平成 26 年度

卒業論文

題目 UAV を利用した自律航行に関する研究

学科名 海事システム工学科
（情報システムコース）

学籍番号 1121024

氏名 須合翔一

指導教員 久保信明

目次

1　序論

1-1　研究背景

無人航空機:Unmanned Aerial Vehicle(以下 UAV)は第一次世界大戦中からの存在し、発展を続けている。軍用機、民生機が存在するだけでなく、大きさも手のひらに乗るような小型の物から全長 35m クラスの物まで存在する。また、形状も固定翼機と回転翼機共に存在する。

近年、民生用 UAV の販売が加速し、数千円で手に入る玩具レベルの物から、数十万円の業務用レベルまで幅広いレンジで販売されている。民生用小型 UAV の発展は特に目覚ましく、購入時のままの状態で Waypoint 飛行を行えるモデルや、機体の上下が反転しても継続して飛行可能なモデルも登場してきている。

民生用 UAV の販売が加速されると同時に UAV の実利用も増えてきている。

利用内容としては、災害調査、地形調査、高所での画像撮影、遠隔監視等と人間が行えない事を代行させる形の物が多い。

また、米 Amazon 社をはじめ、業務で UAV の利用を検討する企業が増えてきており、UAV の利用増加は確実である。

本学で船舶について学んだ私は、UAV を船舶で利用して何か支援出来たら便利であると考えた。そこで、調査を行った所、船舶の航行中に船舶を俯瞰出来るシステムや、極地において最適経路で航行するため先回りして海氷の様子を調べられるシステムが必要とされている事がわかった。現状の航海計器では船舶全体を俯瞰する事は出来ず、マストにカメラを付けて船舶を監視しても死角が存在するという報告が上がっている。

一方、図 1-1 は南極探査船「しらせ」が接岸を目指しているところの写真である。極域では非常に氷が厚く船舶が進むのに多くの時間を要する。また、2013、2014 年シーズンでは海氷状況が厳しく、2 年連続接岸を断念した。UAV を用いて空撮を行い海氷の少ない経路を見つけ出せれば接岸がスムーズに行えるかもしれない。

上述のような理由から、UAV を用いてどこまで航行支援を行えるか調査研究を行うことにした。

図 1-1 南極探査船「しらせ(砕氷艦)」

1-2　研究目的

UAV の船舶利用では、上空から船舶及び海上を撮影した映像が重要となる。

また､海上での UAV 利用時は船舶が動いているので UAV が船舶を追尾する必要があると考えた。

それらを満たす UAV を制作し、実際に飛行及び空撮をさせて、前項で上げたような船舶支援をどこまで行えるか確かめる事が目的である。

本研究では、実験場所や時間的制約のため小型 UAV を扱った。以後、本文では UAV とは小型 UAV の事を指す。

2　UAV の制作

2-1　参考機体

今回、UAV を制作するにあたって、DJI 社製 Phantom 2 Vision+(図 2-1、図 2-2)（以下、Phantom2）を性能の目標とした。

この機体の特徴は、購入した状態で空撮、緯度経度を指定しての Waypoint 飛行が可能である。空撮も HD 画質で撮影が可能。Waypoint 飛行時の緯度経度も小数点以下 5 桁まで入力可能である。

製作する UAV は大きく分けて UAV 本体、制御部、通信部、撮影部から構成される。

各部位毎に参考機体と同等の性能を確保することを目標とし製作にあたった。

Phantom2 の使用詳細は表 2-1 に記す。

図 2-1 Phantom2 vision+ 機体

図 2-2 Phantom2 vision+ 送信機

表 2-1 Phantom2 の仕様

機体	重量	1242g
	全長	対角線;350mm
	ホバリング精度	垂直: 0.8m; 水平: 2.5m
	飛行時間	約 25 分
カメラ	解像度	4384×3288
	HD 録画	1080p30
	画角	110 °/ 85°
制御	手動	付属送信機:動作周波数 922.7MHz～927.7MHz
	自動	スマートフォン用アプリ DJI VISION にて WayPoint 飛行

2-2　UAV 本体

参考機体である Phantom2 は 4 枚羽のヘリコプターで、最大 25 分の飛行が可能である。製作する機体も Phantom2 同様 4 枚羽にしようとしたが､機体の安定性を考えて羽を多くする事とした。

6 枚羽と 8 枚羽の機体が存在し、後者の方が機体安定性は高いが、羽根の枚数が増えるため飛行時間が短くなってしまう。Phantom2 と大きく飛行時間に差異が出ないようにするため、機体は 6 枚羽のヘリコプターとする事とした。羽の枚数と機体動作に関しては表 2~2 に記載。
また、機体には Waypoint 飛行用の制御装置、空撮用カメラ、ジンバルを載せる必要がある事からそれらを積載可能な大きさの機体を選ぶ事とした。
条件を満たす機体を株式会社アトラスから購入し組み立てた。

表 2-2 羽の枚数と飛行能力

羽の枚数	機体安定性	飛行時間（消費電力）
4 枚	低	長い（低）
6 枚	中	普通（中）
8 枚	高	短い（高）

2-3 制御部

2-3-1 飛行安定化装置

現在 UAV が誰でも手軽に運用出来るのは、飛行制御システムのお陰である。参考機体である Phantom2 にも DJI 飛行安定化装置 NAZA-M(図 2-3）が内部に搭載されており、姿勢保持、位置保持を同システムに依存している。
一から UAV の姿勢保持システムを作るのは膨大な時間を要するので､今回の機体製作においては飛行安定化装置を用いることとし、DJI 社製の飛行制御システム Wookong-M(図 2-4)を使用する事とした。
両者は共に DJI 社製の飛行安定化装置であるが、前者は非常に小さな機体向けの装置なので、Phantom2 より大きい制作機には、大型機対応の後者を選択した。
飛行安定化装置にはコントロールシステム以外にも、ジャイロセンサー、GPS モジュールが搭載されているので機体の姿勢及び位置を容易に保持することが出来る。
制作機で仕様した Wookong-M のホバリング精度は垂直方向±0.5m、水平方向±2.0m であり、Phantom2 搭載の飛行安定化装置 NAZA-M のホバリング精度の垂直方向±0.8m、水平方向±2.5m より優れている。

図 2-3 NAZA-M

図 2-4 Wookong-M

2-3-2　自律飛行用制御装置

自律飛行に用いる制御装置は制御プログラムが書き換え可能であるゼノクロス社製「AP-CUB-DIY LITE」(図 2-5)を使用した。こちらの制御装置はラジコン用プロポ（リモコン）を使用して、制御モードを切り替える事が可能である。制御モードをラジコン入力モード、自動操縦モードと 2 つ製作することで、制御装置を介してもラジコン入力モード時は通常のラジコンヘリコプター同様に手動にて前進後進、上下左右等の移動を行う事が出来る。
また、自動操縦モード時は事前に書き込んだ制御プログラムを実行し、機体を移動させる。
制御プログラムは自由に設計出来るので、ラジコン入力による飛行と自動操縦を組み合わせた設計も可能である。また、GPS モジュールを搭載しているので、位置データを仕様することも可能である。

図 2-5 制御装置「AP-CUB-DIY LITE」

2-4　通信部

2-4-1　送受信機

UAV 機体の RC 送受信機には JR PROPO 社製 XG8(図 2-6)、RG831B(図 2-7)を使用した。
送受信機間の通信周波数は 2.4GHz 帯。8ch の送受信が可能であり、これにより機体の制御のみならず、制御モードの切替、カメラ固定用ジンバルの操作が出来る。

図 2-6 送信機 XG8

図 2-7 受信機 RG831B

2-4-2　通信装置

通信装置には Digi International 社の小型モジュール「XBee PRO S2B」(図 2-8)を使用した。このモジュールは短距離無線通信企画 Zigbee 規格に基づいて、最大 3.5km のレンジでシリアル通信を行う事が出来る。
本研究においては UAV の飛行中 PC に対して、緯度経度の位置情報を始めとした、速度、高度、針路、WP 番号等の値を制御装置からリアルタイムに送信している。PC 側で受信したデータは管制ソフトでリアルタイム監視するか、記録ログを見ることで確認することが可能である。

図 2-8 ZigBee 受信機「XBee PRO S2B」(PC 側)

2-5　撮影部

制作した機体にはカメラは搭載されていないので、外部カメラを積載する。
カメラの固定は、UAV 機体のスキッド部にジンバルを取り付ける形で行った。
積載するカメラの重量は飛行時間に影響するので、HD 画質での撮影が可能かつ軽量である事が望ましい。
また、ジンバルで固定する必要があるので、対応アクセサリーが多く、それらの条件を満たす GoPro 社の HERO 3 使用した。また、FeiyuTeac 社の 2 軸ジンバルを使用した。

図 2-9 HeiyuTeac 社製 2 軸ジンバルと固定中 GoPro HERO3

2-6　管制ソフト

UAV の飛行状況確認や Waypoint の設定を行う飛行管制ソフトはゼノクロス社製 Flight viewer を使用した。
このソフトは機体制御部から送られてきた位置情報、各種出力変数、飛行状況等をリアルタイムで確認する事が出来る。
また、地図を設定することで、図 2-10 のように UAV の現在地及び周辺情報に加え、UAV の飛行の軌跡を表示することが可能である。
PC 側でこのデータを参考にして UAV の制御を容易に把握することが出来るので、関数に代入する制御値の最適化及び、不具合発生時対応の効率化に寄与している

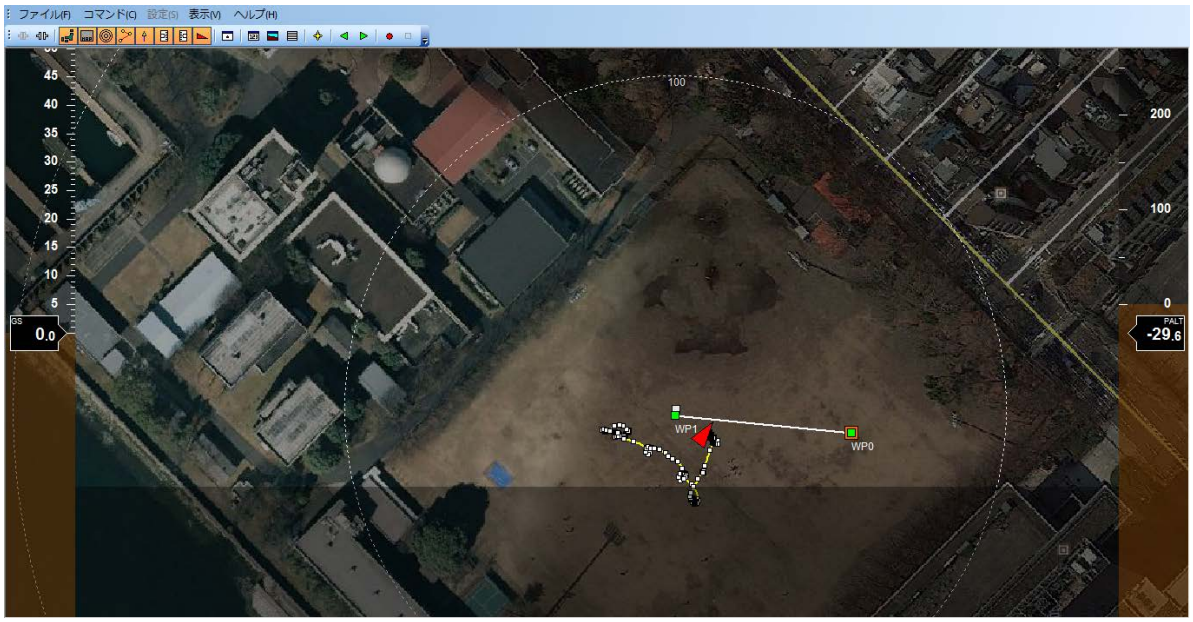

図 2-10 FlightViewer 実行画面

2-7 完成した機体

Phantom2 を参考に制作した機体(図 2-11)である。制作した機体の仕様は表 2-3 の通りである。

図 2-11 制作した機体

表 2-3 制作した機体の仕様

機体	重量	4410g
	全長	対角線;900mm
	ホバリング精度	垂直: 0.5m; 水平: 2.0m
	飛行時間	約 20 分
カメラ	解像度	4000×3000
	HD 録画	4K15 /1080p60
	画角	122.6°/ 94.4°
制御	手動	XG8: 2.4GHz 帯
	自動	AP-CUB-DIY LITE にて制御。管制ソフト Flight viewer

