read_word_2c(ADXL345_DATAZ0)
227         return self.Zraw
228 
229     # G related readings in g
230     def getXg(self,plf = 1.0) :
231         self.Xg = (self.getRawX() * self.Xcalibr + self.Xoffset) * plf +
    (1.0 - plf) * self.Xg
232         return self.Xg
233 
234     def getYg(self,plf = 1.0) :
```

```
235         self.Yg = (self.getRawY() * self.Ycalibr + self.Yoffset) * plf +
    (1.0 - plf) * self.Yg
236         return self.Yg
237 
238     def getZg(self,plf = 1.0) :
239         self.Zg = (self.getRawZ() * self.Zcalibr + self.Zoffset) * plf +
    (1.0 - plf) * self.Zg
240         return self.Zg
241 
242     # Absolute reading in m/s2
243     def getX(self,plf = 1.0) :
244         self.X = self.getXg(plf) * EARTH_GRAVITY_MS2
245         return self.X
246 
247     def getY(self,plf = 1.0) :
248         self.Y = self.getYg(plf) * EARTH_GRAVITY_MS2
249         return self.Y
250 
251     def getZ(self,plf = 1.0) :
252         self.Z = self.getZg(plf) * EARTH_GRAVITY_MS2
253         return self.Z
254 
255     def getPitch(self) :
256         aX = self.getXg()
257         aY = self.getYg()
258         aZ = self.getZg()
259         self.pitch = atan2(aX,sqrt(aY*aY+aZ*aZ)) * 180.0/pi
260         return self.pitch
261 
262     def getRoll(self) :
263         aX = self.getXg()
264         aY = self.getYg()
265         aZ = self.getZg()
266         self.roll = atan2(aY,(sqrt(aX*aX+aZ*aZ))) * 180.0/pi
267         return self.roll
268 
269 class L3G4200D(IMU):
270 
271     ADDRESS = L3G4200D_ADDRESS
272 
273     def __init__(self) :
274         #Class Properties
275         self.Xraw = 0.0
276         self.Yraw = 0.0
277         self.Zraw = 0.0
278         self.X = 0.0
279         self.Y = 0.0
280         self.Z = 0.0
281         self.Xangle = 0.0
282         self.Yangle = 0.0
283         self.Zangle = 0.0
284         self.gain_std = 0.00875 # dps/digit
285         self.t0x = None
286         self.t0y = None
287         self.t0z = None
288 
```

```
289         self.write_byte(L3G4200D_CTRL_REG1, 0x0F)   # 0x0F(15)  Normal mode
    , X, Y, Z-Axis enabled 0xB0
290         self.write_byte(L3G4200D_CTRL_REG4, 0x80)   # 0x30(48)  Block non
    continous update, Data LSB at lower address
291                                         # FSR 250dps, Self test disabled,
    4-wire interface
292         #write(L3G4200D_CTRL_REG4, 0x30)    # 0x30(48)  Continous update,
    Data LSB at lower address
293                                         ## FSR 2000dps, Self test disabled,
     4-wire interface
294         self.setCalibration()
295
296     def setCalibration(self) :
297         gyr_r = self.read_byte(L3G4200D_CTRL_REG4)
298
299         self.gain = 2 ** ( gyr_r & 48 >> 4) * self.gain_std
300
301     def getRawX(self):
302         self.Xraw = self.read_word_2c(L3G4200D_OUT_X_L)
303         return self.Xraw
304
305     def getRawY(self):
306         self.Yraw = self.read_word_2c(L3G4200D_OUT_Y_L)
307         return self.Yraw
308
309     def getRawZ(self):
310         self.Zraw = self.read_word_2c(L3G4200D_OUT_Z_L)
311         return self.Zraw
312
313     def getX(self,plf = 1.0):
314         self.X = ( self.getRawX() * self.gain ) * plf + (1.0 - plf) * self.
    X
315         return self.X
316
317     def getY(self,plf = 1.0):
318         self.Y = ( self.getRawY() * self.gain ) * plf + (1.0 - plf) * self.
    Y
319         return self.Y
320
321     def getZ(self,plf = 1.0):
322         self.Z = ( self.getRawZ() * self.gain ) * plf + (1.0 - plf) * self.
    Z
323         return self.Z
324
325     def getXangle(self,plf = 1.0) :
326         if self.t0x is None : self.t0x = time.time()
327         t1x = time.time()
328         LP = t1x - self.t0x
329         self.t0x = t1x
330         self.Xangle += self.getX(plf) * LP
331         return self.Xangle
332
333     def getYangle(self,plf = 1.0) :
334         if self.t0y is None : self.t0y = time.time()
335         t1y = time.time()
336         LP = t1y - self.t0y
```

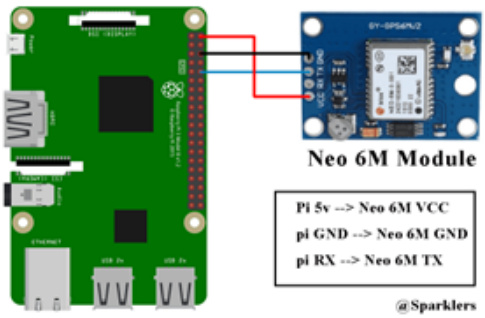
```
337         self.t0y = t1y
338         self.Yangle += self.getY(plf) * LP
339         return self.Yangle
340 
341     def getZangle(self,plf = 1.0) :
342         if self.t0z is None : self.t0z = time.time()
343         t1z = time.time()
344         LP = t1z - self.t0z
345         self.t0z = t1z
346         self.Zangle += self.getZ(plf) * LP
347         return self.Zangle
348 
349 class HMC5883L(IMU):
350 
351     ADDRESS = HMC5883L_ADDRESS
352 
353     def __init__(self) :
354         #Class Properties
355         self.X = None
356         self.Y = None
357         self.Z = None
358         self.angle = None
359         self.Xoffset = 0.0
360         self.Yoffset = 0.0
361         self.Zoffset = 0.0
362         self.angle_offset = 0.0
363 
364         self.scale = 0.92
365 
366         self.write_byte(HMC5883L_CRA, 0b01110000)   # Set to 8 samples @ 15
    Hz
367         self.write_byte(HMC5883L_CRB, 0b00100000)   # 1.3 gain LSb / Gauss
    1090 (default)
368         self.write_byte(HMC5883L_MR, 0b00000000)    # Continuous sampling
369 
370     def getX(self):
371         self.X = (self.read_word_2c(HMC5883L_DO_X_H) - self.Xoffset) * self
    .scale
372         return self.X
373 
374     def getY(self):
375         self.Y = (self.read_word_2c(HMC5883L_DO_Y_H) - self.Yoffset) * self
    .scale
376         return self.Y
377 
378     def getZ(self):
379         self.Z = (self.read_word_2c(HMC5883L_DO_Z_H) - self.Zoffset) * self
    .scale
380         return self.Z
381 
382     def getAngle(self):
383         bearing  = degrees(atan2(self.getY(), self.getX())) + self.
    angle_offset
384         if (bearing < 0):
385             bearing += 360
386         bearing += self.angle_offset
```

```
        if (bearing < 0):
            bearing += 360
        if (bearing > 360):
            bearing -= 360
        self.angle = bearing
        return self.angle

class BMP180(IMU):

    ADDRESS = BMP180_ADDRESS

    def __init__(self) :
        #Class Properties
        self.tempC = None
        self.tempF = None
        self.press = None
        self.altitude = None

        self.oversampling = 3        # 0..3

        self._read_calibratio_params()

    def _read_calibratio_params(self) :
        self.ac1_val = self.read_word_2c(BMP180_AC1,0)
        self.ac2_val = self.read_word_2c(BMP180_AC2,0)
        self.ac3_val = self.read_word_2c(BMP180_AC3,0)
        self.ac4_val = self.read_word(BMP180_AC4,0)
        self.ac5_val = self.read_word(BMP180_AC5,0)
        self.ac6_val = self.read_word(BMP180_AC6,0)
        self.b1_val = self.read_word_2c(BMP180_B1,0)
        self.b2_val = self.read_word_2c(BMP180_B2,0)
        self.mc_val = self.read_word_2c(BMP180_MC,0)
        self.md_val = self.read_word_2c(BMP180_MD,0)

    def getTempC(self) :

        # print ("Calculating temperature...")
        self.write_byte(0xF4, 0x2E)
        time.sleep(0.005)

        ut = self.read_word(0xF6,0)

        x1 = ((ut - self.ac6_val) * self.ac5_val) >> 15
        x2 = (self.mc_val << 11) // (x1 + self.md_val)
        b5 = x1 + x2
        self.tempC = ((b5 + 8) >> 4) / 10.0

        return self.tempC

    def getTempF(self) :
        #print ("Calculating temperature (Fahrenheit)...")
        self.tempF = self.getTempC() * 1.8 + 32

        return self.tempF

    def getPress(self) :
```

```
443
444         # print ("Calculating temperature...")
445         self.write_byte(0xF4, 0x2E)
446         time.sleep(0.005)
447
448         ut = self.read_word(0xF6,0)
449
450         x1 = ((ut - self.ac6_val) * self.ac5_val) >> 15
451         x2 = (self.mc_val << 11) // (x1 + self.md_val)
452         b5 = x1 + x2
453
454         #print ("Calculating pressure...")
455         self.write_byte(0xF4, 0x34 + (self.oversampling << 6))
456         time.sleep(0.04)
457
458         msb = self.read_byte(0xF6)
459         lsb = self.read_byte(0xF7)
460         xsb = self.read_byte(0xF8)
461
462         up = ((msb << 16) + (lsb << 8) + xsb) >> (8 - self.oversampling)
463
464         b6 = b5 - 4000
465         b62 = b6 * b6 >> 12
466         x1 = (self.b2_val * b62) >> 11
467         x2 = self.ac2_val * b6 >> 11
468         x3 = x1 + x2
469         b3 = (((self.ac1_val * 4 + x3) << self.oversampling) + 2) >> 2
470
471         x1 = self.ac3_val * b6 >> 13
472         x2 = (self.b1_val * b62) >> 16
473         x3 = ((x1 + x2) + 2) >> 2
474         b4 = (self.ac4_val * (x3 + 32768)) >> 15
475         b7 = (up - b3) * (50000 >> self.oversampling)
476
477         press = (b7 * 2) // b4
478         #press = (b7 / b4) * 2
479
480         x1 = (press >> 8) * (press >> 8)
481         x1 = (x1 * 3038) >> 16
482         x2 = (-7357 * press) >> 16
483         self.press = ( press + ((x1 + x2 + 3791) >> 4) ) / 100.0
484
485         return self.press
486
487     def getAltitude(self) :
488         #   print ("Calculating altitude...")
489         self.altitude = 44330 * (1 - ((self.getPress() / STANDARD_PRESSURE)
    ** 0.1903))
490         return self.altitude
491
492 if __name__ == "__main__":
493     # if run directly we'll just create an instance of the class and output
494     # the current readings
495
496     sensors = gy801()
497     adxl345 = sensors.accel
```

