**UAS(Unmanned Autonomous Ship)主控系统开发日志**

--李东豫

**Estello UAS 概要总结**

1. 亮点概要：
   1. 系统架构方面：
      1. STM32采用类操作系统式框架，任务可分不同频率被调用。优点是该框架可以提高系统灵活性，降低系统耦合性。这么做的还有许多无人机飞控厂家，如crazypony等。
      2. 下位机工程可选择编译，只需要改动一个宏定义，系统即可进行相应改变，自动选择传感器元件。主要应用于对GY901与MPU6050+HML5883两套惯性测量装置同时兼容，无缝切换。
      3. 传感器上电自动自检，自动校准，保障数据精度
      4. GPS模块采用先进的UBX协议，系统上电前自动配置GPS输出必要信息，滤除不必要信息，大幅提高数据处理效率
   2. 系统核心方面：
      1. 上、下位机内部数据缓冲区采用先进的环形缓冲区，大幅降低数据高速传输时的丢包率，提高上下位机对数据处理速度
      2. STM32采用有限状态机设计，可通过遥控器进入不同状态，执行不同操作。提高系统灵活性。
      3. 采用先进的Madgwick梯度下降姿态解算算法，较以前的卡尔曼滤波方法大幅降低了下位机对数据迭代的次数，采用此方法仅需10Hz的采样速度即可收敛至当前载具的姿态
      4. GPS模块由于需要实时接收大量数据，严重占用MCU任务处理时间，因此加入了DMA。并且在工程中配置为DMA与中断处理两种可选方式选择编译，提高了效率、灵活性。
   3. 功能拓展方面：
      1. 可设置的载具模型(汽车模型、海上模型)可最优化使用GPS输出数据，GPS输出数据将根据所选模型进行适配。
      2. 可在Google Map上标点直接导出为载具轨迹（详情请看下面的实地测试2019-8-23日志）

**为未来的参阅者写的序：**

*自动导航系统是怎么制作出来的？*

首先无人船要“无人”，就得能自己跑；自己跑，就得知道它自己的GPS坐标和它自己的姿态。正如飞机一样，可能飞机在地球的某一经\纬度的数值是一样的，但是一架飞机是往天上飞(Upwards)，另一架飞机是往地上撞(Downwards) ；这么一来，对一架飞机来说实际上有6个自由度：三维空间的坐标为3个自由度，绕三维空间的三个轴旋转的角度为3个自由度，结合了这6个自由度，我们才能准确的直到一架飞机当前的具体状态，管这个具体状态，行话叫“姿态(Attitude)”。我们利用各种传感器测出当前姿态的这个过程，行话叫“姿态解算”。

对咱的船来说，咱的船一般不会像飞机一样go upwards\downwards，不然不是翻船就是宇宙飞“船”。目前我的程序就忽略了传感器测定这两种角度，因为如果真出现了，那没办法，跑个无人船还遇着海啸了，翻就翻吧。

所以就只考虑了船头向左偏还是向右偏，这个偏离的角度，咱行话叫“偏航角(Yaw)”，好了，船知道了目前的四个自由度就可以导航了。利用GNSS定位，常说的GPS只是老美的那一套定位系统，其实还有咱的北斗、欧洲的伽利略，俄罗斯的Glonass定位系统，这一大堆统称起来叫GNSS，Global Navigation Satellite System。好了，GNSS接收机是咱们装在船上的，它能输出当前接收机模块所在地点的经纬度。但是不同的定位系统基于的坐标系不同，GPS基于WGS84坐标系，北斗基于一套咱国家的坐标系。不能乱用经纬度数值，不然偏差能差到夏威夷。

一般的GNSS模块是Ublox牌的 M8N，之前比赛的时候有个人说咱的GNSS模块不咋地，但其实M8N老好用了，很多汽车都用装它。有的定位系统还不一定赶上M8N (售价200)

唯一的问题是M8N假货横行，我上面的介绍全部基于买了个正品货的情况下。买这玩意的时候我还没来这个实验室，也就不知道它是真是假

好了，获取了一定坐标系下的经纬度，还差最后一个自由度：Yaw。为了获取Yaw，有两种方法：  
1.利用GNSS。让小船下水之后往前走两步，咱就能利用两点连成一条直线的原理就知道船头朝哪了。  
2.利用磁力计。有的GNSS模块上边写的一行小字：with compass.  这个compass就是指的磁力计，其实也就是电子指南针啦！利用霍尔原理采集三轴上的磁数据。问题是：它可不知道啥是地磁啥是人造磁，你把无人船放到两块大吸铁石旁边，分分钟这船就得撞残废。这是问题之一。第二点就是咱的电路板上的高频数字通信，比如IIC协议，也会散播磁力。影响船的导航。基于这个原因，磁力计受争议比较大。我的想法是结合磁力计和GPS，但是具体怎么结合以后再做。首先搭好框架最重要。

传感器介绍完了。假装咱们已经做好传感器了，然后读取数据，我们直接用读取回来的数据控制无人船。你将会看到的现象是：三、二、一，把小船放下水，上电…这船咋一直哆嗦呢？

出现这种状况的原因是电子传感器直接获取的数据噪声很大，需要滤波。在此，神奇的卡尔曼滤波出来了。输入传感器数据中的杂乱数据，它就能给你输出很接近真实数据的结果。当然啦，我们还可以采用互补滤波，这个以后再讲。我做的程序目前就是试图采用互补滤波。互补滤波是指加速度计的误差是高频误差，陀螺仪误差是低频误差。好了，这俩一结合：啪的一下，他俩的误差就很小了。但是加速度计是为了获取Pitch\Roll角数据的，咱的船用不上。所以....也不算是互补滤波。（补充：最终在Eastar船的系统里也没有用互补滤波，而是使用的Madgwick梯度下降解算）

有了这些，就是处理数据的事了。给你两个经纬度目标点，你怎么规划出一条路线？这里是由树莓派处理。我正在这一方面工作(此时为2019年7月21日)。详细内容请查阅开发日志下文。

