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 نظامی‌ـ 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 آ‌ـجی‌پی‌اس[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.

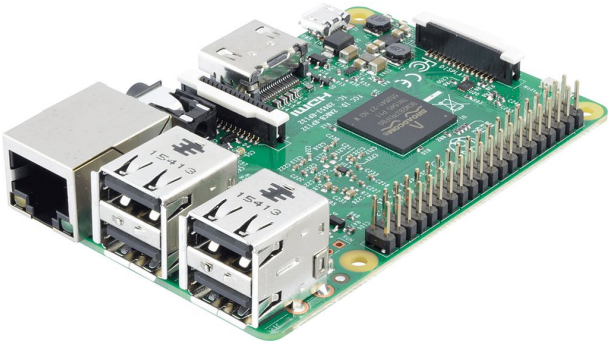
در ادامه این گزارش ابتدا به بررسی اجمالی تحقیق انجام شده می‌پردازیم و معماری آن را تشریح می‌کنیم. سپس به نحوه اتصال به سیستم جهت استفاده از آن می‌پردازیم و در نهایت در مورد ابعاد فنی پروژه صحبت می‌کنیم. در نهایت نیز با بررسی قیمت محصول، جمع‌بندی را انجام می‌دهیم.

---

[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

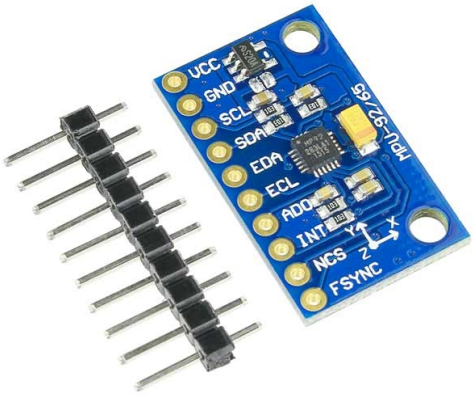
معماری محصول

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.

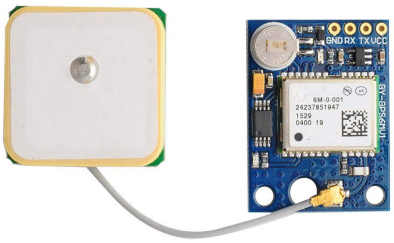
برای سنجش حرکت در این پروژه از ژیروسکوپ ۹ محوره MPU9250 استفاده کرده‌ایم که تصویر آن را در ادامه مشاهده می‌فرمایید.



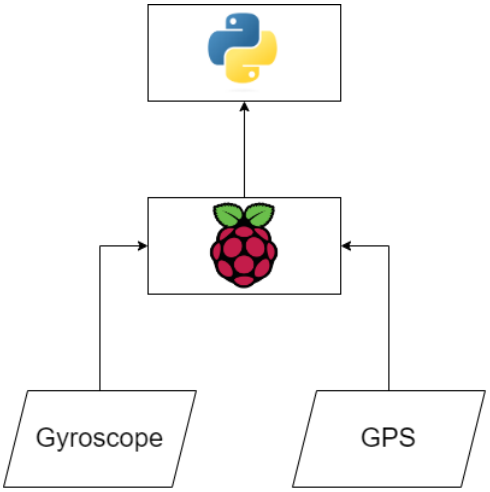
شکل ۲: تصویر ماژول MPU9250 Gyroscope که در پروژه استفاده شده است.

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.



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



شکل ۴: معماری پروژه و ماژول‌های مورد استفاده در آن.

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
دقت عمکرد	در دست بهبود

راه‌اندازی و راهنمای کاربری

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

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.

پس از اتصال به رزبری در مسیر پیش‌فرض می‌توانیم با استفاده از دستور

```
python3 view.py
```

خروجی را مشاهده کنیم.

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

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.