```
498
499     print ("\033[1;34;40mADXL345 on address 0x%x:" % (ADXL345_ADDRESS))
500     print ("   x = %.3f m/s2" % ( adxl345.getX() ))
501     print ("   y = %.3f m/s2" % ( adxl345.getY() ))
502     print ("   z = %.3f m/s2" % ( adxl345.getZ() ))
503     print ("   x = %.3fG" % ( adxl345.Xg ))
504     print ("   y = %.3fG" % ( adxl345.Yg ))
505     print ("   z = %.3fG" % ( adxl345.Zg ))
506     print ("   x = %.3f" % ( adxl345.Xraw ))
507     print ("   y = %.3f" % ( adxl345.Yraw ))
508     print ("   z = %.3f" % ( adxl345.Zraw ))
509     print ("   pitch = %.3f" % ( adxl345.getPitch() ))
510     print ("   roll = %.3f" % ( adxl345.getRoll() ))
511
512     gyro = sensors.gyro
513
514     gyro.getXangle()
515     gyro.getYangle()
516     gyro.getZangle()
517
518     print ("\033[1;33;40mL3G4200D on address 0x%x:" % (L3G4200D_ADDRESS))
519     print ("   Xangle = %.3f deg" % ( gyro.getXangle() ))
520     print ("   Yangle = %.3f deg" % ( gyro.getYangle() ))
521     print ("   Zangle = %.3f deg" % ( gyro.getZangle() ))
522
523     print ("   Xangle = %.3f" % ( gyro.getX() ))
524     print ("   Yangle = %.3f" % ( gyro.getY() ))
525     print ("   Zangle = %.3f" % ( gyro.getZ() ))
526
527     print ("   Xangle = %.3f raw" % ( gyro.getRawX() ))
528     print ("   Yangle = %.3f raw" % ( gyro.getRawY() ))
529     print ("   Zangle = %.3f raw" % ( gyro.getRawZ() ))
530
531     compass = sensors.compass
532
533     print ("\033[1;32;40mHMC5883L on address 0x%x:" % (HMC5883L_ADDRESS))
534     print ("   X = %.3f " % ( compass.getX() ))
535     print ("   Y = %.3f " % ( compass.getY() ))
536     print ("   Z = %.3f " % ( compass.getZ() ))
537     print ("   Angle = %.3f deg" % ( compass.getAngle() ))
538
539     barometer = sensors.baro
540
541     tempC = barometer.getTempC()
542     tempF = barometer.getTempF()
543     press = barometer.getPress()
544     altitude = barometer.getAltitude()
545
546     print ("\033[1;31;40mBMP180 on address 0x%x:" % (BMP180_ADDRESS))
547     print ("   Temp: %f C (%f F)" %(tempC,tempF))
548     print ("   Press: %f (hPa)" %(press))
549     print ("   Altitude: %f m s.l.m" %(altitude))
550     print ("\033[0m")
```

اتصال GPS به رزبری



شکل ۱۱: اتصال GPS به Raspberry

Pin On Raspberry	Pin on GPS
Raspberry pi 5V	Neo 6M VCC
Raspberry pi GND	Neo 6M GND
Raspberry pi TX	Neo 6M RX
Raspberry pi RX	Neo 6M TX

حال برای شناساندن ماژول به رزبری خطوط زیر را به فایل /boot/config سیستم‌عامل اضافه می‌کنیم.

```
$ sudo nano /boot/config.txt
```

```
dtparam=spi=on
dtoverlay=pi3-disable-bt
core_freq=250
enable_uart=1
force_turbo=1
```

سپس خط زیر را در فایل /boot/cmdline.txt جایگزین می‌کنیم.

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles
```

سپس سیستم را reboot می‌کنیم.

سپس از طریق دستورات زیر، خروجی سریال tty0AMA0 را غیرفعال می‌کنیم تا بتوانیم از طریق کد پایتون با آن ارتباط برقرار کنیم.

$ sudo systemctl stop serial-getty@ttyAMA0.service $ sudo systemctl disable serial-getty@ttyAMA0.service

حال با استفاده از قطعه کد زیر می‌توانیم نتایج را از GPS دریافت کنیم. دقت بفرمایید که این GPS در محیط‌های سرپوشیده عمل نخواهد کرد.

این کد در هر مرحله ورودی سریال را می‌خواند و اگر سرآیند آن نمایان‌گر دیتای GPS بود، آن پیام را parse می‌کند و درخت نحوی آن را می‌سازد و در نهایت طول و عرض جغرافیایی آن را می‌خواند.

```
import serial
import time
import string
import pynmea2

while True:
    port="/dev/ttyAMA0"
    ser=serial.Serial(port, baudrate=9600, timeout=0.5)
    dataout = pynmea2.NMEAStreamReader()
    newdata=ser.readline()

    if newdata[0:6] == "$GPRMC":
        newmsg=pynmea2.parse(newdata)
        lat=newmsg.latitude
        lng=newmsg.longitude
        gps = "Latitude=" + str(lat) + "and Longitude=" + str(lng)
        print(gps)
```

```
Latitude=35.7035515and Longitude=51.35109466666667
Latitude=35.7035545and Longitude=51.3510985
```

تخمین مکان

گام‌های الگوریتم عبارت است از

۱. در ابتدا می‌بایست شتاب‌سنج را در یک موقعیت مکانی و جهتی مشخص که آن را می‌دانیم قرار داد.

۲. داده‌ها را از شتاب‌سنج می‌خوانیم.

۳. در این گام با استفاده از یکی از الگوریتم‌هایی که در ادامه مطرح می‌شود جهت (orientation) را بدست می‌آوریم. جهت را به فرم quaternion نمایش می‌دهیم. پس از پایان الگوریتم درباره‌ی quaternion توضیحاتی داده خواهد شد.

۴. با استفاده از جهت فعلی بدست آمده یک ماتریس چرخش می‌سازیم (rotation matrix). این ماتریس مبدا داده‌های شتاب‌سنج را که نسبت به خود این دستگاه هستند را به مبدا دنیای حقیقی (۳ محور مکان) تبدیل می‌کند. در فرمول

زیر R همان ماتریس چرخش است که در شتابی که از دستگاه خوانده می‌شود ضرب می‌شود. سمت چپ تساوی نیز شتاب‌های خوانده شده از دستگاه در دستگاه مختصات مبدا است.

۵. با استفاده از انتگرال گیری روی داده‌های تبدیل شده سرعت دستگاه را بدست می‌آوریم.

۶. دوباره با استفاده از انتگرال گیری روی سرعت بدست آمده شتاب دستگاه را بدست می‌آوریم.

۷. برو به گام ۲

الگوریتم فوق ساده‌ترین راه استفاده از خروجی شتاب‌سنج برای بدست آوردن مکان است. با این حال، همه سنسورها دارای سوگیری هستند، بنابراین وقتی خروجی محاسبه می‌شود، در تخمین‌های سرعت، موقعیت و جهت‌گیری ممکن است دچار خطا شویم (زیرا آنها تخمین هستند و اندازه‌گیری نیستند). به طور خاص در گام تخمین برای محاسبه جهت‌گیری روش‌های گوناگونی وجود دارد اما پایه تمامی آنها بدست آوردن سرعت‌های زاویه‌ای است. الگوریتم‌های مختلف سعی دارند تا با استفاده از پارامترهای دیگر که شتاب‌سنج خروجی می‌دهد به اصلاح خطاهای موجود در محاسبه این سرعت‌های زاویه‌ای بپردازند. برای مثال، فیلتر Madgwick در تخمین جهت‌گیری بهتر از انتگرال گیری از خروجی شتاب‌سنج عمل می‌کند، زیرا فیلتر Madgwick از خروجی بردار جاذبه‌های شتاب‌سنج برای حذف خطاهای محاسبه جهت استفاده می‌کند. در این گزارش ما از الگوریتم Madgwick استفاده میکنیم و در قسمت‌های بعد پروژه به بررسی الگوریتم‌های Mahony و Extended Kalman Filter و انتگرال‌گیری می‌پردازیم.

در گام ۳ام الگوریتم هدف محاسبه جهت‌گیری در فرمت چهارگانه[۱۱] است. چهارگانه‌های واحد، یک نماد ریاضی مناسب برای نمایش جهت‌گیری‌های فضایی و چرخش عناصر در فضای سه بعدی ارائه می دهند. به طور خاص، آنها اطلاعات مربوط به یک چرخش محور-زاویه حول یک محور دلخواه را رمزگذاری می کنند. هنگامی که برای نشان دادن یک جهت (چرخش نسبت به یک سیستم مختصات مرجع) استفاده می شود، به آنها چهارتایی جهت یا چهارتایی نگرش می‌گویند.

در فضای سه بعدی، طبق قضیه چرخش اویلر، هر چرخش یا دنباله‌ای از چرخش یک جسم صلب یا سیستم مختصات حول یک نقطه ثابت، معادل یک چرخش منفرد در یک زاویه معین در مورد یک نقطه ثابت است. محور (به نام محور اویلر) که از نقطه ثابت می گذرد. محور اویلر معمولاً با یک بردار واحد نشان داده می‌شود. بنابراین، هر چرخش در سه بعدی را می توان به صورت ترکیبی از یک بردار u و یک اسکالر نشان داد.

چهارگانه‌ها راه ساده‌ای را برای رمزگذاری این نمایش محور-زاویه در چهار عدد ارائه می دهند و می توان از آنها برای اعمال (محاسبه) چرخش متناظر به بردار موقعیت (x, y, z) استفاده کرد که نشان دهنده یک نقطه نسبت به مبدا در فضای سه بعدی است.

نسخه اولیه کد عبارت است از

[۱۱] quaternion

```
import time
from math import radians

import numpy as np
import quaternion
from ahrs.filters import Madgwick
from magno_gy import gy801


sensors = gy801()
gyro = sensors.gyro
accel = sensors.accel
compass = sensors.compass
barometer = sensors.baro

def read_accel():
    return accel.getX(), accel.getY(), accel.getZ()

def read_gyro():
    return radians(gyro.getX()), radians(gyro.getY()), radians(gyro.getZ())

def read_magnet():
    return compass.getX(), compass.getY(), compass.getZ()

madgwick_wo_magnet = Madgwick()
madgwick_w_magnet = Madgwick()
Q_w_magnet = [1., 0., 0., 0.]

t = 0
v_wo = np.array([0., 0., 0.])
v_w = np.array([0., 0., 0.])

x_wo = np.array([0., 0., 0.])
x_w = np.array([0., 0., 0.])


def calcualte_rotation_matrix(Q):
    np_quaternion = np.quaternion(Q[0], Q[1], Q[2], Q[3])
    return quaternion.as_rotation_matrix(np_quaternion)

class IMUData:
    def __init__(self, gyr, acc, mag) -> None:
        self.gyr = gyr
        self.acc = acc
        self.mag = mag

def get_imu_data() -> IMUData:
    acc = read_accel()
    gyr = read_gyro()
    mag = read_magnet()

    return IMUData(gyr, acc, mag)

class AccelartionCalculator:
    def __init__(self):
        self.q_prev = [1., 0., 0., 0.]
```

```
57
58      def get_q(self, imu_data: IMUData, dt):
59          pass
60
61      def get_acceleration(self, imu_data: IMUData, dt, a0=np.array([0, 0,
    0])):
62          q = self.get_q(imu_data, dt)
63          rotation_matrix = calcualte_rotation_matrix(q)
64          a = np.dot(rotation_matrix, imu_data.acc) - a0
65
66          self.q_prev = q
67          return a
68
69  class AccelartionCalculatorWithOutMagnet(AccelartionCalculator):
70      def get_q(self, imu_data: IMUData, dt):
71          q = madgwick_w_magnet.updateIMU(self.q_prev, gyr=imu_data.gyr, acc=
    imu_data.acc, dt=dt)
72          return q
73
74  class AccelartionCalculatorWithMagnet(AccelartionCalculator):
75      def get_q(self, imu_data: IMUData, dt):
76          q = madgwick_w_magnet.updateMARG(self.q_prev, gyr=imu_data.gyr, acc
    =imu_data.acc, mag=imu_data.mag, dt=dt)
77          return q
78
79
80  a_calculator_w = AccelartionCalculatorWithMagnet()
81  a_calculator_wo = AccelartionCalculatorWithOutMagnet()
82
83  prev_time = time.time()
84  imu_data = get_imu_data()
85  dt = time.time() - prev_time
86  prev_time = time.time()
87  a0_w = a_calculator_w.get_acceleration(imu_data, dt)
88  a0_wo = a_calculator_wo.get_acceleration(imu_data, dt)
89
90
91  prev_time = time.time()
92  while True:
93      t += 1
94      curr_time = time.time()
95      dt = curr_time - prev_time
96      prev_time = curr_time
97      imu_data = get_imu_data()
98
99      a_wo = a_calculator_wo.get_acceleration(imu_data, dt, a0=a0_wo)
100     a_w = a_calculator_w.get_acceleration(imu_data, dt, a0=a0_w)
101
102     v_wo += a_wo * dt
103     v_w += a_w * dt
104
105     x_wo += v_wo * dt
106     x_w += v_w * dt
107
108     if t  % 100 == 0:
109         print(f'acceleration without magnometer:\t\t{a_wo}')
```

```
        print(f'acceleration with magnometer:\t\t{a_w}')
        print('*'*30)
        print(f'velocity without magnometer:\t\t{v_wo}')
        print(f'velocity with magnometer:\t\t{v_w}')
        print('*'*30)
        print(f'location without magnometer:\t\t{x_wo}')
        print(f'location with magnometer:\t\t{x_w}')

        print('\n'*4)
```