3　飛行実験及び結果

3-1　実験概要

UAV で船舶の追尾を行う前に、UAV を自動で制御出来るようにする必要がある。制御例として WayPoint 飛行を行う事とした。

Phantom2 の waypoint 飛行を参考にして、プログラムを制作し制作機で waypoint 飛行を行い、Phantom2 と比較してどれ位の精度で waypoint 飛行できているかを検証する。

3-2　事前実験

3-2-1　UAV による静止時測位

製作した UAV による自律飛行実験を行う前に、位置情報がどの程度の精度で計測出来るか測定した。

そこで、本学第四実験棟屋上にて約 30 分間 UAV 機体を設置し、静止時データの採取を行った。採取したデータは図 3-1 AP-CUB-DIY LITE の静止時観測データのようになった。計測した緯度経度の平均値を座標中心とし、座標中心からの距離を表している。

グラフは取得されたデータの平均位置を基準点とし、基準点からの距離を表している。

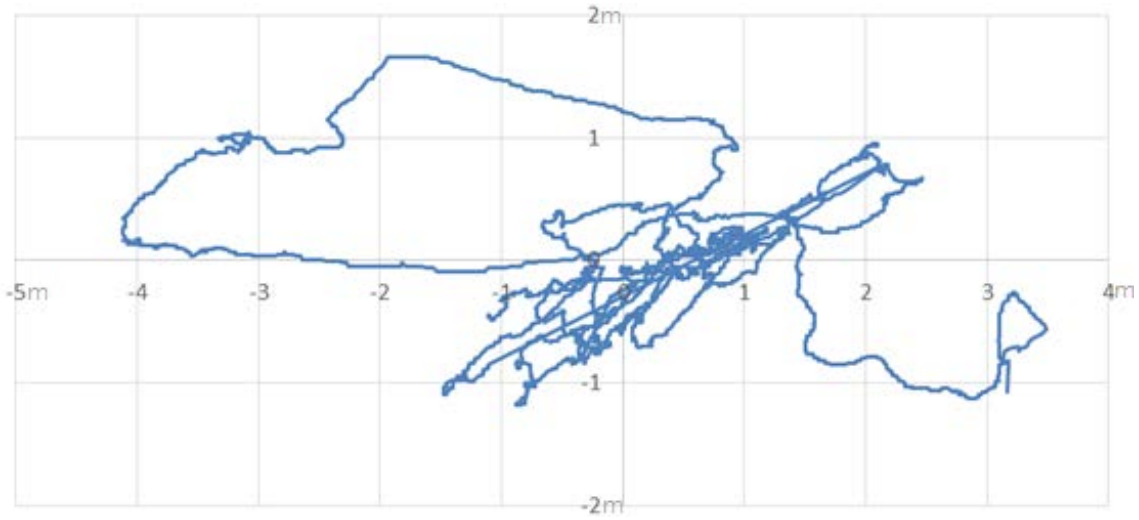

図 3-1 AP-CUB-DIY LITE の静止時観測データ

3-2-2　GPS ロガーによる静止時測位

本研究にて製作した UAV は制御装置「AP-CUB-DIY LITE」によって位置情報を記録することができるが、Phantom2 は位置情報を記録することができない。

Phantom2 と自作機の自律飛行の精度を比較するために、Phantom2 には GPS ロガーを積載し、ロガーで測位した位置情報を利用することとした。

積載する GPS ロガーは Transystem 社の「747pro」(図 3-2)である。こちらは 47 × 72 × 20 (mm)と非常に小型ながら、水平位置誤差±3.0m (メーカー公表値)で位置情報を測定可能である。

前項の UAV の静止データ計測時同様、屋上にロガーを設置し、5 分間静止データを計測した。

計測したデータは図 3-3 に示す。計測の結果、水平誤差は約 2m の範囲で収まる事が確認された。

図 3-2 Transystem 747pro

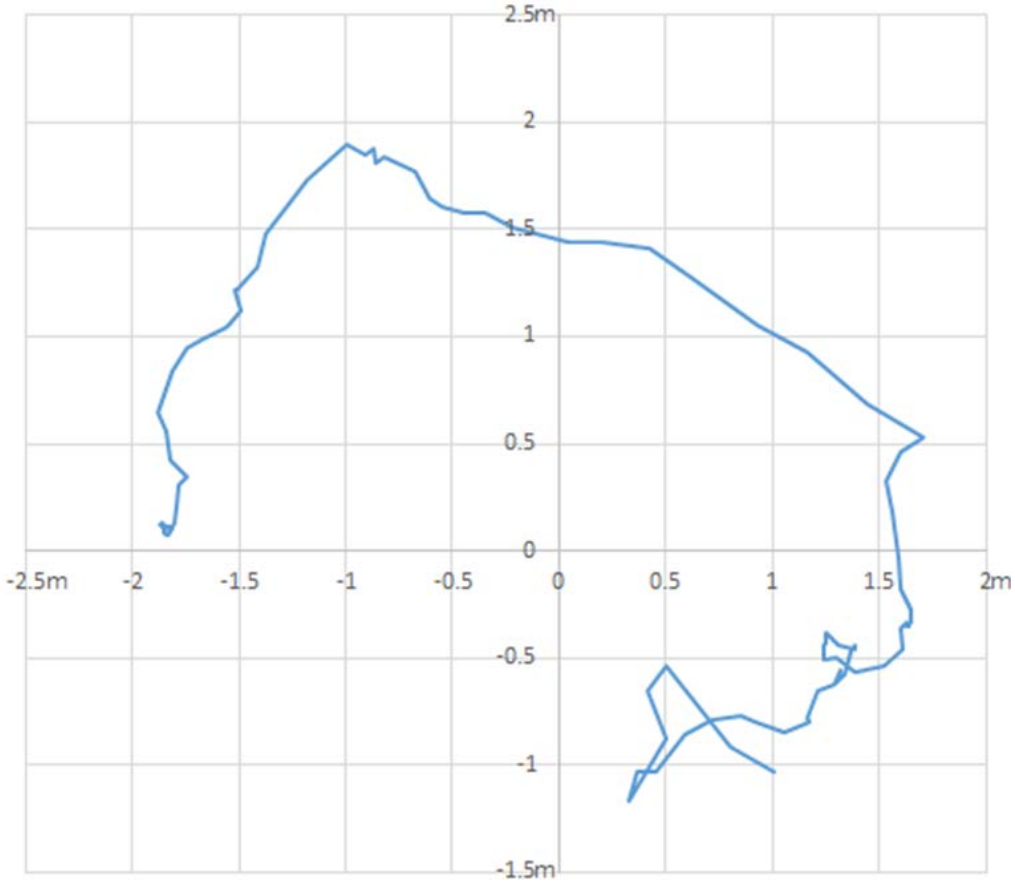

図 3-3 747pro の静止時観測データ

3-3　　市販機体による自律飛行実験

自作機体による自律飛行実験を行う前に、Phantom2 による自律飛行実験を行った。

Phantom2 はスマートフォン用専用アプリ「DJI VISION」をスマートフォンにインストールするだけで、購入したままの状態で自律飛行を行う事が可能である。

「DJI VISION」では図 3-4 のように画面上をタッチする事で waypoint を指定出来る。Waypoint を更にタッチすると、waypoint の緯度経度高度の数値入力が可能となる。

「DJI VISION」を使用して Waypoint 飛行を行うには、GPS 衛星が 6 基以上受かる必要がある。購入時の Phantom2 の GPS 受信感度では GPS 衛星が 6 基受からない時があり、Waypoint 飛行が行えない事がある。

そこで Phantom2 を分解すると図 3-5 のようになっており、グレー部の所に GPS 用アンテナが搭載されている。受信感度を向上させ、6 基以上の GPS 衛星が受かるように GPS アンテナ格納部をアルミテープで被う加工を施した(図 3-6)。

Phantom2 では電波強度を確認する事はできないので、効果を定量的に示す事できない。

しかし、衛星配置によっては 10 機以上の GPS 衛星が受かるようになったので、有用であった事は確実である。

上述の加工を施した後、学内グラウンドにて waypoint 飛行試験を行った。

実験日時 2015 年 1 月 10 日　17 時頃

Phantom2 単体では飛行位置のログを記録できないので、747pro を Phantom2 に積載し実験を行った。

スタート地点から WP4 点を飛行し再びスタート地点に戻ってくる形の飛行を 2 回連続で行った。Waypoint 飛行時、機体は waypoint 方向に機体を回頭させた後 waypoint まで直進し、waypoint 到達後、次の waypoint へ機体を再び回頭させる動作をスタート地点に帰還するまで繰り返した。

2 回の施行における軌跡は図 3-7、図 3-8 に示す。

また、設定した各 Waypoint の座標及び、再接近距離は表 3-1 のようになった。

図 3-4 DJI VISION 起動画面

図 3-5 Phantom2 加工前

図 3-6 Phantom2 加工後

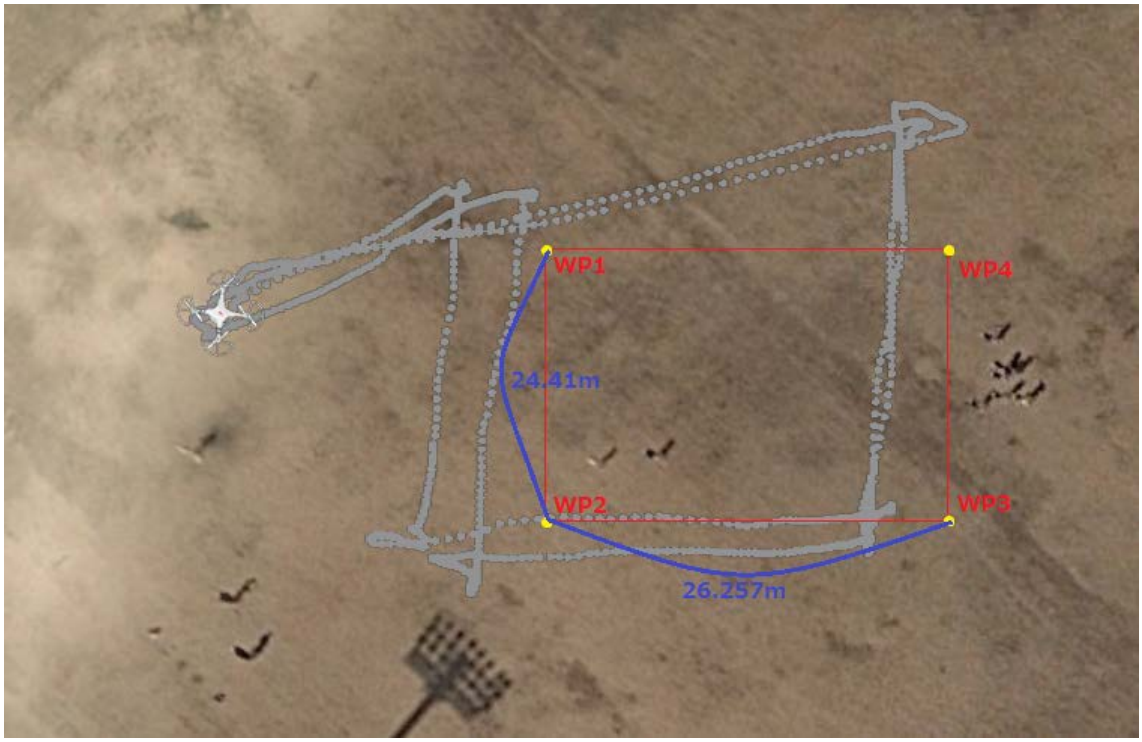

図 3-7 waypoint4 点と Phantom2 の軌跡(地図上)

