公立はこだて未来大学 2020 年度 システム情報科学実習
グループ報告書

Future University-Hakodate 2020 Systems Information Science Practice
Group Report

プロジェクト名

めざせ宇宙開発 - 自律移動ロボット飛行プロジェクト

Project Name

Space Development - Autonomous movement robot Flight Project

グループ名

グループ B

Group Name

Group B

プロジェクト番号/Project No.

17-B

プロジェクトリーダ/Project Leader

比留川満洋　Mitsuhiro Hirukawa

グループリーダ/Group Leader

小林浩也　Hiroya Kobayashi

グループメンバ/Group Member

村上拓馬　Takuma Murakami
小林浩也　Hiroya Kobayashi
永井彬博　Akihiro Nagai
比留川満洋　Mitsuhiro Hirukawa
佐々木光流　Hikaru Sasaki

指導教員

大沢英一, 三上貞芳, 和田雅昭

Advisor

Ei-Ichi Osawa, Sadayoshi Mikami, Masaaki Wada

提出日

2021 年 1 月 14 日

Date of Submission

January 14, 2021

概要

近年、数学や英語などの特定の科目を勉強するだけではなく、目標達成のために取り組む学習形式としてプロジェクト学習が中学校や高等学校、大学へ導入され始めている。プロジェクト学習のテーマには社会的な問題から科学分野、自然の生態系、宇宙工学など多種多様なものがある。その中で、宇宙工学の取り組みとして CanSat があげられる。CanSat とは缶を意味する Can と人工衛星を意味する Satellite とを組み合わせた省略語であり、缶のような大きさと形をした超小型模擬人工衛星のことを指す。CanSat では実際の人工衛星開発でも直面するような多くの問題に対して解決案を模索していく必要がある。

本グループでは一般的な 350ml クラスサイズとオープンクラスサイズのうち、オープンクラスサイズの CanSat を設計、運用を検討してきた。オープンクラスサイズとは直径 146mm、高さ 240mm、重量 1050g 以下の大きさと重量を制限した CanSat の区分を指す。我々は 5 月から設計、開発などの活動予定を立てていた。しかし、新型コロナウイルス感染症（COVID-19）の影響により、登校制限が設けられ、学校での活動が行えなかった。そのため予定を大きく変更して主にオンラインで活動を行った。また、登校制限だけではなく活動開始の時期もやや遅れてしまったため、思い通りの円滑な活動は行えなかった。その結果として準備、発想の段階であったため基板や構造、電子部品の組み立てや具体的な設計は行えず、設定した課題についての成果は得られなかった。しかし、CanSat の基礎知識や過去の大会出場者が考察した CanSat の構造や問題・課題やその修正点について事前調査、それを踏まえて本グループでは製作する CanSat の構造やミッションについてブレインストーミングなどを行い、具体的な意見をまとめることができた。

夏期休暇後、10 月から登校制限が一部解除された。これまで通りオンラインでの活動が多い中、約一週間半に一度メンバーが集まり大学の工房を利用して具体的な部品の作製が行えるようになった。メンバーは通信、飛行物、電装、構造といった分野ごとに担当を決め、制作を進めていったが、あくまでその分野の責任者ということであったため分野にかかわらず、わからない部分はメンバー同士で助け合って作業を進めていった。本格的な作業開始から最終発表までの期間は 2 か月ほどしかなかったが、発表前に成果物を完成させることができた。完成後は走行実験や笹流ダムでの落下実験を行い調整を繰り返した。その実験結果から、今後の展望等についての考察をすることもできた。

Keyword　宇宙工学, CanSat, 超小型模擬人工衛星, COVID-19, オンライン

（※文責: 村上拓馬）

Abstract

In recent years, project learning has begun to be introduced not only in specific subjects such as mathematics and English, but also in junior high schools, high schools, and universities as a learning format to tackle achievement of goals. There are a wide variety of themes for project learning, from social issues to scientific fields, natural ecosystems, and space engineering. Among them, CanSat is an example of space engineering efforts. CanSat is an abbreviation that combines Can, which means a can, and Satellite, which means a satellite, and refers to a micro-simulated artificial satellite shaped and shaped like a can. CanSat needs to search for solutions to many problems that are encountered in actual satellite development.

This group has been designing and operating CanSat, which is an open class size of the general 350ml class size and open class size. Open class size refers to the CanSat category with a diameter of 146mm, a height of 240mm, and a weight of 1050g or less and a limited weight. We have been planning activities such as design and development since May. However, due to the influence of the new coronavirus infection(COVID-19), school restrictions were set and school activities were not possible. Therefore, the schedule was changed drastically and activities were mainly conducted online. In addition, not only the school attendance restrictions but also the start time of the activities was slightly delayed, so it was not possible to carry out smooth activities as expected. As a result, since it was in the stage of preparation and idea, it was not possible to assemble the base and structure, electronic parts and concrete design, and the results for the set issues could not be obtained. However, the basic knowledge of CanSat and the structure, problems and issues of CanSat considered by participants in the past competitions, and their correction points were preliminarily researched, and based on this, we conducted brainstorming on the structure and mission of CanSat to be produced , we were able to put together a concrete opinion.

After the summer vacation, some school attendance restrictions were lifted from October. While there are still many online activities, members gather once every one and a half weeks and can now make concrete parts using the university's workshop. The members decided to take charge of each field such as communication, flying objects, electrical equipment, and structure, and proceeded with the production, but since they were the person in charge of that field to the last, regardless of the field, the members helped each other. We were working on it. The period from the start of full-scale work to the final announcement was only about two months, but we were able to complete the deliverables before the announcement. After completion, we conducted running experiments and drop experiments at Sasanagare Dam and repeated adjustments. From the experimental results, we were able to consider future prospects.

Keyword Space Engineering, CanSat, Micro Satellite, COVID-19, Online

（※文責: 比留川満洋）

目次

第 1 章　はじめに

この章では、本プロジェクトの背景と目的、従来の例や問題点、課題について説明する。

（※文責: 比留川満洋）

1.1　背景

現在、宇宙開発は、科学的知識の向上だけでなく社会インフラへと貢献を重ねている。しかし、高コスト、複雑化、高信頼化への要求に伴い、安全性を保障し実験しなければならず、学生による宇宙開発は困難とされてきた。そこで、学生の宇宙工学における教育を目的として始められたのが、CanSat プロジェクトである [1]。

CanSat とは、空き缶（Can）サイズの衛星（Satellite）をモデルとし名付けられた超小型模擬人工衛星のことである。1998 年 11 月ハワイで開かれた日米大学宇宙システム会議 (USSS'98 : University Space Systems Symposium'98) でスタンフォード大学の Bob Twiggs 教授の提唱のもと始められた [1]。これを機に、アメリカや日本の大学を中心として、CanSat プロジェクトが盛んに行われている。

CanSat プロジェクトにはいくつかの競技大会があり、代表的な大会に ARLISS(A Rocket Launch for International Student Satellites) と呼ばれるものがある [2]。アメリカ合衆国ネバタ州のブラックロック砂漠でアマチュアロケット団体（AeroPAC）の協力のもと、CanSat の打ち上げ実験を毎年開催している。ARLISS の競技の 1 つであるカムバックコンペティション (Comeback Competition) では、まず CanSat を搭載したロケットを高度約 4,000m まで打ち上げ、CanSat を放出する。放出された CanSat は地上に着陸後、あらかじめ設定された目標地点を目指して自律制御移動を始め、最終的にどれだけ目標地点に近い距離で到着できるかを競う [2]。

本プロジェクトでは、このカムバックコンペティションの競技をもとに、CanSat の製作をする。

（※文責: 比留川満洋）

1.2　目的

本プロジェクトの目的は、CanSat の設計・構築・運用を通して、航空宇宙工学に関連した、電子工学、情報工学および通信工学技術といった人工衛星開発に必要な知識と技術を習得するとともに、人工衛星の打ち上げ、運用というプロジェクト管理について学習することである。

本プロジェクト全体の目標は「目標地点へ到達できる、姿勢制御および自律移動アルゴリズムを取り入れた機体の製作」である。人工衛星を運用するうえで、目標地点へ到達することは最も重要なミッションの 1 つである。そのため、本プロジェクトでは CanSat を無事に目標地点へ到達させることを掲げ、さらに各グループごとにミッションの追加を行っていくものとする。

今年度得た経験や技術をもとに、来年度以降 ARLISS などの CanSat 競技大会出場を目指している。

（※文責: 比留川満洋）

1.3 従来の例

CanSat には 350ml クラスとオープンクラスの 2 つのサイズ規定がある。350m クラスは重量 350g 以下、高さ 240mm 以下、直径 66mm 以下であり、オープンクラスは重量 1050g 以下、高さ 240mm 以下、直径 146mm 以下である。本グループではオープンクラスの規定に沿い、製作を行う。なお、オープンクラスは、ARLISS の大会において多くの大学が採用している。

ここで過去の実験例を ARLISS2012 慶応義塾大学の報告書 [3] より紹介する。ARLISS では各チームが規定のサイズに収めた CanSat をアマチュアロケットに搭載し、上空約 4,000m へ打ち上げる。打ち上げ後、CanSat はアマチュアロケットから放出され、CanSat の通信を ON にし、位置情報を送信ならびに自律制御を開始する。光センサを用いて、パラシュートを展開し、地上に軟着陸する。軟着陸後、気圧センサによる判定から、CanSat はパラシュートを切り離し、目標地点に向けて走行を始める。走行中は GPS センサを用いて現在地点と目標地点を確認し走行する。目標地点付近まで到達したら、カメラによる画像認識をもとにさらに正確な位置まで走行しゴールをする。

（※文責: 比留川満洋）

1.4 従来の問題点

従来の問題点には以下の要素がある。

空中での姿勢制御　上空での制御にパラシュートやパラフォイルを利用する実験が多くある。しかし、このパラシュートやパラフォイルが空中で展開する際に CanSat に絡まってしまうことがよくある。このため、高度約 4,000m からの自由落下により、着陸時における衝撃が大きくなることから、CanSat の破損が多く報告されている [4]。

制御ログ　CanSat の競技大会である ARLISS では CanSat の制御ログを取らなければ大会は失格となる。実際にログの履歴確認が取ずに、失格となるケースが多く報告されている [5]。また、ログ履歴の破損に備えた冗長性に関しても考慮する必要がある。

サイズ規定　重量、高さ、直径、これらを規定以内に収めるのが容易ではない。電気回路、筐体、飛行物の製作に必要最低限な材料に加えて、軽材料を選別し、設計する必要がある。また、重量が重すぎると空中における自由落下の危険性も高まるので要注意である。

このように、CanSat 製作には多くの問題点があり、事前に段階実験を通して検証・改善していく必要がある。

（※文責: 比留川満洋）

1.5 課題

従来の問題点を参考に、問題の分析と改善策の考案、そして CanSat の自律制御による目標地点への到達を課題とする。また、今年度は新型コロナウイルス感染症（COVID-19）の拡大防止の影響により、本学での活動が制限された。そのため、スケジュール管理は大きな課題となる。

（※文責: 比留川満洋）

第 2 章　プロジェクト学習の概要

この章ではまず、ミッションと問題の設定として第 1 章で説明した目的と課題から考えられるミッションと問題について説明する。次に、課題の設定として設定した問題に対する課題とアプローチを説明する。

（※文責: 小林浩也）

2.1　ミッションと問題の設定

本グループでは、壊れやすいものを壊れないように運ぶことをグループのミッションとした。その上で、正常に動作するソフトウェアとハードウェアの作成方法、強い衝撃の回避と振動の緩和、及び様々な意見から一つの決定を導く事がグループにおける問題であると定めた。

（※文責: 小林浩也）

2.2　課題の設定

本グループでは、正常に動作するソフトウェアとハードウェアの作成方法についての課題とアプローチとして、予想される不具合を可能な限り事前に排除し予想外の不具合の解決に集中するために、全体及び領域毎の設計を十分に行った上で作成の各段階でテストを実施する事にした。また、強い衝撃の回避と振動の緩和についての課題とアプローチとして、ローバー内の運搬コンテナは内容物への衝撃や振動を緩和するよう設計し、降下システムは着陸時の衝撃がなるべく小さくなるように設計する事とした。また、様々な意見から一つの決定を導く事についての課題とアプローチとしては、実際に活動を行っていく中で方向性を模索する事とした。

（※文責: 小林浩也）

第 3 章　インターワーキング

この章ではプロジェクト全体の進捗やグループ相互間の活動を円滑に進行するためにプロジェクト内における活動について述べる。

（※文責: 比留川満洋）

3.1　プロジェクト内のインターワーキング

今年度は、新型コロナウイルス感染症（COVID-19）の影響により、オンラインでの活動を主に行った。会議は Zoom を用いて行い、全体の進捗管理や情報交換を行った。さらに、コミュニケーションツールとして Slack を用いて意見交換を行った。分野ごとにチャンネルを作成し、資料共有を行った。本プロジェクトでは、2 グループに分かれて製作を行ったため、各グループで共通する部品は共用した。担当する学習分野は飛行物系、プログラム系、構造系、電装系、通信系の 5 つの分野にて分かれて活動し、各グループで情報共有を積極的にした。

また、本プロジェクトを開始するにあたって、CanSat の理解を深めるため、体験型学習として i-CanSat と呼ばれる教育用人工衛星開発キットを購入しプロジェクト全体で共有して学習を行った。i-CanSat ははんだづけや電子部品の仕組み理解に用いた。大学にはプロジェクト学習のうち 3 回に 1 回での登校が認められ、3 回に 2 回はオンラインでの活動となった。そのため、思うように実機の製作は行えなかったが、適宜スケジュール調整を行い、限られた時間内で実機完成に向けて 2 グループとも活動した。

（※文責: 比留川満洋）

3.2　グループ内のインターワーキング

本プロジェクトでは、小さな筐体を 1 チーム 5 人で製作をするため、各担当の作業内容が他担当の作業内容へ大きく影響する。そのため、調査した内容に問題点や疑問点を感じた際には、互いに Slack や Zoom を用いて共有し、問題解決を図ると同時に情報共有を綿密に行った。

本グループでは、前期に Google スプレッドシートを用いて、グライダーやローバー、電子パーツ、プログラムについての設計・製作・調査状況についての進捗管理を行った。後期では Slack を多用し、進捗確認と問題点の共有をし、メンバー同士で助け合いながらグループ全体的に協力的に活動をした。

（※文責: 比留川満洋）

第 4 章　活動内容

4.1　到達レベル

この章では、本グループのミッションの概要、到達レベル、課題の割り当て、活動スケジュールや作業詳細について説明する。

（※文責: 佐々木光流）

第 2 章第 1 節で示したミッションを達成するために、6 つの到達レベルを設定した。ミッションの各段階において、どのような問題が存在し、それに対してどのような到達レベルを設定したのかをまとめたものである。

1. 空中で CanSat の無線通信をオンにすること

   CanSat は、打ち上げる前は無線通信がオフの状態でなければならない。これは、ARLISS の大会規定で定められている。したがって、空中で分離するときに無線通信をオンにする必要がある。今回、打ち上げにはドローンを使用する。

   ここでの到達レベルは、ドローンから分離する際に確実に無線通信をオンにできるようにすることである。

2. 落下の際の減速機構と着地時の衝撃吸収

   CanSat を自由落下させてしまうと、着地の際に大きく破損してしまうため、減速機構を用いて自由落下を防ぐ必要がある。また、着地時も本体や電子機器に大きな衝撃が加わるため、衝撃を吸収する機構を用意して破損を防ぐ必要がある。

   ここでの到達レベルは、機体や電子機器が破損せずに着地ができることである。

3. GPS による位置情報の取得

   機体が着地後に自律走行するためには現在の自機の位置を知る必要があり、そのために GPS は必要である。前述したように、CanSat は分離時に電源を入れるため、事前に GPS 衛星を捕捉することは不可能であり、捕捉は落下中に行うことになる。また今回、目標地点も GPS 情報として捕捉させるため、困難であることが予測される。加えて、機体に搭載された電子機器や無線通信機の発する電波もノイズになる可能性があるため、注意する必要がある。ただし、私たちが製作する機体の性質上、空中では位置情報を必要としないため、着地までに GPS 衛星を捕捉できればよい。しかし落下時間は短時間であるため、できるだけ素早く GPS 衛星を捕捉することが必要である。

   ここでの到達レベルは、GPS 衛星の電波を素早く捕捉することである。

4. マイクロコンピュータによる制御

   GPS によって位置情報を取得した後、マイクロコンピュータによってモーターを制御し、機体を目標地点まで走行させる。この時に、制御に物理的に失敗したり、制御アルゴリズムに欠点があると、目標地点まで走行できない可能性がある。また、目標地点に到達したこと

を判定することも必要である。

ここでの到達レベルは、目標地点まで確実に走行できるような制御アルゴリズムを構築することである。

5. 地上局との無線通信

これはミッションとは直接関係ないが、人工衛星を打ち上げるための学習という点では必要である。無線通信に関しては、CanSat が自律走行できているかを確認するためにログを取り、それを地上局に送信することを想定している。

ここでの到達レベルは、機体から送られてくる情報を正しく地上局で受信することである。

6. トラブルに対する対応

本プロジェクトでは、製作と実験を何度も行うため、様々な想定外のトラブルが起こる可能性がある。過去の例を見ても、すべて計画通りに実験が完了した例はほとんどない。また、今年度は新型コロナウイルス感染症 (COVID-19) の流行により、前期はほとんど実機製作に取り掛かることができなかったため、よりトラブルが発生する可能性が高い。そのため、トラブルが発生してもすぐに対応できるように、様々な可能性を想定し、対策する必要がある。

ここでの到達レベルは、起こりうるトラブルや障害をできる限り想定し、その対策案を講じることである。

また、本プロジェクトでは、ミッションをいくつかの段階に分け、実験結果がミッションに対してどの程度の成功であったかを評価する。本グループでは、ミッションをミニマムサクセス、ミドルサクセス、フルサクセス、アドバンスドサクセスの 4 段階に分け、実験がどこまで成功したかの判断基準とした。各成功基準を設定した表は以下のとおりである。

成功基準 (%)	ミッション内容
ミニマムサクセス (60%)	軟着陸ができて、CanSat 本体が走行を開始できる
ミドルサクセス (80%)	目標地点に到達できる
フルサクセス (100%)	運搬物を壊すことなく目標地点に到達する
アドバンスドサクセス (120%)	ゴール地点を 1 周する

表 4.1　成功基準とミッション内容

本グループでは、フルサクセスまで達成できればミッションを達成したものとする。

（※文責: 佐々木光流）

4.2　課題の割り当て

第 2 章第 1 節で示したミッションを達成できる CanSat を製作するにあたって、以下の項目に分類して課題の洗い出しを行った。ハードウェアに関しては、飛行系と筐体系に分けて分類した。

飛行物系　一般的な CanSat では、降下の際の減速機構としてパラフォイルを採用する。しかし本グループでは、降下時の縦方向の衝撃を緩和できることや降下中の芸術性から、降下と同時にグライダーを展開し、螺旋を描くように降下させることにした。この時の課題としては、材料や設計はどうするのか、降下中にグライダーが展開するかどうか、グライダーが機体の重量に耐えられるかどうか、うまく螺旋飛行するかどうかなどがあげられる。今回提案者が飛行模型について一定の知識や経験があるので、そのメンバーを中心に設計、製作を進めていく。なお、重量や形状などについてローバー構造系にも関係するため、連携をとりながら進めていく。