（写于2019年7月21日）

（2019年8月19日修正）

*关于自动导航系统的一些算法简述*

其实一切控制系统，都始于这样的一个框架：观察->输出->反馈->观察。拿吃饭举个例子，吃饭也是这么一个闭环控制系统，饿了（观察），接着吃饭（输出），吃了没饱（反馈+观察），接着再吃（输出）。这一套控制算法有很多种，首先是PID算法——Proportional, Integral, Derivative.即比例积分微分算法。P\I\D分别代表三种不同的控制方法。还是以吃饭为例，越饿吃得越多，这是比例控制；饿的时间越长，吃的也就越多，这是积分控制；坐飞机之前怕自己饿着，先多吃点饭，这是微分控制。如果你在看这篇日志时，原型车还没被拆掉的话，你可以观察一下原型车的电机控制部分代码，采用的就是PI控制，这可以使路面摩擦力大时，车速还是保持稳定；路面打滑，车就会自动减速。此外与PID算法并行的还有滑模控制器等等。此处就先不展开篇幅介绍。

Kalman滤波算法

Madgwick梯度下降姿态解算算法，这是最终我在Eastar小车上实现的算法。首先它的好处就是对迭代次数、传感器采样频率要求很低，而这恰好弥补了Kalman滤波器的缺点——吃性能。由于下位机采用STM32F1系列，时钟频率最高72Mhz，这也就使想要在32上同时采用姿态解算算法和控制是不太可行的。梯度下降实际上是一种在机器学习中应用的比较广泛的算法，普遍用在优化问题上。也就是说，梯度下降中的“下降”可以理解为使一个函数的输出值与理论值之间误差逐步下降，用在姿态解算上也就是使解算姿态与真实姿态之间误差逐步下降。

**UAS开发第一阶段**

说明：

1. **黑体字**表示需要搭起的框架。优先级0。
2. 正常字体表示正常接口。优先级1。
3. *斜体字*表示在主框架搭建完毕后可拓展的功能。优先级2。
4. **重要信息**以加粗+下横线的形式标出。
5. **主要目标：**
   1. 上位机仅执行#裁决#功能。使用遥控器时给上位机传输数据，上位机告诉下位机该怎么走。下位机只需要去做。
   2. *上位机由于某些未知原因突然下线，则下位机全权掌握控制权，立马停机。*
   3. *遥控器直接对下位机发出信号，由下位机转达给上位机此时处于遥控状态。这样可以提高载具的响应速度。*
   4. *电压、电池电量监测模块*，下位机获得电量数据后传输给上位机，上位机实时计算已走出的路径（Course over Ground），当电池电量过低时（电量比载具恰好能够返航所要消耗的电量稍微多一些，例如，10%），通知下位机返航。功能可开可关。
   5. 给出固定路径点，可以有两种前进模式：
      1. Perpendicular模式，即全部走直线。直线走到下一个路径点，拐弯，再直着走到下一个路径点。
      2. *Tangent模式*，即走一个带曲率的弧线，比较自然。用曲线连接起所有路径点。
   6. 通过读取俯仰角、滚转角，判断船身晃动情况，开发抗风浪模式。为了提前为抗风浪模式的开发做好准备，这里在Alpha0.1版本里，虽然并没有校正（甚至没有必要去读取）Pitch、Roll数据，但是却保留了Pitch、Roll读出的接口。以备后面使用。

(2019-8-23注：Alpha1.0版本中直接输出了Pitch与Roll角，可直接使用)

1. 为什么不用Kalman Filter 而用**Mahony Complementary filter**?
   1. The Arducopter developers chose the Direction Cosine Matrix (DCM)/complementary filter method of attitude estimation for precisely the reason that it is an order of magnitude less computationally intensive than the Kalman filter approach.   
        
      Here is a collection of discussion and papers on the topic. See this early paper "Bizard, Paul, William Premerlani. DIRECTION COSINE MATRIX IMU: THEORY. 17 May 2009. "   
      As far as I know, most self-contained attitude estimation units, like the UM6 from CHRobotics, use dedicated, fast processors like the 32bit ARM to implement Kalman filters. These units are becoming so inexpensive that it makes no sense to write your own, unless you want the (painful) learning experience
   2. <http://www.chrobotics.com/products/inertial-and-orientation-sensors>

——forum.arduino.cc

* 1. Apart from issues of processing power,  you run into a fundamental problem with Kalman because the Kalman method is based on some assumptions about the linearity of the problem,  which do not apply to the flight dynamics problem in three dimensions,  except in a limited sense,  such as only processing a single input,  or reducing the problem to one dimension.
  2. 我的想法：第一，Kalman Filter运算量大，迭代时间较长。如果以后有什么高科技主控（比如Nvidia Jetson Nano），再考虑用迭代算法吧。第二：船目前只有一个自由度，即偏航Yaw,而Yaw可由磁力计、GPS、陀螺仪一起测出，而Yaw对误差不敏感，使用Mahony的方法足以解决较高精度的问题。第三，由于微控的截断误差等各种误差，Kalman Filter不见得精度就会高。~~因此，选择Mahony算法。~~

*注：最终开发实现的版本中也没有使用Mahony算法，原因是Mahony算法是为IMU模块开发的，但船还需使用到Magnetometer，因此最终使用了Madgwick的Gradient Decent算法*

* 1. 互补滤波：利用加速度的测量值修正陀螺仪的输出，再通过修正后的角速度更新四元数，进而解算得到姿态角。由于仍然有噪声的干扰，因此可以在上位机采用**卡尔曼滤波算法对姿态角进行滤波**，**同时引入互补滤波与卡尔曼滤波算法**，从而得到正确的姿态角

1. 应该有一个当地重力加速度测量装置。设想：用两个完全一样的加速度计，一个装在船上，一个用作校准当地重力加速度；或者干脆用船上的加速度计校准。把加速度计静止放置，加速度计输出的三轴X\Y\Z数据即重力加速度向量。取模…?可行性待测。

*注：最终开发的版本中选择忽略了暂时性的加速度（如船的前进加速度），直接利用加速度计结果对陀螺仪进行纠正。实验结果比较理想，此外也因为在船整个行驶的过程中加速度前后抵消。*

* 1. 加速度计仅仅测量的是重力加速度,而重力加速度与E坐标系（Earth Frame）是固连的,通过这种关系,可以得到加速度计所在平面与地面的角度关系。
  2. --- Accelerometers are used for roll-pitch drift correction because they have zero drift.