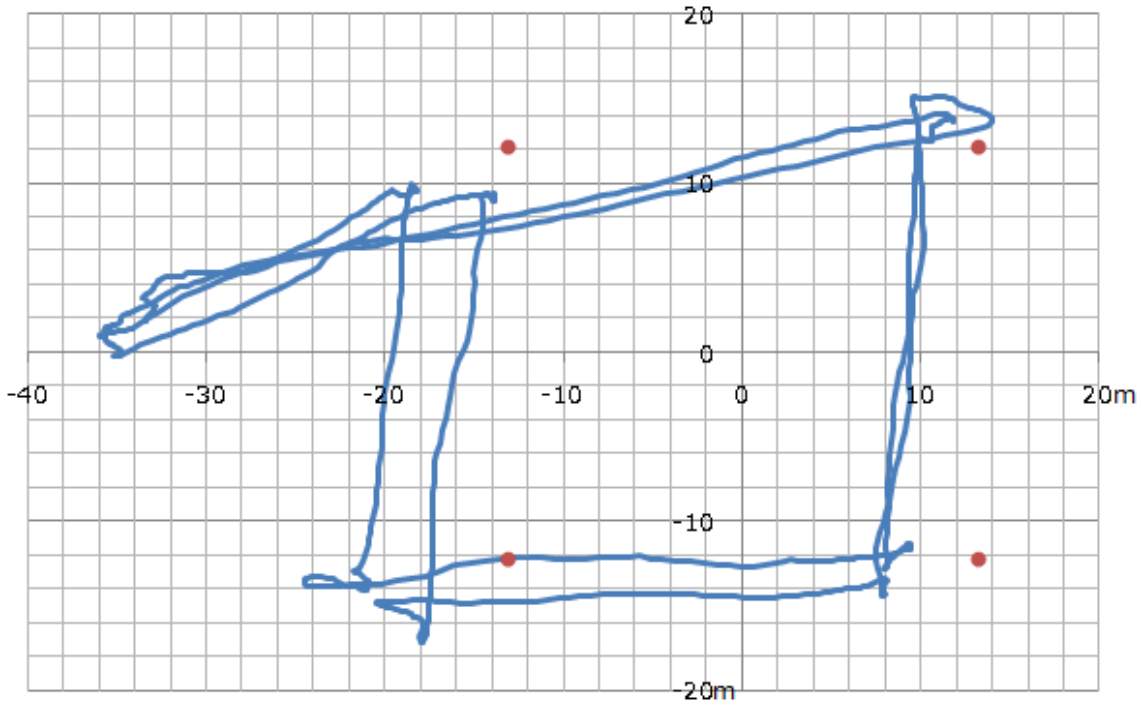

図 3-8 waypoint4 点及び Phantom2 の軌跡(プロット)

表 3-1 各 WayPoint の緯度経度及び再短接近距離

	緯度(°)	経度(°)	1 週目(m)	2 週目(m)
WP1	北緯 35.66576	東経 139.79361	5.73	2.93
WP2	北緯 35.66554	東経 139.79361	0.34	2.55
WP3	北緯 35.66554	東経 139.79390	3.72	5.18
WP4	北緯 35.66576	東経 139.79390	1.96	0.99

3-4 自律飛行プログラム

3-4-1 プログラム概要

本研究で作成したプログラムは、GPS を用いて算出された UAV の位置情報を参考にして、waypoint の緯度経度との距離差方位差を求めて、機体を回頭及び前進させる。UAV が waypoint に設定値より接近した場合、次の waypoint に向かって進む。

UAV 最終目標に到達した場合、再び最初の waypoint を目指し飛行するプログラムとした。

今回作成したプログラムで制御するのは、回頭、前後進、高度である。

制御装置「AP-CUB-DIY LITE」では、前述の制御以外にも左右移動させる事も可能である。

それぞれの制御値は 1875 を中心とした値で表され、1250 から 2500 の間の値を取り、値に応じた移動を行う。それぞれの制御値と数値は表 3-2 のような対応である。

表 3-2 各出力と制御値の対応

	1250	1875	2500
エレベーター	後進	中立	前進
エルロン	左移動	中立	右移動
スロットル	下降	中立	上昇
ラダー	左回頭	中立	右回頭

3-4-2　プログラムのアルゴリズム

ラジコン飛行モードの時に現在高度値を保持する。自動操縦モードに切り替えた場合、その直前の現在高度値を保持する。

プログラムが実行されると最初に、UAV の高度と waypoint の高度差、緯度方向差、経度方向差それぞれ計算する。この時計算された緯度方向差と経度方向差は度数で計算されているので、計算結果に緯度経度 1 度あたりの距離をかけて m 単位に直す。

m単位で表された緯度距離、経度距離を用いて水平方向直線距離を算出する。

次に、緯度経度距離を元に UAV から目標 waypoint までの方位を計算する。ここで、出てくる値はラジアン値なので、度数に変換する。目標方位と現針路の差分を計算。

制御値として入力するために角度は 0 度~360 度ではなく、-180 度~180 度とした。なので、計算した差分値が正負 180 度を超えたら、計算値に 360 を加減するものとした。

高度差、水平距離、針路差をもとに、ゲインをかけた値を定数値 1875 に加えた後に出力する。

各制御変数出力後に、水平距離が設定値より小さければ、目標 waypoint を次の waypoint へと変更する。ただし、目標 waypoint が最終 waypoint であった場合、変更後の waypoint を最初の waypoint とする。

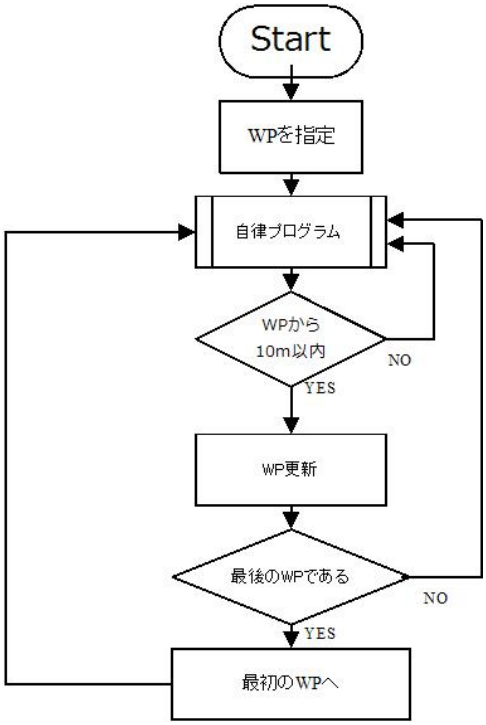

図 3-9 制御フローチャート

3-5　自律飛行実験

3-5-1　地上試験

Phantom2 による Waypoint 飛行実験のデータを参考に、製作した UAV で Waypoint 飛行を行うにあたり、事前に地上での動作確認を行った。

UAV は空中を移動するため、誤動作が起きると即時に墜落や暴走による激突等の事故につながる。そのため、新たな自動飛行プログラムを作成したら、機体の羽を外して人力で UAV 機体を移動させる形で試験を行う。

図 3-10 は地上にて、羽を外した機体を台車で移動させながら試験を行っている時の様子である。水平方向の動作検証を行う時は、上述のように台車を用いて動作試験を行い、鉛直方向の検証を行う時は機体を抱えながら階段等を昇降し試験する。

試験時はモーター音の変化を耳で確認するだけでなく、管制ソフト「Flight Viewer」を用いて各種出力値に異常が無いか確かめる。ここで異常が確認された場合、プログラムを書き直し、再度同様に試験を行う。異常が確認されなければ羽を機体に取り付けて実試験に移る。

図 3-10 地上実験様子

3-5-2　高度制御実験

地上試験で異常が確認されなくても、機体が思うように動作をしない事が起こる。その際、各種出力値を同時に変更していたら、どこに問題があるか確認するのに時間を要する。

そこで、最初に自律飛行時に高度のみを制御するプログラムを作成した。

プログラムの内容は、プロポで制御モード制御装置のモードを自律飛行モードに切り替える直前の高度から 5m だけ上昇させるというものだ。

実験では、手動制御にて目視しやすい高さまで上昇させ、機体の安定が確認とれた後に、制御モードを自動に切り替える。

そして、目標地点に到達したら、手動制御で高度を元に戻し、再度自律制御にて上昇させる実験を行った。
結果は図 3-11 のようになり、丸で囲んだ部分のスロットル出力と気圧高度を見ると、制御が行われているのが確認できる。

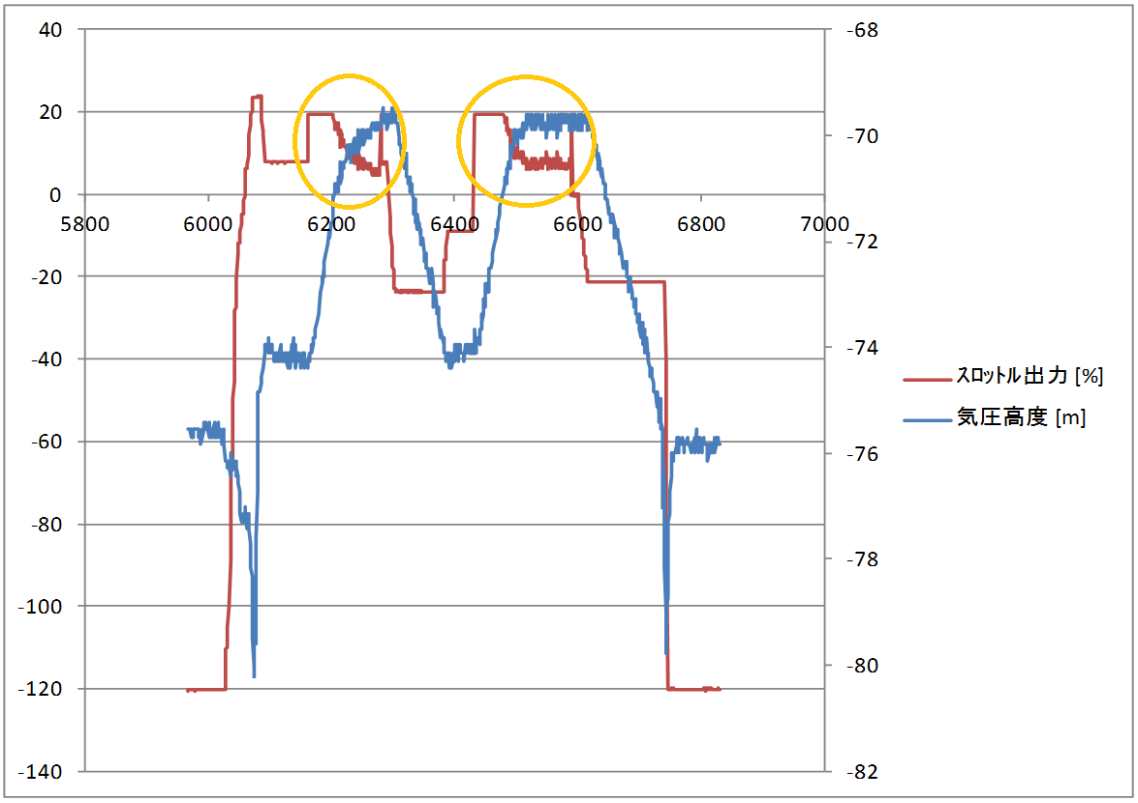

図 3-11 高度制御実験グラフ

3-5-3 WP 飛行実験(1 回目)

実験日時 2015 年 1 月 13 日　16 時頃
前項の高度制御試験により、プログラム通りに機体が動く事が確認できたので、水平方向の自動飛行実験を行う事とした。
高度制御試験時のプログラムをベースに高度を保持、自動で回頭、前進するものを作成したが、安全を考えて、エレベーター(前進)入力は手動で行う物に変更した。
実験時の軌跡は図 3-12 の通りである。軌跡を見る限り waypoint にかなり接近しているように見えるが、実験時に Waypoint と機体の距離が 10m 以下になった時点で目標 waypoint を変更するプログラムとしていたので、次の waypoint へ向かう途中に、前 waypoint に近づいただけである。また、waypoint へ向けて移動する際に機体が大きく外回りしているのがわかる。実験中に「Fright viewer」及び目視で機体の様子を確認すると機体方向と waypoint の方向に常に差があることが確認された。実験時の機体方位(磁方位)と目標方位を確認すると、平均して約 12 度の差があることが確認された。(図 3-14)