فیلتر جهت مجویک[۱۲]

الگوریتم مجویک که توسط سباستین مجویک ارائه شده است، برای ژیروسکوپ و شتاب‌سنج‌های سه محوره قابل اعمال است.

این فیلتر از یک نمایش چهارگانه جهت برای توصیف ماهیت جهت‌گیری‌ها در فضای سه بعدی استفاده می‌کند و مشمول تکینگی‌های مرتبط با نمایش زاویه اویلر نمی‌شود و اجازه می‌دهد تا داده‌های شتاب‌سنج و مغناطیس‌سنج در یک الگوریتم گرادیان-نزولی به‌صورت تحلیلی مشتق‌شده و بهینه‌شده برای محاسبه جهت خطای اندازه گیری ژیروسکوپ استفاده شوند.

جنبه های خلاقانه این فیلتر عبارتند از

۱. یک پارامتر منفرد قابل تنظیم که توسط ویژگی های سیستم های قابل مشاهده تعریف می شود.

۲. یک الگوریتم گرادیان نزولی که به صورت تحلیلی مشتق شده و بهینه شده است که عملکرد را در نرخ نمونه‌برداری پایین ممکن می‌سازد.

۳. الگوریتم جبران اعوجاج مغناطیسی آنلاین

۴. جبران رانش سوگیری ژیروسکوپ

توجه کنید که سوگیری زمین نسبت به سنسور را اگر یک بردار چهار بعدی $\mathbf{q}_{\omega,t} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}$ در زمان t در نظر بگیریم، می‌توانیم با استفاده از دادگان زاویه‌ای ${}^{S}\omega = \begin{bmatrix} \cdot & \omega_x & \omega_y & \omega_z \end{bmatrix}$ بدست آمده در بازه زمانی Δt و انتگرال گیری نسبت به مشتق $\dot{\mathbf{q}}_t = \frac{1}{2}\mathbf{q}_{t-1}\omega_t$ عبارت

$$\begin{aligned} \mathbf{q}_{\omega,t} &= \mathbf{q}_{t-1} + \dot{\mathbf{q}}_{\omega,t}\Delta t \\ &= \mathbf{q}_{t-1} + \frac{1}{2}\left(\mathbf{q}_{t-1}\,{}^{\mathbf{S}}\omega_{\mathbf{t}}\right)\Delta t \end{aligned}$$

[۱۲] Madgwick Orientation Filter

مقادیر آنها را بدست آوریم.

اگر فرض کنیم که زمین یک مرجع ثابت باشد که $^{E}\mathbf{d} = \begin{bmatrix} . & d_x & d_y & d_z \end{bmatrix}$ و سرعت نسبت به آن به صورت $^{S}\mathbf{s} = \begin{bmatrix} . & s_x & s_y & s_z \end{bmatrix}$ اندازه‌گیری شود، می‌توانیم تابع هدف را به صورت

$$\begin{aligned} f(\mathbf{q}, {}^{E}\mathbf{d}, {}^{S}\mathbf{s}) &= \mathbf{q}^{*}\,{}^{E}\mathbf{d}\,\mathbf{q} - {}^{S}\mathbf{s} \\ &= \begin{bmatrix} ۲d_x(\frac{۱}{۲} - q_y^۲ q_z^۲) + ۲d_y(q_w q_z + q_x q_y) + ۲d_z(q_x q_z - q_w q_y) - s_x \\ ۲d_x(q_x q_y - q_w q_z) + ۲d_y(\frac{۱}{۲} - q_x^۲ q_z^۲) + ۲d_z(q_w q_x + q_y q_z) - s_y \\ ۲d_x(q_w q_y + q_x q_z) + ۲d_y(q_y q_z - q_w q_x) + ۲d_z(\frac{۱}{۲} - q_x^۲ q_y^۲) - s_z \end{bmatrix} \end{aligned}$$

بنویسیم به نحوی که جواب مساله با کمینه‌سازی این تابع حاصل آید. برای کمینه‌سازی این جواب از الگوریتم گرادیان-نزولی استفاده می‌کنیم.

روش ماهونی[۱۳]

این روش توسط رابرت ماهونی در سال ۲۰۰۸ معرفی شد. فرض این روش بر این است که ژیروسکوپ سرعت زاویه‌ای را در یک چارچون ثابت با خطای نویزی μ بدست می‌آورد.

$$\Omega_y = \Omega + b + \mu \in R^۳$$

یک شتاب‌سنج آرمانی سرعت را در چارچوبی که شتاب گرانش سمت پایین آن باشد اندازه می‌گیرد. اما در شرایط حقیقی،الزامی برای اینکه محورهای ژیروسکوپ هم راستا با محور های زمینی باشند وجود ندارد. بنابرین مقدار خروجی این فیلتر برابر خواهد بود با

$$\mathbf{a} = \mathbf{R}^T(\dot{\mathbf{v}} - \mathbf{g}.) + \mathbf{b}_a + \mu_a$$

که در آن $g.$ شتاب جاذبه زمین است.

همچنین برای سادگی محاسبات می‌توانیم مقادیر را نرمالایز کنیم.

$$\mathbf{v}_a = \frac{\mathbf{a}}{|\mathbf{a}|} \approx -\mathbf{R}^T e_۳$$

[۱۳] Mahoni Method

$$e_۳ = \frac{\mathbf{g}_۰}{|\mathbf{g}_۰|} = \begin{bmatrix} ۰ & ۰ & ۱ \end{bmatrix}^T$$

توجه کنید که میدان مغناطیسی را از طریق رابطه زیر با توجه به ماتریس دوران می‌توانیم بدست آوریم.

$$\mathbf{m} = \mathbf{R}^T\mathbf{h} + \mathbf{B}_m + \mu_b$$

به صورت مشابه نرمال‌سازی را انجام می‌دهیم.

$$\mathbf{v}_m = \frac{\mathbf{m}}{|\mathbf{m}|}$$

حال با استفاده از مقادیر فوق سعی می‌کنیم تا ماتریس دوران را تخمین بزنیم.

$$\mathbf{R} \approx \mathbf{R}_y = \underset{\mathbf{R}\in SO(۳)}{\arg\min}(\lambda_۱|e_۳ - \mathbf{R}\mathbf{v}_a|^۲ + \lambda_۲|\mathbf{v}_m^* - \mathbf{R}\mathbf{v}_m|^۲)$$

که ضرایب در آن با توجه به اطمینان به خروجی سنسورها انتخاب خواهند شد.

دو درجه آزادی که برای ماتریس دوران در نظر گرفته شده است با توجه به غلطش و جهت که از شتاب‌سنج و قطب‌نما بدست می‌آید حل می‌شود.

در نظر بگیرید که که

$$\dot{\mathbf{R}} = \mathbf{R}\lfloor\Omega\rfloor_\times$$

$$\Omega = \begin{bmatrix} \Omega_X & \Omega_Y & \Omega_Z \end{bmatrix}^T$$

$$\lfloor\Omega\rfloor_\times = \begin{bmatrix} ۰ & -\Omega_Z & \Omega_Y \\ \Omega_Z & ۰ & -\Omega_X \\ -\Omega_Y & \Omega_X & ۰ \end{bmatrix}$$

توجه کنید که خواهیم داشت

$$\lfloor \Omega \rfloor_\times \Omega = \Omega$$

بنابرین فرم کینماتیک برابر خواهد شد با

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}\mathbf{p}(\Omega)$$

که در آن $\mathbf{q} = \begin{pmatrix} q_w & \mathbf{q}_v \end{pmatrix} = \begin{pmatrix} q_w & q_x & q_y & q_z \end{pmatrix}$ چارگانه یکانی است و $\mathbf{p}(\Omega) = \begin{pmatrix} 0 & \Omega_X & \Omega_Y & \Omega_Z \end{pmatrix}$ چهارگانه متناظر با سرعت زاویه‌ای خواهد بود.

فیلتر کالمن توسعه‌یافته[14]

Filter Kalman Extended یکی از پرکاربردترین الگوریتم‌های جهان است و از آن برای محاسبه جهت به فرمت چهارگانه با مشاهدات ژیروسکوپ‌های سه محوری، شتاب‌سنج‌ها و مغناطیس‌سنج‌ها استفاده می‌شود.

در این روش حالت سیستم که با حرف x نمایش داده می‌شود، تابع خطی حالت قبلی است و مقادیری که در هر مرحله بدست می‌آید تابع حالت است که با استفاده از روابط زیر حاصل می‌آیند.

$$\begin{aligned} \mathbf{x}_t &= \mathbf{F}\mathbf{x}_{t-1} + \mathbf{w}_x \\ \mathbf{z}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{w}_z \end{aligned}$$

در روابط بالا x و z به ترتیب حالت سیستم و خروجی سیستم هستند و w_x و w_z نویز فرآیند و اندازه گیری هستند که از توزیع طبیعی بدست می آیند و F ماتریس انتقال و H نیز ماتریس مدل کننده اندازه است.

با استفاده از مدل‌های خطی متعددی از جمله مدل زیر با ماتریس‌های ثابت A و L می‌توان مدل‌سازی را انجام داد.

$$\dot{\mathbf{x}}_t = \frac{d\mathbf{x}_t}{dt} = \mathbf{A}\mathbf{x}_t + \mathbf{L}\mathbf{w}_t$$

با استفاده از فرم نمایی ماتریس می‌توانیم ماتریس F را به صورت زیر بدست آوریم.

$$\mathbf{F} = e^{\mathbf{A}\Delta t} = \mathbf{I} + \mathbf{A}\Delta t + \frac{(\mathbf{A}\Delta t)^2}{2!} + \frac{(\mathbf{A}\Delta t)^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{(\mathbf{A}\Delta t)^k}{k!}$$

[14] Extended Kalman Filter

روش کالمن روشی است که در سال ۱۹۶۰ برای تبدیل معادلات دیفرانسیل به فرم ماتریسی و در نتیجه آن به فرم بازگشتی است. این روش از دو بخش تشکیل شده است. بخش اول که پیش‌بینی را با توجه توجه به حالت فعلی انجام می‌دهد و بخش تصحیح که در آن تخمین اصلی انجام می‌شود.

$$\begin{aligned}\hat{\mathbf{x}}_t &= \mathbf{F}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t \\ \hat{\mathbf{P}}_t &= \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q}_t\end{aligned}$$

$$\begin{aligned}\mathbf{v}_t &= \mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t \\ \mathbf{S}_t &= \mathbf{H}\hat{\mathbf{P}}_t\mathbf{H}^T + \mathbf{R} \\ \mathbf{K}_t &= \hat{\mathbf{P}}_t\mathbf{H}^T\mathbf{S}_t^{-1} \\ \mathbf{x}_t &= \hat{\mathbf{x}}_t + \mathbf{K}_t\mathbf{v}_t \\ \mathbf{P}_t &= \hat{\mathbf{P}}_t - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^T\end{aligned}$$

حال میخواهیم در این بخش با استفاده از چهارگانه q محورها را در فیلتر کالمن تخمین بزنیم.