1. 加速度计测量的是全部的加速度，包括重力加速度与船引起的加速度。但是实际上仅想用加速度计测量三轴的重力加速度分量，通过**叉乘**得出当前pitch\roll.
   1. 解决方案？ 想法：用GPS读取船的速度，如果船在以0.5s为步长的1s间隔内，船速上下浮动不超过5%，则可近似认为此时船是匀速行驶，没有附加的加速度，只有重力加速度。此时加速度计数据可用来校正陀螺仪pitch\roll数据。
   2. *Direction Cosine Matrix IMU: Theory*中写道：

All is not lost. On average, a plane does not accelerate in the forward direction. There are times when it speeds up and when it slows down, but the accelerations cancel out. A plane cannot accelerate for long in the forward direction until aerodynamic drag prevents it from going any faster. **A plane cannot decelerate for long without stopping and falling from the sky. （注：也就是说机体引起的加速度只是暂时量）** As longas we are not depending on an accelerometer for fast transient response, we can use it for roll-pitch correction of gyro drift, because the accelerometer does not drift.

也就是说应用在无人机上修正pitch\Roll数据的方法就是单纯的认为无人机加减速的平均值会相互抵消。大概这也就是无人机在向前(Forward)飞行的时候，机头会往下沉的原因吧。因为加速度计误以为向前的加速度也是重力加速度的一部分，因此机身矫正系统正在努力“纠偏”。

1. 最简单的船只需要1个自由度，即Yaw,因此也只需要陀螺仪、磁力计、GPS.加速度计在此时用不上。*可能在未来需要加上抗风浪功能的时候需要考虑Pitch\Roll.*
2. ~~GNSS模块中有一块地磁传感器，IMU里也有一块地磁传感器，这里为了稳健起见我对两个磁力计的数据全都读取，IMU内的Mag输出频率是50Hz；GNSS内的Mag输出频率是75H。每秒计算一次两个地磁传感器的输出数值的方差，方差小的那个Mag作为当前的主Mag使用。~~考虑后认为这一步并没有必要。输出数据如果真的大到这种方法可以修正的地步的话，~~基本就可以去把半导体厂家告上法庭了~~。因此功能被删减。只需读取IMU中的Mag数据即可。
3. **用Mag作为陀螺仪Yaw偏角纠正的问题**：
   1. “由于地磁传感器极易受到各种干扰，而此算法又将地磁传感器所指示的方向过度的融入到了姿态当中，导致实际使用中数据会非常不稳定。怪不得crazypony的算法中没有使用磁力计修正Yaw，这应该就是原因所在吧。”

——《**[Pixhawk代码分析-姿态解算篇B](https://www.ncnynl.com/archives/201607/309.html)**》

* 1. 《An Improvement Strategy on Direction Cosine Matrix based Attitude Estimation for Multi-Rotor Autopilot》by Nazia Ahsan Dilshad & Mohammed Kawser Jahan指出，如果使用Mag作为Yaw校正的Primary Choice,必须要考虑几个因素:Hard Iron影响与Soft Iron影响；Tilt compensation。而Tilt补偿必须得知船的Pitch与Roll.目前来看，要得出Pitch与Roll必须使用加速度计测重力加速度，这么一来又与上面冲突。想法：*是否可以取一步近似：船在水面上航行，假设船的重量分布均衡，近似船的Pitch与Roll都为0？* 但是问题是——虽然我没见过，但是听说过——有船会往水里扎猛子的现象。如果说船往水里扎猛子是一种Pitch的极限情况，那么也就是说Pitch的确不为0，且不可近似为0？ 待考虑。这里考虑**下一步做实验测试**

(2019-8-23注：实验发现，在陆地上用小车测试时发现Pitch与Roll很少有大于30度的情况，且船在水上时由于处于水平面，两个角度应该更加小。因此是可以忽略的；此外，Madgwick 梯度下降姿态解算算法直接解出了Pitch与Roll,可以将其应用在有较高精度要求的场所对Tilt进行补偿。)

* 1. 电子罗盘如  HMC5883L或者 MPU9250的AK8963测方位角时，经常会发现测量角度抖动很厉害。即使不动，角度也在不停的小范围内变化。这时候就需要使用陀螺仪对方位角进行补偿融合，得出稳定的数据。这里只讨论2D空间的方位角，就是电子罗盘X, Y两轴数据融合陀螺仪Z轴数据。
     1. 电子罗盘：0—360度的方位角，也称为 yaw 。角度为arctan2(y,x)
     2. 陀螺仪：围绕某轴旋转的角速度，这里我们只考虑Ｚ轴，相当于方位角的变化速度。#符合上面的假设!#
     3. 融合[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库)公式：

在某个时间点 Tx 上, 电子罗盘值为 YawCompassTx, 陀螺仪值为YawGyroTx, 融合后的数据为 YawFuzeTx

下一个时间点 Tx+1,  陀螺仪计算的角度变化为

delta\_Yaw = YawGyroTx+1\* delta\_t,  其中 delta\_t =Tx+1 - Tx

YawFuzeTx+1 = K \* ( YawFuzeTx+ delta\_Yaw ) + ( 1 - Ｋ ) \* YawCompassTx+1

K一般取0.9-1，表示对陀螺仪数据的信任程度

* 1. 此外一个问题：电路板等会产生Hard Iron影响，会使磁力计测量原点脱离，此时必须将Hard Iron影响消除。方法：将陀螺仪沿Z轴向上的形式匀速旋转360°，记录X\Y轴上的最大值、最小值Xmax，Xmin，Ymax，Ymin。然后将Z轴放置水平，匀速旋转360°，记录最大最小值。最后：

Xoffset = (Xmax+Xmin)/2

Yoffset = (Ymax+Ymin)/2

Zoffset = (Zmax+Zmin)/2

之后的各个测量值均减去这三个轴的Offset值即可。

*问题：只校正X、Y轴是否可行？Z轴的作用？*

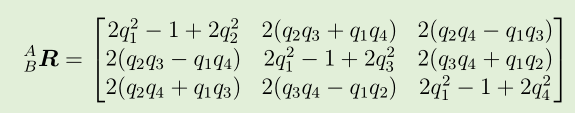
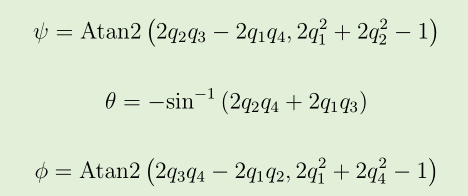
解答：方位角=arctan2(y,x)关系式是在检测仪器与地表面平行时才成立。当仪器发生倾斜时，方位值的准确性将要受到很大的影响，该误差的大小取决于仪器所处的位置和倾斜角的大小。Z轴作误差校正的作用。