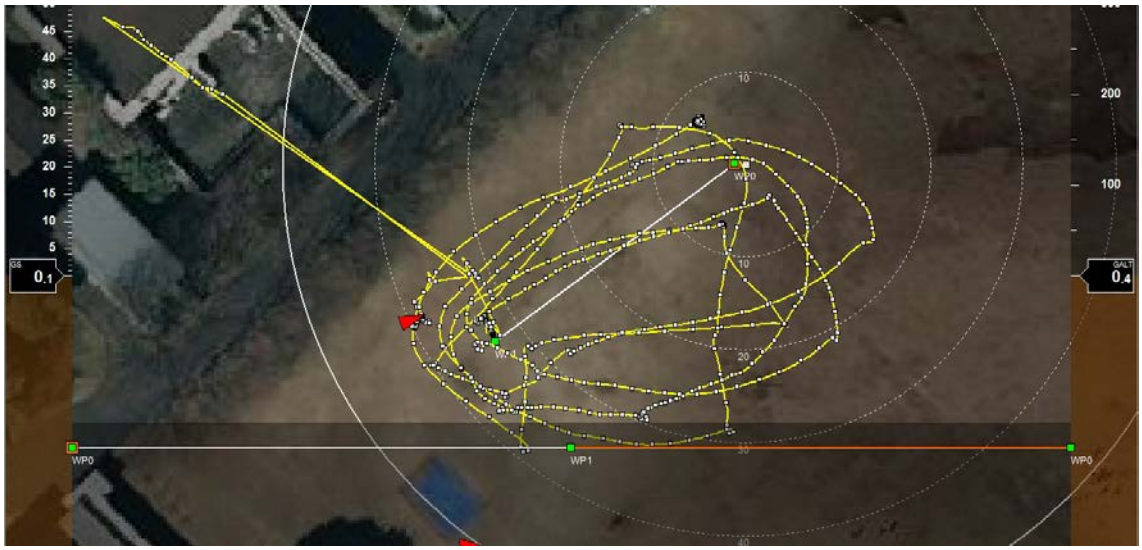

図 3-12 waypoint 飛行(1 回目)航跡 FlightViewer より

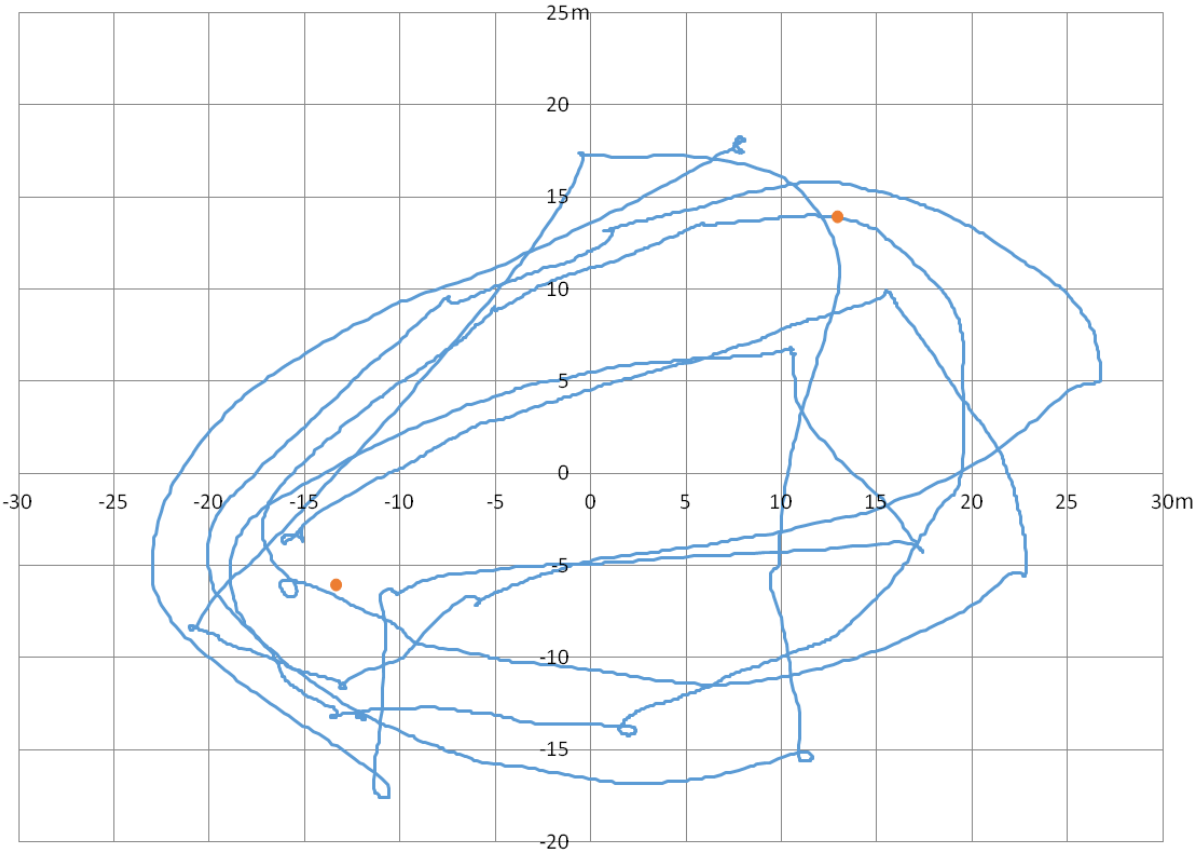

図 3-13waypoint 飛行(1 回目) プロット

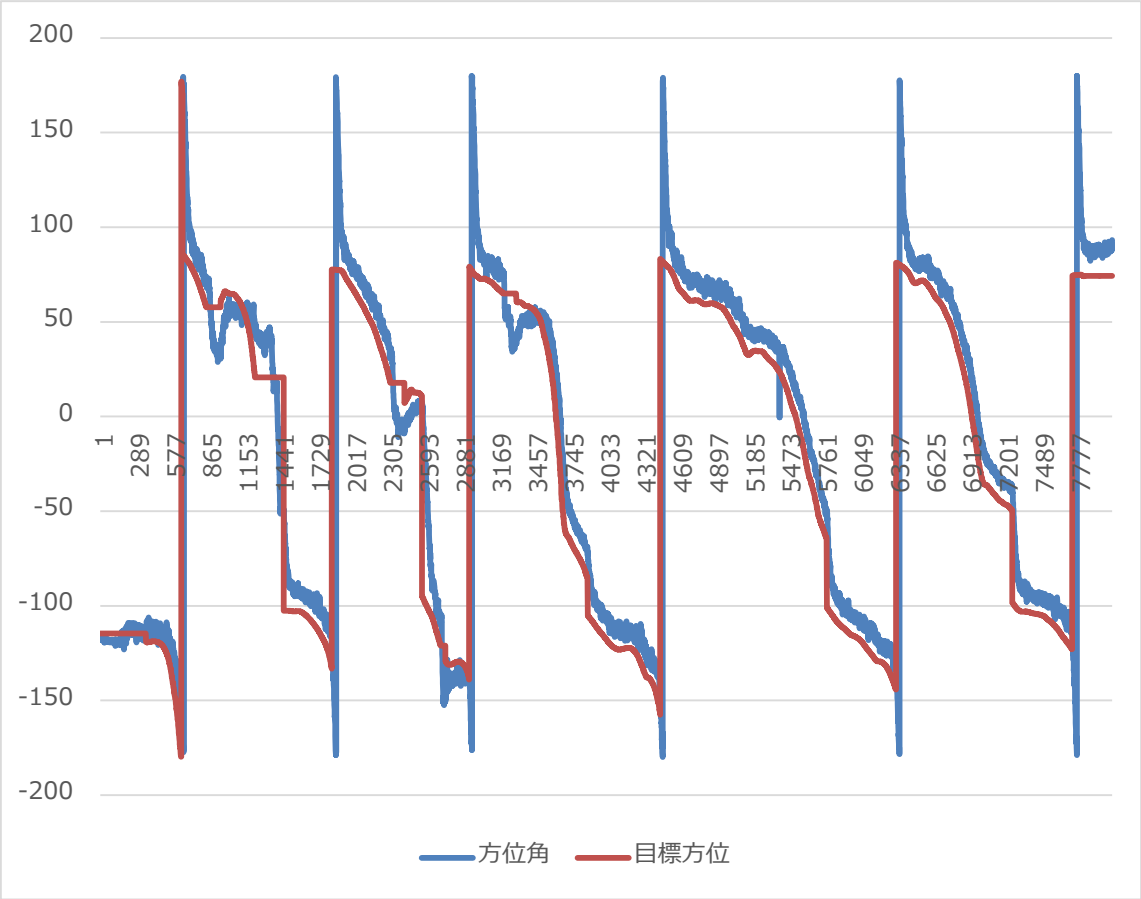

図 3-14 waypoint 飛行(1 回目)方位グラフ

表 3-3 各 waypoint の緯度経度

	緯度(度)	経度(度)
WP1	35.66594	139.79352
WP2	35.66576	139.79323

表 3-4 各 waypoint 再接近距離

	1 週目(m)	2 週目(m)	3 週目(m)	4 週目(m)	5 週目(m)
WP1	7.48	0.67	1.81	0.14	4.78
WP2	2.41	2.73	4.02	4.86	0.56

3-5-4 WP 飛行実験(2 回目)

実験日時 2015 年 1 月 19 日　16 時頃

1 回目の自律航行実験を終えて、目標方位と機体の方位角の間に差がある事が判明した。

機体の方位角は、制御装置で求めている。1 回目の自律飛行実験が終わるまで、機体と制御装置は固定されておらず、実験毎に設置位置が違った。

制御装置の位置が実験毎に違うということは、方位角の値に実験毎に異なる誤差が含まれるとい

う事だ。
そこで、制御装置を設置しているアクリル板に加工を施し、制御装置を図 3-15 のように固定し、実験毎の誤差を取り除いた。
加えて、制御装置が持つ自差を考慮してプログラムを設計していなかったので、自差を求めることとした。
本学第四実験棟屋上にて図 3-16 のように GNSS 受信機を 2 箇所に設置し、2 点間の方位の真値を求める。2 点間に線を引き、線と制御装置の方向を重ねて機体を設置し方位を測定。
真方位と制御装置の方位の平均値はそれぞれ 315.09 度、335.31 度（図 3-17、図 3-18）となり両者の間に 20 度近い差があるのが確認された。東京では真方位と磁方位の間に約 7 度の差があるので、それを考慮しても 13 度近い差があること判明した。これを修正したプログラムで、再度自律飛行実験を行った。
実験時の軌跡(図 3-19、図 3-20)及び、方位は図 3-17、図 3-18 の通りである。
航跡を見る限り、一度目の実験から改善が見られなかった。
そこで、再度 UAV を屋上に持って行き再度磁方位を計測した。
計測結果は図 3-21 に示す。平均磁方位は 327.71 度となった。この結果は前回計測した値とおよそ 7.6 度異なる。UAV は手で置いたので、1~2 度の誤差は考えられるが、約 7.6 度の誤差は起動時毎の自差の影響が大きいと判断する。