تصور می‌شود در این روش تمام سنسورها در گام زمانی مشخص که با فرکانس f مشخص می‌شود داده خود را ارسال می‌کنند.

$$f = \frac{1}{\Delta t}$$

معادلا

$$t_n = t_. + n\Delta t$$

حال در این سیستم فرض می‌کنیم که q حالت سیستم و ω کنترل‌کننده سیستم باشد. بنابرین خواهیم داشت.

$$\mathbf{x} \quad \mathbf{q} = \begin{bmatrix} q_w & \mathbf{q}_v \end{bmatrix}^T = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T$$

$$\mathbf{u} \quad \boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

بنابرین انتقال را می‌توانیم به صورت زیر مدل کنیم.

$$\dot{\mathbf{q}}_t = \mathbf{A}\mathbf{q}_{t-1} + \mathbf{B}\boldsymbol{\omega}_t + \mathbf{w}_{\mathbf{q}}$$

$$\hat{\mathbf{q}}_t = \mathbf{F}\mathbf{q}_{t-1} = e^{\mathbf{A}\Delta t}\mathbf{q}_{t-1}$$

مرحله تخمین را می‌توانیم به صورت زیر با انتگرال گیری انجام دهیم.

$$\hat{\mathbf{q}}_t = \mathbf{q}_{t-1} + \int_{t-1}^{t} \boldsymbol{\omega}\, dt$$

با استفاده از فرم اویلری می‌توانیم چهارگانه q را به صورت زیر بازنویسی کنیم.

$$\hat{\mathbf{q}}_t = \left[\cos\left(\frac{\|\boldsymbol{\omega}\|\Delta t}{2}\right)\mathbf{I}_4 + \frac{2}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|\Delta t}{2}\right)\boldsymbol{\Omega}_t\right]\mathbf{q}_{t-1}$$

که در آن

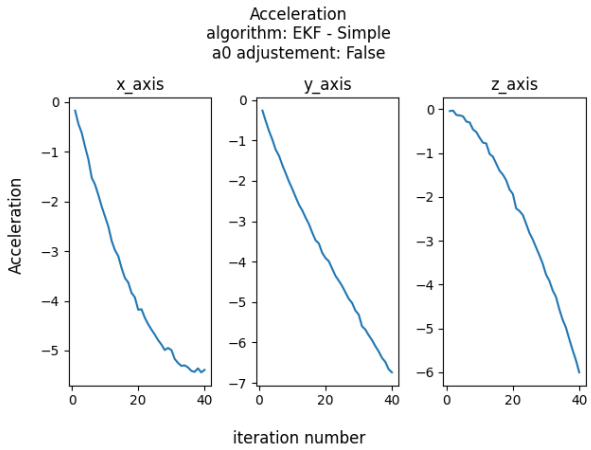
$$\boldsymbol{\Omega}_t = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & \lfloor\boldsymbol{\omega}\rfloor_\times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

با استفاده از بسط تیلور می‌توانیم بنویسیم.

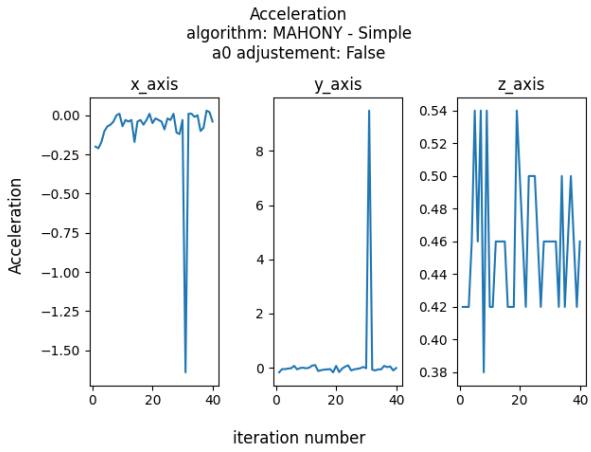
$$\begin{aligned} \hat{\mathbf{q}}_t = & \left(\mathbf{I}_4 + \tfrac{1}{2}\boldsymbol{\Omega}_t\Delta t + \tfrac{1}{2!}\left(\tfrac{1}{2}\boldsymbol{\Omega}_t\Delta t\right)^2 + \tfrac{1}{3!}\left(\tfrac{1}{2}\boldsymbol{\Omega}_t\Delta t\right)^3 + \cdots\right)\mathbf{q}_{t-1} \\ & + \tfrac{1}{4}\dot{\boldsymbol{\Omega}}_t\Delta t^2\mathbf{q}_{t-1} + \left[\tfrac{1}{12}\dot{\boldsymbol{\Omega}}_t\boldsymbol{\Omega}_t + \tfrac{1}{24}\boldsymbol{\Omega}_t\dot{\boldsymbol{\Omega}}_t + \tfrac{1}{12}\ddot{\boldsymbol{\Omega}}_t\right]\Delta t^3\mathbf{q}_{t-1} + \cdots \end{aligned}$$

نتایج اولیه

در شکل‌های زیر نتایج اولیه اجرای ۳ الگوریتم ذکر شده را مشاهده می‌کنیم. این داده‌ها در حالی بدست آمده که ژیروسکوپ به طور ثابت در یک مکان قرار داشته است.

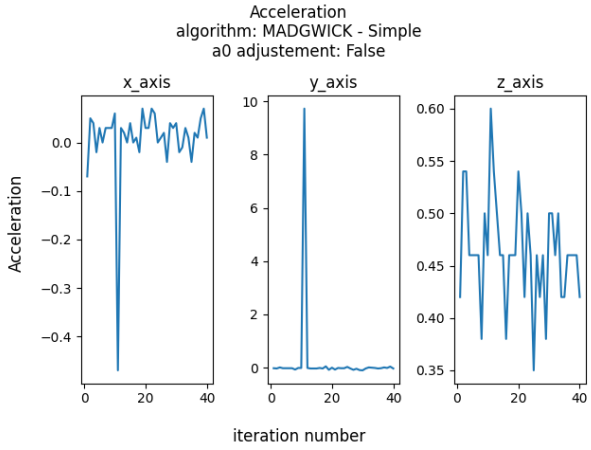


شکل ۱۲: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



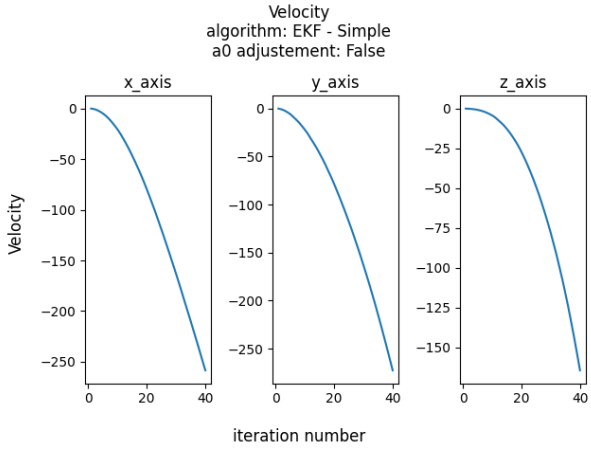
شکل ۱۳: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی

همانطور که مشاهده می‌شود، فیلتر کالمن خطای زیادی در شتاب دارد و پس از مدتی اندازه شتاب در هر یک از محورهای اصلی شروع به افزایش کرده‌اند. همچنین در الگوریتم‌های ماهونی و مجویک در محورهای x و y بعضا نویز زیادی وجود دارد. به طوری که باعث شده که اندازه شتاب در حالت سکون تا ۸ متر بر مجذور ثانیه افزایش پیدا کند. علاوه بر این، به طور کلی در این دو الگوریتم اخیر مشاهده می‌کنیم که علی رغم ساکن بودن دستگاه، هر سه محور دارای نویز کمی هستند و باعث شده تا مقدار شتاب در هر یک از جهات بین ۰ و ۱ نوسان داشته باشد. با گذشت زمان این خطاها روی هم انباشته می‌شوند و باعث می‌شوند که موقعیت و سرعت در طی مدت کوتاهی تغییرات زیادی داشته باشند.



شکل ۱۴: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک

در ادامه به بررسی سرعت و مکان بدست آمده از نمودار شتاب‌های بالا می‌پردازیم:

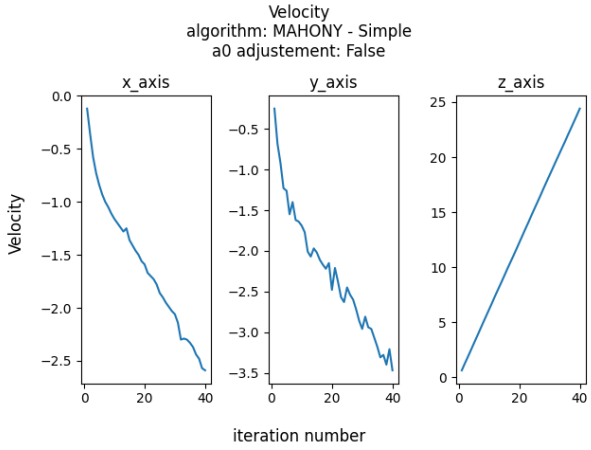


شکل ۱۵: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن

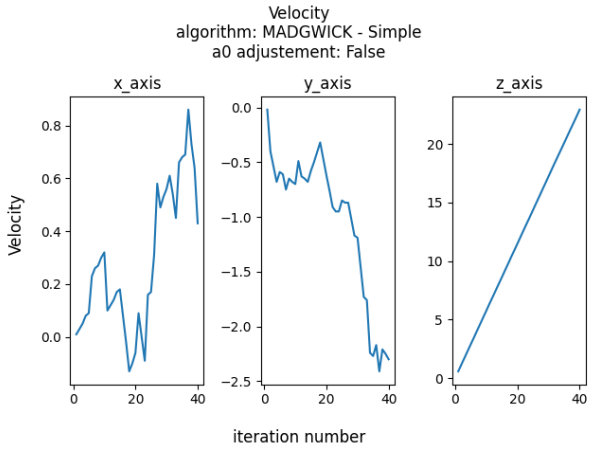
همانطور که انتظار می‌رفت با توجه به خطای زیاد کالمن فیلتر سرعت و مکان به شدیدا دچار انجراف شده‌اند. به طوری مکان در هر محور بالغ بر هزار متر جابه‌جا شده است در حالی که دستگاه ثابت بوده است.