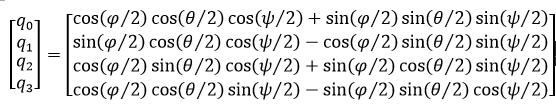
1. **坐标系说明**：
   1. JY901 IMU模块使用的是东北天坐标系
   2. GPS输出速度数据基于北东地坐标系
   3. 为了统一坐标系，**Alpha0.1基于东北天坐标系进行导航**。
   4. 机体坐标系只有前左上和前右下，**东北天地理坐标系常与右前上机体坐标系对应**，
   5. 欧拉角旋转顺序：偏航Z------横滚Y------ 俯仰X,注意东北天坐标系偏航角带一个负号（-psi）！
   6. Madgwick’s Algorithm中的坐标系：

Euler angles ψ, θ and φ in the so called aerospace sequence [36] describe an orientation of frame B achieved by the sequential rotations, from alignment with frame A, of ψ aroundˆz , θ around ˆ y B , and φ around ˆ x，即载体坐标为右前上坐标

1. 为了统一Madgwick算法和惯例，**Alpha0.1采用右前上坐标作为机体坐标系**。

右:x轴 前：y轴 上：z轴

1. 四元数矩阵：
2. 欧拉角：
3. 欧拉角到四元数：



1. **说明**：
   1. **以上讨论均在z-y-x坐标系中进行。**
   2. 东北天坐标系:
      1. X轴（东）：俯仰角Pitch；Y轴（北）：横滚角Roll；Z轴（天）：航向角Yaw
      2. 俯仰角和横滚角：姿态角方向与对应轴向转动方向**一致**.绕对应轴**正**转，相应姿态为**正**；绕对应轴**反**转，相应姿态为**负**。
      3. 航向角：姿态角方向与对应轴向转动方向**相反**。
   3. 载体坐标系原点位于载体质心，通常选取右前上坐标系，即其 x 轴沿载体横轴向右，其y 轴沿载体纵轴向前， 其 z轴沿载体竖轴向上。
   4. Alpha0.1系统中以四元数存储当前机体姿态，用欧拉角进行调整姿态。与Unity3D运作方式相同。可节省内存空间与计算量。
2. WGS->NED坐标系：最重要的问题时算出导航在往下一个点前进时的偏航角。似乎之前的船是把WGS坐标转换到utm坐标…? 这里试图给出两个WGS经纬度点，直接算出它们之间的夹角。

公式：θ = atan2( sin Δλ ⋅ cos φ2 , cos φ1 ⋅ sin φ2 − sin φ1 ⋅ cos φ2 ⋅ cos Δλ )

|  |  |
| --- | --- |
| *where* | φ1*,*λ1*is the start point,*φ2*,*λ2*the end point (*Δλ*is the difference in longitude)* |
|  |  |

----<https://www.movable-type.co.uk/scripts/latlong.html>

Since atan2 returns values in the range -π ... +π (that is, -180° ... +180°), to normalise the result to a compass bearing (in the range 0° ... 360°, with −ve values transformed into the range 180° ... 360°), convert to degrees and then use (θ+360) % 360, where % is (floating point) modulo.

1. 导航过程：
   1. 系统上电，上位机发送下位机发送定位数据（不在路径点中）至上位机
   2. 上位机计算当前GPS坐标与下一目标点之间的Yaw角、距离与坐标点类型Ptype(可以通过改变配置的形式直接控制是否需要拐点类型)
   3. 根据上位机返回Yaw，下位机判断当前方位角与目标方位角之差，当差值大于等于MAXIMUM度时修正航向至目标方位角的MAXIMUM/4（或者其他数值）范围内。
   4. 根据上位机返回的距离数值大小与Ptype控制航速例如：
      1. 当Ptype为正常航点时，UAS径直加速，直至最大速度的90%。到达GPS坐标的1%范围内，前往下一个航点。
      2. 当Ptype为拐点时（是否为拐点的判断由上位机进行计算，例如当两个点之间的夹角大于45度时认为是拐点），是拐点的话通过一个函数控制拐角大小与航速的关系（上位机计算出这两个拐点所围成的圆弧的圆心位置，或者拐角角度越大，航速越低，同时偏角逐渐递增或递减至所要求Yaw）。
2. 写了一个Python脚本，可以直接在Google Earth上标点，标好点之后输入给脚本，脚本直接将kml文件转化为带有标点名称（用来标号）和经纬度、高度的txt文档。

示例格式：1:120.520163178378,36.89206629053305,0

这样一来可以直接在Google Earth上标好点之后让船去跑。输出坐标点基于WGS84坐标系。

1. 7月20日明天要做的事：
   * + - 1. 把GPS拿出去试试精度与UBX协议传输的数据质量.
         2. IMU输出的数据貌似在之前的船上被配置的只输出角度值。要是换了一个比较智障的IMU的话不就废了。所以明天把IMU改成输出原始数据（磁场、加速度、角速度）。
2. 7月21日调试中发现的问题：
   * + - 1. IMU输出信息目前是一次33Bits，但实测发现数据处理的速度没有数据发送的速度快，导致缓冲区很快饱和。（实际现象是缓冲区越大，饱和到来的时间越滞后）。Brainstorm之后认为这是Debug过程中会出现的问题，实际过程中STM32一次处理数据为11Bits,只要在不超过调度时间的基础上多处理几帧数据，就没有太大问题。
         2. 之前始终不相信JY901只能用TTL电平，然后直接把JY901插在电脑的USB串口上了。直到JY901没有了反应。看来的确是不支持RS电平。有些操之过急了。会好好反省的。
3. 整体系统架构：



1. 下位机->上位机通信协议(Eastar Protocol 0.5)：(向量图)



1. 上位机->下位机通信协议(Eastar Protocol 0.5)：(向量图)