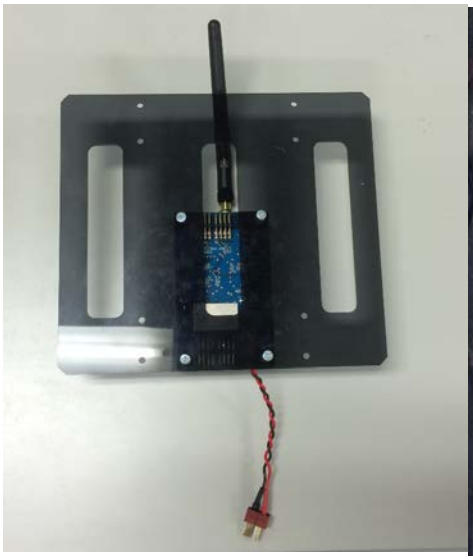

図 3-15 制御装置固定後

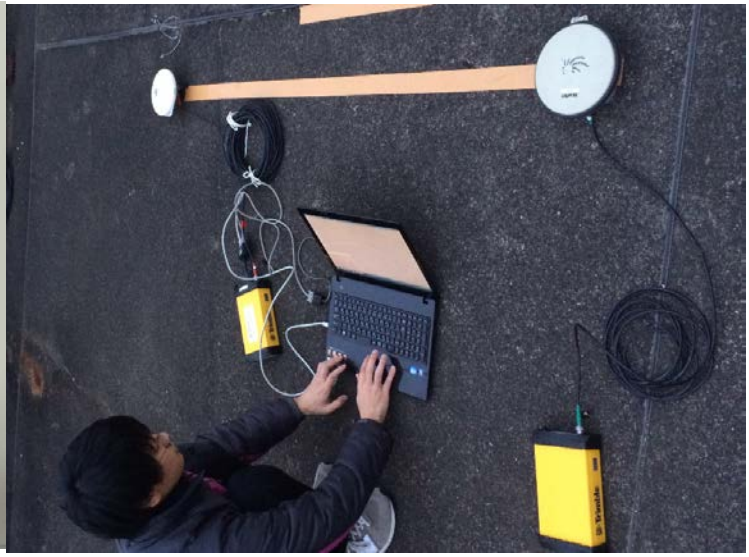

図 3-16 真方位の計測

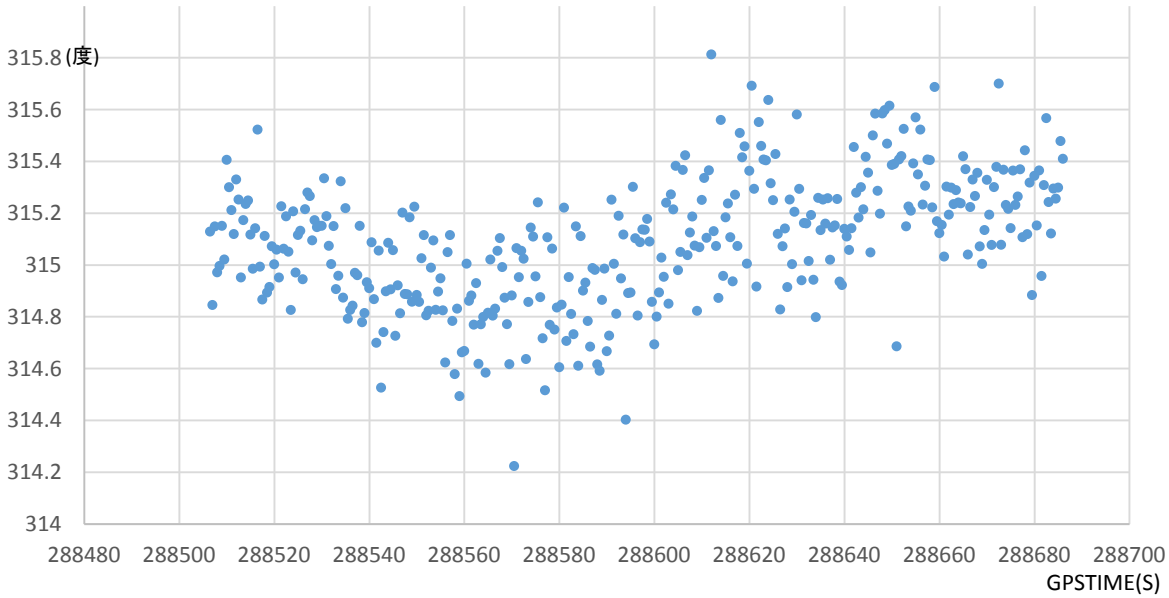

図 3-17 計測された真方位

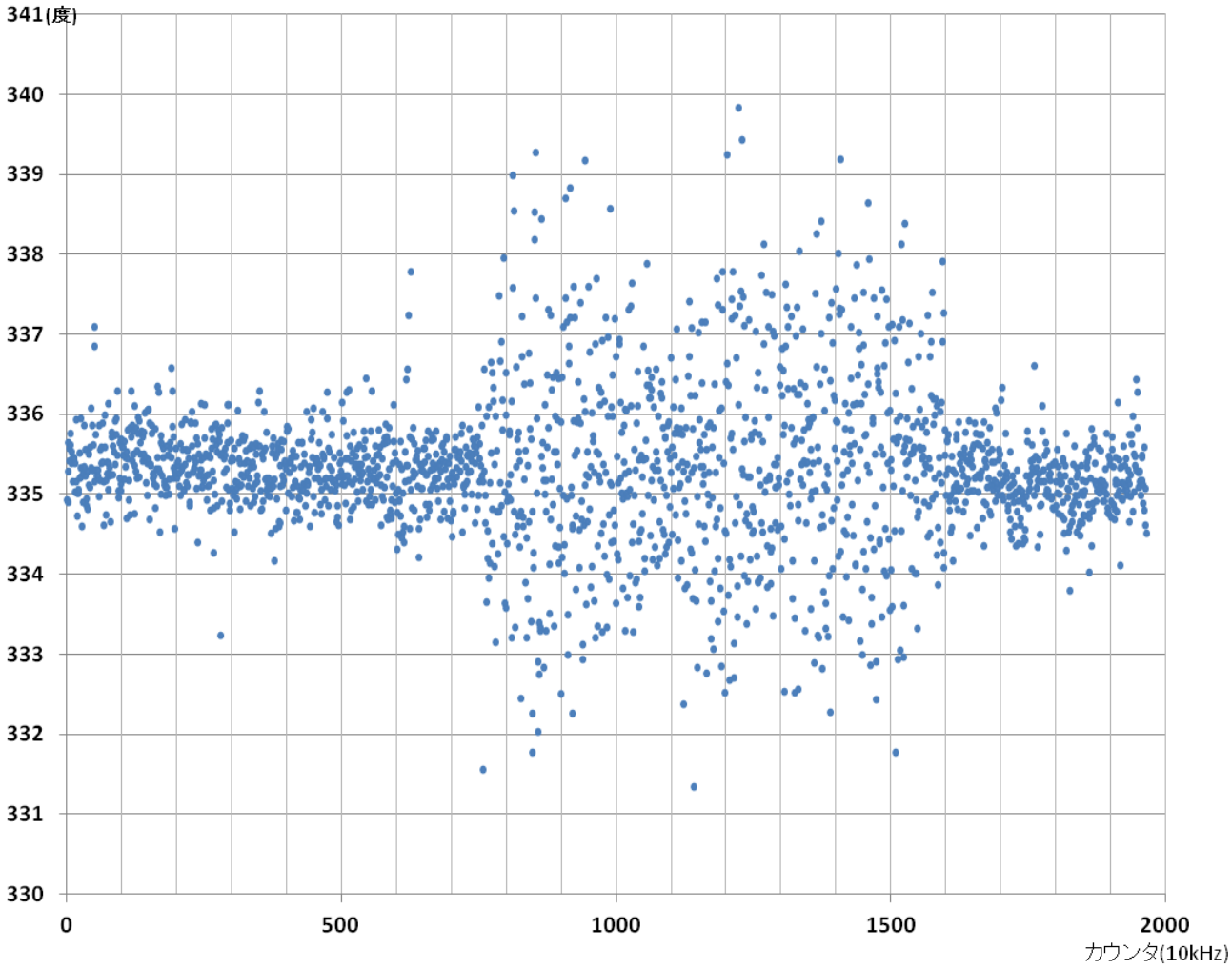

図 3-18 計測された磁方位

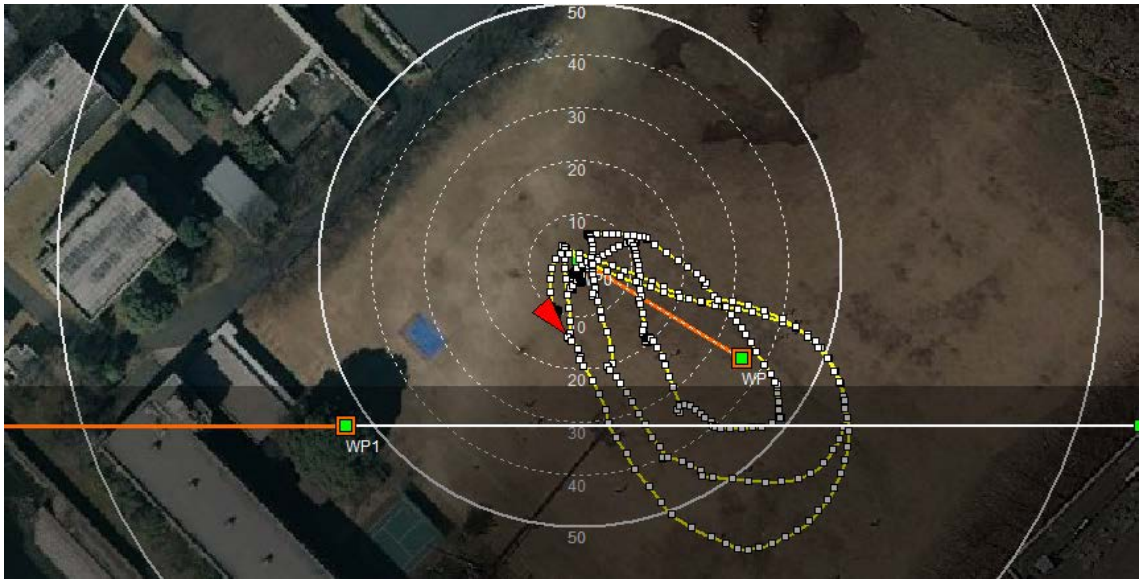

図 3-19 WayPoint 飛行(2 回目)航跡 FlightViewer より

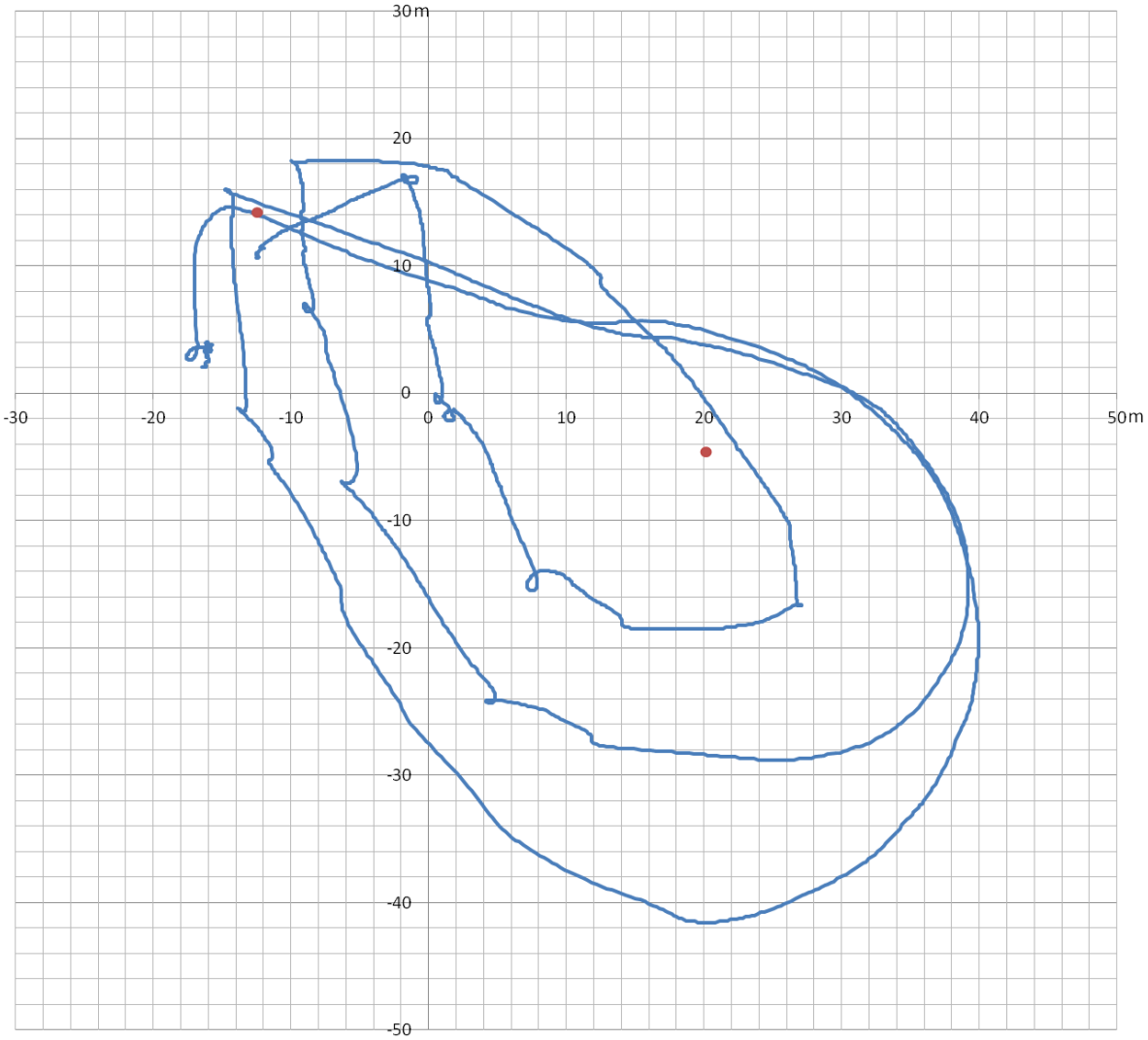

図 3-20 waypoint 飛行(2 回目)軌跡(プロット)