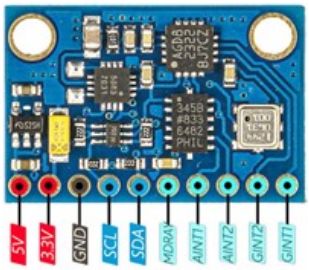
pi:<password_hash>

راه‌اندازی واسط‌های مربوطه

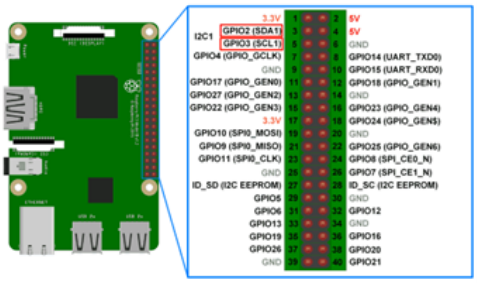
برای این بخش ما دو واسط Wi-Fi و I2C را با استفاده از دستور raspi-config فعال کردیم. این دستور یک محیط شبه گرافیکی پویا در اختیار قرار می‌دهد که به راحتی می‌توان در قسمت interface واسط‌های مذکور را فعال کرد.

وصل کردن ژیروسکوپ به رزبری

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.



شکل ۵: تصویر پین‌های ماژول ژیروسکوپ

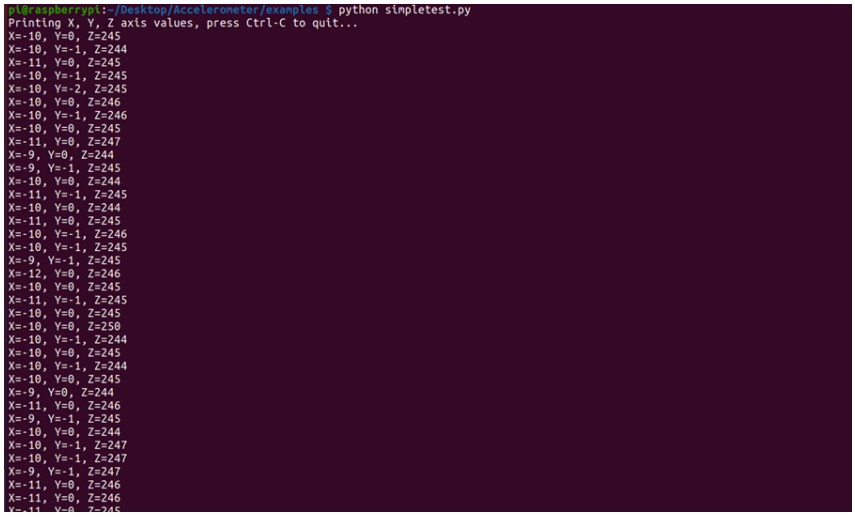


شکل ۶: تصویر پین‌های ماژول رزبری

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.