برای دو الگوریتم دیگر همچنان خطا داریم ولی این خطا به نسبت کمتر از فیلتر کالمن است. برای مثال در محور مکان با استفاده از این دو فیلتر در بازه زمانی مشخص شده حداکثر ۶۰۰ متر جابه‌جا شده‌ایم.

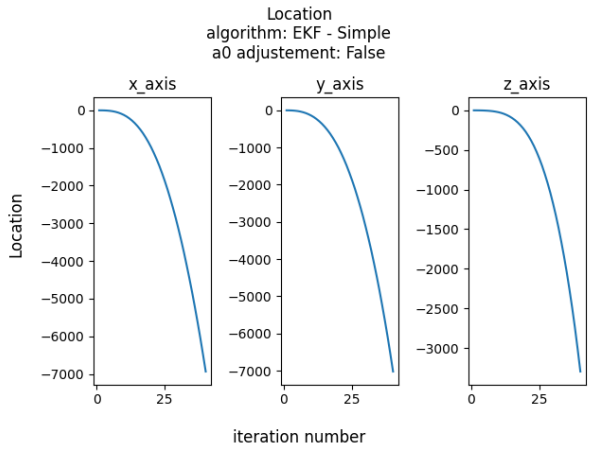
برای رفع این خطا در گام بعدی ما از روشی استفاده می‌کنیم تا شتاب‌اولیه دستگاه را دقیق تر تخمین بزنیم. بدیهی است که شتاب اولیه یا همان شتاب کرده زمین می‌بایست به عنوان یک ثابت از بردار شتاب محاسبه شده کم شود. در حال حاضر شتاب اولیه برابر است با اولین شتاب خوانده شده از دستگاه.



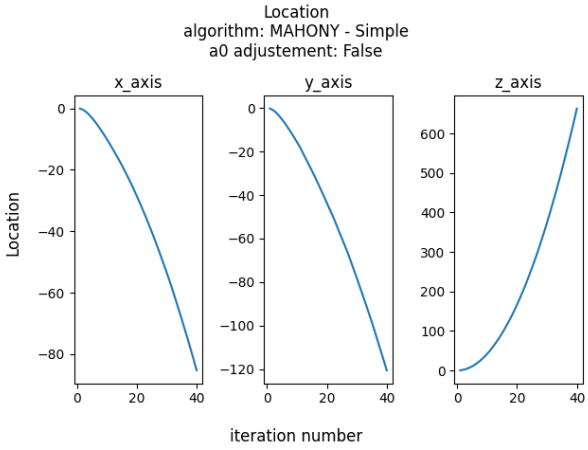
شکل ۱۶: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



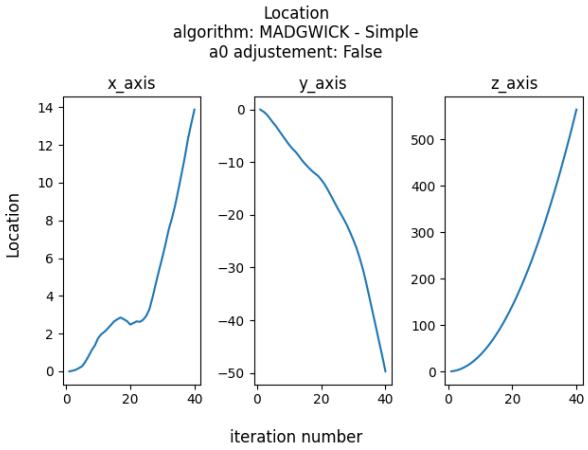
شکل ۱۷: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



شکل ۱۸: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



شکل ۱۹: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



شکل ۲۰: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک

بررسی توان ورودی دستگاه

با توجه به خطای زیاد در دادگان، برای اطمینان از صحت عملکرد دستگاه ما به بررسی توان ورودی ماژول ژیروسکوپ پرداختیم. در حالت فعلی این دستگاه از پایه ۳.۳ ولت رزبری‌پای برق خود را تامین می‌کند. با توجه به اینکه این احتمال وجود دارد که رزبری‌پای نتواند جریان مورد نیاز ژیروسکوپ را تامین کند، ممکن است دستگاه با خطا روبرو شود. برای همین ما ورودی‌های دستگاه را به ترانس متصل کردیم و از هر دو ورودی ۵ و ۳.۳ نیز استفاده کردیم. هرچند تغییری در عملکرد دستگاه ایجاد نشد. در نتیجه تا حد خوبی اطمینان حاصل کردیم که خطا دستگاه به خاطر مشکلات توانی نیست.

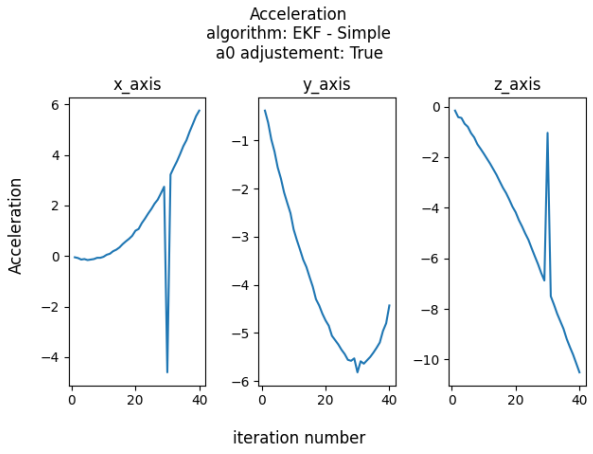
تخمین دقیق‌تر شتاب اولیه

با توجه به مشاهدات ما در قسمت قبل می‌توان دید که دستگاه در حالت ساکن نویز زیادی دارد. در نتیجه محاسبه شتاب با استفاده از تنها یک داده خطای زیادی دارد. برای حل این مشکل ما شتاب اولیه محاسبه شده را به صورت مرحله‌ای بهبود می‌دهیم. در ابتدا شتاب خوانده شده از دستگاه را به عنوان شتاب ثابت اولیه درنظر میگیرم. در مرحله بعد شتاب دستگاه در حالت ساکن را با کم کردن شتاب ثابت اولیه محاسبه می‌کنیم. با توجه به اینکه که دستگاه ثابت است انتظار می‌رود تا نتیجه برابر بردار صفر باشد، اما خروجی محاسبه شده این گونه نیست. در نتیجه شتاب محاسبه شده را می‌توان به عنوان خطای محاسبه شتاب ثابت در نظر گرفت و شتاب ثابت محاسبه شده را به اندازه درصدی از این خطا تصحیح کرد.

$$a_{gravity} = a_{gravity} + a_{calculated} * C$$

در محاسبات ما ثابت C را برابر ۱.۰ در نظر گرفتیم. اگر مقدار C خیلی بزرگ باشد باعث می‌شود که شتاب ثابت نتواند به مقدار حقیقی خود میل کند. از طرفی اگر این مقدار خیلی کوچک باشد مقدار تصحیح در هر مرحله بسیار کم می‌شود که باعث کند شدن این فرایند می‌شود. در بین مقادیر مختلف، ۱.۰ مقداری بود که برای هر دو شرایط عملکرد مناسبی داشت.

در نمودارهای زیر نتایج حاصل از این تغییر را مشاهده می‌کنیم.

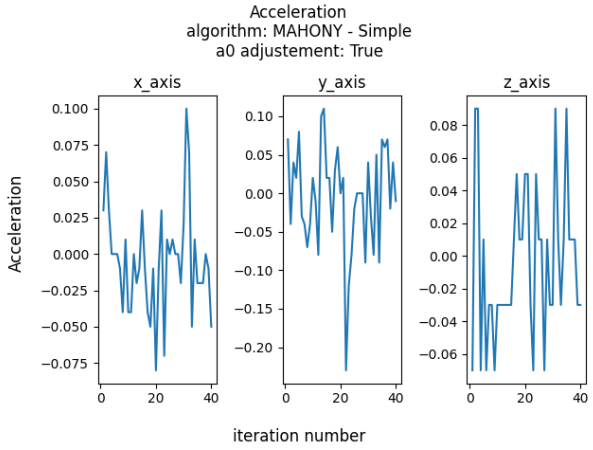


شکل ۲۱: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن

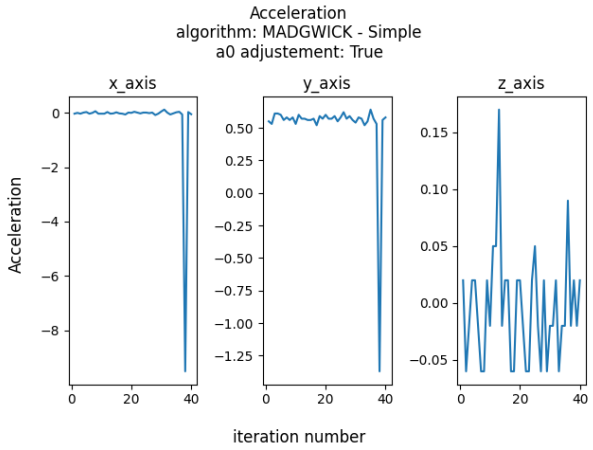
همانطور که مشاهده می‌کنید، میزان خطا در الگوریتم ماهونی و مجویک بسیار کاهش یافت. ولی مثلا در الگوریتم مجویک در محور x یکبار دچار نویز شد که باعث شد اندازه شتاب موقتا به ۸ متر بر ثانیه افزایش پیدا کند و بازگردد.

در ادامه به بررسی پارامترهای سرعت و مکان بدست آمده از این شتاب‌ها می‌پردازیم.

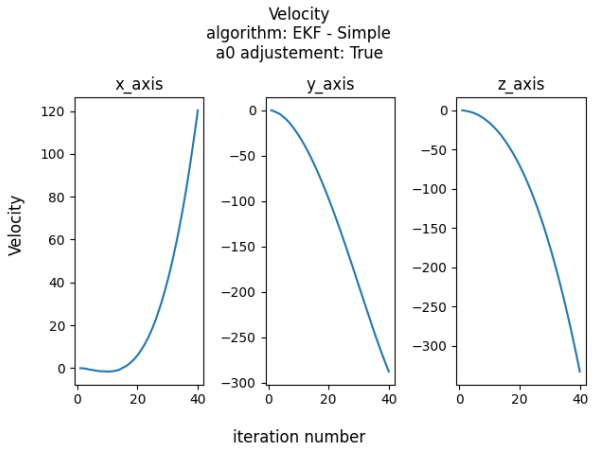
همانطور که دیده می‌شود پارامترهای سرعت و مکان نسبت به حالت قبلی بسیار بهبود یافتند. به طوری که در قسمت مکان در الگوریتم ماهونی در هر محور به اندازه حداکثر ۱۰ متر جابه‌جا شده است. هر چند که این خطا همچنان بسیار زیاد است، در



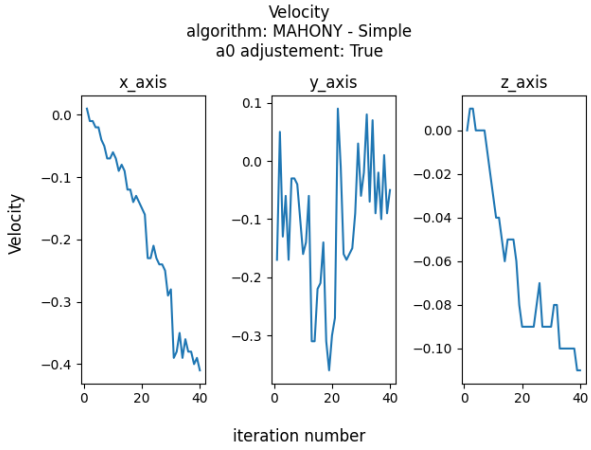
شکل ۲۲: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