プログラム系　着地を判定し走行を開始する、GPS から位置情報を取得する、目標地点を認識しそこへ向かうように制御を行う、ログをとり地上局に送信するなどを実行するプログラムを作成する。使用する電子部品のサンプルコードなどを参考に、CanSat がうまく動作するようなプログラムを作成していく。

ローバー電装系　地上に着地後に走行を行うローバーの電子部品の選定、配置設計や製作を行う。なお、部品の種類や配置がプログラム系やローバー構造系の作業に影響する可能性があるので、各分野と連携しながら作業を進めていく。

ローバー構造系　地上に着地後に走行を行うローバーのタイヤなどの電子部品以外の材料の選定や設計、製作を行う。ARLISS の大会規定では、オープンクラスの CanSat の重量は降下機構を含めて 1050g が上限であるが、今回降下機構にグライダーを採用したため、パラフォイルの場合よりも降下機構が重くなる。そのため、ローバーはできる限り軽量化する必要がある。そのうえで、グライダーで降下したとしても着地時に衝撃が加わるため、強度も追及する必要がある。これらのことを考慮しつつ、飛行物系やローバー電装系と連携して、なるべく軽量かつ十分な強度を持ち、安定して走行が可能なローバーを設計・製作していく。

これらの課題を解決して製作を進めるにあたって、グループメンバーで各分野を分担して勉強や設計、製作を進めた。なお、製作するうえで密接な連携が必要と考えられる分野には、両方を同じメンバーが担当するように割り当てを行った。

飛行物系　小林、村上
プログラム系　永井、比留川、小林
ローバー電装系　村上、佐々木、比留川
ローバー構造系　佐々木、小林、永井

（※文責: 佐々木光流）

4.3　課題解決のプロセス

この節では本グループの課題解決を行うための活動内容を説明する。

（※文責: 永井彬博）

4.3.1 前期スケジュール (作業内容)

5 月

グループ内の管理職を決め、グループの目標までの問題点を列挙していき、過去の製作物の搭載機器などを調べ、さらに使用する材料を考えた。

6 月上旬

目標を達成する上で必要な課題に対する解決策についてそれぞれのメンバーが考え、その考えについて意見を述べあって、良い案を採用した。課題解決の考案と並行し、実験時、問題点を実験の流れを把握することで、問題点がどの部分にあるのかを明確化した。加えて、新たな課題を見つけていき、その課題についても解決策を考えていった。ミッションの流れや課題解決を考えていく中で知識の不足などを感じ、その度にメンバー全員で疑問点に関する資料を探し、情報交換を行った。

6 月中旬

製作物の具体的な構成案について考えた。考えている中で自分たちの課題が漠然としすぎていると感じたので、課題の再検討およびサクセスクライテリア (成功基準) を設定した。他にも、課題を達成していくうえで、段階的に実験を行えるよう設定を行った。次に構成について考えている中で製作時にどのような問題があるかを考え、その問題を解決していくための必要な知識・構造などをメンバー各々検討した。具体的構成案が完成後、構造に必要な材料や機能を考え、その後各分野の役割担当を決定した。各分野はそれぞれ飛行物系・ローバー電装系・ローバー構造系・プログラム系の 4 つに最終的に分野の担当に分かれた。

6 月下旬

中間発表に向けて、Web サイトの製作や報告書の章立てを優先した。グループ内での活動は少なくなったが、少ない中で具体的製作物の材料の検討やそれにかかる費用などを考えていった。随時決まってきたマイクロコンピュータなどの機器の検討や配置、製作物の構成、課題解決のプロセスについて Web サイトやポスターに追加して中間発表会の準備を進めた。

7 月上旬

引き続き中間発表会に向けて準備を進めた。それと並行して、具体的な基板に設置する部品について検討をして、決まったものから発注をした。その後、設計に向けて役割分担を行い、その中で役割ごとにリーダーを決めて、開発工程案を検討することに決定した。設置する部品を検討する際に、課題解決のプロセスに沿っているものであるかなどを考えて、材料の検討を慎重に行った。

構成を練っている間に、意見の食い違いや情報の共有不足があったので、グループ会議時や Slack などのコミュニケーションツールを用いて、しっかりと構成に必要な条件などを共有して、食い違いが残らないように情報共有を行った。中間発表会で掲載する Web サイトの内容が違っていないか、文章に誤りがないかをメンバーで確かめ合った。

7 月中旬

中間発表会終了後、Google Forms で作成した評価アンケートを用いて本グループの評価を確認し、良い点、悪い点、そして今後の作業に影響しそうな問題がないかなど、第 3 者の意見を確認し改善するところを検討した。その後、メンバー同士のフィードバックを行い、意見などを交換し合った。

7 月下旬

オンラインで発注するものとホーマックに行って直接購入できるもので分け、構造物に必要な材料を引き続き検討した。夏期休暇が開始前に直近の予定と後期予定について本グループの予定を検討した。役割分担した 4 つの役割ごとに開発工程を考えてくることにした。そして、グループ全体で不安点・問題点について話し合い、不安点の解消を行った。

（※文責: 永井彬博）

4.3.2 後期スケジュール (作業内容)

8 月上旬

夏期休業の時期に入り中間報告書の締め切りも近づいてきていたので 8 月の基本的作業は実際に回路を組むための学習や機体の部品の材料の購入、Tex のインストールや基本操作など、ローバーの製作や報告書の作成を中心に作業を進行した。ローバーの作製では中間発表での製作の際の問題点での解決策を考慮しながら材料を検討した。夏季休業前に購入したセンサ類に関して勉強不足と判断したためサイトなどでセンサについて学んだ。

8 月中旬

センサ類に関して、いまだ不透明な部分があるのでそこを明確にしてセンサに関して理解を深めた。飛行物に関して、ローバーのサイズを胴体の空気抵抗を考えるとサイズが大きいという問題が出てきた。構造系ではタイヤのサイズは購入したものをそのまま使用したいという意見が出ており、そこの意見の合致のために議論を行った。結果できるだけ軽量な材料を使用し、製作を進め飛行物の担当に随時進捗を確認し、サイズや重量について話し合いを行った。機体の製作を中心としているため各担当に分かれた役割を進行していくこととなった。それを同時並行で、報告書の章立ての見直しと担当分けを行って、文章を作成していくこととなった。

8 月下旬

報告書の作成を中心に作業を進行した。夏期休業中に報告書を仮提出するためにメンバーが考えてきた文章を TeX のファイルに変換させ、PDF 形式にまとめて仮提出が可能な形に進めた。学校での作業日を申請し工房で、購入していたセンサ類の単体の動作確認を大まかに確認することができ、具体的に回路の設計などに取り掛かることができるようになった。回路の設計と平行して役割ごとに分けているローバーの構造班はどのような配置にすれば走行時に問題がないかなどを検討した。

9 月上旬

8 月下旬と同じく、報告書の作成・添削を中心に進めていった。TeX ファイルに変換した時に、不自然になっているものがないか、レイアウトや見出しをつける際に節や章の補足説明をするかなどを話し合い、特に言葉の使い方について統一性を意識して作成を行った。例を挙げるならば制作と製作、新型コロナウイルス（COVID-19）感染症と新型コロナウイルス感染症（COVID-19）などの細かな部分にも気を遣うようにした。その後、先生に報告書を仮提出し、フィードバックをいただきそのフィードバックから、長い文章を項目ごとに分け、見やすくするなどの読み手が読みやすいように報告書を作成していった。実機の製作については今後の予定を見て、どの部分を優先して進行するかなどを検討した。加えて構造の部分的なものに関して必要かどうかの再検討を行った結果、ローバーの 3 輪目はそのままなくさずに、姿勢安定の向上として設置することを決定した。

9 月中旬

8 月下旬から作成していた報告書を一度先生に提出してフィードバックを 9 月の上旬に受け取り修正を行った。修正を行った後グループメンバーで誤字や脱字がないか、正式名称、統一した主語などの確認を厳密に行い、先生に仮提出をもう一度行った。そこで先生から問題ないとフィードバックを受け取ったのでプロジェクト学習 WG に本提出を行い、メールでサインの代替を行った。

9 月下旬

スケジュールを確認する中で、できるものが限られていることを認知しており、グループ内で今後の方針について議論を行った。結果としてグライダーの展開機構を省略し、羽が開いた状態で実験を行い、実験する際にドローンをもちいて実験を行う予定だったが、ドローンと機体との切り離しについても考慮すべき点であり、限られた時間の中では困難であると考え笹流ダムの高所から投射する形となった。そして、プログラムに関してはタスクの分散のために各関数をメンバー内で分担してタスクの振り分けを行った。

10 月上旬

本格的にプログラムをメンバーで作成するために変数規則や命名規則などを決め各自受け持った関数を分担して行うこととなった。学校での作業ではプログラムと同様に本格的に機体の製作を行った。学外で実験をするためにその申請書を先生に提出し確認してもらい、その実験に間に合うように完成で来るようにスケジュールを組んだ。工房では他のプロジェクトも使用しているため配慮を持って製作を行った。工房での作業ではプロジェクト時間内という制約があるため時間を有効活用しなければならない。そのために作業の前のプロジェクト活動で学校での作業内容を大きく見積もり、製作を効率よく行えるように主なスケジュールを立てた。

10 月中旬

学校での作業は 3 回に 1 回なので、作業ができない場合はオンラインでの活動となっているのでオンライン上でできることを進行した。オンライン上ではプログラムの作成を主に進行した。学校での作業と並行でプログラムの作成を行い、オンライン上のプログラムの作成以外は機体の製作をする上での問題点の検討や、学校での作業をするためにどのように製作

を行えば、効率が良いのかを議論した。

10 月下旬

センサの 1 つである GPS の不具合が発生し、動作を確認することができなかったので壊れている可能性を考慮し、新しく GPS を購入した。学校での作業でモーター 2 つをモータードライバで制御して回すときバッテリーの電力不足が問題となり、急遽バッテリーを 1 つ追加した。そして、飛行物ではローバーを入れるためのコンテナの設計・製作に取り掛かった。学校での作業で GPS を新しくはんだ付けする予定であったが、GPS データを取得するのに時間がかかりその時間が長時間であったため、故障だと勘違いをしていた。GPS センサは問題なく作動した。11 月からは実験を開始することとなっているので、プログラム・実機の製作を間に合わせるように進行した。

11 月上旬

オンライン上での作業は報告書の仮作成に向けた構成と役割分担と成果発表会に向けた Web サイトの構成・役割分担を決め、できる部分から作業に取り掛かった。実験も予定に入り込んでおり、1 回目の実験はコンパイル動作がセンサの影響でうまくいかずに失敗した。飛行物に関しては 1 回目の実験には完成せず、実験を行うことができなかった。実験後学校での作業が重なっていたため、不具合の調整を行ったところ、モータードライバに異常があり新調しなければならなくなった。問題の検討をするためにオンライン上で議論を詳細に行った。

11 月中旬

11 月上旬と同様にオンライン上では成果発表会の Web サイトの作成を進めていった。前回の実験でモータードライバに不具合がありモータードライバを変更しなければならなかった。さらに、モーターのトルクが足りないという問題もあり、モーターも新しく高トルクのものを用意しなければならない状況になった。よって、実験に 2 回目は走行実験ができず飛行物の落下実験のみを行う予定であった。しかし、実験場所へ飛行物を運ぶ際にパーツの一部をなくし、天候も悪くなり実験を断念することとなった。その後、空いた時間に新しくしたモーターの動作確認を行い、正常に動くことを確認した。

11 月下旬

モータードライバの故障部分を新しいものに交換し、その後動作確認をして正常に動作することを確認し、はんだ付けが必要な部分を接着し電装系は完成した。飛行物のパーツの一部をなくしてしまったため、代用品を検討し再度製作しなおし、その後コンテナの脱出機構を製作した。モーターを新しく変更したためモーターを支える土台とモーターのタイヤの固定部分を変更しなければならず、工房の 3D プリンターやレーザーカッターなどを用いて再度、設計を行い製作した。

3 回目の実験には間に合わず走行実験はできずに 2 回目同様に飛行物のみの実験となった。4 回目の実験を行う前にローバーの構造は完成したがプログラムがうまくいかず調整が必要となった。4 回目の最終実験ではローバーも目標地点に向かうような振る舞いをし、飛行物も姿勢は安定するようになった。その後、飛行物のコンテナの中に実際にローバーを収納し、落下実験を行った結果、着陸の衝撃によってローバーのモーターを支える土台部分の

パーツが外れ走行不可能となり、目標を達成することができなかった。

成果発表会までに学校での作業が 1 回あるのでそこで最終調整をし、走行実験だけでも成功させようと進行した。

12 月

成果発表会前のプロジェクト活動で最後の走行実験を行った。飛行実験は場所の制限上実験をすることできないので、コンテナからローバーが出たところから実験を始めた。走行実験は学校の前の中庭で行った。障害物がない状態で GPS の誤差に影響が出ない場所で行った。実験は 5 回行い、その中の 3 回が目的地につくことができた。2 回の失敗はモーターとタイヤの接合部分が外れてしまったことが原因であった。その後最終発表会に向けて Web サイト、ポスターの添削を行い、完成していない部分に関しては早く仕上げることにした。メンバー同士で共有していない情報に関しても質疑応答のときに答えられるようにした。成果発表が終わった後、アンケート結果を見て反省を行った後、各役割に分けていた報告書の作成を進めた。中間発表時に気をつけていたことを思い出して、読み手が見やすいような報告書を作成した。

1 月

報告書のフィードバックを確認し、訂正及び添削をした後、最終報告書を完成させ提出した。

（※文責: 永井彬博）

4.4 作業詳細

4.4.1 飛行物

飛行物は、ローバーを上空から安全に地上に届けるためのユニットである。ここでは、飛行物を開発した手順について説明する。

（※文責: 小林浩也）

設計

飛行物を設計する上で、まずは運搬物を壊さないというミッションに着目した。なぜなら、運搬物が壊れる大きな要因の一つとして着陸時の衝撃があげられるからである。この課題を解決する為に、パラシュートよりも着地の衝撃が小さいと考えられるグライダーを飛行物として採用する事とした。また、そこから、完成系は白い大きな鳥のようになると考え、グライダーにアルバトロス (あほうどり) と命名した。

グライダーを飛行物に採用するにあたって考えられる大きな課題は 4 つある。1 つ目は、非操縦状態でも飛行姿勢を安定させる事。2 つ目は、ローバーの重量 (約 500g を想定) を支えられる飛行能力を有する事。3 つ目は、ローバー (直径約 10cm 高さ約 20cm の円筒型を想定) を格納できるコンテナを有する事。4 つ目は、CanSat 規定サイズ (長さ 240mm、直径 146mm の円筒) に折りたためる事である。

これらの課題を解決する為グライダーの構造は、基本形状をエンテ型とし、主翼は 3 段階折りたたみ式とした。また、主翼下にコンテナを配置し、折り畳みはコンテナを包み込むように行う事とした。まず、エンテ型を採用した理由は、ローバーの重さが与える機体のバランスへの影響を小さくするためにコンテナを重心位置付近に配置しなければならないが、一般的な飛行機の形状ではコンテナを重心位置付近に配置しつつ折りたたみの機構を実装することが困難だったためである。つぎに、主翼を 3 段階折りたたみ式にした理由は、飛行能力の目安の一つである翼面荷重 (翼が単位面積あたりに支える重量) が一般的な模型飛行機と同程度 (35g/dm^2 前後) になるためには翼面積は 20dm^2 程度必要であるが、主翼を 3 段階折りたたみ式にして主翼の長さを約 1.5m 程度にしなければ翼面積が不十分なためである。実際の設計のイメージ図を以下に示す。

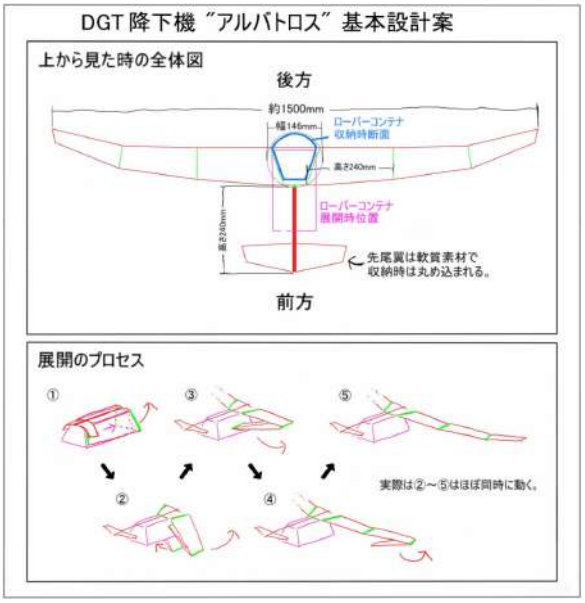

図 4.1　アルバトロス基本設計案

また、機体作製は 7 つの段階実験を通して作る予定であった。実験の内容は、1 つ目は、低高度での飛行試験 (基本飛行性能の確認)。2 つ目は、低高度での展開試験 (展開機能の確認)。3 つめは、中高度での飛行試験 (継続的な飛行性能の確認)。4 つ目は、中高度での展開飛行試験 (展開から飛行への移行の確認)。5 つ目は、高高度での飛行試験 (上空の風の飛行への影響の確認)。6 つ目は高高度での展開試験 (上空の風の展開への影響の確認)。7 つ目は高高度展開飛行試験 (上空での展開から飛行への移行の確認) である。しかし、この時点で新型コロナウイルス感染症対策の影響により、予定していた実験や作業を大幅に削減しなければならない事が分かったため、折り畳み機構の実装を見送る事とした。これにより、グライダーは CanSat 規定サイズから逸脱する事となり、グループ全体の方向性としてサイズ規定は重要視しないという方向に変更された。また、段階実験についても前述の 7 回から、1,2 回へと大幅に減らすことを余儀なくされた。

グライダーの具体的な構造については、カーボンと EPP(発砲ポリプロピレン) を用いて強度の確保と軽量化を図った。カーボンは炭素繊維が用いられた材質で、軽量かつ剛性に優れる。カーボンの欠点としては、やや高価な事が上げられる。EPP は柔らかく崩れない発泡スチロールのような材質で、軽量かつ衝撃吸収性に優れており加工も容易である。EPP の欠点としては引っ張り方

向に力が加わると簡単に破れてしまう点があげられる。EPP には発泡倍率というパラメータがあり、倍率が高いほど軽量で柔らかく、倍率が低いほど重く硬くなる。今回は、45 倍という中程度の発泡倍率の EPP を使用した。また主翼の構造も、EPP ブロックからの切り出しではなく骨組みに表張りをする構造にする事で軽量化を図った。

（※文責: 小林浩也）

作製

グライダーを作製する上で、まずは四分の一サイズの紙飛行機模型を作製して、翼の調整や重心位置等の基本的な空力特性を検証した。紙飛行機模型の作製には、3D モデリングソフト「Metasequoia 4」[6] と、ペーパークラフト作製ソフト「ペパクラデザイナー 4」[7] を用いた。これらのソフトを用いた理由は、グライダー担当者に使用経験があったためである。