表 3-5 各 waypoint の緯度経度

	緯度(度)	経度(度)
WP1	35.66575	139.79348
WP2	35.66558	139.79384

表 3-6 各 waypoint との再接近距離

	1 回目(m)	2 回目(m)	3 回目(m)
WP1	3.17	0.62	0.22
WP2	2.11	8.12	9.02

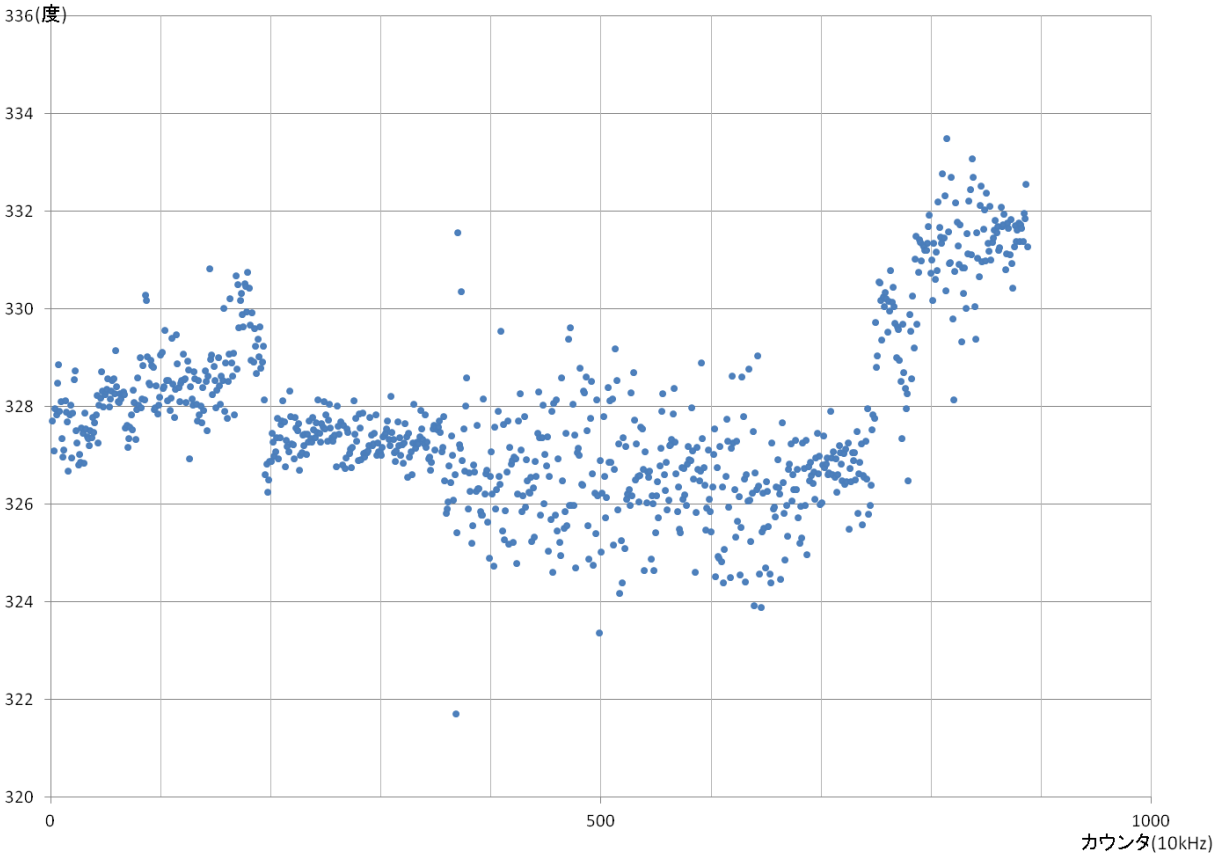

図 3-21 再度屋上で計測した真方位

4　空撮実験

4-1　学内グラウンドでの空撮

本学所有の実験船汐路丸で空撮を行う前に、本学グラウンドにて空撮を行った。目的は全長49.93m 幅 10m の汐路丸がカメラにどのように映るかの検証だ。

本学グラウンドでメジャーを用いて 10m を計測し、両端にパイロンを設置する。そのパイロンを写すように空撮する形で行った。

汐路丸への持ち込みを想定し、空撮は Phantom2 で行った。

撮影後、パイロンの間隔を参考して、汐路丸相当の大きさの長方形(50m×10m)を作成し、実際に汐路丸で空撮した場合、どの程度の高さまで飛ばす必要があるか検証した。

本学グラウンドにて撮影した画像が高さ別に以下の図 4-1、図 4-2 に示す。順に高度 45m、70m である。

汐路丸を映すだけで良いなら高度 45m あれば足りるが、汐路丸の周囲もある程度映るように撮影するのであれば最低でも 70m 程度は必要であることが確認できた。

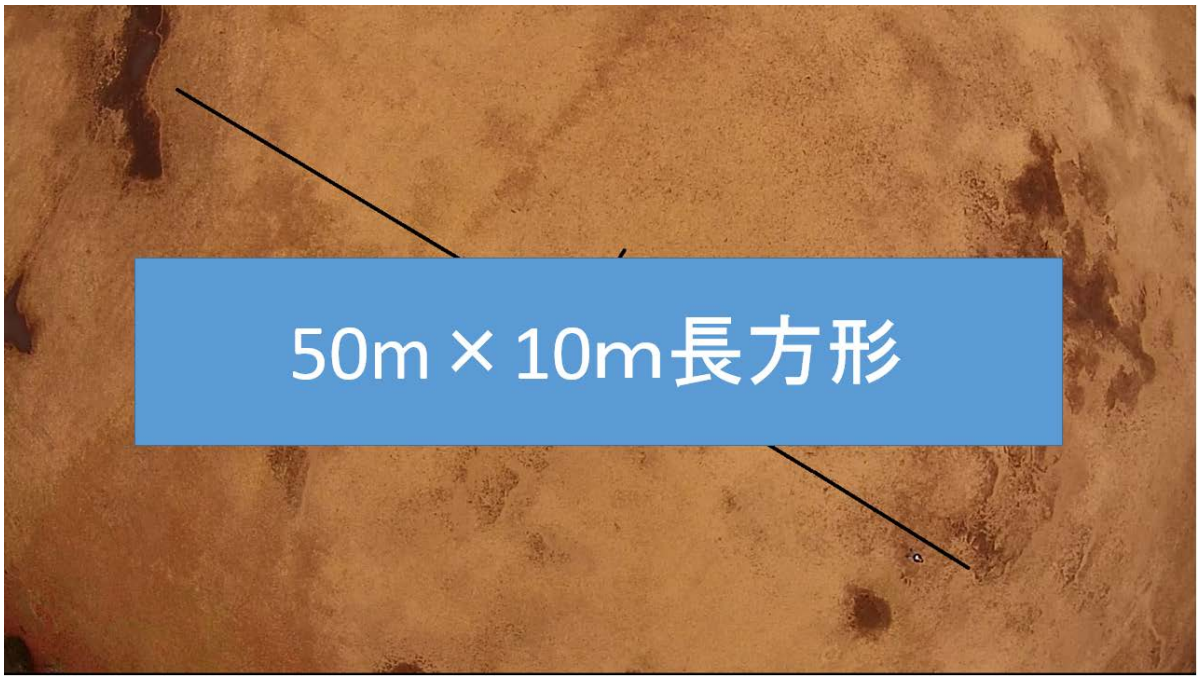

図 4-1 学内空撮(45m)

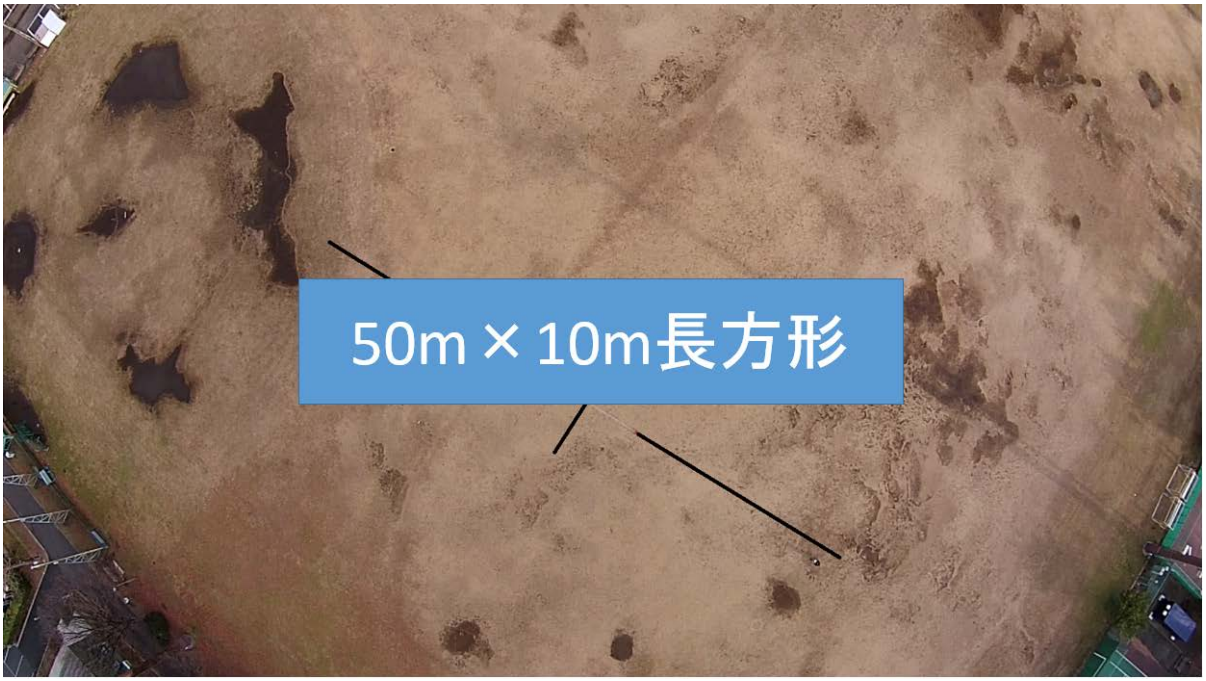

図 4-2 学内空撮(70m)

4-2　　汐路丸での空撮

本学グラウンドでの事前空撮で、汐路丸がどのように映るかはある程度確認できた。事前空撮によるイメージと差異は無いか、あるいは差異があってもどの程度であるか調べるため、実際に汐路丸に乗船し空撮を行った。