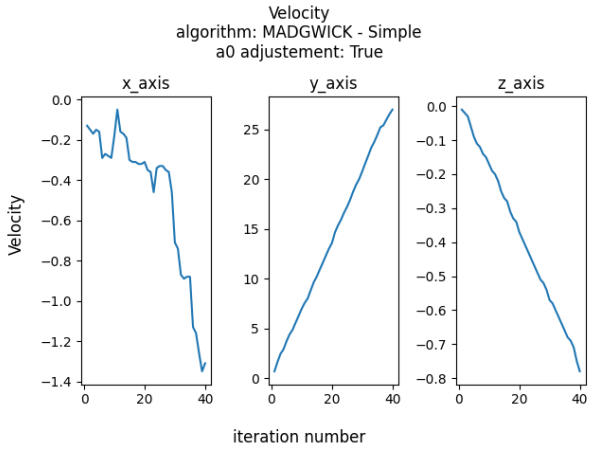
شکل ۲۳: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



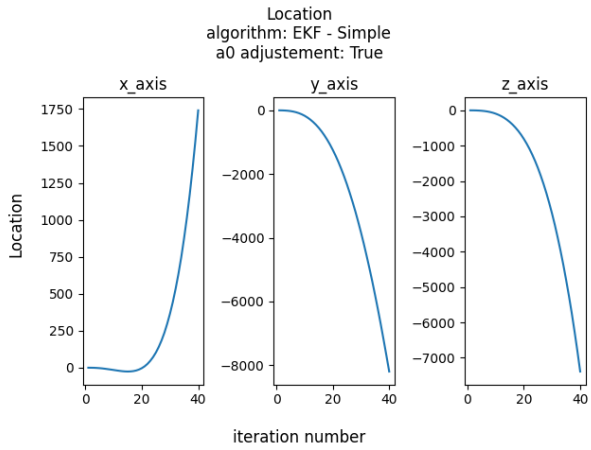
شکل ۲۴: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



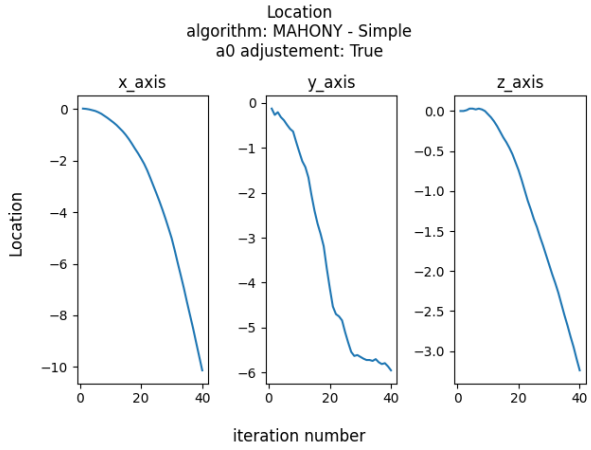
شکل ۲۵: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



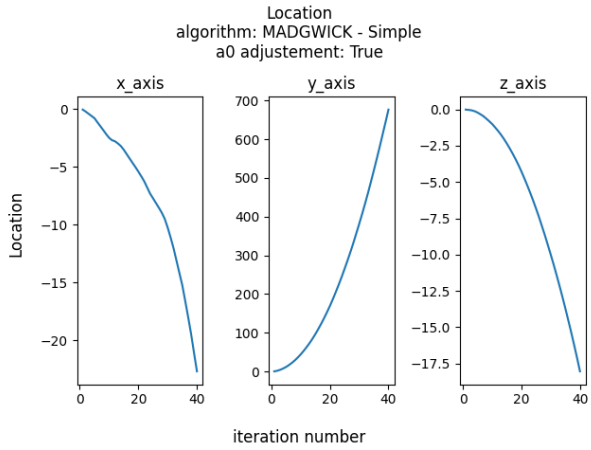
شکل ۲۶: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



شکل ۲۷: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



شکل ۲۸: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



شکل ۲۹: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک

قسمت های بعد روش‌هایی ارایه می‌دهیم تا خطا را کم کنیم.

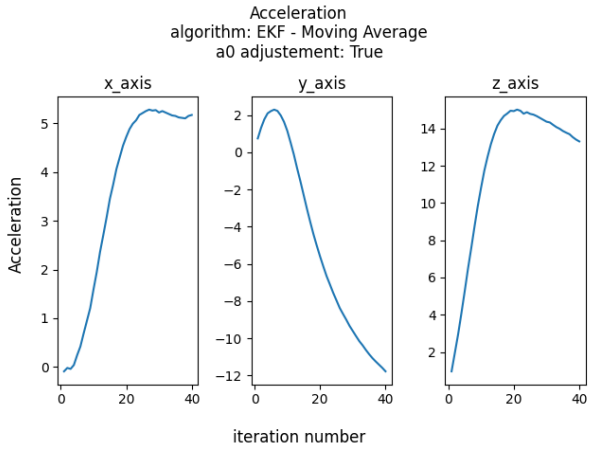
استفاده از میانگین لغزان

یکی از روش‌های مرسوم برای حذف یا کم‌اثر کردن داده‌های نویز استفاده از میانگین لغزان است. ما برای محاسبه شتاب از این روش استفاده کردیم. پس از این که شتاب را با استفاده از شتاب ثابت اولیه بدست‌اوردیم، بردار حاصله را با درصدی از شتاب قبلی جمع می‌کنیم. این کار را هر مرحله برای محاسبه شتاب انجام می‌دهیم.

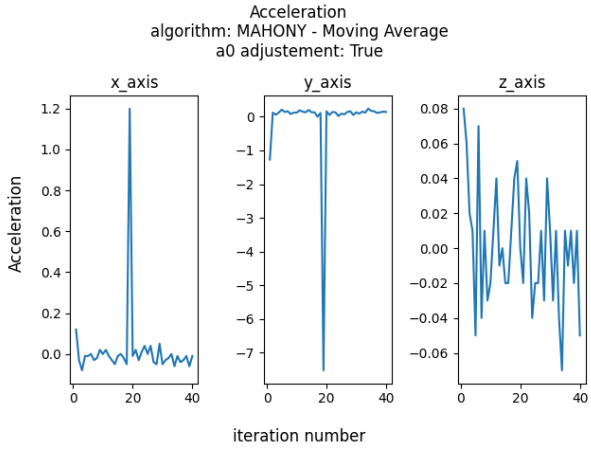
$$a = a_{calculated} * RATE + a_{previous} * (۱ - RATE)$$

این روش معادل میانگین لغزان نمایی است، چراکه هر بار که می‌خواهیم شتاب محاسبه شده را با شتاب جدید اضافه کنیم آن را در یک ضریبی ضرب میکنیم. نتایج شتاب، سرعت، مکان را ادامه مشاهده می‌کنیم.

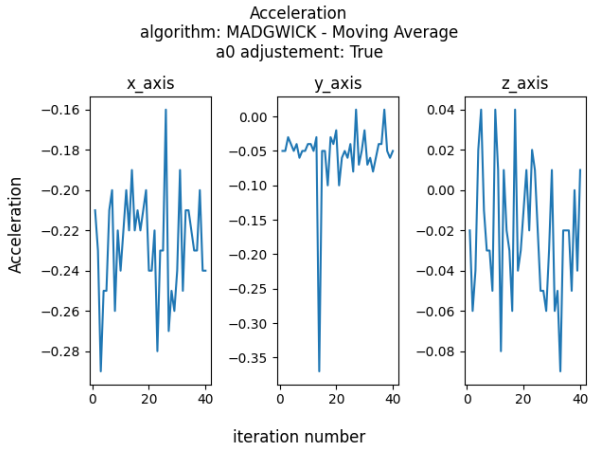
در این روش اندکی بهبود در نتایج داشتیم. همانگونه که مشاهده می‌کنید، با الگوریتم کالمن، نتایج خوبی بدست نمی‌آید و شی دائما در حال حرکت است. در روش‌های ماهونی و مجویک، شتاب بدست آمده در اطراف صفر است اما در نهایت برآیند آن ایجاد سرعتی میکند که قابل چشم‌پوشی نیست. و در نهایت این موضوع باعث می‌شود که مکان دارای تغییر باشد اما تغییر آن در بازه‌های زمانی کوتاه قابل صرف نظر است.



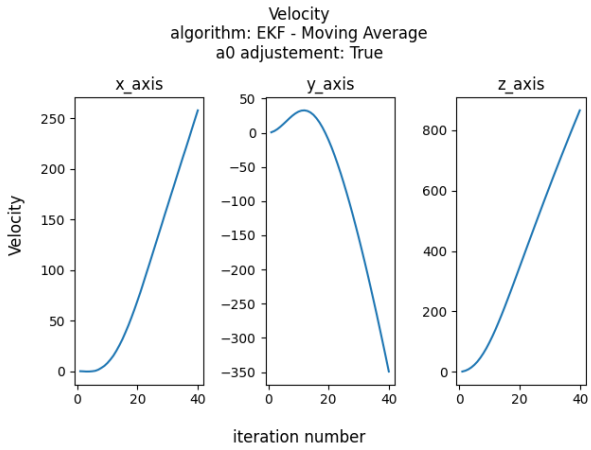
شکل ۳۰: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



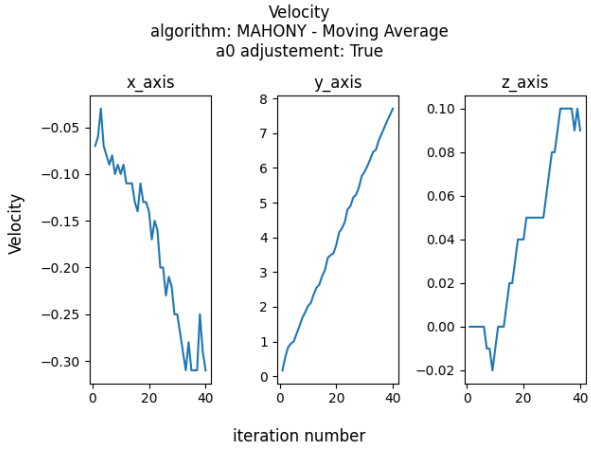
شکل ۳۱: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



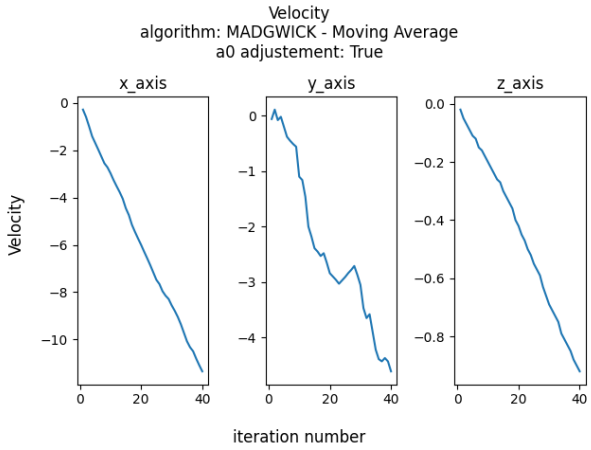
شکل ۳۲: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