作製した紙飛行機模型を公園にて飛行テストし、空力特性を検証した結果、次のような事がわかった。まず、主翼の翼断面はかなりしっかりキャンバー (翼断面の上への偏り) を付けたほうが飛行が安定した。次に、先尾翼は逆キャンバー (翼断面の下への偏り) にして迎え角 (横から見た時の翼の取り付け角度) を強めにつける事で、ピッチ (機首の上げ下げ) 方向の安定を得られた。次に、先尾翼にかなりきつめの下反角（正面から見た時の翼の取り付け角度）を付ける事で、飛行姿勢を復元する力が働いた。次に、主翼は中央から程よい上反角を付ける事でロール (翼の傾き) 方向の安定性を得られた。次に、主翼先端の上反角は少し強くすることでヨー (水平回転) 方向の安定性を得られた。最後に、重心は主翼付け根の前縁から後ろに 5.5mm(実機サイズに直すと 22mm 程度) で安定して飛行する事が出来た。実際の紙飛行機模型の写真と、調整後の翼の状態の参考画像を以下に示す。

図 4.2　紙飛行機模型

図 4.3　調整後の翼の状態

これらの結果を元に、実際のグライダーの作製に着手した。胴体はカーボンパイプを用いて作製した。主翼の骨組みは、厚さ 10mm の EPP で骨組みを作りカーボンスパー (細長い板状) とカーボンロッド (棒状) で補強をした。主翼の表張りは厚さ 3mm の EPP を用いて破れそうな箇所はグラステープを張って補強をした。先尾翼は最初は EPP とカーボンで作製したが、実験中に紛失してしまい作り直すにはカーボンが足りなかったため、カーボンを竹串で代用して作り直した。また、特に主翼と先尾翼を作る時は、ゆがみが出ないように気を付けながら、紙飛行機でのテスト結果で得られたセッティングになるべく近づくように作製した。コンテナ本体は、厚さ 8mmEPP で側面を除く箱を作り、前面の丸い部分は厚さ 3mm の EPP で、側面は厚さ 3mm の EPP と厚さ 1mm のバルサ材を組み合わせて作製した。また、後述する開閉の仕組みの為に、左側上面と側面に穴をあけた。最終的にグライダーの重量は 150g 程度になった。実際に作製したグライダーの写

真を以下に示す。

図 4.4　主翼の骨組み

図 4.5　グライダー正面から見た図

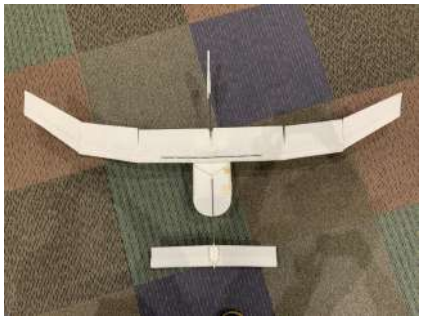

図 4.6　グライダー上から見た図

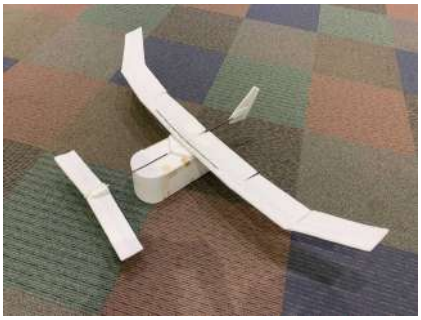

図 4.7　グライダー全体の様子

図 4.8　グライダーの重量

コンテナの開閉機構については、当初はニクロム線でテグスを焼き切って開けるという設計だったが、ニクロム線の温度が足りないことが判明したためサーボモーターを用いて開閉する仕様に変更した。コンテナを開閉する手順は、まずコンテナにローバーを入れ、コンテナ上部の穴からテグスを通し、ローバーのサーボモーターによって閉じているフック部分に掛ける。次に、側面の穴からも同様にテグスを通しフック部分に掛ける。最後に、上部から出たテグスと側面から出たテグスをしっかりと引っ張ってから結ぶ。これにより、ローバーがサーボモーターを使ってフックを開くと、テグスの引っ掛かりが外れ側面の壁が解放される、という仕組みである。コンテナを閉じている様子を以下に示す。

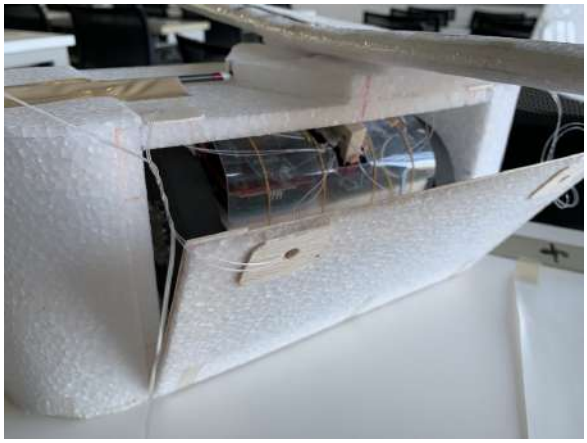

図 4.9　コンテナを閉じている様子

（※文責: 小林浩也）

実験

地上で機体を水平に投げてある程度調整した後、最初の実験を行った。実験の内容としては、水を入れたペットボトルをローバーに見立ててコンテナの中に固定し、高さ 25.3m のダムの上から飛ばした。結果として、機体の姿勢は安定していたものの全く前に進まずほぼ真下に落ちた。この結果については、重心が後ろすぎる事と、コンテナ形状がブレーキになっている事が原因だと考えられた。この実験の結果から前進しない問題の解決策として、疑似的に重心を前にするために先尾翼を少し後ろにして、コンテナ後方を空気が流れやすい形状にした。2 回目の実験では、実際のローバーをコンテナに入れて、同じく高さ 25.3m のダムの上から飛ばした。結果として、以前よりは前に進むようになったものの、依然としてほぼ真下に落ちた。原因を追求するため、コンテナに何も入れずにダムの上から飛ばしたところ、グライダーは下向きに半分宙返りしたあと、さかさまの状態で安定飛行した。

このことから、グライダーの空力特性（特にピッチについての特性）に問題がある事が分かった。原因として、下部につけたコンテナの空気抵抗が頭を下げる力を生んでいると考えられる。そのため、コンテナ形状を工夫する必要があると考えた。また、縮小した紙飛行機模型でのテストでは、大気に対する機体のスケールの関係で完全に空力を再現できなかったという事も原因として考えられた。このことから、今回はスケージュールの都合上段階実験を極端に少なくして開発したが、実物大スケールでの空力テストを増やすべきであったと考えた。

（※文責: 小林浩也）

4.4.2　構造系

本グループの大きな課題は「壊れやすい物体を安全に運搬する」である。そのため、構造についても、ある程度の衝撃から CanSat 本体と運搬物を守るような構造や機構が必要であると、議論の際に早期から意見が出ていた。本来、大会におけるカムバックコンペティションでは、轍や砂塵が巻き上がるような劣悪で不安定な環境下で走行することが多く、落下時の衝撃ほどではないにしろ走行時にかかる衝撃も小さくない。加えて、他大学の報告書や資料には、走行中に轍などに乗り上

げた衝撃で電装部位やセンサーに不具合が起こり、走行停止に追い込まれた実例も報告されている。

（※文責: 村上拓馬）

本グループでは ARLISS の規定によりオープンクラスは重量 1050g 以下、高さ 240mm 以下、直径 146mm 以下となっている。重量を規定内に収めることとなっているが本グループでは飛行物にパラフォイル・パラシュートを使用せず、グライダーを用いたため翼面荷重の問題もあり、できるだけローバーを軽量にしなければならない。グライダーの問題は作業詳細の飛行物系で説明している。グライダーの予想重量は 300g となっており、最低でも 700g 以内に収めなければ規定違反となってしまう。また、グライダーの翼面荷重を考慮すると、規定では 1050g 以下であるが、それ以上に軽量に作製しなければグライダーが飛行できなくなってしまう。そのため、ローバーではできるだけ軽く、そして衝撃にも耐えられる丈夫な材料を用いることを検討した。

次に使用した材料と作製物について説明する。

- 車体ベース
  軽量であり加工のしやすいのでバルサ板を用いた。大きさとしては縦 8cm、横 15cm、厚さ 1cm にカットし、車体のベースとして設計を行った。
- タイヤ
  低反発ウレタンフォーム（厚さ 2cm、半径 5cm）を使用した。当初の予定では 3D プリンターを用いて、作製する予定であったが衝撃を受ける際に ABS 樹脂は不安があったため、衝撃吸収にも役に立つ他にも、軽量ということでこの素材を用いた。
- モーター固定土台
  バルサ板にモーターを直接つけることは難しくモーターの土台を作製することにした。設計するときに使用したアプリケーションソフトは Blender[8] と Autodesk 社の Fusion 360[9] を使用した。Blender と MakerBot[10] の 3D プリンターが相性が悪く stl ファイルに直してプリントしてみると設計したようにいかず、作成が失敗となった。そこで Fusion360 を用いて設計をし、3D プリンターで製作すると設計どうりに作製することができた。以下に Blender で設計したモノと Fusion360 で設計した画像を示す。

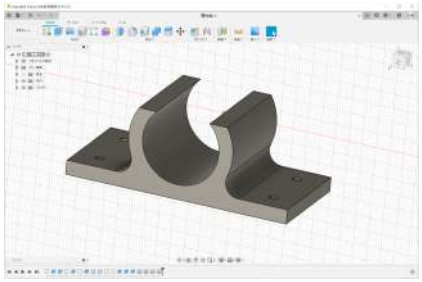

図 4.10　Blender で設計した土台

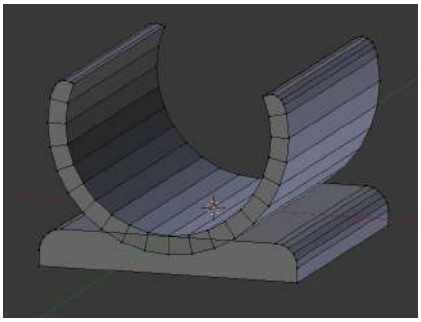

図 4.11　Fusion360 で設計した土台

しかしながら、モーターの問題により新しくモーターを新しくすることとなり、モーターの土台

も 3D プリンターで同じく作製することとなった。使用した材料は前回と同様、ABS 樹脂を使用し、3D プリンター MakerBot を用いて作製を行った。新しいモーターは長方形型のモーターであり、2 層である構造の支えにもなるように作製を行った。設計したものと実際に組み立てたものを下に示す。

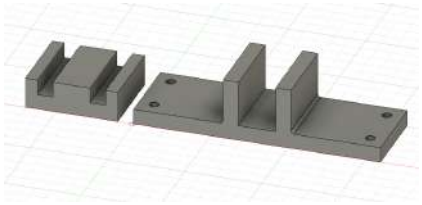

図 4.12　土台 2 つ目

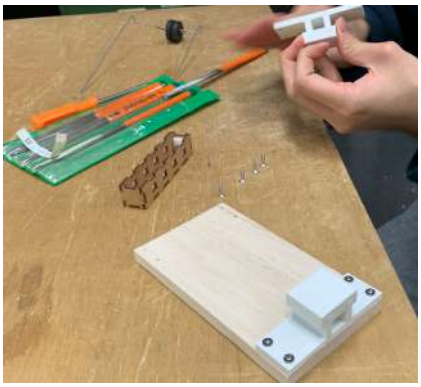

図 4.13　取り付けている土台

（※文責: 永井彬博）

- 収納箱

  本グループのミッションである、「物を壊さないように運ぶ」を達成するための運搬物を入れる箱を作製した。今回、私たちが運搬物に設定したのは市販のチョークであったため、これを衝撃吸収材とともに収納できるように設計した。使用した材料は、大学工房の端材にあった MDF(Medium Density Fiberboard) を使用させていただいた。また、衝撃吸収材には、メラミンスポンジを薄く切って詰め、収納箱の中でチョークが壁に直接当たらないようにした。設計には、無料の設計ソフトである MakerCase[11] を利用して SVG ファイルを作製し、そのファイルを Adobe illustrator[12] で開いて調整を加えたものを 3D プリンターで切り出して作製した。Adobe illustrator で作製した設計図と完成した収納箱を下に示す。

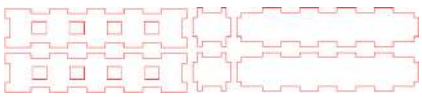

図 4.14　収納箱の設計図

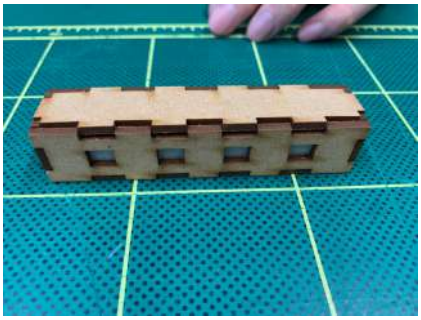

図 4.15　実際に作製した収納箱

- モーター固定具

  今回使用したタイヤはウレタンフォームであるため、モーターシャフトを直接タイヤに差し込むことが可能であるが、そのまま差し込むと固定ができず、走行中にタイヤがモーターから外れてしまう危険性があったため、固定具をタイヤに取り付けることによってタイヤを固

定し、ローバーが安定して走行できるようにした。材料には、大学工房の端材にあったアクリルを利用させていただいた。設計は、収納箱と同じく Adobe illustrator を用いて設計した。固定具の形状は、ローバー全体の重量をできるだけ軽くしたかったため、円形ではなく十字型を採用した。周囲 4 ヶ所の穴でタイヤにねじ止めをして、中央の穴にモーターのシャフトを差し込む設計になっている。なお、中央の穴の形状については、当初の設計では通常の円形であったが、モーターを途中で別のものに変更することになり、シャフトの形状が変わってしまったため、それに合わせて円形の一部を切り取ったような形状に設計を変更している。最終的な設計図と完成した固定具をタイヤに取り付けた様子を以下に示す。

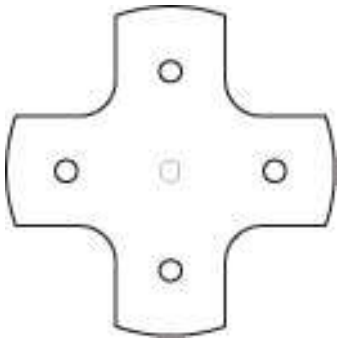

図 4.16　タイヤとモーター固定具の設計図

図 4.17　実際のタイヤとモーターの固定具

- 3 輪目構造

  ローバーの後方に小さいタイヤを取り付けることで、走行中の姿勢を安定させたり、路面の凹凸によるローバーへの衝撃を緩和させることが目的である。こちらの材料は、針金を折り曲げて骨組みを作り、そこに市販のタイヤを差し込んで作製した。しかし、実際にローバーを走行させてみると、タイヤが思うように回転していなかった。また、ローバーが一時停止した際に、前方に転倒してしまったので、前方にも針金を折り曲げて作製した簡易的なスタビライザーを取り付けた。

今回、ローバーに搭載する部品の数や大きさを考慮し、部品の搭載部分を 2 層構造とした。下段にモーター、バッテリー、そして収納箱を置き、上段に Arduino や基盤を積んでいる。上段と下段の接続部分に関しては、グライダー用の部品であるカーボンロッドの一部を使用した。上段と下段それぞれの画像を以下に示す。

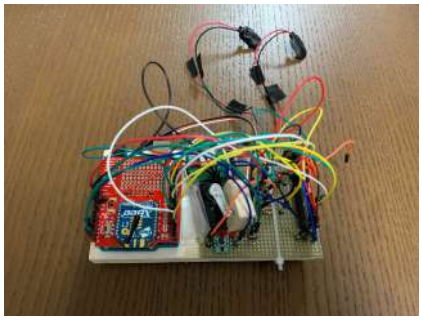

図 4.18　ローバー上層

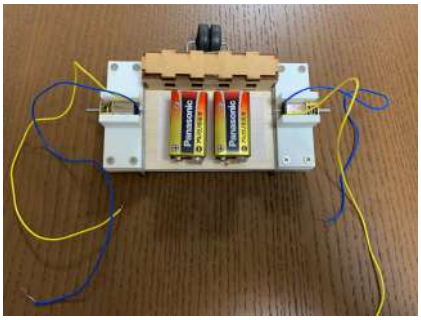

図 4.19　ローバー下層

これら 2 つを合体させた結果、ローバーの重量は、最初の段階では 512g となった。しかし、走行テストの結果、モーターのトルクが不足していたため走行することができなかったため、モーターをより高トルクかつ小型のものに変更した。また、グライダーのコンテナに収納する際に、ジャンパ線がまとまっておらず収納できなかったので、プラスチック板を使用してコンテナに収納できるようにした。その結果、総重量は 408g となり、100g 以上の軽量化に成功した。最終的なローバーとその重量を測った画像を以下に示す。

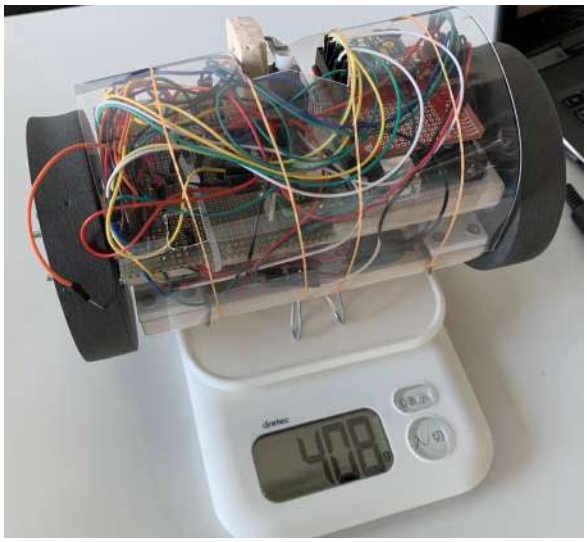

図 4.20　ローバー重量

（※文責: 佐々木光流）

4.4.3　電装系

電装部分についてはいくつかの電子パーツを採用することが決定している。大まかな現在位置を特定することで、飛行・走行を支援するための GPS 装置、加速度、傾きなど本体に何が起こっているのかを探知するための 9 軸 IMU、そして電子部品の中でも心臓部となるマイクロコンピュータ、CanSat の動力部となるモーターとその駆動を制御するモータードライバである。予め入力した命令や指示の処理動作を保存する EEPROM や CanSat 本体と基地局との通信を可能にする XBee モジュールなど CanSat の運用に必要な基本的な電子部品は本グループの課題と共にどのパーツを採用するのかを討論した。

当初は、小型カメラも採用し、GPS による情報と小型カメラからの情報とを組み合わせることで、より高い精度で周りへの状況の把握と安定した位置確認を想定していたが、最終発表の期間と実装においての高い難易度により搭載を断念した。よって、現状では GPS のみを用いて、位置把握と目標地点までの距離算出などを行う。また、頻繁に学校での作業が行えないため、電気回路作成アプリケーションなどを利用して、具体的な配置や実装について検討を進めている。さらに、基板についても既製品ではなく銅板を用いたものを自らで作成するという意見となった。

夏季休暇の間にメンバー間で担当する分野を決められ、基板作成について既製品ではなく自らで作成とのことになっていたため基板の設計を行った。しかし、基板作成の経験や知識もほとんどな

かったことから一からの基板作成が始まった。まず初めに電気回路の作成を行った。前期の時点で具体的にどのような電子部品を採用するのか決定していたため、インターネット上の説明書ファイルや解説ファイルを参考にして以下のような電子回路図を手書きで作成した。