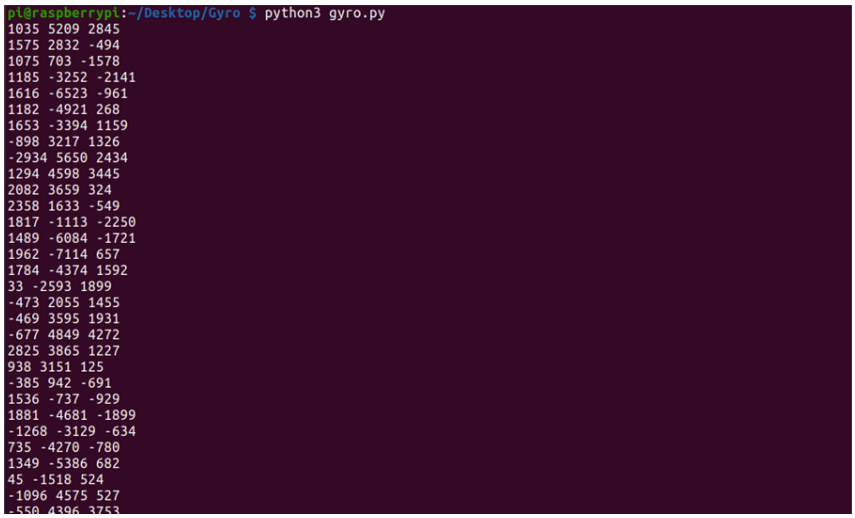
راه‌اندازی ژیروسکوپ

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

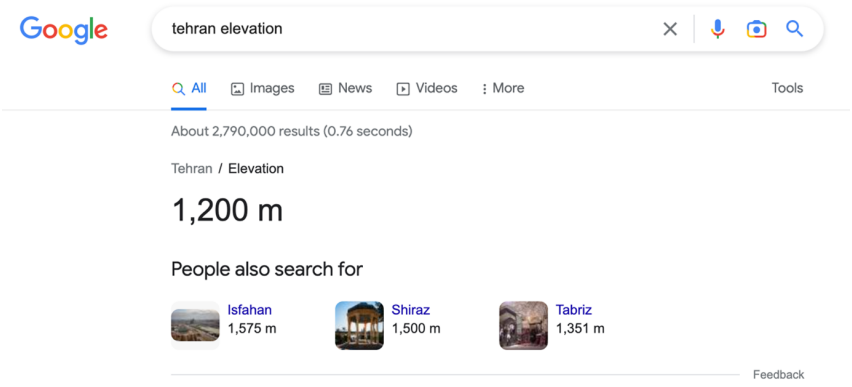
با استفاده از مخزن زیر در گیت‌هاب، کدهای مربوط به شتاب‌سنج را دریافت و آنها را با تغییر مناسب اجرا می‌کنیم. توجه فرمایید برای این کدهای نیاز به نصب پکیج smbus است.

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.

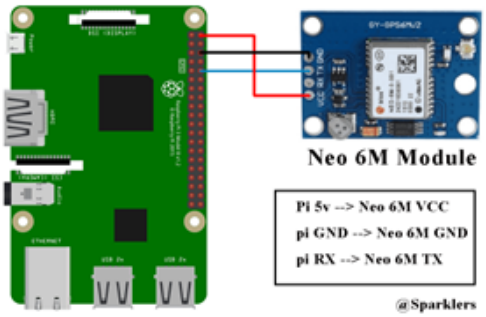
```python
def read_pressure(self):
    UT = self.read_raw_temp()
    UP = self.read_raw_pressure()
    # Datasheet values for debugging:
    # UT = 27898
    # UP = 23843
    # Calculations below are taken straight from section 3.5 of the
    datasheet.
    # Calculate true temperature coefficient B5.
    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

```

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
```

فیلتر جهت مجویک[11]

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 گرادیان‌ـنزولی 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 ${}^{S}\omega = \begin{bmatrix} ۰ & \omega_x & \omega_y & \omega_z \end{bmatrix}$ بدست آمده در بازه زمانی

[11] Madgwick Orientation Filter

Δt و انتگرال گیری نسبت به مشتق $\dot{\mathbf{q}}_t = \frac{1}{2}\mathbf{q}_{t-1}\omega_t$ عبارت

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

مقادیر آنها را بدست آوریم.

اگر فرض کنیم که زمین یک مرجع ثابت باشد که $^E\mathbf{d} = \begin{bmatrix} 0 & d_x & d_y & d_z \end{bmatrix}$ و سرعت نسبت به آن به صورت $^S\mathbf{s} = \begin{bmatrix} 0 & s_x & s_y & s_z \end{bmatrix}$ اندازه‌گیری شود، می‌توانیم تابع هدف را به صورت

$$\begin{aligned} f(\mathbf{q},\, {}^E\mathbf{d},\, {}^S\mathbf{s}) &= \mathbf{q}^*\, {}^E\mathbf{d}\, \mathbf{q} - {}^S\mathbf{s} \\ &= \begin{bmatrix} 2d_x(\frac{1}{2} - q_y^2 q_z^2) + 2d_y(q_w q_z + q_x q_y) + 2d_z(q_x q_z - q_w q_y) - s_x \\ 2d_x(q_x q_y - q_w q_z) + 2d_y(\frac{1}{2} - q_x^2 q_z^2) + 2d_z(q_w q_x + q_y q_z) - s_y \\ 2d_x(q_w q_y + q_x q_z) + 2d_y(q_y q_z - q_w q_x) + 2d_z(\frac{1}{2} - q_x^2 q_y^2) - s_z \end{bmatrix} \end{aligned}$$

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

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 $\mathbf{u}$ و یک اسکالر [؟] نشان داد.

چهارگانه‌ها راه ساده‌ای را برای رمزگذاری این نمایش محور‌ـ 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.

بنابرین گام‌های این 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

تخمین مکان

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 گام‌های بعد پروژه به بررسی الگوریتم‌های Mahony و Extended Kalman Filter و انتگرال‌گیری می‌پردازیم.

نسخه اولیه کد عبارت است از