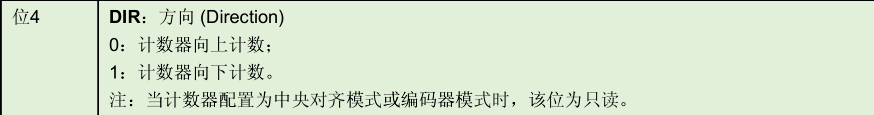
1. 上位机的手动、自动操作模式的切换通过一个状态机实现：



1. 7月25日-31日，上位机的开发：
   1. 虽然上位机框图中采用了FPGA开发方法设计了系统状态机，但是这毕竟是硬件\嵌入式开发的思路，不是计算机的思路。最近发现树莓派可以多线程进行，这样不就避免了在程序跑某一部分的时候卡死吗！并且可以一边接受信息，一边进行控制。所以最近试图把上位机开发为多线程的模式。
   2. Python的一个重要特点是面向对象，和C语言不同。这里我用一个class描述一个环形缓冲区，用来给树莓派读取下位机数据使用。一个问题是环形缓冲区的Head和Tail在某一时刻必然会被重置，这会导致某一时刻的数值不可用。如果是数据流的话还好，如果数据恰好在此时中断，那么环形缓冲区中存储的大量数据都会被认为“不可用”。这个问题以后在解决。目前的解决方案是降低环形缓冲区的重置率。
   3. 在编写线程时发现的问题：Ctrl+C中断程序运行后，线程实际上还在后台运行（现象是在线程中加入了print函数的话，中断主程序后后台程序仍然在一只输出）。查阅资料发现线程有setDaemon与Join方法可用，等待测试。这个问题不解决的话很危险。这个可能也就是上一代智能船突然宕机的时候电机仍然有输出的原因。很危险。玩意出现毛病了，船自动停下了还好。千万别一出毛病就自己跑海里了。
   4. 发现的新问题：如果船在运行过程中，假设已经经过了5个点，此时系统突然掉电后恢复，这个时候路线规划程序会使船直接驶向第一个点，重新开始跑。似乎不太合理。解决方案是每次系统上电计算当前坐标与路线各个坐标点的距离最小值，从最小距离的那个点开始航行。
2. 7月31日，一些感想：
   1. 几天在看别人开发的无人船视频的时候隐约看到视频画面中无人船可以与电脑实时连接，并且可以显示当前状态。视频上软件的字样是Ardupilot。这是一个自动导航库，也算是一个开源的工程。如果直接使用这个工程的话，的确，可以直接拿到更好的效果，看起来也更炫酷。但是，对于这种开源文件，我个人是不太喜欢用的。至少核心技术还是别人开发的，别人免费给你用，一方面可以理解为方便了自己的开发，另一方面也让自己少学习了很多东西。就像自己看一场电影和看了一个5分钟的电影解说视频一样。为什么现在那么多核心科技掌握在老外手里？为什么国内开源的技术少？为什么要去制作一个东西，总是要去看外国人的研究资料？科技和技术不是一码事。所以，Ardupilot这种东西，不去排斥，也不去主动使用。可以吸收里面的思想，“师夷长技以制夷”，也可以在比赛中被使用这种库的赛队打败，因为它的核心并不是这个赛队研发的，而是无数开源开发者研发的。真正能推动科技发展的，绝不是“会用了就行”这种心态。
   2. 真正研发不能只靠臆想就能完成。需要很多后期的测试、修正。甚至可以说后期的测试修正环节比研发环节更为重要。
   3. 研究传感器价格时发现了个有意思的事。。
      1. 陀螺仪+加速度传感器MPU6050裸板价格：25
      2. MPU6050+磁力传感器5883价格：50(顶多)
      3. MPU6050+磁力传感器5883用一个pcb封装 价格：90
      4. 实验室买的MPU6050+5883+一个塑料破壳子，价格：150

暴利了吧。。塑料壳子直接把IIC接口的引脚给封在里面了，只露出来了USART引脚还这么贵

1. 7月31日，新的计划：
   1. 需要东西测试程序。买了一个质量很好（价格也很高，甚至比之前的船壳‘飞鱼号’还贵）的车架子，带上电机、电池、电机驱动，等等。车模选的是后轮双电机、前轮舵机控制转向的，主要原因是考虑到以后想要把船做成舵控制转向，这个车模和以后的预想非常相似，非常合适。等待到货。这两天开始提前编写程序吧。
   2. 把之前在51单片机上开发的Nrf24L01+模块拿回家了，今晚上开发了一晚上，成功完成PC端与树莓派端通信，加上使用的是PA版24L01,实际遥控距离在空旷地带可以达到100米，可以用于数传、控制、等等。很好，进步很大。目前PC端修改数据，到树莓派端更新数据的延迟压缩到了大约0.5秒，接下来尝试进一步压缩。
      1. 在提高24L01响应速度时发现的问题： 23:39 第一次响应速度测试时系统挂机。应该是缓冲区溢出了。结论：不可以过度追求速度，否则导致2401缓冲区溢出，系统死机。
2. 2019-8-1 到2019-8-2 ,发现与解决的问题：
   1. 树莓派的主串口ttyAMA0似乎被蓝牙使用了，因此留下了一个始终不稳定的miniuart端口，一般不要用。解决方案：
      1. In brief, ***dtoverlay=pi3-disable-bt*** to the /boot/config.txt file to enable Device Tree Overlays. Note that the -overlay.dts part of the filename is removed
      2. 然后，禁用蓝牙。（我发现开机的时候蓝牙占用了很长时间。关闭蓝牙可以加速开机）
         1. sudo systemctl disable hciuart.service
         2. sudo systemctl disable bluealsa.service
         3. sudo systemctl disable bluetooth.service
   2. 安装python3-distutils:
      1. 在apt update和apt upgrade之后: sudo apt-get install python3-distutils
   3. 安装pip:
      1. sudo apt-get install python3-dev
      2. sudo apt-get install python3-pip
      3. 更换源：<https://blog.csdn.net/yuisyu/article/details/81587800>
   4. 安装需要的模块:
      1. Sudo pip3 install serial  之后 sudo pip3 install pyserial
   5. 发现树莓派对电源的要求极高，如果电源线不合适、电压源不稳定等等原因，可能导致更新软件、系统运行时（甚至CPU负载较高时）就会触发低压保护而关机，可能导致文件损坏。这一点一定要注意
3. 2019年8月3日：买的小车底盘到了。准备着手测试。
   1. 用驱动板驱动电机的时候的PWM引脚根据我的经验要选一个和串口无关的，因为之前在测试TIM1的pwm信号时发现由于USART1在使用，所以PWM信号根本没有的现象。这二者应该是排他性的。因此尝试用TIM3做PWM发生的时钟源。
   2. 控制器的电源最好和电机电源分开。也就是说用两块电池。这样可以避免干扰，再者可以防止电机挂了的时候控制器也挂。好歹也能做个“植物船“是不？
   3. 发现一个问题：在给电机驱动板供驱动板电压（非功率电压）时，如果接入的是STM32的5V接口，则电机无输出，但驱动板显示已供电；接入STM32的3V3接口时，驱动板显示已供电，且电机可转。这是为什么？原因待查。
   4. 2019年8月4日 00:00:08 ，8月四号啦！今天的第一个问题：pwm输出波形：当前的配置是周期固定，向上计数模式，计数记完之前输出低电平；一旦到达时间则输出高电平。好了晚安！！
4. 2019年8月4日 08:53:01 ：
   1. 电机的编码器知识：
      1. 编码器线数就是编码器的分辨率，也就是一转所发出的脉冲数，编码器没有倍频技术，是接收器处理脉冲时通过编码器输出脉冲（A与B相）的相位差关系实现倍频技术的。富春山号的编码器是1560线。
      2. 360度除以线数就是一个脉冲表示的度数，比如1000线的旋转编码器，那么一个脉冲表示转了0.36度。
   2. STM32编码器：定时器初始化好以后,任何时候CNT寄存器的值就是编码器的位置信息,正转他会加反转他会减这部分是不需要软件干预的,初始化时给的TIM\_Period 值应该是码盘整圈的刻度值,在减溢出会自动修正为这个数.加超过此数值就回0。
   3. 方向信息在CR1的DIR位里：(TIM2->CR1&0X10)>>4