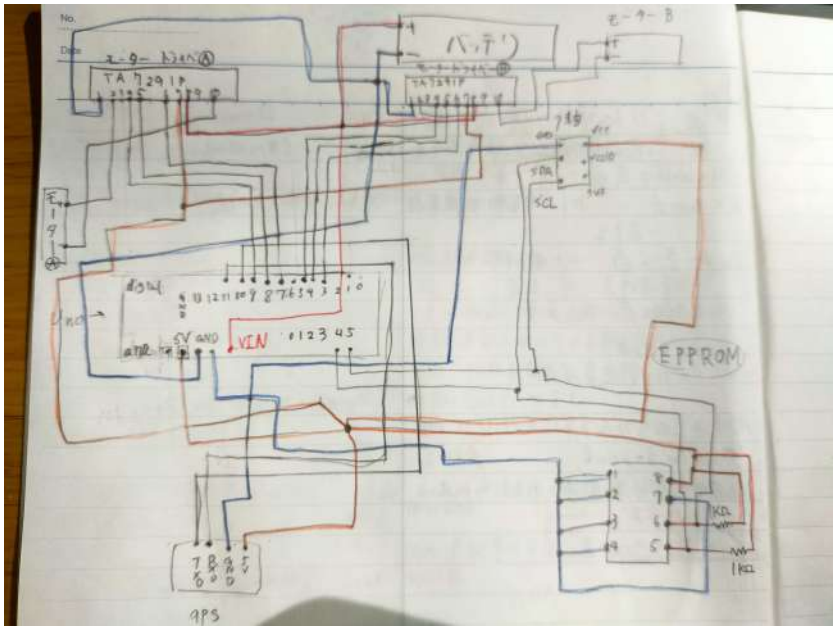

図 4.21　手書き回路図

手書きであったためやや見にくい電子回路図であったが、この電子回路図を参考に基板設計用アプリケーションを用いた電気回路の清書と具体的な基板設計を行った。CanSat 本体の大きさは決定されているため基板を搭載するスペースも限られている。そこで構造系のメンバー達と相談し $8cm \times 10cm$ の大きさの基板を作成していくこととなった。使用した基板設計アプリケーション「EAGLE」はその自由度の高さと複雑さから短期間で使いこなすことは非常に困難であったが電気回路図の清書は何とか完了し、基板の設計データも苦戦しながらも着々と進んでいた。以下がその清書した回路図と作製途中の基板データの画像である。

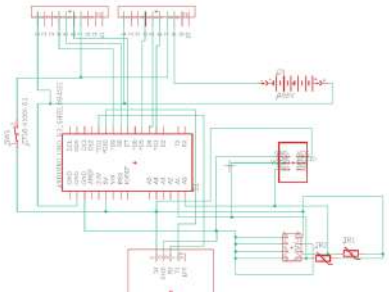

図 4.22　設計した回路図

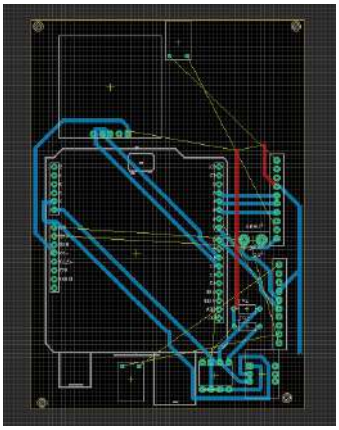

図 4.23　ガーバーデータ

しかし、購入した物資を受け取り確認すると購入したものは基板の素材にならないという事が判明した。購入したものはユニバーサル基板と呼ばれるもので実験、試作、および電子工作での配線、組立に用いられ、特徴として回路を自由に作成できるというものであった。当初は基板自体を一から作成するという方針であったが、基板作成に全く使えないということではないため、はんだと金属線を用いて電気回路を作成していく方針となった。

まず行ったことは注文したユニバーサル基板及び電子部品の確認とその配置についてである。インターネット上の説明書ファイルや解説ファイルからおおよそのサイズ感は把握していたがやはり実物を確認しながらの方がより具体的なイメージにつながった。

基板上の電子部品の配置を考察する作業と並行して電子部品の動作確認を行った。こちらは他のメンバーにも協力してもらい、全て部品に不備が無いことを確認した。その後、電子部品を基板にはんだ付けし電気回路を作成した。この作成時に最も工夫したことはジャンパー線用の接続ピンをいくつか採用したことである。このジャンパー線とは接続ピン同士をつなぐできて気軽に取り外しができる特徴を持った電線で、arduino 系などの規格にも採用されているものである。接続ピンを採用することで電子部品ごとに不具合や問題が発生した場合に回路を阻害することなく確認が行えるというメリットがあり、メンテナンス性が大きく向上することから基板作成にジャンパー線用の接続ピンを採用した。以下が作成途中の基板の画像である。

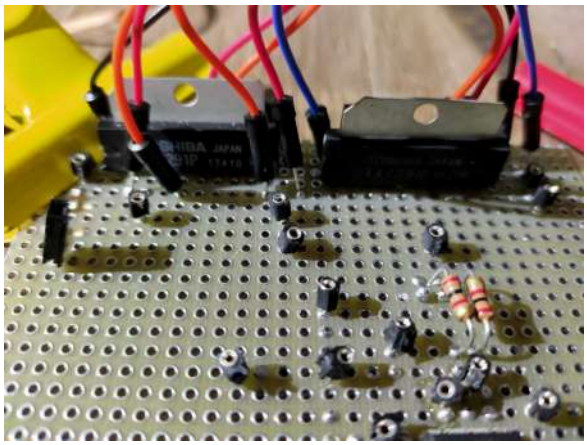

図 4.24　作成中の基板

しかし、次の問題が発生した。CanSat 本体が完成し、走行テストを行った際本体が動かないという問題である。原因はモーターのトルク出力不足であり電力バッテリーを増やして対応したがやはりトルク出力が CanSat 本体の重量を上回らず走行が確認できなかった。また、この時２つのモータードライバのうち１つに不具合が発生し、片方のモーターが動かなくなってしまっていた。そこで基板に直接はんだ付けしていたモータードライバのピンを破壊し、新しいものと交換した。元々メンテナンス性の向上を基板作成時の工夫点としていたが今回のモータードライバ交換時に基板に直接はんだ付けしていたことが負担となり失敗であったことが分かった。このような事態にならないためにも電子部品を直接基板にはんだ付けするのではなく部品をはめ込むだけで接続できるソケットを利用してよりメンテナンス性を向上したいと考えている。

また、モーターについても以前よりも重量が軽く、トルク出力が大きいものを購入し搭載した。再度、走行テストを行ったところ今度はプログラムに問題があったが何とか走行できた。その後、作バッテリーと基板をつなぐバッテリースナップの電線がはんだ付けした部分の根元から切れることが何度かあり、その部分をはんだ付けし直すことがあった。最終実験日当日、グライダーの CanSat 本体を格納する際にジャンパー線の量が多すぎてしまいそのままではコンテナに入らないといった問題が発生した。この問題に対しプラスチック板で上から押さえつけ何とか入るようにした。

完成後の電装部分の反省点として次のような点が挙げられる。まず、メンテナンス性である。先程も述べたように接続ピンを採用することで部品の不具合を確認しやすくしたが、部品を基板に直接はんだ付けをしてしまったため部品の交換に手間がかかってしまった点である。
次がジャンパー線の採用数と長さである。基板の接続ピン同士や arduino とを接続した際にジャンパー線や接続ピンが多く、複雑すぎてどこに何を接続すればよいのか設計担当以外分からなくなってしまっていた点である。加えて、ジャンパー線が多すぎて CanSat 本体の全長が想定より高くなってしまいそのままではグライダーのコンテナに入らなくなってしまうなどといった点も挙げられる。

以上のことから電子部品をはんだ付けする際には直接基板に行うのでなく部品用のソケットをはんだ付けしそこへ電子部品をはめ込むようにする。また、基板上の接続ピン同士をジャンパー線でつなぐ際にはできるだけ短いものを利用し接続する際のジャンパー線の色を統一、整理することでジャンパー線がかさばることなくまとめられるのではないかと考えられる。さらに、ジャンパー線の長さをできるだけ短くすることで CanSat 本体の全長を超えない程度に抑えることができると考えられる。

（※文責: 村上拓馬）

4.4.4　プログラム系

言語について、まずマイクロコンピュータを Arduino UNO を使用するため、C 言語ならびに Arduino 言語を用いてプログラム作成することとなった。はじめは Raspberry Pi を考えたが、新型コロナウイルス感染症（COVID-19）の拡大防止により、大学での作業があまりできない中、慣れないマイクロコンピュータを用いるのは時間的にも厳しいと考えた。また、Arduino は 1 年時に開講された情報表現基礎 I の講義より、ピタゴラ装置制作時に用いたため、グループメンバー全員が Arduino の使用に多少慣れていると判断した。そのため、CanSat 製作には Arduino UNO を採用した。

プログラミングは、まず電装部品である各センサの動作が行えるように、Arduino 言語でプログラム作成をはじめた。ローバーの走行に関しては 9 軸 IMU に含まれる加速度センサとモーターを連動させ、時間制御を用いたプログラムの作成を検討している。目標地点の判定を、GPS センサを用いて目標地点から半径 10m まで来たら動作を停止させるように設定する。これは、GPS センサの機能上、正確な位置情報を取得することは困難であり、目標に対して半径 10 mの計測が限度であるためである。

（※文責: 比留川満洋）

次にローバーがどのような動きをするのかをフローチャートとして示す。

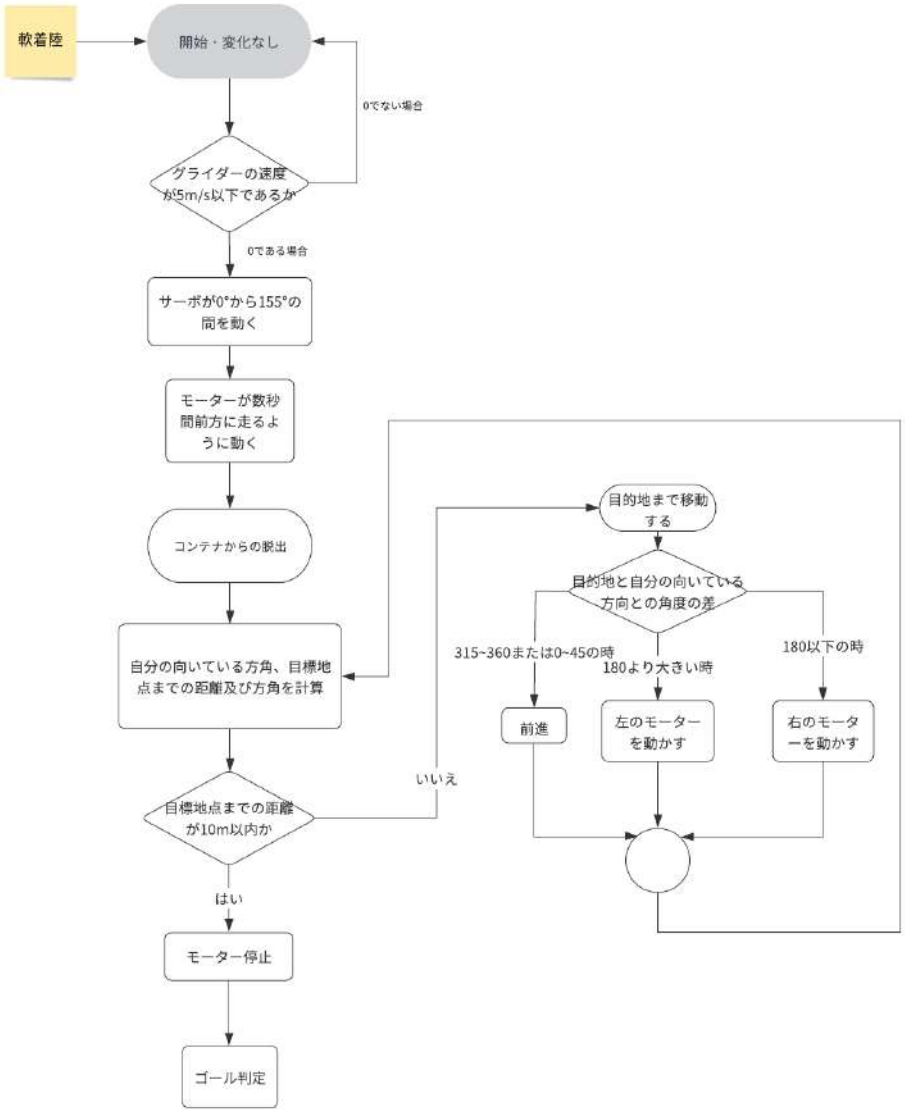

図 4.25　フローチャート

ローバーの動きの判断基準は以下の通りになっている。

- グライダーが着陸しているか（速度が 5m/s 以下であるか）
- 目標地点までの距離が 10m 以内であるか
- 目標地点と自分の向いている方向がどの角度であるか

この 3 つが主な判断基準となっている。

続いて使用したライブラリの説明を行い、実際に作製したプログラムの構造体及び各関数の説明をする。

始めに使用したライブラリについてまとめたものを以下の表に示す。

ライブラリ名	役割
TinyGPS++[13]	GPS の情報を受け取るためのライブラリ
math.h	数学で用いられる関数や定数を使用して計算するライブラリ
Wire.h	2 線による通信を行うための I2C 通信ライブラリ
EEPROM.h	EEPROM のアドレスに書き込む際に使用
stdlib.h	基本的なプログラムで使用する関数を使うためのライブラリ
string.h	文字列操作に関する関数が定義されているライブラリ
NeoSWSerial.h[15]	Servo.h と併用可能な SoftwareSerial.h の代替ライブラリ

表 4.2　ライブラリの説明

次に主要構造体の仕様として緯度・経度・高度・速度を表わす構造体と目的地への情報を表わす構造体を作成した。以下に構造体の説明について記述する。

```
struct GPSData{
double lat;
double long;
double alt
double mps;
};
```

lat は緯度, long は経度, alt は高度、mps は速度を示すものとして扱う。

```
  struct WayInfo{
double distance;
double direct;
}:
```

distance は目的地までの距離の値、direct は目的地の方角として扱う。

次に各関数についての説明をする。

GPSData GetGPS()

現在の GPS 情報を取得するための関数である。GPS を取得するために Tiny GPS++ というライブラリを用いて GPS 情報を取得した。Tiny GPS++ ライブラリは後程詳しく説明する。Tiny GPS++ ライブラリで機体の緯度、経度、高度そして速度の情報を取得することができこのデータでそれぞれ判断基準の条件分岐を行う。加えて、GPS の高さ情報が非常に誤差が大きい事が実験を行って判明したので、直近 100 回の値の平均を用いる事にした。

WayInfo CalcWayInfo(GPSData nowData, GPSData targetData)

現在地と目的地の情報から、目的地への方角と距離を計算する関数である。GetGPS() 関数で取得した緯度・経度と目標地点の緯度・経度を Tiny GPS++ ライブラリを用いて、現在地と目的地の情報から、目的地への方角と距離を計算する関数である。計算をした値を用いて実際に目的地に行くための条件分岐のために大きく影響される。

double GetDirection()

現在の機体の方角を取得するための関数である。機体自身の向いている向きを取得するための関数である。これは BMX055 使用 9 軸センサモジュール [14] を用いて機体の向きを度数として取得する。プログラムはサンプルコードを使用していた。しかし、実際に実験を行い出力結果を解析したところ、その中に含まれる値の修正が誤作動していると判明したため修正を行い、磁気センサーにはキャリブレートが必要であると判明した。そのためコンテナから脱出下のち、その場で旋回し、キャリブレートを行う動作を追加した。

void SetEEPROM(char[data)]

EEPROM に data の文字列を書き込むための関数である。制御ログを残すため、GPS センサと 9 軸 IMU の値を String data で文字列として取得する。取得した値は byte 型の配列 (buf) にコピーし、文字列の長さに合わせて動的にアドレスを確保して、EEPROM.write でアドレスにデータを書き込むことをした。また、データの読み出しでは、EEPROM.read で行った。

void SetXBee()

XBee に Arduino から取得したデータを書き込むための関数である。XBee は Arduino でシリアルモニタにデータを出力させ、PC に送信をすることができるのでシリアルモニタに GPS などのデータをこの関数でシリアルモニタに表示させ PC に通信するのは通信系作業詳細で詳しく説明する。

void ServoOpen()

サーボモーターを動かすための関数である。プログラムの内容としてはサーボを 0(基準値) から 155 まで開くようなプログラムである。サーボモーターを動かす際に、Servo.h というライブラリを通常使用するが今回、GPS シリアル通信を行うときに使用する、SoftwareSerial.h とが競合するため正常に動作しなかった。そのため上の 2 つのライブラリの役割を持つ NeoSWSerial.h というライブラリを宣言することで競合を避けることができた。

void SetMotor(double l, double r)

LR のモーターの速度を設定する（l,r それぞれ、-255〜255 の値を取る。関数内では、正回転・逆回転・停止を判断し、制御する）ための関数である。前進・右回転・左回転・停止を判断し制御するため、右のタイヤと左のタイヤそれぞれで制御を行った。前進の最大速度の設定は 255 とし、逆回転の最大を-255 と設定して制御を行った。

bool EscapeContainer(GPSData nowData)

条件を満たした時コンテナから脱出し、脱出が完了すれば true を返す関数である。条件の判定は目標地点と自分のいる地点の高度の差と自身の速度が 5m/s 以下となっているかが判断条件としている。条件を満たした時、コンテナの扉とつないでいるテグスがサーボモーターが起動することで緩まり、コンテナの扉が開くようになる。

GoToTarget(WayInfo way, double direction)

道のりの情報と機体の向きの情報を用いて、目的地へ接近するような関数である。ゴール判定は 10m 圏内に入れば終了となっている。どのように角度を知るかは自分の向いている向きと目標地点の向きとの差をとり、その値に数値 360 を加え、360 で割った余りで判断する。360 を加えることによって余りを用いる前位に負の値をとることが内容にするためである。

次にどのような条件分岐になっているのかを説明する。

フローチャート図にも記載されているが主に 3 つの条件となっている。

1. 計算した値が 315 から 360 または、0 から 45 の時：直進

2. 計算した値が 180 以下の時：左回転

3. 計算した値が 180 より大きい時：右回転

この 3 つの動きによって目標地点まで移動することとなっている。

最後に、解決できなかった問題について説明する。
解決できなかった問題は大きく 4 つある。

1. ローバーがコンテナから脱出が失敗
   ログを確認したところ、脱出のモードへ移行していなかったため、ソフトウェアに問題があると考える。これは、技術的検証が不十分なまま終了した。

2. EEPROM を用いたデータ記録
   技術的検証が不十分なまま終了した。

3. 位置推定の誤差
   今回位置推定には GPS のみを用いましたが誤差が非常に大きく、特に高度については全く利用できないレベルであった。よって、加速度センサや気圧センサを用いるなどして、位置推定のアルゴリズムを改善する必要があると考える。

4. 障害物に引っかかって目的地へたどり着かないという問題
   今回は意図的に移動アルゴリズムを単純にしたため予想の範囲ではある。しかし、今後より良いシステムにするためには移動アルゴリズムの改善が必要であると考えた。

（※文責: 永井彬博）

4.4.5 通信系

概要と目的

CanSat の競技大会規定より、CanSat が目的地到達のための制御を行えたか、制御履歴の確認を行う必要がある。そこで、本グループでは CanSat に無線通信モジュールの搭載をしパソコンとの通信を行うことで制御履歴を取ることとした。

通信は GPS センサと 9 軸センサの値をパソコンにリアルタイムで出力させ、得られたデータをログとして残し、CanSat の現在位置と目標地点までの距離を測定、分析するのに利用した。パソコンへの具体的な出力内容は、通信の経過時間、CanSat の現在地の座標値、CanSat の現在の高度値、ローバーの向いてる方向角度、目標地点の座標と現在地の座標の差、目標地点までの距離、ローバーの動きをそれぞれ数値や文字で表示し、各センサが正常に動作しているかの制御履歴の確認を行った。