実験日時　1 月 20 日 15 時頃、1 月 21 日 8 時頃
撮影場所　東京湾館山沖
使用機体　Phantom2
風速 5m/s

高さ別の空撮画像は図 4-3、図 4-4、図 4-5 に示す。撮影した画像に写った汐路丸の大きさは、事前空撮の結果から想定されたものと大きな差はなかった。
今回空撮を行ったカメラでは、短辺に写る範囲が高度の 0.8~1.0 倍程度、長編に写る範囲が高度の 1.3~1.5 倍程度である。

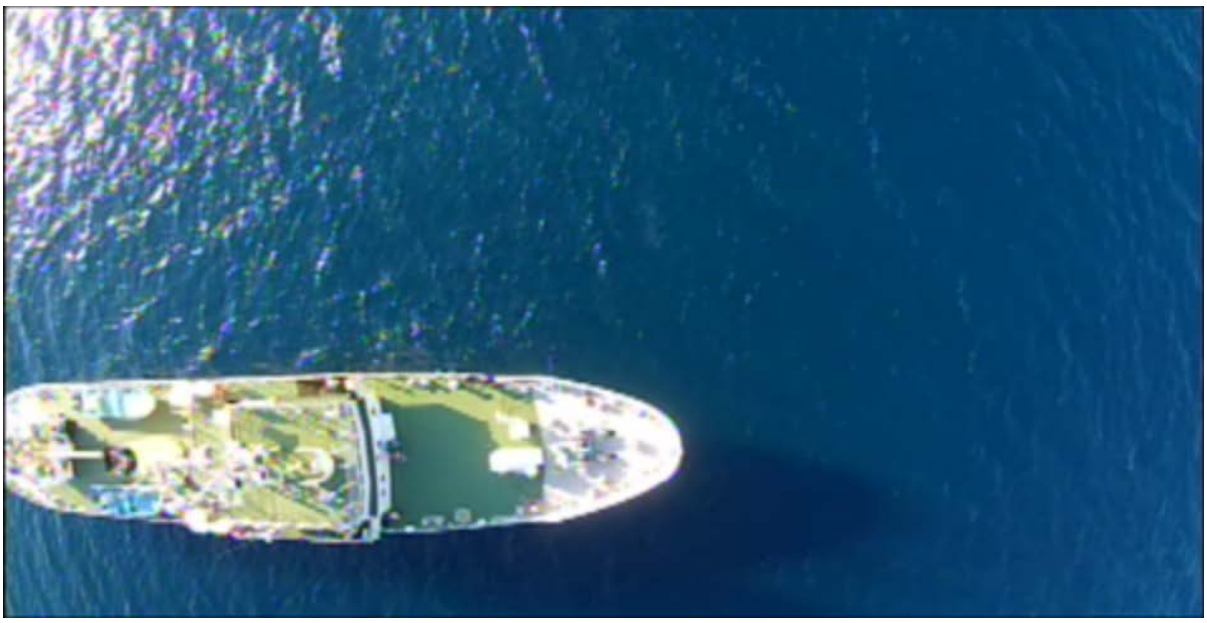

図 4-3 汐路丸空撮(45m)

図 4-4 汐路丸空撮(70m)

図 4-5 汐路丸空撮(100m)

5　結論

5-1　まとめ

UAV を海上で利用できたら便利だろうと思い UAV の試作、運用を行ったが難しさを知らされる結果となった。市販機で簡単に WayPoint 飛行が行えるので、自分でも同様のものを直ぐに作れると思ったが非常に難しかった。

WayPoint へ向けての動作も、単純に直進させれば大丈夫であると思っていたが、外乱の影響などにより機体が流される事を考慮できていなかった。

それでも、ある程度形になり少し改良を加えれば実用化出来る目処は立った。

海上での撮影も行い、実際の船舶がどのように写るかを確認できた事は非常に大きい。しかし、機材の性能や法律等の問題で実用化は厳しいだろうと考えられる面もある。

5-2　今後の課題

今後の課題としては、制御プログラムの改良をはじめ、実海域で自律航行を行い、更にデータ採取する必要がある。

また、実務での利用ではカメラの性能がどの程度要求されるのか、どの範囲まで映ればいいのか調査、研究していく必要がある。

巻末付録

3-5-4 WP 飛行実験(2 回目)<20 ページ>にて使用した自動飛行プログラム(user.c)を記載する。

```
#include "global.h"
#include "sci.h"
#include "interface.h"
#include "flightviewer.h"

#include <math.h>


//--------------------------------------------------------------------------------
// ｾﾝｻｰ値
//--------------------------------------------------------------------------------
// 変数名          変数型          内容              単位     備考
//--------------------------------------------------------------------------------
// g_ushRcv[0]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH1    [-]      1250(-150%) ～ 2500(+150%)
// g_ushRcv[1]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH2    [-]      1250(-150%) ～ 2500(+150%)
// g_ushRcv[2]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH3    [-]      1250(-150%) ～ 2500(+150%)
// g_ushRcv[3]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH4    [-]      1250(-150%) ～ 2500(+150%)
// g_ushRcv[4]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH5    [-]      1250(-150%) ～ 2500(+150%)
// g_ushRcv[5]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH6    [-]      1250(-150%) ～ 2500(+150%)
// g_ushRcv[6]     unsigned short  ﾌﾟﾛﾎﾟ舵角 CH7    [-]      1250(-150%) ～ 2500(+150%)
// g_nCtrl         int             自動/手動         [-]      0=MANUAL, 1=AUTO
// g_dAttAx        double          加速度 X 軸       [m/s2]
// g_dAttAy        double          加速度 Y 軸       [m/s2]
// g_dAttAz        double          加速度 Z 軸       [m/s2]
// g_dAttP         double          角速度 X 軸       [deg/s]
// g_dAttQ         double          角速度 Y 軸       [deg/s]
// g_dAttR         double          角速度 Z 軸       [deg/s]
// g_dAttPhi       double          ﾊﾞﾝｸ角            [deg]
// g_dAttTht       double          ﾋﾟｯﾁ角            [deg]
// g_dAttPsi       double          方位角            [deg]
// g_dAltP         double          気圧高度          [m]
// g_dAirspeed     double          対気速度          [m/s]
// g_dGpsLat       double          GPS 緯度          [deg]
// g_dGpsLon       double          GPS 経度          [deg]
```

// g_dGpsAlt	double	GPS 高度	[m]	WGS84 楕円体面からの高度
// g_nUTChour	int	UTC 時	[hour]	
// g_nUTCmin	int	UTC 分	[min]	
// g_dUTCsec	double	UTC 秒	[sec]	
// g_nGpsStatus	int	GPS ｽﾃｰﾀｽ	[-]	0:有効　1:無効
// g_nGpsMode	int	GPS ﾓｰﾄﾞ	[-]	0:単独測位 1:DGPS 2:無効
// g_dGpsHDOP	double	HDOP	[m]	
// g_dGpsGs	int	GPS 対地速度	[m/s]	
// g_dGpsCrs	double	GPS ｺｰｽ[deg]		
// g_nGpsNsv	int	GPS 衛星数	[-]	

//----------

// 飛行管制ｿﾌﾄ

//----------

// 変数名	変数型	内容	単位	備考
// g_dWpLat[*]	double	飛行計画 緯度	[deg]	
// g_dWpLon[*]	double	飛行計画 経度	[deg]	
// g_dWpAlt[*]	double	飛行計画 高度	[m]	
// g_uchWpFlg[*]	unsigned char	飛行計画 ﾌﾗｸﾞ	[-]	
// g_nWpTo	int	目標 WP ｲﾝﾃﾞｯｸｽ[-]		
// g_nWpNum	int	設定 WP 数	[-]	

//----------

// 誘導出力

//----------

// 変数名	変数型	内容	単位	備考
// g_dPhiCmd	double	ﾊﾞﾝｸ角指令	[deg]	FlightViewer 表示用
// g_dThtCmd	double	ﾋﾟｯﾁ角指令	[deg]	FlightViewer 表示用
// g_dPsiCmd	double	方位角指令	[deg]	FlightViewer 表示用
// g_dVelCmd	double	速度指令	[deg]	FlightViewer 表示用
// g_dAltCmd	double	高度指令	[m]	FlightViewer 表示用