* 1. 舵机经过测试，PWM脉冲为50Hz、高电平时间为1.25ms时，车头为正前方。Pwm参数范围为：(车头左偏67.5°)15-（车头正前）25-（车头右偏67.5°）35
  2. 左右轮为镜像关系，一个为顺时针转时，另一个必为逆时针转。因此编码器输出有一个负号。尝试一下AB相反接。
  3. 因为采用了舵机的控制方式，因此左右轮必须转向相同，不允许出现转向相反的情况！
  4. 左右轮共用TIM3的两个通道，因此无法实现差速！如果有必要，回头会改造一下。留好了差速控制的Motor\_Control函数接口。
  5. 电机转速有时会突变，实验发现问题出在Round计数器多加或者多减上。
     1. 耐用性测试：正、反转时，电机保持PID偏差不大于1cm/s稳定运行时间
  6. MPU6050的低通滤波器一般设置为采样率的一半左右。

1. 调整IMU模块：

MPU6050与HMC5883均采用右前上坐标。

加速度计模块默认为左前下坐标，这里给强行设为右前上坐标：右:x轴(Pitch) 前：y轴(Roll) 上：z轴(Yaw)。

* 1. GPS的磁力计箭头指向为x轴，反装。

1. 2019年8月8日 16:41:00
   1. 下位机->上位机通讯模块终于做好了。接下来时解析速度测试。测试时发现会有溢出的情况，实验发现实际上是环形缓冲区数据未处理完就被串口输入给冲刷掉了。解决方案：
      1. 尝试加入环形缓冲区输入写保护，写入数据不得冲掉未处理数据
      2. 加速读取速度，减慢输出速度
   2. 解决了PID调整电机在电机正反转交替的时候会出现peak值的问题。首先尝试了低通滤波器->PID，发现这样会减慢系统的动态响应。因此最后不得不放弃了。采取的解决办法是在PID内加入了一个专用剔除peak值语句.似乎效果不大..?后面用别的方法解决了这个问题。
   3. 加入了磁力计Hard Iron校正，陀螺仪静差校正。
   4. PID算法迭代频率不够高的话容易出问题。
   5. 加入了上位机下线保护功能。当上位机由于意外突然下线时，保证载具不会仍然一直跑。
   6. 电机编码器读取数据总有peak值。
   7. 已知的问题：（划横线表示已解决）
      1. ~~PID的peak；~~
      2. ~~Madgwick偏航角开机后不收敛；~~
      3. ~~32不解析数据；~~
      4. ~~舵机抖动；~~
         1. 可能原因：数字模拟不共地
      5. ~~24L01不回传数据~~
      6. ~~初始偏航角算法不算精致。~~
      7. ~~Yaw梯度下降算法漂移：~~**尝试增大滤波增益beta**
      8. 一些思考：做控制，一定不要急于求成。有问题，立马解决，不要留着。最近由于已经能看出做出来的实际效果了，因此有些浮躁，程序也出现了很多bug。这怎么能行？突破不只是体现在口头或者ppt上，最重要的是体现在实物上。规格严格功夫到家，不是白话的。

务必稳扎稳打。

1. 遥控模式：**遥控模式应该直接对下位机操作**，而不需上位机的参与。这样上位机如果挂了，下位机依然可以遥控回来。
2. 本着对无线的执着，准备在树莓派与32之间也加入Long Range模块，这样可以在测试的时候树莓派在手里，只有32在水下跑，即使船泡水了，至少还救了Raspi.

项目试验成功（2019-8-23注：最终放弃了LoRa模块，原因是延时较长，且增加了系统的不稳定性）

**UAS开发第二阶段**

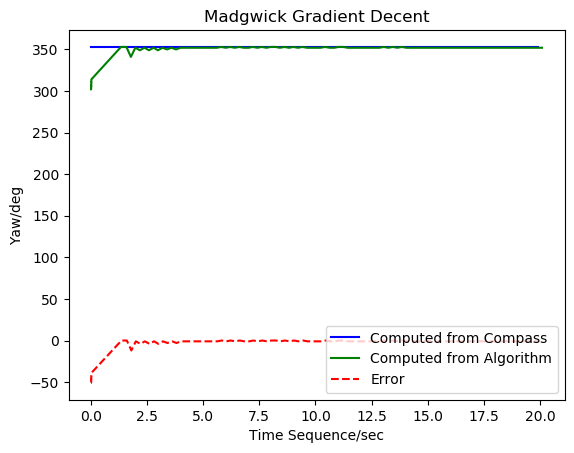
1. 基本框架已搭建完毕，进入以稳健性升级以及外设改装为主题的开发阶段。
2. 为了确保系统的稳健性，进行系统耐用性测试。
   1. 动力稳健性测试标准：在一直运行在标准状态下时，原型机突然进入停机状态的频率不高于0.1Hz,时间不超过一秒则为正常运行。否则计为系统中断。

|  |  |
| --- | --- |
| 测试时间 | 不间断运行最长时间 |
| 2019-8-13 10:22:34 | 3Min58Sec |
| 2019-8-14 14:48:53 | 5Min21Sec |
| 2019-8-15 08:25:08 | 20Min23Sec  注：此次计时结束的原因是电池电量耗尽。实际可运行时间长于此数值。对此结果满意。 |