これらを行うために、XBee と呼ばれる通信モジュールを利用した。

（※文責: 比留川満洋）

XBee とは

XBee は Digi International 社が販売している無線モジュールである。無線通信用の小型モジュールとして、屋外で最大 1.2km までの距離を通信可能とし、CanSat の競技大会で多く利用されている。小型で重量が軽く、価格も安価で無線通信の初心者でも比較的に簡単に取り扱うことが出来る通信モジュールである。電波法に関する無線通信技師などの資格は不要であり、消費電力がとても低く、管理や制御がしやすいのが特徴となっている。

（※文責: 比留川満洋）

使用部品

使用した部品は以下である。

- XBee ZB(S2C)/ワイヤアンテナ型
- XBee シールド (SparkFun 社製)
- XBee USB アダプター ver.2
- XBee 用 2.54mm ピッチ変換基板

XBee ZB(S2C)/ワイヤアンテナ型は通信を行うモジュール本体であり、大会規定 (FCC 認証かつ 100[mW] 以下の機器は電源 OFF にしなくてよい) を遵守しているものを利用した。これにより、実験時開始より電源を ON にした状態で搭載することができる。XBee シールドは Arduino と XBee モジュールとの間でロジックレベル変換をするために使用した。XBee USB アダプターはパソコンと XBee を接続するために必要であった。XBee 用 2.54mm ピッチ変換基板は XBee を基板やブレッドボードに接続する際に用い、主に接続テストに使用した。

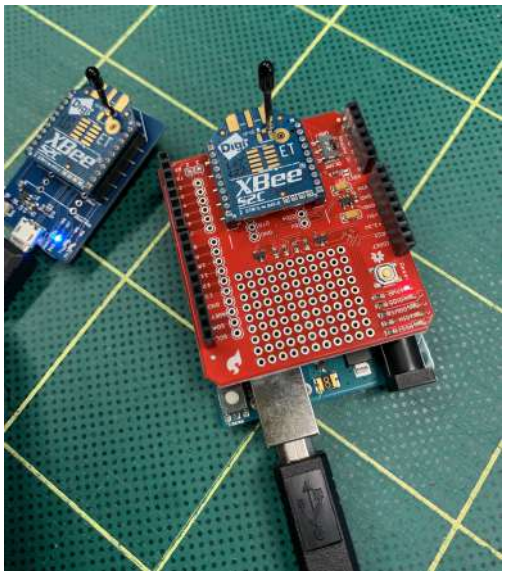

図 4.26　使用した XBee の写真

（※文責: 比留川満洋）

通信の仕組み

XBee 通信を行うには、XBee を 2 つ用意し、Coordinator（送信側）と Router（受信側）で使用を分ける必要がある。

- Coordinator（送信側）
  ネットワークに一つ存在し、ネットワークの制御を行う端末

- Router（受信側）
  データの中継機能を持つ端末

この設定を行うために、XBee の本体を設定するソフトウェア「X-CTU」をパソコンにインストールし、XBee の ID 登録や通信モードの設定を行うことで、シリアル通信の状態を確認することができる便利ツールである。通信モードには 2 通りあり、AT モードと API モードがある。

- AT モード（透過モード）
  シリアル通信を無線で通信する方法

- API モード
  API フレームで決められた構造でデータ通信を行う方法

本グループではシンプルに無線通信が行える AT モードで通信を行った。AT モードでは Arduino のシリアルモニタに出力されたシリアル通信データをそのまま XBee が無線でパソコンにデータ送信してくれる。これにより、XBee と Arduino 間でのシリアル通信データが同様であることを容易に確認することもでき、動作確認がしやすくなるメリットもある。

実際に XCTU を用いた通信様子は以下に示す。

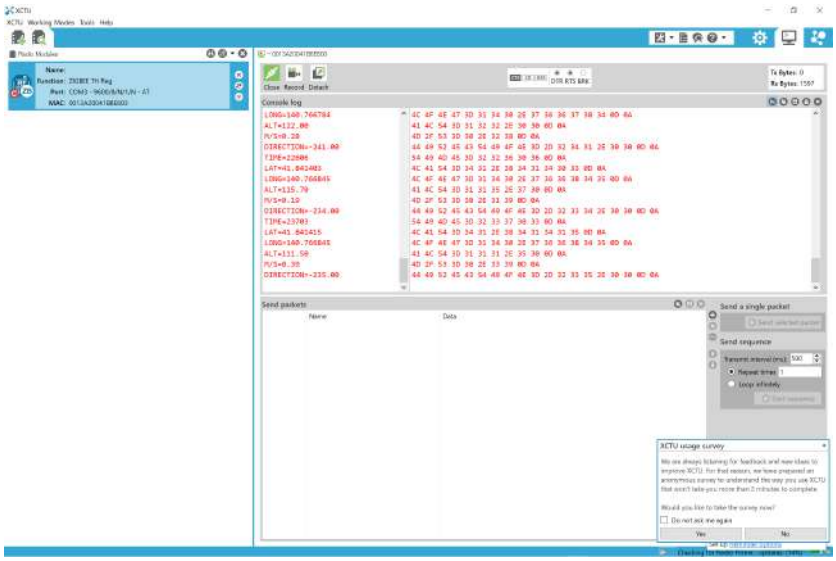

図 4.27　XCTU での受信の様子

しかし、XCTU のコンソール画面に表示されるデータ数には限りがあり、データ受信開始から終了まで、すべてのデータを保存することが出来ず、安定したデータ収集を得ることが出来ない。そこで、実験では Tera Term を利用したシリアル通信を行い、十分なデータを獲得することにした。

実際に Tera Term を用いた通信様子は以下に示す。

図 4.28　Tera Term での受信の様子

（※文責: 比留川満洋）

4.4.6　分析

ここでは、実験の動画やデータから分析を行った手順と詳細について説明する。

（※文責: 小林浩也）

動画と Kinovea を用いたグライダーの落下速度の分析

グライダーの落下速度の分析には、定点固定したカメラと動画分析ソフト Kinovea[16] を用いた。Kinovea は動画内の動体を半自動で追跡し、得られた軌道の分析が行えるソフトウェアである。グライダーの落下速度分析においては、ダムの正面にカメラを設置して動画を撮影し、落下するグライダーについて分析を行った。

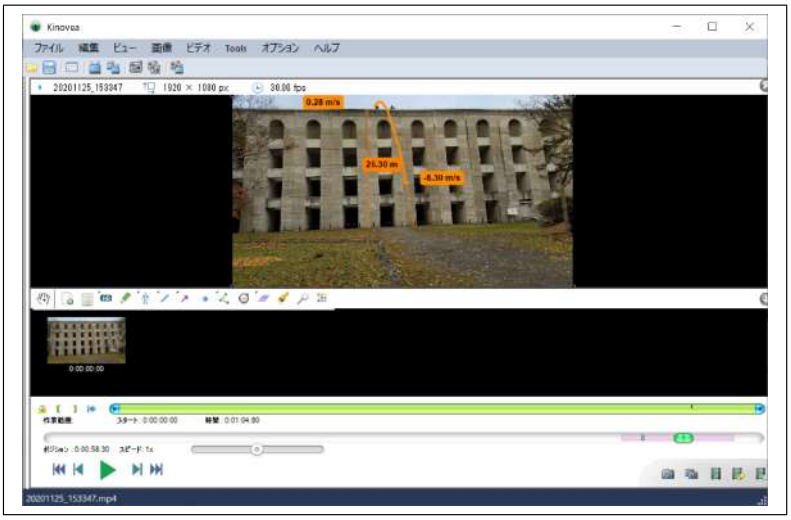

図 4.29　Kinovea 分析画面

Kinovea による基本的な軌道分析の手順を説明する。まず Kinovea に分析したい動画を読み込む。次に、追跡開始する時間まで移動して、追跡したいものの上で右クリックしてメニューを開き、軌道追跡を選択する。次に、四角い枠が現れるので、ダブルクリックをして、Object window と Search window を適切に設定する。最後に、動画を進めると自動追跡が行われる。ただし、この自動追跡は完全ではないので、追跡に失敗しているフレームがあれば手動で修正する必要がある。正しく追跡出来ている事が確認できれば、メニューバーの Tools にある Linear kinematics から、各種グラフの作成やデータの出力が可能となる。

Kinovea による分析を行う際、主に 3 つの問題点があったが、最終的に解決し目的の落下速度を得る事が出来た。その 3 つの問題と原因、解決策について説明する。

1 つ目の問題はうまくグライダーを追跡できないという問題であった。これは、Object window と Search window の設定が適切でないことが原因であった。解決策としては、Object window を対象のグライダーよりもかなり小さめに設定し、Search window を Object window の縦横 3 倍程度にする事で背景の境界など一部を除き、正しく追跡する事が出来た。

図 4.30　追跡できる設定と追跡できない設定の比較

2 つ目の問題は速度が m/s で得られないという問題であった。これは、高さの基準を設定していない事が原因であった。解決策としては、ダムの高さを表す線を引き、右クリックメニューから較正を選択し、実際のダムの高さを入力する事で解決することが出来た。

3 つ目の問題は、高さの基準がずれるという問題であった。これは、座標の原点が初期設定の画面中央のままである事が原因であった。解決策としては、メニューバーの Tools から座標を選択し、画面上のグリッドをドラッグして正しい位置に修正する事で解決することが出来た。

最終的に作成したグラフを以下に示す。

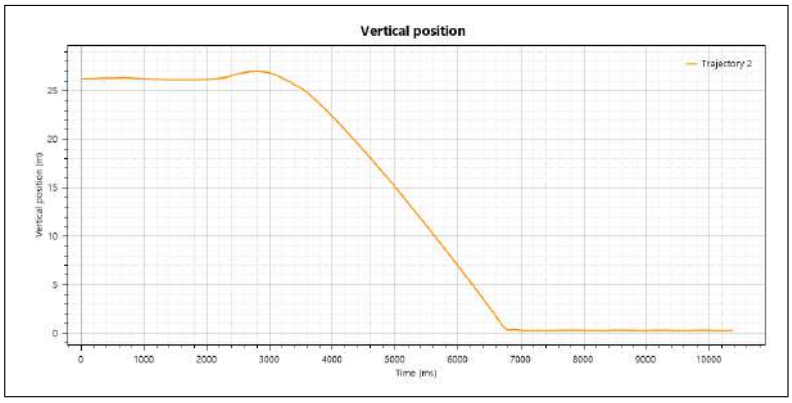

図 4.31　Kinovea で作成したグライダーの高さのグラフ

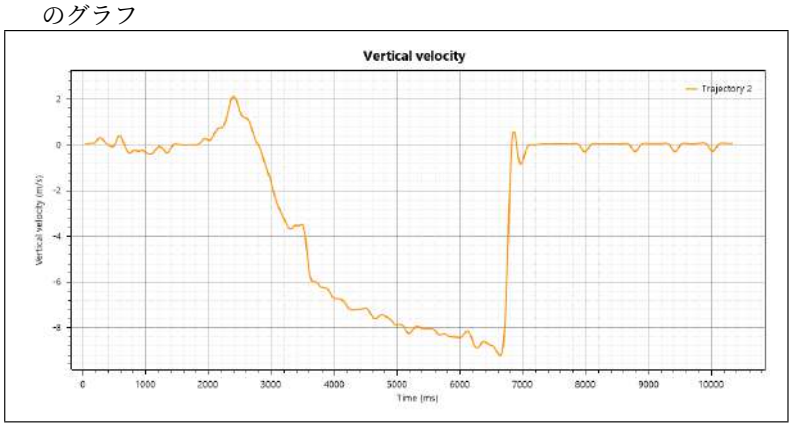

図 4.32　Kinovea で作成したグライダーの落下速度のグラフ

（※文責: 小林浩也）

XBee 通信で得られたログと Gnuplot を用いたローバーの制御と経路の分析

ローバーの制御と経路の分析は、実験中ローバーから XBee 通信で送信されるログを自作の変換プログラムで数値データに変換し、Gnuplot[17] でグラフ化と一部の計算を行った。

まず、ローバーから送られてくるログについて説明する。ログの各行は“[キーワード]=[値][改行]”の書式で記述され、14 種のキーワードと値が数 100ms 毎にローバーから送信される。それをコンピュータで受け取りファイルとして保存したものを分析に用いた。

```
#--------------------------------------------------
TIME=6195
GPS_LAT=41.8399772644
GPS_LONG=140.7657470703
GPS_ALT=111.3000030517
GPS_NEW_ALT=0.0000000000
GPS_M/S=1.21
MAG_DIRECTION=99.35
TARGET_LAT=0.0001678467
TARGET_LONG=-0.0005950928
TARGET_ALT=7.4000015258
TARGET_DIR=110.49
TARGET_DIST=52.64
EEPROM=GOTO_TARGET : Forward;
EEPROM_INDEX=0
#--------------------------------------------------
TIME=6823
```

図 4.33　ログの見本

次に、ログを数値データに変換するプログラムについて説明する。変換プログラムは生産性に重点を置いて C#で作成した。この変換プログラムはログのデータから必要な情報のみを抽出し、Gnuplot で読み込みやすい書式に変換するものである。入出力には標準入出力 (Console) を用いた。具体的な処理の内容は、まず標準入力で受け取ったログからデータを抽出するが、その際にデータを一時的に保持する変数を用意しておく。そして、タイムスタンプが更新される度に半角スペース区切りで全ての変数と改行を出力する。このように変数を用いる事で、通信の不具合でデータが部分的に飛んでしまっても、正常な数値データに変換出来るようにした。また、ログファイルから標準入力へ入力し、標準出力からデータファイルへ出力する為に、ツールとして簡単なバッチファイルを作成して用いた。また、最初の数行のデータはセンサが初期化されていない等の理由で正常な値が出ない為、手作業で削除した。

次に、Gnuplot でのグラフ化と計算について説明する。Gnuplot は、データをプロット (グラフ化) する為のソフトウェアである。ここでは Gnuplot の基本的な操作についての説明は省略する。Gnuplot では 4 種類のグラフを作成した。それぞれのグラフについて、グラフの内容と工夫した点を説明する。

1 つ目のグラフは、x 軸に時間、y 軸にローバーの現在地から目標地点までの距離をとったグラフである。このグラフの作成においては特に工夫はしておらず、using コマンドを用いて使用するデータの行を指定してプロットしているのみである。以下に実際に作成したグラフを示す。

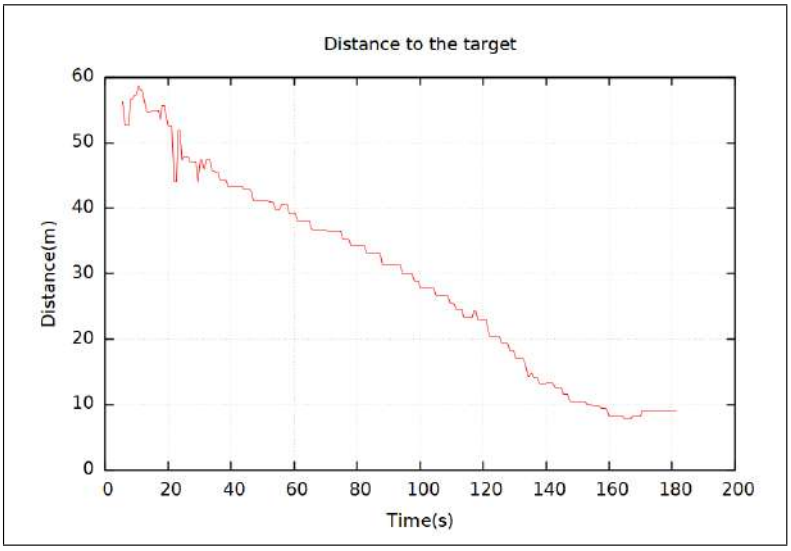

図 4.34　目標地点までの距離の時系列グラフ

2 つ目のグラフは、x 軸に時間、y 軸にローバーから見た目標地点の方向及びローバーの旋回方向をとったグラフである。このグラフの作成においては、元のデータでは目標地点の方向とローバー自身の方向を表す値のみであり、結果を 180 から-180 の値にしたかったので、using を用いて計算した結果をプロットした。また、旋回方向は-1,0,1 の値でデータ化しているがそのままだと変化が小さくすぎて見づらく、このデータにおいて重要なのは値の符号のみであるため、こちらも using を用いて拡大した。加えて、旋回の基準である ±45 の線が分かるように、ytics をセットし、grid を表示した。また、今回のように複数回のプロットを png 等で出力する場合、replot で直接出力する事は出来ないので、全てのグラフを plot し終えてから png 等を出力に設定して replot する必要がある。以下に実際に作成したグラフを示す。

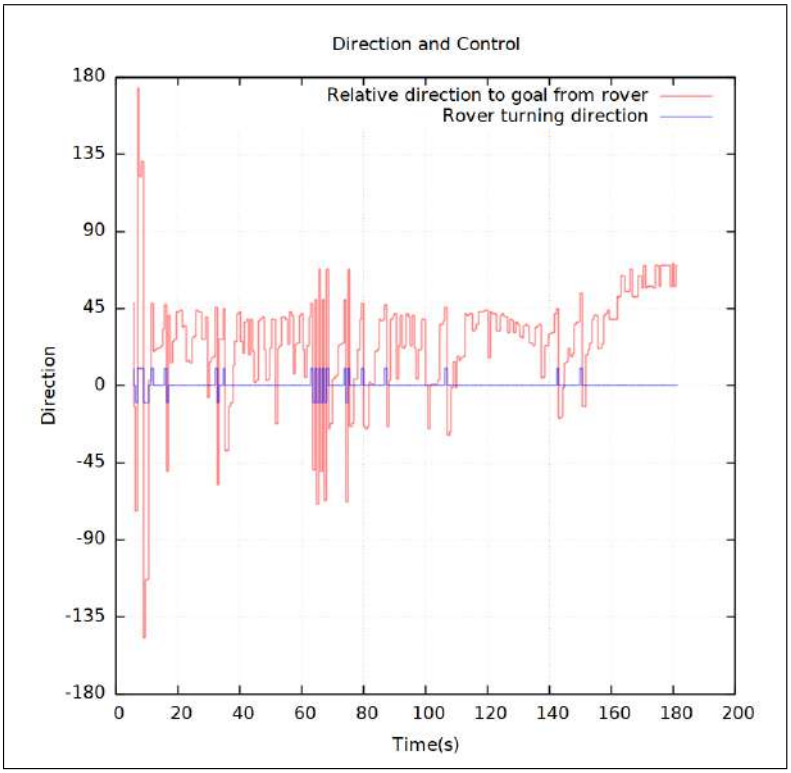

図 4.35　目標地点の相対方向と旋回方向の時系列グラフ

3 つ目のグラフは、x 軸に GPS の緯度、y 軸に GPS の経度、z 軸 (カラーバー) に時間をとったグラフである。このグラフの作成においては、lc palette のコマンドを用いる事で線の色を時間 (z 軸) によって変化させ、目標地点へ接近している事がわかるようにした。また、数値の表示が見やすくなるように、format xy コマンドを用いて表示桁数を指定し、xtics rotate コマンドで x 軸数値の表示方向を変更した。以下に実際に作成したグラフを示す。