//----------

// 制御計算

//----------

```
void userfunc()
{
	//--------------------------------
	// ﾁｬﾝﾈﾙ番号	出力変数
	//--------------------------------
	// CH1		g_ushCmd[0]
	// CH2		g_ushCmd[1]
	// CH3		g_ushCmd[2]
	// CH4		g_ushCmd[3]
	// CH5		g_ushCmd[4]
	// CH6		g_ushCmd[5]

	//--------------------------------
	// ｻｰﾎﾞ出力	舵角出力値
	//--------------------------------
	// -150%		1250
	//	0%		1875
	// +150%		2500

	static unsigned short s_ushCh1;
	static unsigned short s_ushCh2;
	static unsigned short s_ushCh3;
	static unsigned short s_ushCh4;
	static unsigned short s_ushCh5;
	static unsigned short s_ushCh6;

	double dTmp;
	double
dLat,dLon,dAlt,dDis,m_dLat,m_dLon,g_dLat,dr11,dr12,abs_Deg,gpsdHdg,dDeg,dEle;
	static double s_dAlt = 0;

	double dAltCmd;


	if (g_nCtrl == MANUAL) {
		// ﾏﾆｭｱﾙ操縦

		// ﾏﾆｭｱﾙ操縦時のﾌﾟﾛﾎﾟ入力を保存
```

```
                s_ushCh1 = g_ushRcv[0];
                s_ushCh2 = g_ushRcv[1];
                s_ushCh3 = g_ushRcv[2];
                s_ushCh4 = g_ushRcv[3];
                s_ushCh5 = g_ushRcv[4];
                s_ushCh6 = g_ushRcv[5];

#if 0
                // 気圧高度を使用する場合
                s_dAlt = g_dAltP;
#else
                // GPS 高度を使用する場合
                s_dAlt = g_dGpsAlt;
#endif


                // 出力
                g_ushCmd[0] = g_ushRcv[0];
                g_ushCmd[1] = g_ushRcv[1];
                g_ushCmd[2] = g_ushRcv[2];
                g_ushCmd[3] = g_ushRcv[3];
                g_ushCmd[4] = g_ushRcv[4];
                g_ushCmd[5] = g_ushRcv[5];

        } else {

                // 自動操縦

                // 高度指令
#if 0
                // WP 高度を使用する場合
                dAltCmd = g_dWpAlt[g_nWpTo];
#else
                // 自動操縦 ON 時の高度を使用する場合
                dAltCmd = s_dAlt;
#endif
                // 飛行管制ｿﾌﾄ表示用
                g_dAltCmd = dAltCmd;
```

```
//---------------------------------------------------



// 位置誤差 deg
dLat = g_dWpLat[g_nWpTo] - g_dGpsLat;
dLon = g_dWpLon[g_nWpTo] - g_dGpsLon;

//---------------------------------------------------
#if 0
// 気圧高度を使用する場合
dTmp = dAltCmd - g_dAltP;
#else
// GPS 高度を使用する場合
dTmp = dAltCmd - g_dGpsAlt;
#endif
//---------------------------------------------------

// 位置誤差 deg -> m
m_dLat = dLat* 90541.70352;
m_dLon = dLon* 110952.8317;

// 目標ｳｪｲﾎﾟｲﾝﾄまでの距離
dDis = sqrt(m_dLat * m_dLat + m_dLon * m_dLon);//水平方向直線距離(m)



if (dLat == 0 && dLon == 0){
        // atan2 不定回避処理
        dr11 = 0;

} else {

        // 自機から見た目標ｳｪｲﾎﾟｲﾝﾄまでの方位（真方位） rad
        //dr11 = atan2(m_dLon, m_dLat);
        dr11 = atan2(dLon, dLat);

}
```

```
		// 自機から見た目標ｳｪｲﾎﾟｲﾝﾄまでの方位 rad -> deg
		dr12 = dr11 * 180.0 / PI;

		// 飛行管制ｿﾌﾄ表示用
		//-------------------------------------------------
		g_dPsiCmd = dr12 + 7.0;
		//g_dPsiCmd = dr12;
		//-------------------------------------------------
#if 1
		// 磁方位計を使用する場合
		gpsdHdg = g_dAttPsi - 7.0;		// 真方位座標系に変換

		gpsdHdg = gpsdHdg - 22.28; //基線方位との誤差が 22.28 度だったので 22.28
を入れて修正

#else
		// GPS ｺｰｽを使用する場合
		if (180 < g_dGpsCrs) {
			gpsdHdg = g_dGpsCrs - 360.0;
		} else {
			gpsdHdg = g_dGpsCrs;
		}
#endif

		// 方位誤差 deg
		dDeg = (dr12 - gpsdHdg);

		if (dDeg < -180.0){
			dDeg = dDeg + 360.0;
		} else if(dDeg > 180.0){
			dDeg = dDeg - 360.0;
		}

		// ラダー出力
		dDeg = -(dDeg * 2.0) + 1875;

		// ラダーリミット処理
```

```
if (dDeg > 1955) {
        dDeg = 1955;
} else if (dDeg < 1795) {
        dDeg = 1795;
}

// 出力(水平方向)
dEle = (dDis * 40.0) + 1875;

// ﾘﾐｯﾀ
if (dEle > 1955) {
        dEle = 1955;
} else if (dEle < 1795) {
        dEle = 1795;
}


// 出力(高度方向)
dTmp = (dTmp * 40.0) + 1875;
// ﾘﾐｯﾀ
if (dTmp > 1955) {
        dTmp = 1955;
} else if (dTmp < 1795) {
        dTmp = 1795;
}

// 自動操縦に入る直前のﾌﾟﾛﾎﾟ入力を出力

// CH1          エルロン
s_ushCh1 = g_ushRcv[0];

//-------------------------------------------------
// CH2          エレベータ
//s_ushCh2 = (unsigned short)dDis;今回、前進は手動でかける
s_ushCh2 = g_ushRcv[1];
//-------------------------------------------------
```

```
			// CH3			高度制御
			s_ushCh3 = (unsigned short)dTmp;

			// CH4			方位制御
			s_ushCh4 = (unsigned short)dDeg;

			// CH5
			s_ushCh5 = g_ushRcv[4];

			// CH6
			s_ushCh6 = g_ushRcv[5];



			if (dDis < 10.0) {
				g_nWpTo = g_nWpTo + 1;
			}

			if (g_nWpNum == g_nWpTo){
				g_nWpTo = 0;
			}

			// 出力
			g_ushCmd[0] = s_ushCh1;
			g_ushCmd[1] = s_ushCh2;
			g_ushCmd[2] = s_ushCh3;
			g_ushCmd[3] = s_ushCh4;
			g_ushCmd[4] = s_ushCh5;
			g_ushCmd[5] = s_ushCh6;
		}
}
//↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑


//------------------------------------------------
//
// ﾀﾞｳﾝﾘﾝｸ
//
```

```
//-----------------------------------------------
void downlink()
{
	int i;
	unsigned char uchMsg[100];
	unsigned char uchID, uchRS;
	unsigned char uchCS;
	unsigned short ushTmp;
	unsigned long ulTmp;
	double dDeg;

	// ﾍｯﾀﾞｰ
	uchMsg[0] = 0xff;
	uchMsg[1] = 0xff;

	// ｶｳﾝﾀ
	ushTmp = g_ushCnt++;
	uchMsg[2] = (unsigned char)(ushTmp >> 8 & 0x00ff);
	uchMsg[3] = (unsigned char)(ushTmp & 0x00ff);

	// AX
	ushTmp = (unsigned short)((g_dAttAx + 50.0) / 100.0 * 65535.0);
	uchMsg[4] = (unsigned char)((ushTmp >> 8) & 0x00ff);
	uchMsg[5] = (unsigned char)(ushTmp & 0x00ff);

	// AY
	ushTmp = (unsigned short)((g_dAttAy + 50.0) / 100.0 * 65535.0);
	uchMsg[6] = (unsigned char)((ushTmp >> 8) & 0x00ff);
	uchMsg[7] = (unsigned char)(ushTmp & 0x00ff);

	// AZ
	ushTmp = (unsigned short)((g_dAttAz + 50.0) / 100.0 * 65535.0);
	uchMsg[8] = (unsigned char)((ushTmp >> 8) & 0x00ff);
	uchMsg[9] = (unsigned char)(ushTmp & 0x00ff);

	// P
	ushTmp = (unsigned short)((g_dAttP + 300.0) / 600.0 * 65535.0);
	uchMsg[10] = (unsigned char)((ushTmp >> 8) & 0x00ff);
```

```
uchMsg[11] = (unsigned char)(ushTmp & 0x00ff);

// Q
ushTmp = (unsigned short)((g_dAttQ + 300.0) / 600.0 * 65535.0);
uchMsg[12] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[13] = (unsigned char)(ushTmp & 0x00ff);

// R
ushTmp = (unsigned short)((g_dAttR + 300.0) / 600.0 * 65535.0);
uchMsg[14] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[15] = (unsigned char)(ushTmp & 0x00ff);

// PHI
ushTmp = (unsigned short)((g_dAttPhi + 180.0) / 360.0 * 65535.0);
uchMsg[16] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[17] = (unsigned char)(ushTmp & 0x00ff);

// THT
ushTmp = (unsigned short)((g_dAttTht + 180.0) / 360.0 * 65535.0);
uchMsg[18] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[19] = (unsigned char)(ushTmp & 0x00ff);

// PSI
ushTmp = (unsigned short)((g_dAttPsi + 180.0) / 360.0 * 65535.0);
uchMsg[20] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[21] = (unsigned char)(ushTmp & 0x00ff);

// PALT
ushTmp = (unsigned short)((g_dAltP + 500.0) / 9500.0 * 65535.0);
uchMsg[22] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[23] = (unsigned char)(ushTmp & 0x00ff);

// IAS
ushTmp = (unsigned short)(g_dAirspeed / 200.0 * 65535.0);
uchMsg[24] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[25] = (unsigned char)(ushTmp & 0x00ff);

// GPS LAT
```

```
dDeg = g_dGpsLat;
uchMsg[26] = (unsigned char)dDeg;
ulTmp = (unsigned long)((dDeg - (int)dDeg) * 16777215.0);
uchMsg[27] = (unsigned char)((ulTmp >> 16) & 0x000000ff);
uchMsg[28] = (unsigned char)((ulTmp >> 8) & 0x000000ff);
uchMsg[29] = (unsigned char)(ulTmp & 0x000000ff);

// GPS LON
dDeg = g_dGpsLon;
uchMsg[30] = (unsigned char)dDeg;
ulTmp = (unsigned long)((dDeg - (int)dDeg) * 16777215.0);
uchMsg[31] = (unsigned char)((ulTmp >> 16) & 0x000000ff);
uchMsg[32] = (unsigned char)((ulTmp >> 8) & 0x000000ff);
uchMsg[33] = (unsigned char)(ulTmp & 0x000000ff);

// GPS ALT
ushTmp = (unsigned short)((g_dGpsAlt + 500.0) / 9500.0 * 65535.0);
uchMsg[34] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[35] = (unsigned char)(ushTmp & 0x00ff);

// GPS GS
ushTmp = (unsigned short)(g_dGpsGs * 10.0);
uchMsg[36] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[37] = (unsigned char)(ushTmp & 0x00ff);

// GPS Crs
ushTmp = (unsigned short)(g_dGpsCrs * 10.0);
uchMsg[38] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[39] = (unsigned char)(ushTmp & 0x00ff);

// HDOP
ushTmp = (unsigned short)(g_dGpsHDOP * 10.0);
uchMsg[40] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[41] = (unsigned char)(ushTmp & 0x00ff);

// UTC
uchMsg[42] = (unsigned char)g_nUTChour;
uchMsg[43] = (unsigned char)g_nUTCmin;
```

```
ushTmp = g_ushUTCsec;
uchMsg[44] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[45] = (unsigned char)(ushTmp & 0x00ff);

// Status
uchMsg[46] = (unsigned char)g_nGpsStatus;

// Mode
uchMsg[47] = (unsigned char)g_nGpsMode;

// NSV
uchMsg[48] = (unsigned char)g_nGpsNsv;

// 制御計算(CH1)
ushTmp = g_ushCmd[0];
uchMsg[49] = (unsigned char)(ushTmp >> 8 & 0x00ff);
uchMsg[50] = (unsigned char)(ushTmp & 0x00ff);

// 制御計算(CH2)
ushTmp = g_ushCmd[1];
uchMsg[51] = (unsigned char)(ushTmp >> 8 & 0x00ff);
uchMsg[52] = (unsigned char)(ushTmp & 0x00ff);

// 制御計算(CH3)
ushTmp = g_ushCmd[2];
uchMsg[53] = (unsigned char)(ushTmp >> 8 & 0x00ff);
uchMsg[54] = (unsigned char)(ushTmp & 0x00ff);

// 制御計算(CH4)
ushTmp = g_ushCmd[3];
uchMsg[55] = (unsigned char)(ushTmp >> 8 & 0x00ff);
uchMsg[56] = (unsigned char)(ushTmp & 0x00ff);

// 制御計算(CH5)
ushTmp = g_ushCmd[4];
uchMsg[57] = (unsigned char)(ushTmp >> 8 & 0x00ff);
uchMsg[58] = (unsigned char)(ushTmp & 0x00ff);
```

```
// 制御計算(CH6)
ushTmp = g_ushCmd[5];
uchMsg[59] = (unsigned char)(ushTmp >> 8 & 0x00ff);
uchMsg[60] = (unsigned char)(ushTmp & 0x00ff);

// 制御ﾌﾗｸﾞ
uchMsg[61] = (unsigned char)g_nCtrl;

// 目標ｳｪｲﾎﾟｲﾝﾄ
uchMsg[62] = (unsigned char)g_nWpTo;

// ｳｪｲﾎﾟｲﾝﾄ数
uchMsg[63] = (unsigned char)g_nWpNum;

// ﾊﾞﾝｸ角指令
ushTmp = (unsigned short)((g_dPhiCmd + 180.0) / 360.0 * 65535.0);
uchMsg[64] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[65] = (unsigned char)(ushTmp & 0x00ff);

// ﾋﾟｯﾁ角指令
ushTmp = (unsigned short)((g_dThtCmd + 180.0) / 360.0 * 65535.0);
uchMsg[66] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[67] = (unsigned char)(ushTmp & 0x00ff);

// 方位角指令
ushTmp = (unsigned short)((g_dPsiCmd + 180.0) / 360.0 * 65535.0);
uchMsg[68] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[69] = (unsigned char)(ushTmp & 0x00ff);

// 速度指令
ushTmp = (unsigned short)(g_dVelCmd / 200.0 * 65535.0);
uchMsg[70] = (unsigned char)((ushTmp >> 8) & 0x00ff);
uchMsg[71] = (unsigned char)(ushTmp & 0x00ff);

// 高度指令
ushTmp = (unsigned short)((g_dAltCmd + 500.0) / 9500.0 * 65535.0);
uchMsg[72] = (unsigned char)((ushTmp >> 8) & 0x00ff);
```

```
        uchMsg[73] = (unsigned char)(ushTmp & 0x00ff);

        // 応答 1
        getFIFO(&uchID, &uchRS);
        uchMsg[74] = uchID;
        uchMsg[75] = uchRS;

        // 応答 2
        getFIFO(&uchID, &uchRS);
        uchMsg[76] = uchID;
        uchMsg[77] = uchRS;

        // 応答 3
        getFIFO(&uchID, &uchRS);
        uchMsg[78] = uchID;
        uchMsg[79] = uchRS;

        // ﾊﾟﾘﾃｨ
        uchCS = 0;
        for (i = 2; i <= 79; i++) {
                uchCS ^= uchMsg[i];
        }
        uchMsg[80] = uchCS;

        // ﾌｯﾀｰ
        uchMsg[81] = 0x0d;
        uchMsg[82] = 0x0a;

        // 送信
        sendXBee(uchMsg, 83);

        // SD ｶｰﾄﾞ記録
        for (i = 0; i < 83; i++) {
                sci2_writeChar(uchMsg[i]);
        }
}
```