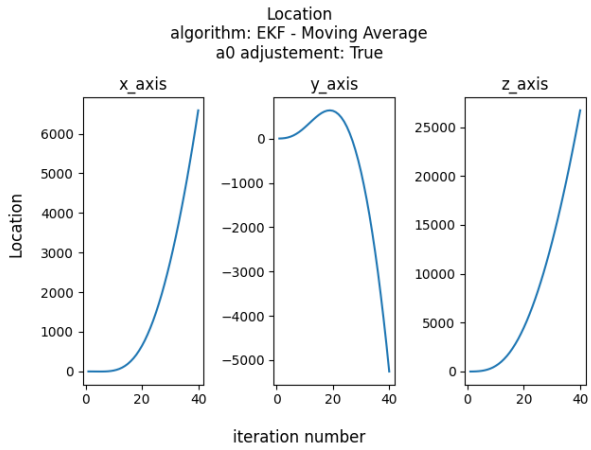
شکل ۳۳: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



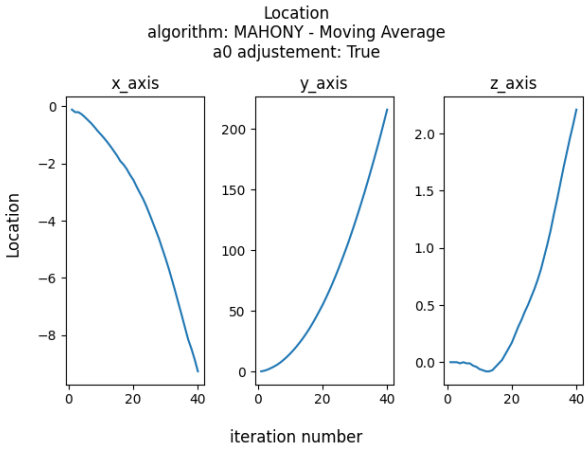
شکل ۳۴: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



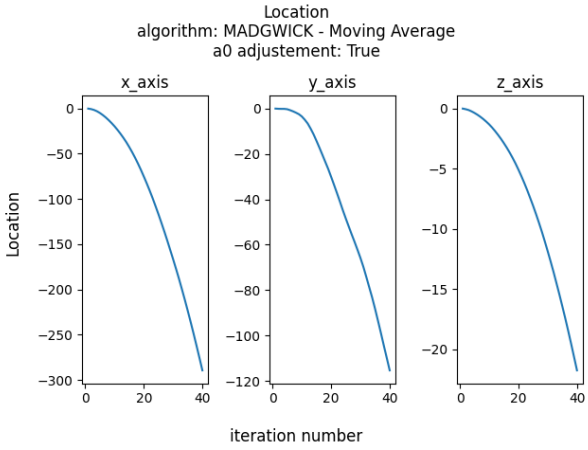
شکل ۳۵: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



شکل ۳۶: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



شکل ۳۷: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



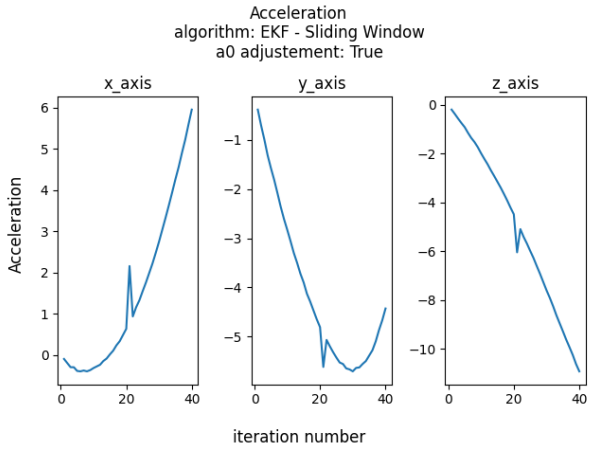
شکل ۳۸: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک

استفاده از پنجره لغزان

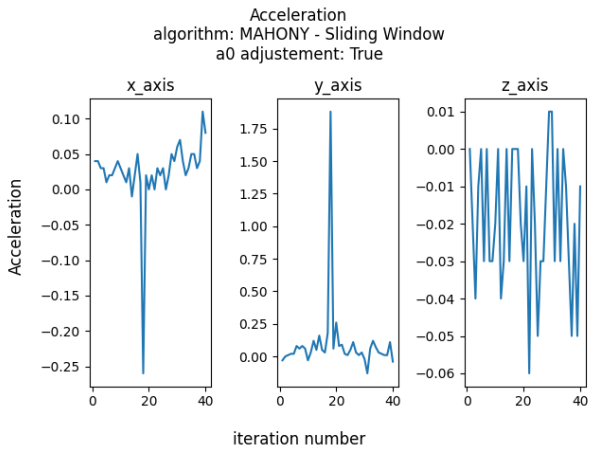
این روش نیز مانند روش میانگین لغزان از روش‌هایی است که به حذف نویز از دادگان کمک می‌کند. برای محاسبه شتاب با استفاده از این روش به این صورت عمل می‌کنیم که شتاب محاسبه شده فعلی را با شتاب‌هایی که در مرحله قبل بدست آمده جمع می‌کنیم و میانگین می‌گیریم. به طور خاص اندازه پنجره‌ای که در نظر گرفتیم ۵ بود.

در این روش نیز مشابه حالت قبل، روش کالمن نتایج با خطای بالایی را بدست می‌دهد. اما در مقابل روش ماهونی و مخصوصا روش مجدویک نتایج خوبی را ایجاد می‌کند. اما مشابه آن ما نیز در اینجا خطاهای قابل توجهی داریم که باعث ایجاد سرعت و در نتیجه تغییر مکان قابل توجهی می‌شوند که البته در بازه زمانی کوتاه قابل چشم‌پوشی است.

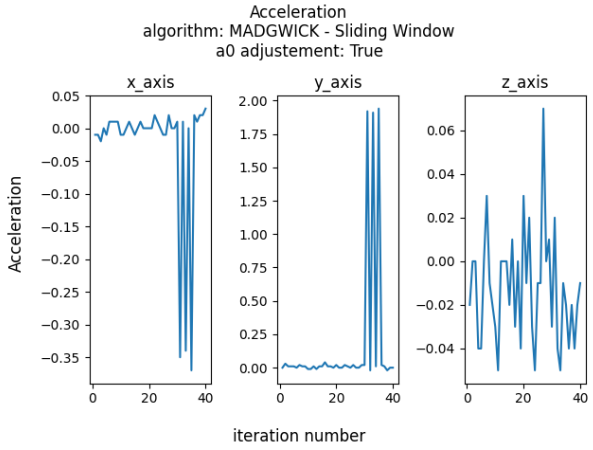
نتایج حاصل از این روش در ادامه آمده است.



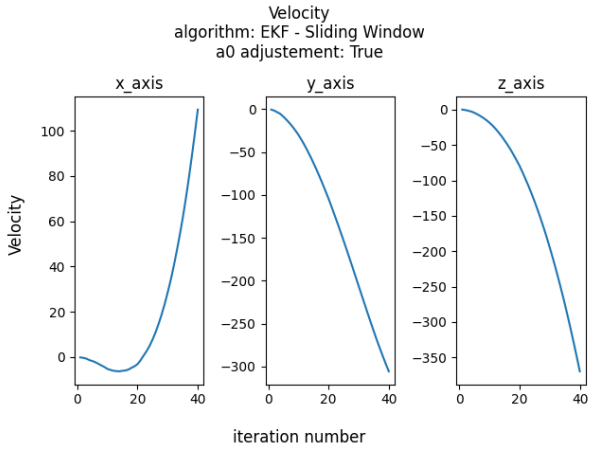
شکل ۳۹: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



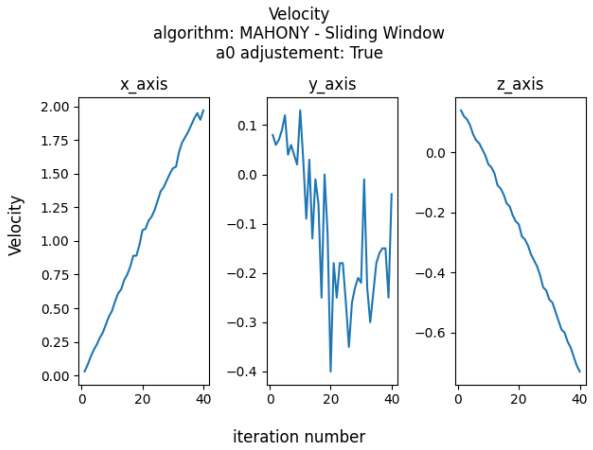
شکل ۴۰: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



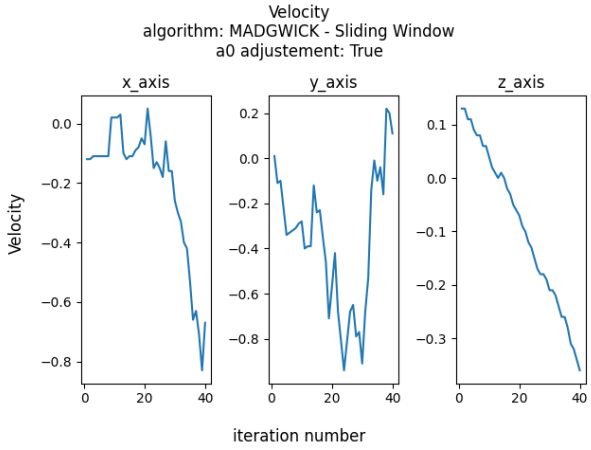
شکل ۴۱: نمودار شتاب بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



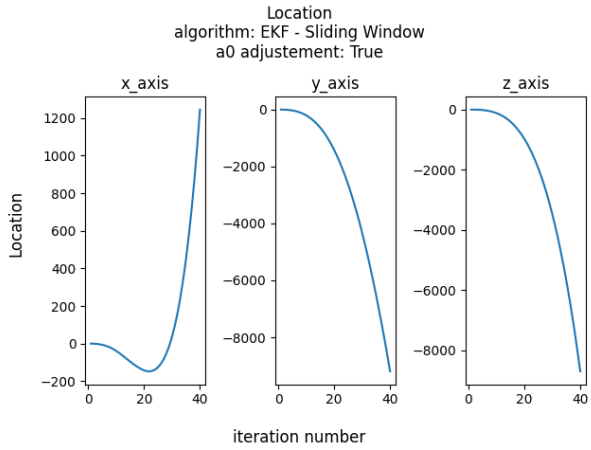
شکل ۴۲: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



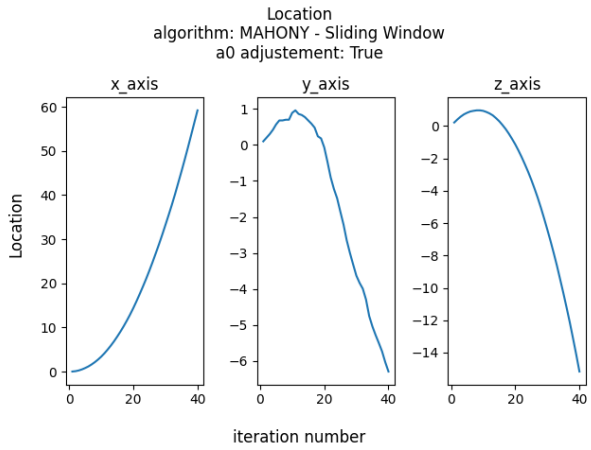
شکل ۴۳: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