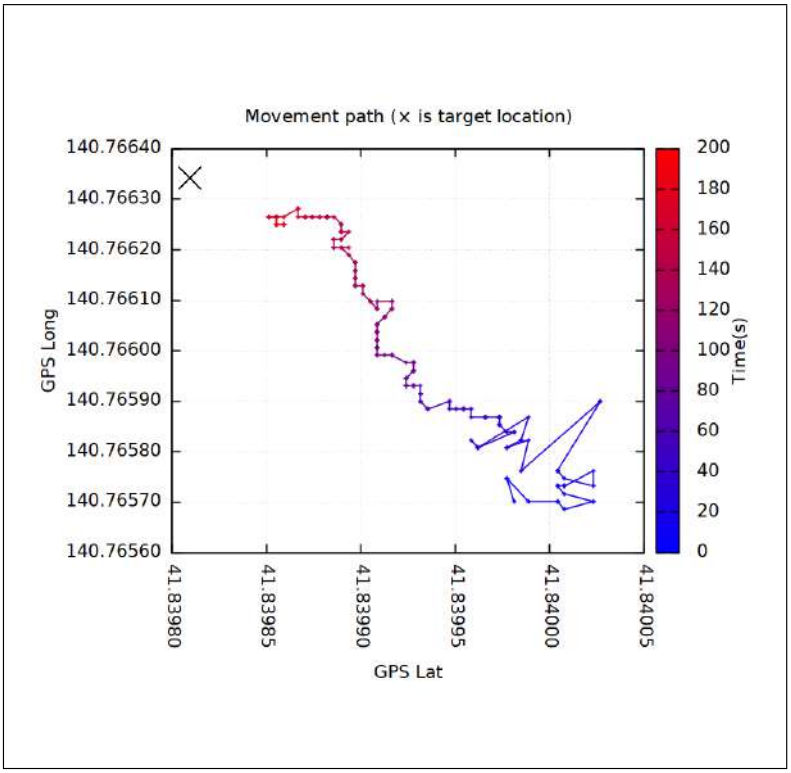

図 4.36　移動経路のグラフ

4 つ目のグラフは、1 つ目のグラフから、線形回帰によって目標地点への到達速度を求めたものである。Gnuplot で線形回帰をするには、まず近似させる関数を定義する。次に、fit コマンドを用いて、元データ、近似させる関数、変化させる変数を指定して近似を計算する。このグラフの作成においては、ゴール判定後はローバーは移動せず、到達速度の計算として不要であったため、every コマンドを用いて該当の時間のデータをスキップした。以下に実際に作成したグラフを示す。

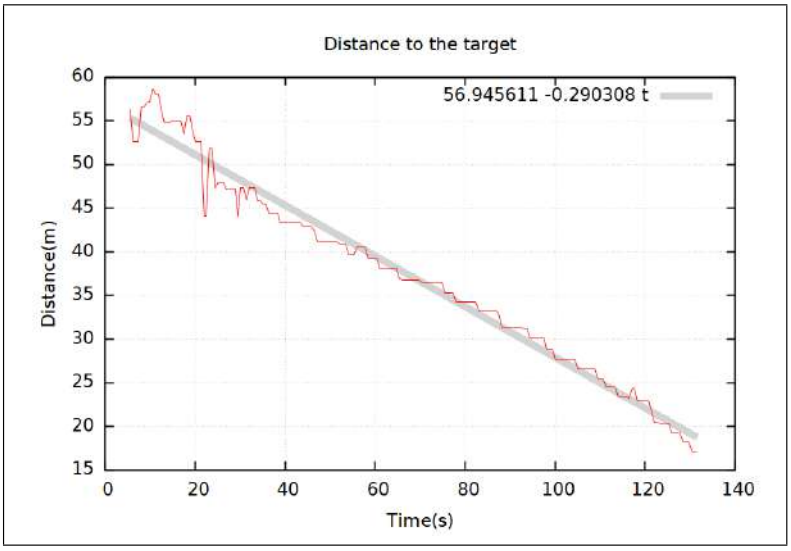

図 4.37　到達速度のグラフ

なお、変換プログラムのソースコード、ファイル入出力の為のバッチスクリプト、プロットの為の Gnuplot スクリプトは付録に示す。

（※文責: 小林浩也）

第 5 章　実験結果

この章では、本プロジェクトの製作した CanSat の実験結果について説明する。

5.1　実験詳細

実験場所は、北海道函館市赤川町にある笹流ダムで行った。日付は 11/6, 13, 20, 25 の計 4 日間、13:30 〜 15:30 の時間帯で行った。実験内容は、以下である。

ローバーにパラシュートあるいはグライダーを装着し、ダムの頂上（高度 25.3mm）から人力で機体を投射し、地面に落下させる。空中では、地面への軟着陸に備え、パラシュートの展開とグライダーによる空中移動をし、着陸後、ローバーはセンサ類の情報から自律走行をはじめ、あらかじめ決めておいた目標地点を目指し走行する。CanSat の打ち上げから目標地点到着まで、CanSat の位置情報などを PC へ通信し、実験記録はセンサを使って記録し、分析を行う。この一連の流れを本番実験とする。

図 5.1　笹流ダム

（※文責: 比留川満洋）

1 回目の実験

CanSat 本体のローバー部分が完成し野外で走行できるかというものであった。しかし、ローバーは微動だにせず、その原因を探した。原因はマイクロコンピュータへのコンパイルエラーの可能性が高いという結論に至った。また、この時点で 2 つのモータードライバのうち 1 つに不具合が発生し、片側のモーターが動かなくなるといった問題も発生した。こちらの原因はこれまでの中で電源を入れた時間が長かったことから、はんだ付けの際の過剰な熱によって弱っていた部分が通電の際の熱により内部が焼き切れてしまったのではないかと考えている。さらに総重量に対してモーターのトルク出力が追い付かず走行できない状態となっていた。第 1 回の実験結果からモータードライバの交換とモーターの変更が課題となった。

2 回目の実験

グライダーの投下を行う予定であったが急激な天候の悪化に加えてグライダーパーツの欠損により中止になった。その後、不具合の発生したモータードライバを交換し、新しく採用したモーターとの接続テストを行い無事に回転することを確認した。

3 回目の実験

新しく採用したモーターを組み込んだ新たなローバーを再び野外で走行させるものであった。また、ローバーにグライダーからの脱出用機構の一部であるサーボモーターを取り付け、動作を確認した。3 回目の実験でローバーは走行に成功した。しかし、GPS と 9 軸 IMU によって本来ならば目的地へ向かい走行するのだが前後運動やその場で回転し続けるなどの動作が見られた。これは目的地を見つけられていないような動きであった。加えて、一度動作を行うはずであったサーボモーターも稼働し続けていた。さらに、3 回目の実験ではグライダーの投下実験も行った。予定ではグライダーは滑空しながら落下するはずが鉛直に落下していた。これによって 3 回目の実験結果からローバーの GPS、9 軸 IMU とサーボモーターのプログラム修正およびグライダーの翼等の調整が課題となった。

最終実験

第 3 回目の実験と同じようにローバーの走行実験とグライダーの投下実験をおこなった。ローバーの走行実験において GPS と 9 軸 IMU の不具合を修正し、目標地点へと向かう様子が確認できた。しかし、GPS の精度の問題で約 10m 前後の誤差があり、実験時の目標地点は立ち入れないような林の中となっていた。その後、グライダーのコンテナにローバーを格納し、脱出機構が機能するかどうかを実験した。ローバーはコンテナから脱出することはできず、その後すぐにプログラムの確認を行った。そして、プログラムの修正後グライダーのコンテナに格納したままグライダーを投下するという最後の実験を行った。

投下されたグライダーは滑空しながらローバー内に積まれた運搬物を破損させることなく落下、一定時間後にローバーがコンテナから脱出し、目標へ走行を始める予定であった。しかし、グライダーは鉛直に落下し、一定時間後、ローバーはコンテナから出てくるようなことはなかった。確認してみると落下の衝撃でローバーが一部分解され、脱出、走行が不可能な状態となっていた。

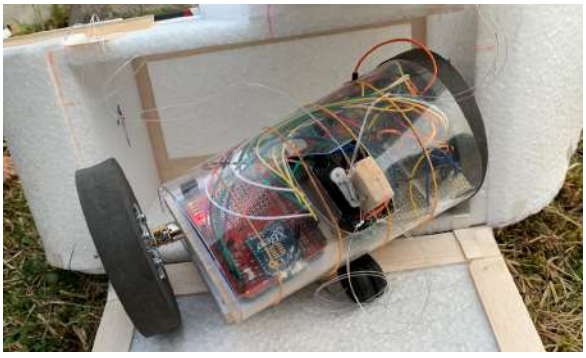

図 5.2　ローバー破損の様子

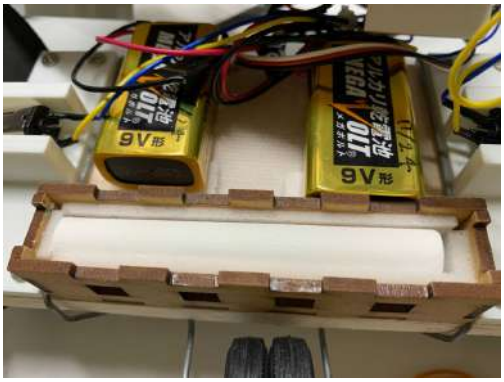

図 5.3　最終実験時の運搬物の様子

衝撃によって上下のパーツが外れ、モーターが飛び出ていることが分かる。こうならないために結束バンドなどで上下パーツをさらに固定することで解決できたのではないかと考えられる。一方でローバーに積まれていた運搬物に破損は見られず、衝撃を抑えることができていたことを確認できた。

以下はグライダーを投下した際の動画からグライダーの状態を分析したものである。

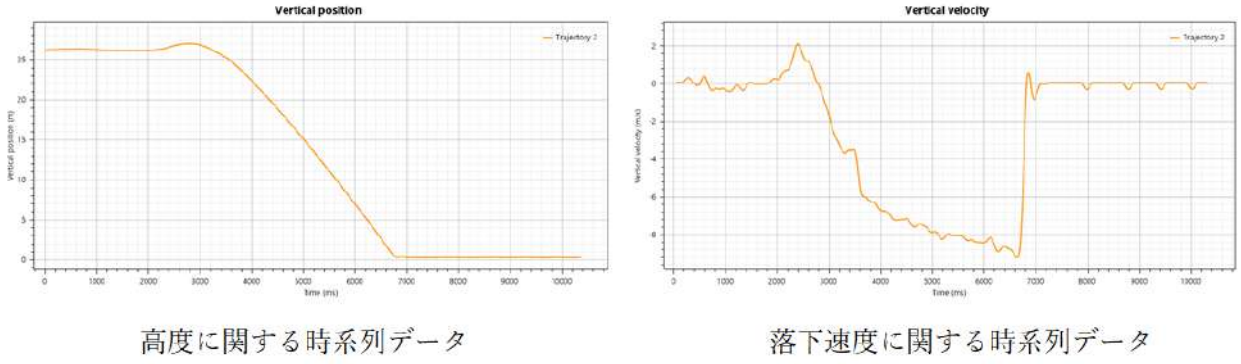

図 5.4　グライダーの落下運動における解析グラフ

グラフより、グライダーは滑らかに落下していることから安定した姿勢で落下したと考えられる。また、落下速度は最大 9m/s となり、想定よりも早く落下した。これはグライダーが滑空せずに真下に落下したのが原因だと考えられる。

大学走行実験

後日、平坦で障害物の少ない大学の中庭で走行実験を行い、GPS センサの誤差影響を小さくした正確なゴール地点の認識が行えているのか確認した。

以下はローバーが走行開始地点から目標地点まで走行した際の状態を分析したものである。

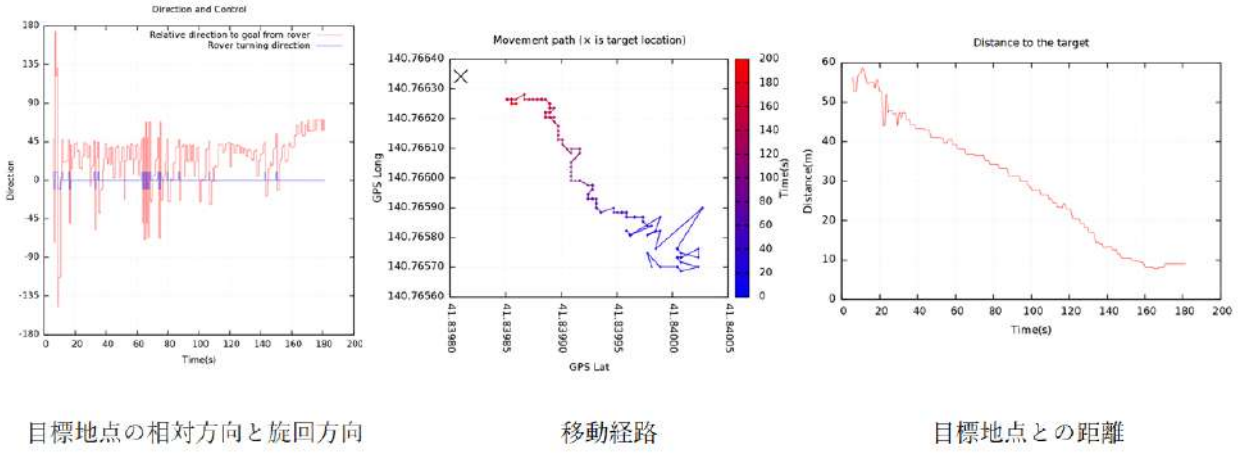

図 5.5　ローバーの走行に関する解析グラフ

グラフより、ローバーは設定した開始地点から目標地点まで約 50m 走行し、目標地点から 8.1m 手前で停止した。結果、GPS センサから約 10m 前後の誤差があるものの目標地点を確認しながら自律移動し、自ら停止行動を行うことができた。

以上より、成果として当初目標にしていたグライダーを落下させ、そこから目標地点まで走行させる、という一連の行動をローバーにさせることは不可能であったが、もう 1 つの目標である運搬物を破損させないという目標は達成できた。また、ローバー単体では誤差があるものの目標地点までの走行に成功した。

（※文責: 村上拓馬）

5.2 解決手段と評価

1 回目の実験ではプログラムのコンパイルエラーとモータードライバの不具合、そしてモーターの交換が課題であった。この解決手段としてコンパイルエラーについては、マイクロコンピュータと XBee シールドによるセンサ同士の接続不備が原因であった。そこで、センサ同士を外し、マイクロコンピュータのみでコンパイルすることでエラー解決した。こちらは初めての起動実験であったこともあり、実験前にある程度のコンパイルチェックを行うべきであったと考える。

モータードライバについてはショートが起こっていないことを確認し、基板に直接はんだ付けしているモータードライバを破壊して新しいものと交換した。こちらははんだ付けの時点で直接はんだ付けしていたため破壊して摘出せざるをえなかった。破壊して取り出したことで不具合の原因が分からなくなってしまったため良い解決手段ではないと考えられる。基板に直接はんだ付けするのではなくソケットを利用することで破壊せずにモータードライバを交換する方法が最善であったと考えられる。

モーターについてはギアボックス型モーターにすることで軽量化と回転数は落ちるもののトルク出力の高上を図った。こちらは妥当な解決手段ではあったが最初のモーターを選ぶ時点で重量やトルク出力からローバーが走行するかを算出した上で採用を行った方が良かったのではないかと考える。

第 3 回目の実験では 9 軸 IMU と GPS、サーボモーターについてのプログラムの修正とグライダーの翼などの調整が課題であった。9 軸 IMU のプログラムについては機体自身の向きを取得するための関数である GetDirection 関数で利用していたサンプルコード中に含まれる値の修正が誤作動していると判明したため修正した。GPS のプログラムについては現在地と目的地の情報から、目的地への方角と距離を計算する関数である WayInfo CalcWayInfo 関数が正常に機能しなかった。そのため GPS を取得する Tiny GPS++ というライブラリを用いた GetGPS 関数との繋がりを再確認して修正した。

サーボモーターのプログラムについてサーボモーターを動かす ServoOpen 関数内で利用した Servo.h というライブラリが GPS データをシリアル通信する際に用いる SoftwareSerial.h と競合するため正常に動作しなかった。そのため、上記 2 つのライブラリの役割を持つ NeoSWSerial.h を宣言することで修正した。この問題は実際に起動して初めて確認できたことであったため、問題の確認後すぐに修正した。

グライダーについては重心が後方向であったことに加えコンテナの形状がブレーキとなっていた。そのため重心を前方向へするため先尾翼をやや後ろに移動させることでコンテナ後方を空気が

流れやすくして滑空しやすくした。グライダーの解決手段についても計算だけで飛行状態を予測するのは難しく投下実験を踏まえなければわからない部分がある。よって、実験後問題点をすぐに修正できた。

（※文責: 村上拓馬）

第 6 章　発表会アンケート結果

この章では、中間と最終成果の二回に分けて行われた発表会の概要と、発表後行ったアンケートの結果について考察する。

（※文責: 小林浩也）

6.1　中間発表

6.1.1　発表会とアンケートについて

中間発表会は 2020 年 7 月 17 日にオンライン上で開催された。中間発表では、このプロジェクトの概要の説明と、当時の設計案や予定についての説明を Web ページ [18] と Zoom ミーティングを用いて行った。アンケートは発表会当日から開始し、2020 年 7 月 19 日に集計を行った。

6.1.2　結果と考察

アンケートの結果、有効な投票は 17 件であった。詳細を以下に示す。

まず、発表技術についての評価のグラフを図 6.1 に示す。平均の評価は 7.9 であった。また、発表技術についてのコメントとしては「聞き取りやすかった」「Web サイトが見やすかった」という意見が多かった。平均評価とコメントから、発表技術については大きな問題もなく良好であったと考えた。

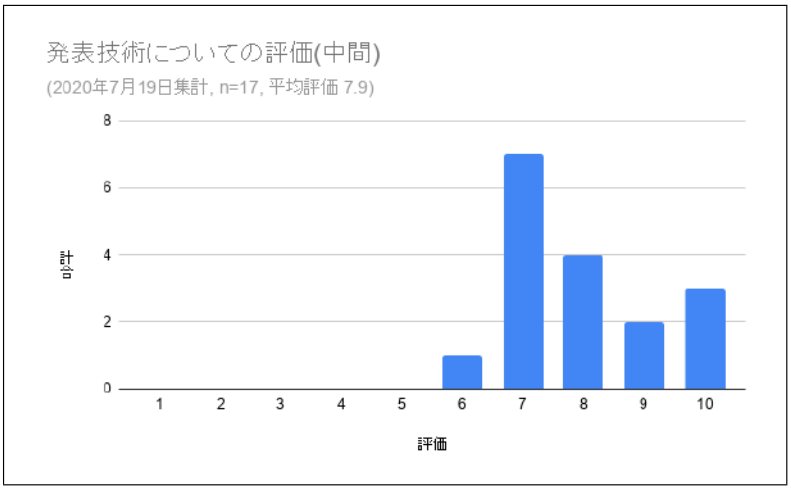

図 6.1　発表技術についての評価 (中間)