```
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.]

    def get_q(self, imu_data: IMUData, dt):
        pass

    def get_acceleration(self, imu_data: IMUData, dt, a0=np.array([0, 0,
    0])):
        q = self.get_q(imu_data, dt)
        rotation_matrix = calcualte_rotation_matrix(q)
        a = np.dot(rotation_matrix, imu_data.acc) - a0

        self.q_prev = q
        return a

```

```
class AccelartionCalculatorWithOutMagnet(AccelartionCalculator):
    def get_q(self, imu_data: IMUData, dt):
        q = madgwick_w_magnet.updateIMU(self.q_prev, gyr=imu_data.gyr, acc=imu_data.acc, dt=dt)
        return q

class AccelartionCalculatorWithMagnet(AccelartionCalculator):
    def get_q(self, imu_data: IMUData, dt):
        q = madgwick_w_magnet.updateMARG(self.q_prev, gyr=imu_data.gyr, acc=imu_data.acc, mag=imu_data.mag, dt=dt)
        return q


a_calculator_w = AccelartionCalculatorWithMagnet()
a_calculator_wo = AccelartionCalculatorWithOutMagnet()

prev_time = time.time()
imu_data = get_imu_data()
dt = time.time() - prev_time
prev_time = time.time()
a0_w = a_calculator_w.get_acceleration(imu_data, dt)
a0_wo = a_calculator_wo.get_acceleration(imu_data, dt)


prev_time = time.time()
while True:
    t += 1
    curr_time = time.time()
    dt = curr_time - prev_time
    prev_time = curr_time
    imu_data = get_imu_data()

    a_wo = a_calculator_wo.get_acceleration(imu_data, dt, a0=a0_wo)
    a_w = a_calculator_w.get_acceleration(imu_data, dt, a0=a0_w)

    v_wo += a_wo * dt
    v_w += a_w * dt

    x_wo += v_wo * dt
    x_w += v_w * dt

    if t  % 100 == 0:
        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)
```

و خروجی آن برابر خواهد بود با

```
acceleration without magnometer:    [-4.25149814e-03 -4.95737249e-02
```

```
   9.42624897e+00]
acceleration with magnometer:       [-0.04666684 -0.82391009  9.39018848]
*****************************
velocity without magnometer:        [-0.08738802 -0.02829634 11.45156396]
velocity with magnometer:           [-0.1709201  -0.47476062 11.4373586 ]
*****************************
location without magnometer:        [-0.0767284  -0.01584075  7.01596472]
location with magnometer:           [-0.12772053 -0.19176933  7.01128514]

acceleration without magnometer:    [-0.02518617 -0.02588419  9.42638894]
acceleration with magnometer:       [ 0.1183487  -1.54551333  9.2981447 ]
*****************************
velocity without magnometer:        [-0.11273455  0.06517335 23.07376832]
velocity with magnometer:           [-0.13394683 -1.81403115 22.97855815]
*****************************
location without magnometer:        [-1.99901281e-01  1.35208854e-02
   2.83287529e+01]
location with magnometer:           [-0.33542242 -1.50777399 28.268977  ]

acceleration without magnometer:    [-2.83333815e-02 -2.27021052e-03
   9.46571874e+00]
acceleration with magnometer:       [ 0.1991406  -2.03568396  9.24213033]
*****************************
velocity without magnometer:        [-0.13622949 -0.20122665 34.72070666]
velocity with magnometer:           [ 0.05955209 -4.32003195 34.351887  ]
*****************************
location without magnometer:        [-3.55068346e-01 -4.62302547e-02
   6.40790289e+01]
location with magnometer:           [-0.38755003 -5.21982079 63.75090291]

acceleration without magnometer:    [ 2.14537055e-03 -8.36774444e-03
   9.42645416e+00]
acceleration with magnometer:       [ 0.2035248  -2.13368564  9.17954661]
*****************************
velocity without magnometer:        [-0.15878657 -0.34168176 46.41142582]
velocity with magnometer:           [ 0.30219398 -7.08084202 45.70950965]
*****************************
location without magnometer:        [ -0.53768706  -0.35373549
   114.40316292]
location with magnometer:           [ -0.15114503 -12.25904527
   113.41877558]