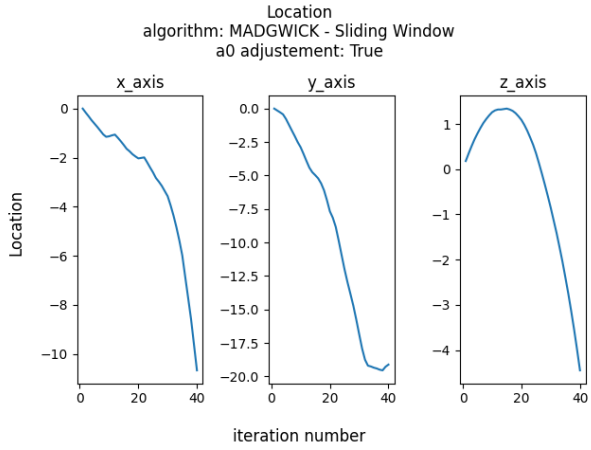
شکل ۴۴: نمودار سرعت بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



شکل ۴۵: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



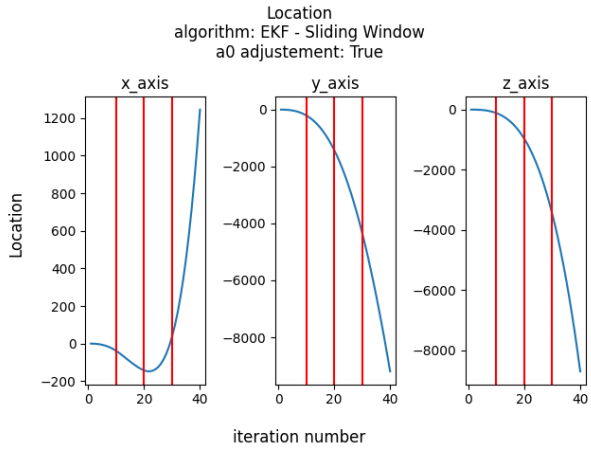
شکل ۴۶: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



شکل ۴۷: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک

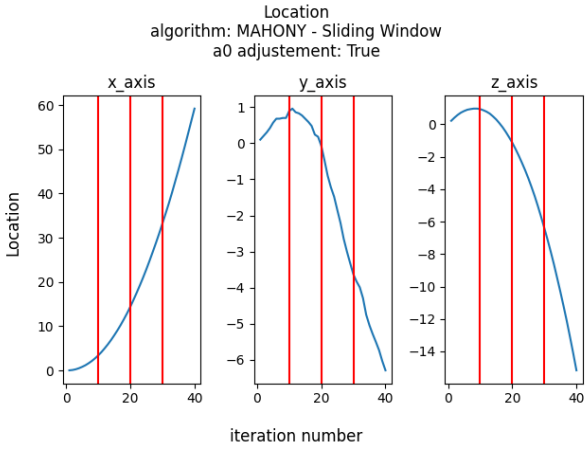
بررسی نتایج بدست آمده

با توجه به اینکه دادگان دریافتی از دستگاه دارای خطای زیادی هستند، هدف ما در این قسمت بررسی نتایج نهایی، یعنی مکان، به طور دقیق تر است. طبق نمودارهای رسم شده در قسمت‌های قبل می‌توان دید که الگوریتم‌های ماهونی و مجویک در کنار روش‌های پنجره لغزان و میانگین لغزان نتایج با خطای کمتری را در اختیار ما قرار می‌دهد. به همین منظور ما نمودارهای مربوط به مکان این الگوریتم‌ها را در ادامه آورده‌ایم.

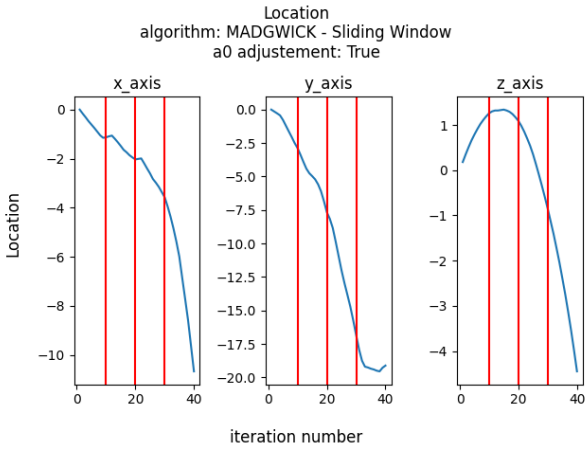


شکل ۴۸: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن

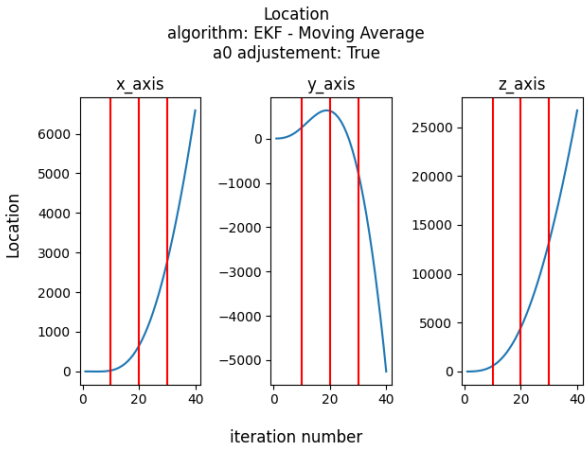
برای بررسی بیشتر، ما هر نمودار را به ۴ قسمت مساوی تقسیم کردیم که با خطوط عمودی قرمز مشخص شده‌اند. همانطور که می‌بینیم میزان جابه‌جایی دستگاه در بازه ۰ الی ۱۰ کمترین مقدار است. حتی در الگوریتم کالمن که بیشترین خطا را دارد نیز این بازه در مقابل بقیه بازه‌ها قابل چشم پوشی است.همچنین در الگوریتم مجویک و ماهونی میزان جابه‌جایی در هر محور زیر



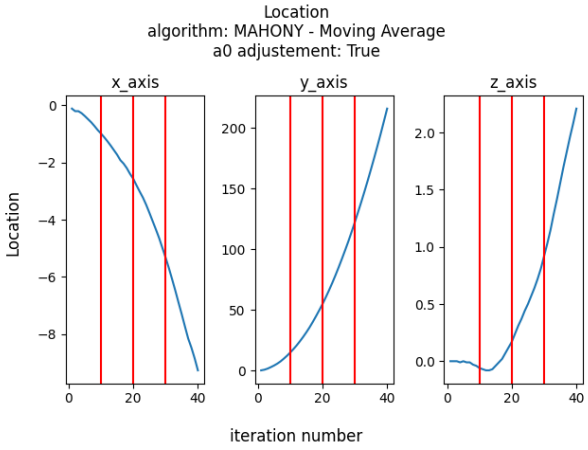
شکل ۴۹: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



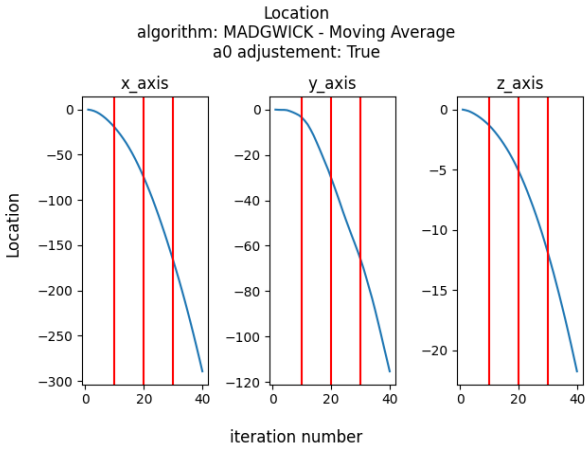
شکل ۵۰: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک



شکل ۵۱: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم کالمن



شکل ۵۲: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم ماهونی



شکل ۵۳: نمودار مکان بر حسب دوره‌ی زمانی سیستم با الگوریتم مجویک

۱۰ متر است.

این مشاهده به ما نشان می‌دهد که اگر بخواهیم از دستگاه ژیروسکوپ فعلی در کنار یک GPS برای تصحیح یا افزایش دقت مکان استفاده کنیم، بازه‌های استفاده می‌بایست در حدود ۱۰ دوره‌ی زمانی باشد که هر دوره‌ی زمانی در حدود نیم الی ۱ و نیم ثانیه طول می‌کشد که در مجموع برابر خواهد بود زمانی در حدود ۱۰ الی ۱۵ ثانیه.

جمع‌بندی

در این پروژه ما به آزمایش الگوریتم‌های گوناگون برای بدست آوردن موقعیت مکانی با استفاده از دادگان دستگاه شتاب‌سنج پرداختیم. آزمایش‌ها نشان می‌هند که دستگاه شتاب‌سنج خطای بالایی دارد و می‌بایست از روش‌های تصحیح خطا و حذف اخلال‌های تصادفی استفاده کرد. در این راستا، ما به ارزیابی ۳ الگوریتم مشهور در این حوزه پرداختیم. علاوه بر این روشی پیشنهاد دادیم تا شتاب گرانشی نسبی دستگاه را با دقت بالاتری محاسبه کنیم. همچنین، برای تصحیح اخلال‌ها از دو روش

میانگین و پنجره لغزان استفاده کردیم. نتایج حاصله از آزمایش‌های ما نشان می‌دهد که می‌توان در بازه‌های کوتاه خطای دستگاه را به مقدار قابل قبولی تقلیل داد. در کارهای آینده می‌توان کارایی شتاب‌سنج‌ها را در کنار GPS در ماشین‌های متحرک که با سرعت بالایی در حال حرکت هستند سنجید.

مراجع

[1] Rudolf Kalman. A New Approach to Linear Filtering and Prediction Problems. 1960.

[2] (1, 2, 3) J. Hartikainen, A. Solin and S. Särkkä. Optimal Filtering with Kalman Filters and Smoothers. 2011

[3] (1, 2, 3, 4) Sabatini, A.M. Kalman-Filter-Based Orientation Determination Using Inertial/Magnetic Sensors: Observability Analysis and Performance Evaluation. Sensors 2011, 11, 9182-9206. (https://www.mdpi.com/1424-8220/11/10/9182)

[4] Roger R. Labbe Jr. Kalman and Bayesian Filters in Python. (https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python)

[5] Lam, Quang Stamatakos, Nick Woodruff, Craig Ashton, Sandy. Gyro Modeling and Estimation of Its Random Noise Sources. AAIA 2003. DOI: 10.2514/6.2003-5562. (https://www.researchgate.net/publication/268554081)

[6] Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. April 30, 2010. http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/

[7] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear Complementary Filters on the Special Orthogonal Group. IEEE Transactions on Automatic Control, Institute of Electrical and Electronics Engineers, 2008, 53 (5), pp.1203-1217. (https://hal.archives-ouvertes.fr/hal-00488376/document)

[8] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Complementary filter design on the special orthogonal group SO(3). Proceedings of the 44th IEEE Confer-

ence on Decision and Control, and the European Control Conference 2005. Seville, Spain. December 12-15, 2005. (https://folk.ntnu.no/skoge/prost/proceedings/cdc-ecc05/pdffiles/papers/1889.pdf)

[9] Mark Euston, Paul W. Coote, Robert E. Mahony, Jonghyuk Kim, and Tarek Hamel. A complementary filter for attitude estimation of a fixed-wing UAV. IEEE/RSJ International Conference on Intelligent Robots and Systems, 340-345. 2008. (http://users.cecs.anu.edu.au/ Jonghyuk.Kim/pdf/$2008_{E}uston_{i}ros_{v}1.04.pdf$)

[10] Tarek Hamel and Robert Mahony. Attitude estimation on SO(3) based on direct inertial measurements. IEEE International Conference on Robotics and Automation. ICRA 2006. pp. 2170-2175 (http://users.cecs.anu.edu.au/ Robert.Mahony/$Mahony_{R}obert/2006_{M}ahHamPfl - C68.pdf$)