次に、プロジェクトの目標設定とスケジュールについての評価のグラフを図 6.2 に示す。平均の評価は 8.0 であった。

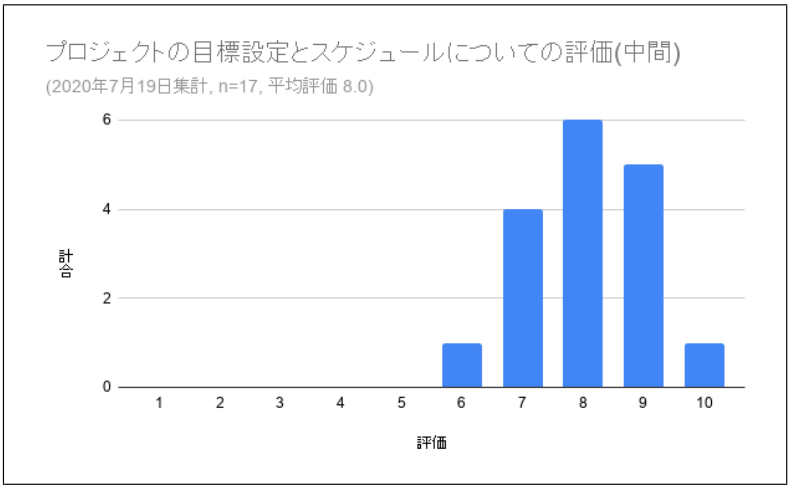

図 6.2　プロジェクトの目標設定とスケジュールについての評価 (中間)

次に、課題解決案についての評価のグラフを図 6.3 に示す。なお、平均の評価は 7.8 であった。

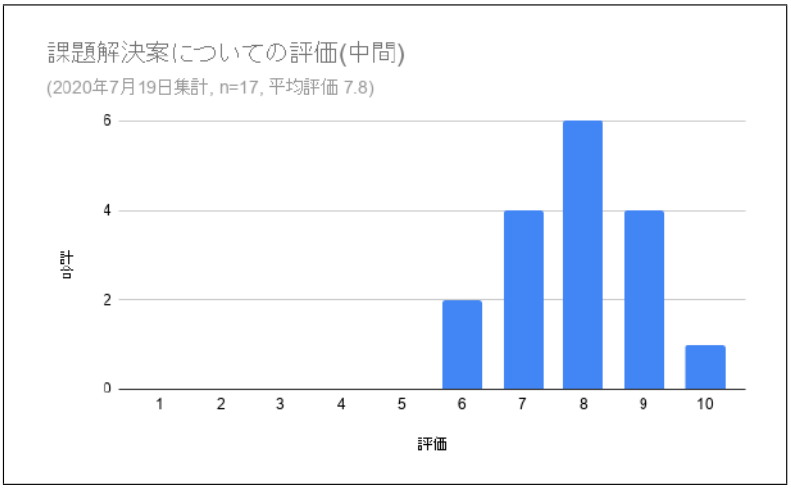

図 6.3　課題解決案についての評価 (中間)

発表内容についてのコメントとしては「詳細まで考えられていてよかった」「コロナの影響もあるので計画通りにはいかないと思った」という意見が多かった。平均評価とコメントから、計画の変更については考慮が必要であると考え、後期では実際に計画を一部変更した。

（※文責: 小林浩也）

6.2　最終成果発表

6.2.1　発表会とアンケートについて

最終成果発表会は 2020 年 12 月 4 日にオンライン上で開催された。最終成果発表では、このプロジェクトを通して得られた技術や成果物、経験についての説明を Web ページ [19] と Zoom ミーティングを用いて行った。アンケートは発表会当日から開始し、2020 年 12 月 18 日に集計を行った。

6.2.2　結果と考察

アンケートの結果、有効な投票は 42 件であった。詳細を以下に示す。

まず、発表技術についての評価のグラフを図 6.4 に示す。平均の評価は 7.6 であった。また、発表技術についてのコメントとしては「問題点と課題が明確であった」「質疑応答が円滑、論理的であった」「Web サイトの情報量が多くどう見ればいいのかわかりづらかった」という意見が多かった。平均評価とコメントから、口頭での発表技術については大きな問題もなく良好であったが、Web ページの分量が中間の時から大幅に増えたため、ページの見方を示すなどの配慮が必要だったと考えた。

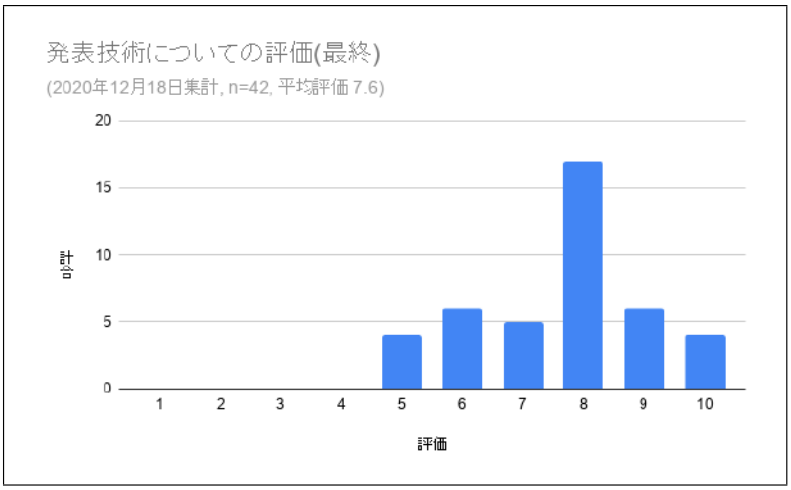

図 6.4　発表技術についての評価 (最終)

次に、発表内容についての評価のグラフを図 6.5 に示す。平均の評価は 7.8 であった。発表内容についてのコメントとしては「コロナの影響下での開発としては良い結果だった」「有益な経験が得られたことが良く分かった」「今後の展望に期待する」という意見が多かった。平均評価とコメントから、当初掲げていた目標の全てを達成する事は出来なかったが、外部から見ても十分に良い結果を残せたと考える。

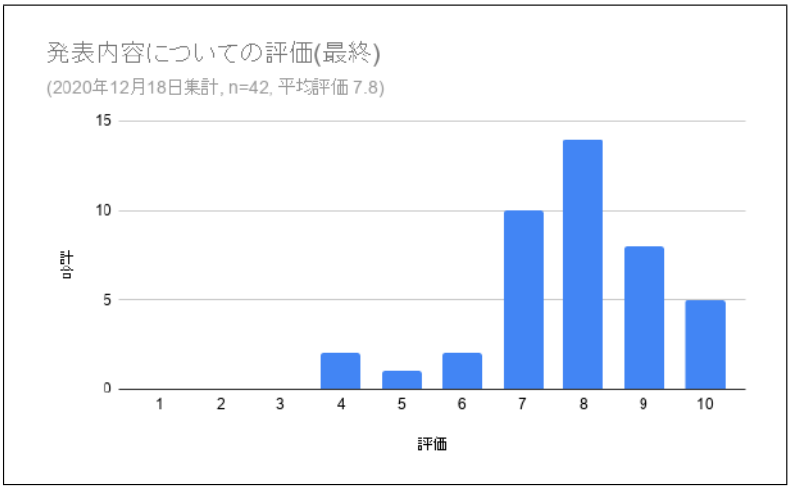

図 6.5　発表内容についての評価 (最終)

（※文責: 小林浩也）

第 7 章　まとめ

この章では、まずプロジェクト全体の成果について簡単に示し、次に今後の展望について説明する。最後に、プロジェクトにおいて自分がどのような役割を担ったと考えるか、各メンバーが振り返り考察する。

7.1　プロジェクトの成果

プロジェクトの成果として、まず活動を通して得られた知識について説明する。活動を通して、設計及び作成する上で一般的に用いられるソフトウェア (EAGLE や Blender 等) や道具 (3D プリンターや基板プリンター等)、材料 (3D プリンター用 ABS フィラメントや EPP 等)、電子部品、ハードウェアを利用するプログラム等についての知識を得ることが出来た。

次に、活動を通して得られた成果物について説明する。前期においては、基本設計と設計思想を固め、後期においては折り畳み機能を持たないグライダーとローバーを作成する事が出来た。

次に、実験結果について説明する。実験においてはグライダーが前進しない、ローバーのサーボが動作しない場合がある等の問題はあったが、着陸時の衝撃からチョークを守る事、目標地点の方向へローバーを直進させて 10m 以内に到達する事は出来た。

最後に、活動を通して得られた経験について説明する。1 つ目に、トラブルや問題解決を通してメンバー間のコミュニケーションの重要性を再認識するとともに、その方法について模索する事が出来た。2 つ目に、設計通りにはいかないハードウェア開発の難しさ、面白さを経験することが出来た。3 つ目に、感染症対策下でのハードウェア開発の場所や時間の厳しさを経験することが出来た。4 つ目に、グループワークでの開発を通して、他のメンバーを尊重・信頼する事の実利的な重要性を知る事が出来た。

（※文責: 小林浩也）

7.2　今後の展望

今後の展望として、まずグライダーについては、ローバーを乗せた状態で前進滑空するように空力設計を見直す事と、今回実装を見送った展開機構を実装する事があげられる。次にローバーについては、本体構造を見直して衝撃で分解しないようにする事、サーボの動作条件を見直して正しく動作するようにする事、ハードウェア・ソフトウェア両方を見直して位置推定の精度を上げる事、障害物を避けられるアルゴリズムにする事等があげられる。プロジェクトとしては、今回の機体をベースに前述のような改良をして、公式な CanSat 競技に参加したいと考えている。また、今回得られた経験や知識を各メンバーが他の場所で活かしていく事も、このプロジェクトにおける重要な今後の展望である。

（※文責: 小林浩也）

7.3　プロジェクトにおける自分の役割

7.3.1　佐々木光流

B グループにおいてローバーの構造系を担当し、収納箱やモーター固定具を設計、作製した。また、プログラム作成も一部を担当し、ローバーが地上を走行する際に自身の方角を取得するプログラムを作成した。会議においては、自分が気になった点はできるだけグループ内で共有し、自分もほかのメンバーもなるべくあいまいな部分を残さないようにした。また、前期の活動では、コミュニケーション不足でメンバーと衝突することがあったため、後期ではそれが起こらないようにしっかりコミュニケーションをとり、意見が分かれたときは相手の意見を一度自分の中で吟味してから意見するように努めた。

（※文責: 佐々木光流）

7.3.2　永井彬博

前期では B グループの書記を担当した。また、全体ミーティングの書記も A グループの書記と交互に行った。メンバーの意見や議論を振り返って見返すときにも、内容がしっかり伝わるような議事録を作成することに努めた。専門的分野に関しては B グループのプログラム作成の担当となった。他にもセンサ類の動作確認や構造系の案などを積極的に出した。
後期では、地上物の機体の製作とプログラムである。地上物の機体は軽量に設計し、できる限り簡単で、製作やメンテナンスに時間をとられないような材料を使用して効率よく製作しようと考えた。機体のモーターを支える土台や、3 輪を設置するための設計、電子基板や、バッテリーなどの機体に設置しなければならないものの設置案などを考案した。プログラムでは走行するための判断基準や、GPS の情報を取得するプログラムを作成した。他にも大まかな機体の動き、フローチャートの作成を行った。

（※文責: 永井彬博）

7.3.3　比留川満洋

プロジェクトリーダーとして、全体の進捗管理、スケジュール管理、外部交渉、担当教員との連絡や材料発注など行った。作業時間の短縮によるスケジュール変更、それに伴い製作物の変更点の取捨選択、実験方法の変更などメンバーに議題を促し、プロジェクトの進行に関して臨機応変に対応しようと心掛けた。また、B グループのローバー作製における通信系とログ管理、モーターの動作プログラム作成、EEPROM と呼ばれる記憶メモリのプログラム作成を担当した。通信ではセンサと PC の設定を行い、PC から CanSat の GPS データやローバーの向いている方角角度、目標地点までの距離などをリアルタイムで数値的に可視化できるようにした。可視化したデータは制御履歴として、実験後、走行の分析を行えるようにログとしてしっかりと残した。しかし、EEPROM による記録データの取得には失敗し、問題が残ったままである。また、ローバーの初期に利用した

モーターの土台作製とグライダーに取り付けるコンテナ製作の手伝いを行った。

（※文責: 比留川満洋）

7.3.4　村上拓馬

前期と比べてプロジェクト対して大きく貢献できた。ミーティング中も積極的に意見を出し、疑問に思った部分はすぐに質問した。また、プロジェクトメンバーとのコミュニケーションを大切にしてムードメーカーの役割もこなすことができた。認識の確認を行うことでミーティング内容を復習を促す役目を行えた。実験の際もいつも参加し、基板の配線接続の他にも写真撮影や他の実験参加メンバーのサポートを行った。最もプロジェクトに貢献できたと思う部分は基板製作である。基板と回路図の設計や基板への電子部品のはんだ付けを行った。また、電子部品の動作確認や基板のメンテナンスを行った。発表の際も詰まることなく発表することができた。

（※文責: 村上拓馬）

7.3.5　小林浩也

担当としては B グループのグループリーダーと B グループの飛行系とプログラム系と解析を主に担当した。グループリーダーとしては、自身の幅広い知識を活かして全体のスケジュールを把握・管理し、他メンバーの考えを取り入れたうえで自身の考えで最終的な判断を下した。飛行系担当としては、グライダーの基本設計から作成を行った。プログラム系としては、主にグループワークで問題となりそうな基本の構造やコーディングルールの策定と、バグの修正を行った。解析としては、グライダーの動画解析とローバーのログ解析を行った。プロジェクトメンバーとしては、様々な成果について批判的視点でチェックし、積極的に意見をした。またその時、問題点と改善案を極力明確にして意見するように心がけた。前期では、考えを十分に伝えることが出来ず、反感を買ってしまう事が多かったが、後期では反省を生かし、他メンバーの意見を尊重するように留意する事で、特にトラブルなく進めることが出来た。

（※文責: 小林浩也）

付録 A　作製したプログラム

以下に示すプログラムは Google drive でも閲覧することができる。
URL：https://drive.google.com/drive/folders/1TcqmnKrk3qG7drcyAPgbWIQxIcWnfqrR?usp=sharing