acceleration without magnometer:    [ 1.29001854e-03 -2.65921315e-03
   9.38816218e+00]
acceleration with magnometer:       [ 0.1189121  -2.0876348   9.15233515]
*****************************
velocity without magnometer:        [-0.29744507 -0.48793365 58.08000237]
velocity with magnometer:           [ 0.33816309 -9.80783865 57.05681006]
*****************************
location without magnometer:        [ -0.84045521  -0.8003985
   179.03378125]
location with magnometer:           [  0.25672646 -22.64583887 176.996302
   ]

acceleration without magnometer:    [-0.02331939 -0.03125536  9.46459583]
acceleration with magnometer:       [ 0.01744526 -2.0486378   9.24028538]
```

```
48 ******************************
49 velocity without magnometer:        [-0.32350558 -0.51723034 69.73941834]
50 velocity with magnometer:           [  0.42745446 -12.3615718
       68.43273926]
51 ******************************
52 location without magnometer:        [ -1.22231378  -1.41875304
       258.01989447]
53 location with magnometer:           [  0.7415673  -36.35719339
       254.54020762]
54
55 acceleration without magnometer:    [-3.23846762e-03 -5.46642539e-03
       9.38808234e+00]
56 acceleration with magnometer:       [-0.04872244 -1.92553817  9.18836544]
57 ******************************
58 velocity without magnometer:        [-0.34674657 -0.54121791 81.44250122]
59 velocity with magnometer:           [  0.40057772 -14.79097339  79.8814177
       ]
60 ******************************
61 location without magnometer:        [ -1.64033198  -2.01828867
       351.81844505]
62 location with magnometer:           [  1.28347787 -53.16669974
       346.56780088]
```

همانطور که مشاهده میکنید بردار عمودی آن دارای شتاب نزدیک به ۱۰ هست که همان شتاب گرانش کره زمین است. در گام بعد این شتاب را از محاسبات حذف میکنیم. برای این کار کافی است تا خطوط ۹۹ و ۱۰۰ از کد را تغییر دهیم

```
1 a_wo = a_calculator_wo.get_acceleration(imu_data, dt)
2 a_w = a_calculator_w.get_acceleration(imu_data, dt)
```

خروجی برابر خواهد شد با

```
1 acceleration without magnometer:    [ 0.21294515 -0.05107898  0.46243406]
2 acceleration with magnometer:       [ 0.17275839 -0.81592309  0.42991005]
3 ******************************
4 velocity without magnometer:        [ 0.18002359 -0.07464414  0.52304873]
5 velocity with magnometer:           [ 0.09035456 -0.51137019  0.50877135]
6 ******************************
7 location without magnometer:        [ 0.08789094 -0.04398274  0.32123195]
8 location with magnometer:           [ 0.03570646 -0.21272872  0.31651788]
9
10 acceleration without magnometer:    [ 0.22391228 -0.09468307  0.50056942]
11 acceleration with magnometer:       [ 0.33809077 -1.66570507  0.35887181]
12 ******************************
13 velocity without magnometer:        [ 0.42873573 -0.03035206  1.05489486]
14 velocity with magnometer:           [ 0.38949088 -1.89878573  0.95883246]
15 ******************************
16 location without magnometer:        [ 0.46382656 -0.10506909  1.29079559]
17 location with magnometer:           [ 0.31357678 -1.59880393  1.23082621]
18
19 acceleration without magnometer:    [ 0.15097011 -0.0447729   0.42603422]
20 acceleration with magnometer:       [ 0.36940456 -2.12815785  0.18960435]
21 ******************************
22 velocity without magnometer:        [ 0.68030172 -0.09790213  1.58801106]
```