* 1. 算法稳健性测试标准：在一直运行在标准状态下，将原型机固定不动。下位机发送给上位机的偏航角数据发生正负5°的漂移时，计为算法出现故障（偏航角漂移）。否则计为系统正常运行。

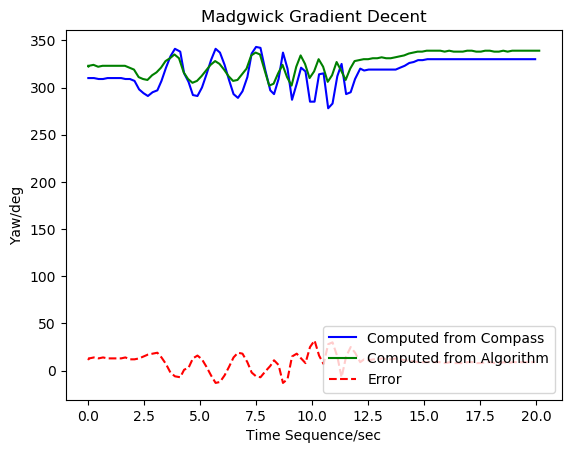
|  |  |
| --- | --- |
| 测试时间 | 可容忍漂移运行最长时间 |
| 2019-8-16 18:18:09 | 1800sec，即30Min内未出现不可容忍偏差 |
| ------ | ------ |
| ------ | ------ |