自律移動プログラムのソースコード (Arduino)

```
/*――――――――CanSat2020_B.ino――――――――*/
#include "_Cansat2020_B.h"
#include <TinyGPS++.h>
#include <NeoSWSerial.h>

#define DELAY_TIME 100

double new_alt = 0.0;
int SetEEPROM_next = 0; //現在のアドレスの位置
TinyGPSPlus gps;
NeoSWSerial mySerial(10, 11); // RX, TX
GPSData targetData = {41.8398094177, 140.7663421630, 103.900, 0.0};
Servo myservo;
GPSData nowData = {0.0, 0.0, 0.0, 0.0};
double direct = 0.0;
bool endTask = false;
unsigned long lastUpdateTime = 0;
unsigned long setXBeeCount = 0;
bool ServoOpen_value = false;
WayInfo info;
String eeprom_str = "";

void setup() {
  Serial.begin(9600);
  GetGPS_setup();
  SetMotor_setup();
  GetDirection_setup();
  // put your setup code here, to run once:

  myservo.attach(13); //myservo.attach(13,500,2400);
  Serial.println("#START");
}

void loop() {

  GetGPS();

  if (lastUpdateTime + DELAY_TIME < millis()) {
    GetDirection();
    info = CalcWayInfo(nowData, targetData);


    if (!endTask) {
      // put your main code here, to run repeatedly:
      if (EscapeContainer(nowData)) {
        GoToTarget(
          info,
          direct
        );
      }
    }

    ServoOpen();

    if (info.distance < 9.0) {
      //終了判定
```

```
      endTask = true;
      SetMotor(0, 0);
      eeprom_str = "Gaol!!; ";
    }

    if (setXBeeCount == 0) {
      SetXBee();
      //SetEEPROM(eeprom_str);
    }

    setXBeeCount++;
    if (setXBeeCount > 2.5) {
      setXBeeCount = 0;
    }

    lastUpdateTime = millis();
  }
}
```

```
/*----------CalcWayInfo.ino----------*/
#include "_Cansat2020_B.h"
#include <TinyGPS++.h>
#include <NeoSWSerial.h>

//現在地と目的地の情報から、目的地への方角と距離を計算する
WayInfo CalcWayInfo(GPSData nowData, GPSData targetData) {

  WayInfo Info = {0.0, 0.0};

  Info.distance =
    TinyGPSPlus::distanceBetween(
      nowData.lat,
      nowData.lon,
      targetData.lat,
      targetData.lon);

  Info.direct =
    TinyGPSPlus::courseTo(
      nowData.lat,
      nowData.lon,
      targetData.lat,
      targetData.lon);

  return Info;
}
```

```
/*EscapeContainer.ino*/
#include "_Cansat2020_B.h"
#include <math.h>


#define EscapeContainer_MPS_AS_STOP 5 //速度がこの値以下になった時、着地と判定
#define EscapeContainer_ALT_AS_STOP 1000000 //目的地との高度差がこの値以下になった時
    、着地と判定
#define EscapeContainer_TIME_AS_STOP 210000 //着地と判定するための時間(ms)
#define EscapeContainer_NICHROME_TIME 5000 //サーボを動かしてからモータが動き出すまで
    の時間
#define EscapeContainer_ESCAPE_TIME (EscapeContainer_NICHROME_TIME + 4000) //脱出時に
    前進する時間
#define EscapeContainer_ROTATE_TIME (EscapeContainer_ESCAPE_TIME + 10000) //脱出後に
    回転（地磁気センサの初期化）する時間
```

```
int EscapeContainer_stopTime = 0;
unsigned long EscapeContainer_stoppingTime = 0;


//条件を満たした時コンテナから脱出し、脱出が完了すればtrueを返す。
bool EscapeContainer(GPSData nowData) {
  return true;
  eeprom_str = "ESCAPE_CONTAINER : ";

  //停止判定と脱出のモード切替
  if (EscapeContainer_stoppingTime == 0) {
    eeprom_str.concat("Wait; ");
    ServoOpen_value = false;
    //停止判定
    if (fabs(targetData.alt - nowData.alt) <= EscapeContainer_ALT_AS_STOP && fabs(
        nowData.mps) <= EscapeContainer_MPS_AS_STOP) {
    //停止継続カウントアップ
EscapeContainer_stopTime += DELAY_TIME;
//停止継続判定
      if (EscapeContainer_stopTime >= EscapeContainer_TIME_AS_STOP) {

//停止時刻保持
        EscapeContainer_stoppingTime = millis();
      }
    } else {

      //停止継続カウントリセット
      EscapeContainer_stopTime = 0;
    }
  } else {
    eeprom_str.concat("Escape; ");
    ServoOpen_value = true;

    if (millis() <= EscapeContainer_stoppingTime + EscapeContainer_NICHROME_TIME) {
    //サーボ動作まち

    } else if (millis() <= EscapeContainer_stoppingTime + EscapeContainer_ESCAPE_TIME
        ) {
    //前進
    SetMotor(-255, -255);

    } else if (millis() <= EscapeContainer_stoppingTime + EscapeContainer_ROTATE_TIME
        ) {
    //回転
    SetMotor(255, -255);

    } else {

//脱出完了
      return true;
    }
  }

  //脱出前または脱出中リザルト
  return false;
}
```

```
/*----------GetDirection.ino----------*/
#include "_Cansat2020_B.h"
#include <Wire.h>
#include <math.h>

// BMX055 加速度センサのI2Cアドレス
#define Addr_Accl 0x19  // (JP1,JP2,JP3 = Openの時)
// BMX055 ジャイロセンサのI2Cアドレス
#define Addr_Gyro 0x69  // (JP1,JP2,JP3 = Openの時)
// BMX055 磁気センサのI2Cアドレス
#define Addr_Mag 0x13   // (JP1,JP2,JP3 = Openの時)
```

```
#define Two_PI 6.28318530718

// センサーの値を保存するグローバル関数
int xMag = 0;
int yMag = 0;

//磁気センサーの最大最小
int mag_x_min = INT_MAX;
int mag_x_max = INT_MIN;
int mag_y_min = INT_MAX;
int mag_y_max = INT_MIN;

void GetDirection_setup()  //BMXの初期化
{
  Wire.begin();
  GetDirection_Init();
  delay(300);
}

//============================================================================//

void GetDirection_Init()
{

  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x4B);  // Select Mag register
  Wire.write(0x83);  // Soft reset
  Wire.endTransmission();
  delay(100);
  //------------------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x4B);  // Select Mag register
  Wire.write(0x01);  // Soft reset
  Wire.endTransmission();
  delay(100);
  //------------------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x4C);  // Select Mag register
  Wire.write(0x00);  // Normal Mode, ODR = 10 Hz
  Wire.endTransmission();
  //------------------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x4E);  // Select Mag register
  Wire.write(0x84);  // X, Y, Z-Axis enabled
  Wire.endTransmission();
  //------------------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x51);  // Select Mag register
  Wire.write(0x04);  // No. of Repetitions for X-Y Axis = 9
  Wire.endTransmission();
  //------------------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x52);  // Select Mag register
  Wire.write(0x16);  // No. of Repetitions for Z-Axis = 15
  Wire.endTransmission();
}

//現在の機体の方角を取得
void GetDirection() {
  direct = 0.0;
  int data[8];
  for (int i = 0; i < 8; i++)
  {
    Wire.beginTransmission(Addr_Mag);
    Wire.write((0x42 + i));     // Select data register
    Wire.endTransmission();
    Wire.requestFrom(Addr_Mag, 1);     // Request 1 byte of data
    // Read 6 bytes of data
    // xMag lsb, xMag msb, yMag lsb, yMag msb, zMag lsb, zMag msb
    if (Wire.available() == 1)
      data[i] = Wire.read();
  }
  // Convert the data
```

```
  xMag = ((data[1] << 8) | (data[0] >> 3));
  yMag = ((data[3] << 8) | (data[2] >> 3));

  //最大最小値更新
  if (xMag < mag_x_min) {
    mag_x_min = xMag;
  }
  if (xMag > mag_x_max) {
    mag_x_max = xMag;
  }
  if (yMag < mag_y_min) {
    mag_y_min = yMag;
  }
  if (yMag > mag_y_max) {
    mag_y_max = yMag;
  }

  int mid_x = (mag_x_max + mag_x_min) / 2;  //中点x
  int mid_y = (mag_y_max + mag_y_min) / 2;  //中点y

  xMag -= mid_x;
  yMag -= mid_y;
  direct = atan2((double)yMag, (double)xMag) * (180.0 / M_PI) + 180;
}
```

```
/*----------GetGPS.ino----------*/

#include "_Cansat2020_B.h"
#include <TinyGPS++.h>
#include <NeoSWSerial.h>
int i = 0;


void GetGPS_setup() {
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  // set the data rate for the SoftwareSerial port
  mySerial.begin(9600);

}

/*現在のGPS情報を取得*/
void GetGPS() {
  while (mySerial.available() > 0) {
    char c = mySerial.read();

    //Serial.print(c);
    gps.encode(c);
    if (gps.location.isUpdated())
    {
      nowData.lat = gps.location.lat();
      nowData.lon = gps.location.lng();
    }
    if (gps.altitude.isUpdated() && gps.altitude.meters() != 0.0) {
      nowData.alt = gps.altitude.meters();
    }

    nowData.mps = gps.speed.mps();
  }
}
```

```
/*----------GoToTarget.ino----------*/
#include "_Cansat2020_B.h"
#include <math.h>

#define TwoPI 360.0 //

/*道のりの情報と機体の向きの情報を用いて、目的地へ接近する*/
/*wayにCalcWayInfoの目標地点までの距離と目標地点の向きのデータが入ってる
  directに現在の自分の向いている方向のデータが入っている*/
/*
  ゴール判定は10m圏内に入れば終了
  ----------角度についての注意点----------
  知りたい角度：自分の向いている向き - 目標地点の向き
  値の範囲0~360
  知りたい角度の範囲：360~720
  余りを用いる前に360足すことでマイナスがつかない
  ----------

  ----------アルゴリズム----------
  動作1：自分の向いている方向 - 目標地点の方向 = 315~360 or 0~45 の時直進
  動作2：自分の向いている方向 - 目標地点の方向 = 45~135 の時左のタイヤを動かす
  動作3：自分の向いている方向 - 目標地点の方向 = 135~225 の時どちらかのタイヤを90度回
      す or 後退
  動作4：自分の向いている方向 - 目標地点の方向 = 225~315 の時右のタイヤを動かす
  ----------
*/


void GoToTarget(WayInfo way, double direct) {
  double GoToTaget_direct = fmod(way.direct - direct + TwoPI, 360);
  eeprom_str = "GOTO_TARGET : ";
  if ((GoToTaget_direct <= 45 && GoToTaget_direct >= 0) || (GoToTaget_direct <= 360
      && GoToTaget_direct > 315)) {
    SetMotor(255, 255); //前進
    eeprom_str += ("Forward; ");
  }
  else if (GoToTaget_direct <= 180) {
    SetMotor(255, -255);//左回転
    eeprom_str += ("Left; ");
  }
  else if (GoToTaget_direct > 180) {
    SetMotor(-255, 255);//右回転
    eeprom_str += ("Right; ");
  }
}
```

```
/*----------ServoOpen.ino----------*/
#include "_Cansat2020_B.h"

void ServoOpen() {
  //pinMode(13, OUTPUT);     //LEDを接続した13番ピンを出力用に設定する
  if (ServoOpen_value == true) {
    myservo.write(0);
  } else {
    myservo.write(155);
  }
}
```

```
/*----------SetEEPROM.ino----------*/
#include "_Cansat2020_B.h"
#include <EEPROM.h>
#include <stdlib.h>
#include <string.h>
```

```
/*
  書き込み
  →読みだしのfor文をコメントアウトする
  読み出し
  →書き込みのfor文をコメントアウトする
*/


void SetEEPROM(String data) {

  byte buf[8192];
  int len = strlen(data.c_str());//文字列の長さ
  // int len = data.length();
  data.getBytes(buf, len); //文字列をbyte型の配列(buf)にコピーします。
       lenはbufのサイズです(int)

  //書き込み
  //(動的メモリ)bufを文字列の長さに合わせて動的に確保
  for (int i = 0; i < len; i++) {
    EEPROM.write(i + SetEEPROM_next, buf[i]); //アドレスにデータを書き込み
  }

  //読みだし
  //  for (int i = 0; i < len; i++) {
  //    EEPROM.read(buf[i]); //アドレスのデータを読み込む
  //    Serial.println(buf[i]);
  //  }

  SetEEPROM_next = SetEEPROM_next + len;
}
```

```
/*----------SetMotor.ino----------*/

#include "_Cansat2020_B.h"

int PIN_R_IN1  = 3;
int PIN_R_IN2  = 4;
int PIN_R_VREF = 5; // PWM  ここがデジタルピン 右モーター

int PIN_L_IN1  = 7;
int PIN_L_IN2  = 8;
int PIN_L_VREF = 9; //  PWM ここがデジタルピン  左モーター

void SetMotor_setup() {
  pinMode(PIN_R_IN1, OUTPUT);
  pinMode(PIN_R_IN2, OUTPUT);
  pinMode(PIN_L_IN1, OUTPUT);
  pinMode(PIN_L_IN2, OUTPUT);

  // モーターの回転速度を最大にする
  analogWrite(PIN_R_VREF, 255);
  analogWrite(PIN_L_VREF, 255);
}

void SetMotor(double l, double r) { //lは左のモーターの速度、
  analogWrite(PIN_L_VREF, (int)(abs(l)));
  analogWrite(PIN_R_VREF, (int)(abs(r)));

  //Lの回転状態
  if (l > 0) { //前進
    digitalWrite(PIN_L_IN1, LOW);
    digitalWrite(PIN_L_IN2, HIGH);
  }
  else if (l < 0) { //後進
    digitalWrite(PIN_L_IN1, HIGH);
    digitalWrite(PIN_L_IN2, LOW);
  }
  else { //止まる
```

```
    digitalWrite(PIN_L_IN1, LOW);
    digitalWrite(PIN_L_IN2, LOW);
  }

  //Rの回転状態
  if (r > 0) { //前進
    digitalWrite(PIN_R_IN1, HIGH);
    digitalWrite(PIN_R_IN2, LOW);
  }
  else if (r < 0) {
    //後進
    digitalWrite(PIN_R_IN1, LOW);
    digitalWrite(PIN_R_IN2, HIGH);
  }
  else { //止まる
    digitalWrite(PIN_R_IN1, LOW);
    digitalWrite(PIN_R_IN2, LOW);
  }
}
```

```
/*----------SetXBee.ino----------*/
#include "_Cansat2020_B.h"

//XBeeに data の文字列を書き込む
void SetXBee() {

  Serial.print("TIME="); Serial.println(millis());
  Serial.print("GPS_LAT="); Serial.println(nowData.lat, 10);
  Serial.print("GPS_LONG="); Serial.println(nowData.lon, 10);
  Serial.print("GPS_ALT="); Serial.println(nowData.alt, 10);
  Serial.print("GPS_NEW_ALT="); Serial.println(new_alt, 10);
  Serial.print("GPS_M/S="); Serial.println(nowData.mps);
  Serial.print("MAG_DIRECTION="); Serial.println(direct);

  Serial.print("TARGET_LAT="); Serial.println(nowData.lat - targetData.lat, 10);
  Serial.print("TARGET_LONG="); Serial.println(nowData.lon - targetData.lon, 10);
  Serial.print("TARGET_ALT="); Serial.println(nowData.alt - targetData.alt, 10);
  Serial.print("TARGET_DIR="); Serial.println(info.direct);
  Serial.print("TARGET_DIST="); Serial.println(info.distance);
  String es = eeprom_str;
  es.trim();
  Serial.print("EEPROM="); Serial.println(es);
  Serial.print("EEPROM_INDEX="); Serial.println(SetEEPROM_next);
  Serial.println("#----------");
}
```

```
/*----------_CanSat2020_B.h----------*/

#ifndef CANSAT_2020_B_H
#define CANSAT_2020_B_H
#include "Servo.h"
#include "mainStructs.h"

WayInfo CalcWayInfo(GPSData nowData, GPSData targetData);
bool EscapeContainer(GPSData nowData);
void GetDirection();
void GetGPS();
void GoToTarget(WayInfo way, double direct);
void ServoOpen();
void SetEEPROM(String data);
void SetMotor(double l, double r);
void SetXBee();

#endif /* CANSAT_2020_B_H */
```

```
/*――――――――――mainStructs.h――――――――――*/

#ifndef MAIN_STRUCT_H
#define MAIN_STRUCT_H

//ＧＰＳの緯度経度高度を表す構造体
struct GPSData {
  double lat;
  double lon;
  double alt;
  double mps;
};

//目的地への情報を表す構造体
struct WayInfo {
  double distance;
  double direct;
};

#endif /* MAIN_STRUCT_H */
```

ログ分析プログラムのソースコード (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CanSat_LogReder
{
    class Program
    {
        static double TIME = 0;
        static double GPS_LAT = 0.0000000000;
        static double GPS_LONG = 0.0000000000;
        static double GPS_MS =0.00;
        static double MAG_DIRECTION = 180.00;
        static double TARGET_LAT = -41.8237609863;
        static double TARGET_LONG = -140.7400512695;
        static double TARGET_DIR = 35.27;
        static double TARGET_DIST = 13929917.00;
        static double EEPROM = 0;
        static double EEPROM2 = 0;

        static void Main(string[] args)
        {
            string line = Console.ReadLine();
            Console.WriteLine("#{TIME} {GPS_LAT} {GPS_LONG} {GPS_MS} {MAG_DIRECTION} {TARGET_LAT} {
                TARGET_LONG} {TARGET_DIR} {TARGET_DIST} {EEPROM} {EEPROM2}");

            while (line != null)
            {
                string[] strs = line.Split('=');

                switch (strs[0])
                {
                    case "TIME":
                        Console.WriteLine($"{TIME} {GPS_LAT} {GPS_LONG} {GPS_MS} {MAG_DIRECTION} {
                            TARGET_LAT} {TARGET_LONG} {TARGET_DIR} {TARGET_DIST} {EEPROM} {EEPROM2
                            }");
                        TIME = int.Parse(strs[1]) / 1000.0;
                        break;

                    case "GPS_LAT":
                        GPS_LAT = double.Parse(strs[1]);
                        break;

                    case "GPS_LONG":
                        GPS_LONG = double.Parse(strs[1]);
                        break;

                    case "GPS_M/S":
                        GPS_MS = double.Parse(strs[1]);
                        break;

                    case "MAG_DIRECTION":
```

```
                        MAG_DIRECTION = double.Parse(strs[1]);
                        break;

                    case "TARGET_LAT":
                        TARGET_LAT = double.Parse(strs[1]);
                        break;

                    case "TARGET_LONG":
                        TARGET_LONG = double.Parse(strs[1]);
                        break;

                    case "TARGET_DIR":
                        TARGET_DIR = double.Parse(strs[1]);
                        break;

                    case "TARGET_DIST":
                        TARGET_DIST = double.Parse(strs[1]);
                        break;

                    case "EEPROM":
                        if (strs[1].Contains("Right")) { EEPROM = 1; }
                        if (strs[1].Contains("Left")) { EEPROM = -1; }
                        if (strs[1].Contains("Forward")) { EEPROM = 0; }
                        if (strs[1].Contains("Gaol")) { EEPROM2 = 1; }
                        break;
                }

                line = Console.ReadLine();
            }
        }
    }
}
```

目標地点までの距離の時系列グラフを作成するスクリプト (Gnuplot)

```
set xlabel 'Time(s)'
set ylabel 'Distance(m)'
set grid

set border 15 lw 2
set title 'Distance to the target'

plot 'data_p2.dat' using 1:9  with lines lc "red" notitle

set term pngcairo
set term pngcairo size 1000, 700
set output "data_p2_dist.png"

replot
```

目標地点の相対方向と旋回方向の時系列グラフを作成するスクリプト (Gnuplot)

```
set xlabel 'Time(s)'
set ylabel 'Direction'
set ytics 45
set grid
set title 'Direction and Control'
set palette defined (-1 '#3333ff', 1'#ff0000')

set border 15 lw 2

plot 'data_p2.dat' u ($1):(int($5 - $8 + 180)%360 - 180) w fsteps lw 2 lc "#88FF0000" title "
     Relative direction to goal from rover"
replot 'data_p2.dat' u ($1):($10 * 10) w fsteps lw 2 lc "#880000FF" title "Rover turning direction"

set term pngcairo
set term pngcairo size 1000, 1000
set output "data_p2_direction.png"

replot
```

移動経路のグラフを作成するスクリプト (Gnuplot)

```
set xlabel 'GPS Lat'
set ylabel 'GPS Long'
set cblabel 'Time(s)'
set title 'Movement path (× is target location)'
set size ratio 1
set format xy "%1.5f"
set xtics rotate by -90
set grid

set border 15 lw 2

set palette defined (0 '#0000ff', 3'#ff0000')

plot 'data_p2.dat' u 2:3:($1) w lp lw 2 lc palette notitle
replot 'data_p2.dat' u (41.8398094177):(140.7663421630) w p ps 5 lc black notitle

set term pngcairo
set term pngcairo size 1000, 1000
set output "data_p2_gps.png"

replot
```

到達速度のグラフを作成するスクリプト (Gnuplot)

```
set xlabel 'Time(s)'
set ylabel 'Distance(m)'
set grid

set border 15 lw 2
set title 'Distance to the target'


f(x)=a + b*x
a = 1
b = 1
fit f(x) 'd1.dat' using 1:9 every 1:1:0:0:200 via a,b

plot f(x) lw 10 lc "light-gray" title sprintf("{%f} {%f} t", a, b)
replot 'd1.dat' using 1:9 every 1:1:0:0:200  with lines lc "red" notitle

set term pngcairo
set term pngcairo size 1000, 700
set output "data_p2_dist2.png"

replot
```

謝辞

本プロジェクトにおいて、新型コロナウイルス感染症 (COVID-19) が流行している中、公立はこだて未来大学工房職員、函館市企業局上下水道部浄水課の担当者ほか、協力していただいた皆様に心から感謝の気持ちとお礼申し上げます。

（※文責: B グループメンバー一同）

参考文献

[1] CanSat 計画　ー日米大学による手作り小型衛星への挑戦ー,
https://www.jstage.jst.go.jp/article/kjsass/48/562/48_589/_pdf/-char/ja,
2020 年 8 月 26 日 アクセス

[2] 新たな宇宙探査機を目指すローバの進化：ARLISS での挑戦,
https://www.jstage.jst.go.jp/article/sicejl/52/6/52_515/_pdf,
2021 年 8 月 26 日 アクセス

[3] ARLISS2012 報告書 慶応義塾大学 Tortoise,
http://www.unisec.jp/history/arliss2012/report/tortoise_rep.pdf,
2020 年 6 月 12 日 アクセス

[4] 缶ロケコラボ〜学生による異なる技術分野のコラボレーション〜 ロケット衛星 WG ,
http://www.unisec.jp/history/ws2010/files/18sat-rocket.pdf,
2020 年 8 月 26 日 アクセス

[5] ARISS2011 報告書 九州大学 宇宙機ダイナミクス研究室 “team IDEA”,
http://www.unisec.jp/history/arliss2011/report/kyushu_rep.pdf,
2020 年 8 月 26 日 アクセス

[6] Metasequoia 4,
https://www.metaseq.net/jp/, 2020 年 12 月 23 日アクセス

[7] ペパクラデザイナー 4,
https://tamasoft.co.jp/pepakura/, 2020 年 12 月 23 日アクセス

[8] Blender,
https://blender.jp/, 2020 年 12 月 23 日アクセス

[9] FUSION 360,
https://www.autodesk.co.jp/products/fusion-360/overview, 2020 年 12 月 23 日アクセス

[10] MakerBot,
https://www.makerbot.com/, 2020 年 12 月 23 日アクセス

[11] MakerCase,
https://ja.makercase.com/#/basicbox, 2020 年 12 月 23 日アクセス

[12] Adobe Illustrator,
https://adobe.ly/3pEuUPs, 2020 年 12 月 23 日アクセス

[13] Tiny GPS++,
http://arduiniana.org/libraries/tinygpsplus/, 2020 年 12 月 23 日アクセス

[14] BMX055 使用 9 軸センサーモジュール,
https://akizukidenshi.com/catalog/g/gK-13010/, 2020 年 12 月 23 日アクセス

[15] NeoSWSerial,
https://github.com/SlashDevin/NeoSWSerial, 2020 年 12 月 23 日アクセス

[16] Kinovea,

https://www.kinovea.org/, 2020 年 12 月 23 日アクセス

[17] Gnuplot,
http://www.gnuplot.info/, 2020 年 12 月 23 日アクセス

[18] B グループ—Project 17 めざせ宇宙開発 - 自律移動ロボット飛行プロジェクト 中間発表,
https://fun-cansat.amebaownd.com/pages/3961682/page_202006122056,
2020 年 12 月 23 日アクセス

[19] B グループ—Project 17 めざせ宇宙開発 - 自律移動ロボット飛行プロジェクト 最終成果発表,
https://fun-cansat2.amebaownd.com/pages/4353777/page_202010301737,
2020 年 12 月 23 日アクセス