```
velocity with magnometer:           [ 0.86183677 -4.22533187  1.26035704]
*****************************
location without magnometer:        [ 1.14623013 -0.18067855  2.91013427]
location with magnometer:           [ 1.07312008 -5.29330579  2.60402268]

acceleration without magnometer:    [ 0.2736358  -0.08744284  0.4612667 ]
acceleration with magnometer:       [ 0.47026358 -2.37752117  0.16340941]
*****************************
velocity without magnometer:        [ 0.935074   -0.16917634  2.12630468]
velocity with magnometer:           [ 1.39086609 -7.01468068  1.46413695]
*****************************
location without magnometer:        [ 2.13461008 -0.34504042  5.18848774]
location with magnometer:           [  2.45360254 -12.16728811   4.2801073
   ]

acceleration without magnometer:    [ 0.21078115 -0.07314695  0.53888497]
acceleration with magnometer:       [ 0.33964242 -2.34659979  0.25362898]
*****************************
velocity without magnometer:        [ 1.20174641 -0.47882824  2.65530114]
velocity with magnometer:           [  1.79661324 -10.10131954
   1.58828864]
*****************************
location without magnometer:        [ 3.44853279 -0.71032858  8.12746389]
location with magnometer:           [  4.43205238 -22.67648808
   6.16054664]

acceleration without magnometer:    [ 0.22118174 -0.05018395  0.50073918]
acceleration with magnometer:       [ 0.25730655 -2.22476348  0.24457954]
*****************************
velocity without magnometer:        [ 1.45657709 -0.66510888  3.18350434]
velocity with magnometer:           [  2.11511748 -12.98412752
   1.76198968]
*****************************
location without magnometer:        [ 5.07719701 -1.46429614 11.70659983]
location with magnometer:           [  6.82642423 -36.88414884   8.2003331
   ]

acceleration without magnometer:    [ 0.19707225 -0.04473991  0.38581495]
acceleration with magnometer:       [ 0.1566871  -2.05454365  0.16542856]
*****************************
velocity without magnometer:        [ 1.71391569 -0.61818085  3.71558581]
velocity with magnometer:           [  2.39740111 -15.4974445
   2.02640531]
*****************************
location without magnometer:        [ 7.01983304 -2.2723039  15.93238891]
location with magnometer:           [  9.59354633 -54.37192878  10.5120584
   ]
```

همانطور که دیده می‌شود این خروجی شتاب اعدادی نزدیک صفر شدند. اما مشکلی که همچنان وجود دارد این است که به خاطر خطای اندک موجود در شتاب‌سنج، سرعت و موقعیت با کمی خطا محاسبه می‌شوند. با گذشت زمان این خطاها روی هم انباشته می‌شوند و باعث می‌شوند که موقعیت و سرعت در طی مدت کوتاهی تغییرات زیادی داشته باشند. ما در گام بعدی پروژه این سعی داریم از الگوریتم‌های دیگر که در بالا معرفی شدند استفاده کنیم

و میزان خطای آنها را با یکدیگر مقایسه کنیم.

قیمت برآوردی

توجه بفرمایید که این یک محصول پژوهشی است و قیمت برای آن چندان معنی ندارد. مهندسین بر روی این پروژه بالغ بر ۱۰۰ ساعت کار کرده‌اند. همچنین لیست قطعات مورد استفاده در این پروژه عبارت است از

ردیف	قطعه	قیمت (تومان)
۱	Raspberry Pi 3b	4,000,000
۲	MPU9250	250,000
۳	GPS U-Blox NEO-6M	200,000

جمع‌بندی

در این پروژه ما موفق شدیم با استفاده از ژیروسکوپ موفق شدیم در شرایط فقدان سیگنال GPS، مکان‌یابی را ادامه دهیم و به صورت تجربی اثبات کنیم که این امر امکان‌پذیر است. نکته حائز اهمیت در این آزمایش در راستای صنعتی‌سازی این است که اگر قرار باشد که این وسیله به صورت صنعتی تولید شود و در مکان‌هایی که در آنها حساسیت وجود دارد استفاده شود، حتما باید از ژیروسکوپ دقیق‌تر و با کیفیت بالاتر استفاده شود. قیمت این قبیل ژیروسکوپ‌ها گاه به هزاران دلار[۱۲] می‌رسد. لذا استفاده از این تکنیک در جایی باید انجام شود که نیاز به دانستن مکان در هر لحظه وجود داشته باشد. البته توجه کنید که در صورت وجود ژیروسکوپ، مثلا در هواپیما، هزینه صرفا به بورد و هزینه تحقیق و توسعه محدود خواهد شد.

[۱۲] دلار ایالات متحده آمریکا