* 1. Madgwick算法评估：
     1. 上电收敛速度：



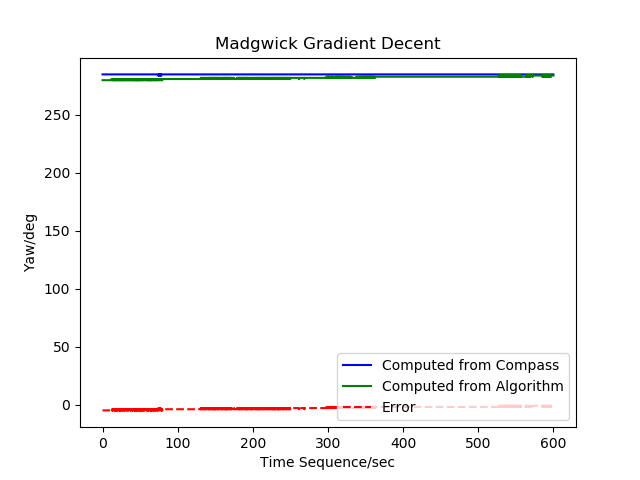
可见在比较随意的参数下可在2.5秒内收敛至当前状态

* + 1. 动态响应情况：



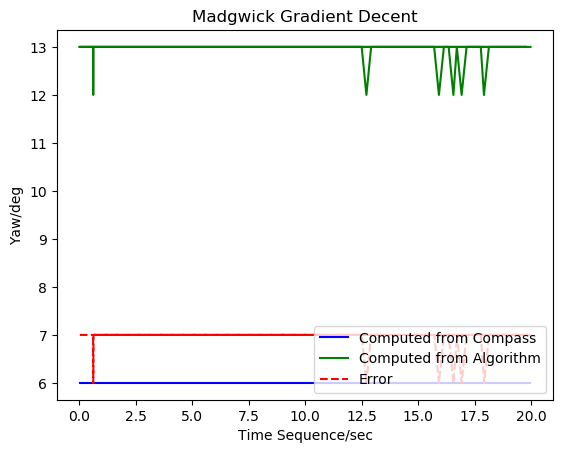
此时参数未细调。可见动态响应速度较快。

* + 1. 零漂情况：通过编写python脚本发现，Madgwick算法存在静态误差，即输出偏航角会随时间偏移。以下为下位机上电后的10分钟内静漂情况：

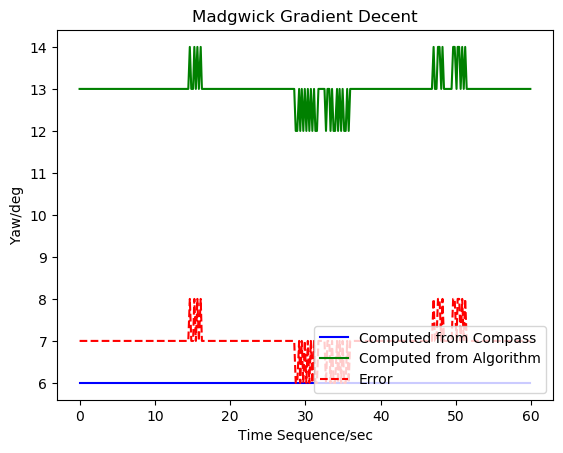


蓝色实线为实际偏航角，绿色实线为Madgwick输出偏航角。红色虚线为误差。实验时将原型机放在静止位置不动。

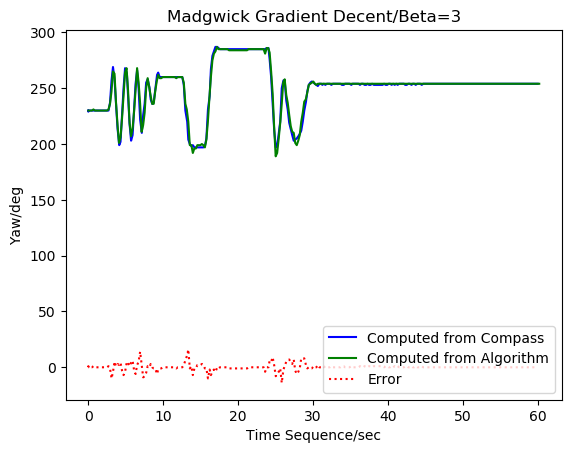
---长时间放置后，零漂很大---



* + 1. 会出现高频抖动。原因未知。

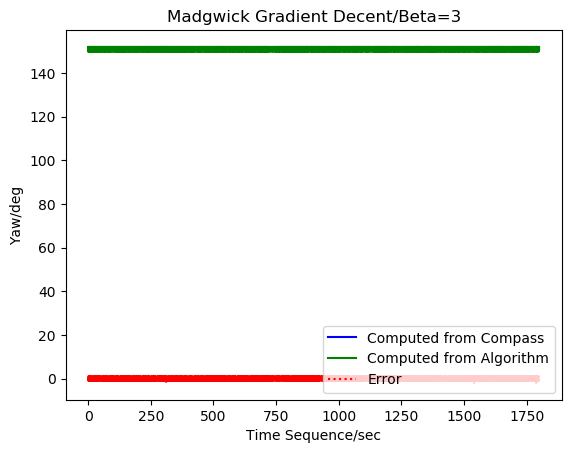


* 1. 2019-8-16 16:38:03。今天到现在为止的主要工作内容：修正Madgwick滤波器；给加速度计和陀螺仪各配备了三通道的数字滤波器。修正了磁力计模块的轴向alignment等。在140次的实验之后终于得出了比较满意的结果。见下图：



响应十分到位、准确。但滤波器增益有一点过大，导致在静止时会有高频震荡（图中可见）。只需要减小滤波器增益就可以解决。

* 1. 零漂解决情况：



实验时间为30分钟。在实验时间内除了滤波器的震荡之外没有显著零漂出现。效果十分满意。

1. 系统进一步升级：
   1. 将串口数据升级为DMA发送，提升系统运行速度，榨干STM32硬件性能。
      1. 难点：DMA在读取时想要与环形缓冲区结合，这样读取数据的Ring Buff Length就变得不好确定，进而影响了缓冲区内数据的读取。
      2. 解决方案：
         1. DMA读取字节数目设置为环形缓冲区的大小
         2. 加入串口空闲中断，当出现空闲中断表示此时一帧数据读取完毕
         3. 使用DMA\_GetCurrDataCounter()函数计算出本次的数据接受长度n，将Ring Buff Head加上n, Ring Buff Length加上n;此外，当Ring Buff Length大于Ring Buff 的大小时，关闭DMA读取。等待缓冲区内数据被读出。
   2. 加装看门狗，防止系统出现意外，程序跑飞。
   3. 在任务驱动器LOOP()函数中加入了有限状态机。用来使下位机在初始化状态、正常运行状态、Debug状态中转换。由于目前处于开发阶段，只加入了有限状态机的驱动器，未加入下一状态的判断函数。
   4. 加装遥控器模块
      1. 三种模式：遥控模式、自主航行模式、采点模式
      2. 遥控模式可脱离上位机使用
      3. 自主航行模式、采点模式必须有上位机
   5. 2019-9-7：配置遥控器信号丢失后的执行序列：继续执行自动导航模式
   6. 树莓派端程序设置为开机自启，注意：
      1. 因为这个开机自启是在机器还没完成初始化进程的时候就启动的，所以在python程序的开头最好加入一个time.sleep函数休眠20秒，强迫程序在系统初始化成功后进行运行。
      2. 将MainSequence.py设置为权限777：

sudo chmod 777

* + 1. 启动开机自启服务：

sudo service my\_start start#启动

* + 1. 关闭开机自启服务：

sudo service my\_start stop#停止

1. 实地测试：
   1. 2019-8-18 19:47:19，实地测试结果：

GPS坐标点由于下位机的原因经常停止给上位机发送， 因此上位机也就无法进行导航。初步认为问题出在与GPS通信串口波特率（921600）过高且\或GPS输出坐标点频率（10Hz）过高导致。已在工程中加入了对这两项的配置。波特率可设置为115200\460800\921600，GPS输出坐标频率可设置为10Hz、5Hz。

* 1. 2019-8-19 22:39:36，实地测试结果：

仍然是GPS输出定位点频率过低的问题。此外在Debug的过程中调低了上位机端环形缓冲区的大小，原因是环形缓冲区过大会使数据有延时，修改之前数据掩饰约为5秒，修正之后降低到了1秒以下。此外调试下位机端的USART1-DMA发送通道成功，USART1-RxDMA未调试，USART3-接收DMA（即GPS接收端）为半调试状态，具体效果还需明天进一步确认。

此外发现无线串口模块经常出现死机的现象，具体出现原因未知，可能的原因有：缓冲区溢出（但无线串口使用的是环形缓冲区，理论上不会）；供给模块的电流不足致使模块错误工作；接线不良。

修正了上位机的Bug:在未选择模式之前就要求读取MapData，会报出Error；删除了上位机的Nrf24L01模块，具体原因是24L01在上位机上用处不大，此外如果需要数传的话需要一个距离更远一些的模块。

* 1. 2019-8-20 18:55:11，实地测试结果：

使用的无线串口模块技术还很不成熟，大批量的数据传输会使缓冲区溢出而进一步使无线串口死机。因此不得不删减无线串口模块，上下位机使用线连的方式。

* 1. 2019-8-21 19:36:14，实地测试结果：

GPS模块传输的数据仍然会在一定时间后停止发送。原因未知，正在分析可能存在于程序中的错误；GPS串口DMA测试正常。

* 1. 2019-8-22 21:42:23，实地测试结果：

舵机会在一段时间之后停止工作，像是受到了干扰。多次试验之后认定：程序正常，而舵机异常、GPS数据异常全部因为模拟地与数字地未分离导致的。此次将模拟地、数字地分离，明天查看效果。

* 1. 2019-8-23 15:06:30，实地测试结果：

已将数字地与模拟地分离。载具表现有很大的提升。

测试分为三个项目。(1)采点模式：原型车可按照采点轨迹前进，且能以较低的误差计算距离下一个轨迹点的距离。(2)在Google Earth上标记定位点，实验结果证明除了Google Earth上标记的点和真实点有大约2m的水平漂移之外，一切正常。(3)导出载具走过的轨迹点，导入Google Earth。采点有些密集，数据正常。



Google Earth取点->载具执行



载具取点->Google Earth重现