令和 7 年度

修士学位論文

論文用テンプレート

信州大学大学院

総合理工学研究科繊維学専攻

機械ロボット分野

指導教員 河村 隆 教授

令和 6 年入学

学籍番号 24FS310F

氏名　　許 鵬飛

目次

第 1 章　緒言

1.1　研究背景

日本の農業では，農業従事者の減少と高齢化の進行により，労働力不足が深刻な問題となっている [1]．平成 27 年から令和 5 年にかけて，基幹的農業従事者は 175.7 万人から 116.4 万人へと減少した．特に 65 歳以上の従事者の割合が高く，平均年齢も上昇している [2]．この問題に対応するため，日本政府は農業現場における農業データの利活用の推進を支持している [3]．例えば，農業現場における土壌データ，気象データなどの農業データを活用することで，農作業の効率化やコストの削減を実現することができる．そこで，我々は桑畑での適用を例として，複数の AGV（Automatic Guided Vehicle，以下，AGV）を使用して圃場を巡回し，桑の生育状態を自動で観察するシステムの構築を目指す．

1.2　課題：農業環境における自律移動ロボットの課題

AGV による圃場モニタリング自動化の可能性を示す。しかし、屋外不整地、特に農業用ハウスのような半構造化環境における自己位置推定の困難性を問題提起する。不整地でのスリップによるオドメトリ誤差。ハウス骨格による GNSS 信号の遮蔽・マルチパス、それに伴う信号品質の動的な変動（君の実験結果をここで軽く触れる）。

1.3　先行研究とその限界

一般的な屋外 SLAM/LIO 技術（FAST-LIO2 等）を紹介し、ドリフト蓄積の問題点を指摘する。GNSS を用いたセンサーフュージョン技術を紹介するが、GNSS 信号が常に良好であることを前提としている研究が多い点を指摘し、ハウス環境への適用限界を示す。本研究室の先行研究（小林さん）に触れ、センサデータ収集は行われたが、位置精度やロバストな自律走行には課題が

残っていたことを明確にする。

1.4　本研究の目的と新規性

目的: GNSS 信号品質が動的に変動する農業用ハウス環境において、3D-LiDAR, IMU, Wheel Odometry, および品質情報付き GNSS をインテリジェントに統合し、途切れなく (continuous)、信頼性の高い (high-integrity)3D 自己位置推定を実現する手法を開発すること。さらに、その高精度な軌跡に基づき、高分解能な生態環境マップを生成すること。

新規性: (1) ハウス環境特有の GNSS 品質変動パターンを実測に基づき分析・モデル化する点。(2) GNSS 品質情報（協方差等）に応じて融合システムへの寄与を動的に調整する「GNSS 品質監視・融合制御」アルゴリズムを提案・実装する点。(3) 上記技術の有効性を実環境データで定量的に実証する点。

1.5　本論文の構成

各章の概要を簡潔に述べる。

第 2 章　高速・高信頼センシングシステムの構築

この章で、ハードウェア/組み込み技術の高さを明確に示す

2.1　システム全体の概要

AGV プラットフォーム（GS02 ベース）、搭載センサ群（Mid360, IMU, ZED-F9P, VESC, 生態センサユニット）、ソフトウェア（ROS 2）からなる全体構成を示す（ブロック図）。

TF ツリーを示し、特に LiDAR の傾斜搭載について言及する。

2.2　センサユニットの開発

本研究で構築するセンサユニットは，植生状態の観察を目的とし，浜松ホトニクス製ミニ分光器 C12880MA を搭載する．生育環境の評価のため，Bosch Sensortec 製 BME280（秋月電子通商製ブレークアウト基板 AE-BME280）および ELT SENSOR 社製 S300L-3V CO_2 センサを統合した．STMicroelectronics 製マイクロコントローラ（MCU）を使用し，センサデータを収集する．本センサユニットの全体構成を Fig. 2.1 に示す．各センサおよび MCU のインタフェース回路を付録に示す（Fig. A.1，Fig. A.2，Fig. A.3(a)，Fig. A.3(b)）．

2.2.1　高速分光センサ駆動モジュールの開発における従来の課題

C12880MA センサは，入力 ST 信号立下り後，TRG 信号を出力し，第 89 番目の TRG（トリガ）信号立上りで Video 信号を出力する．そのタイミングを Fig. 2.2 に示す．

従来は，TRG 信号の立ち上がりで割り込みを発生させ，割り込みサービスルーチン（以下，ISR）内で ADC データを読み取る割り込み駆動方式を用いてきた．本研究室の先行研究では，LPC1768 MCU と外部 AD コンバータ（SPI 接続）を用い，この方式で C12880MA から 50 kHz

Table 2.1　Limits of acquisition frequency with conventional methods

Method	**MCU**	**Achieved frequency**
Interrupt only	STM32F446RE	25.4 kHz
Interrupt + DMA	STM32F446RE	130 kHz

でのデータ取得が報告されている [4]．しかしこの方式では，取得ごとに CPU が ISR を必ず実行する必要があるため，割り込み応答遅延が主要な性能制約となり，センサが持つ数 MHz 帯の性能を引き出すことは困難である．

この制約を再確認するため，内部 ADC が高速な STM32F446RE でも ISR の方式を検証した．結果は Table 2.1 のとおりであり，割り込みのみでは 25.4 kHz 付近で欠落が発生した．さらに DMA（Direct Memory Access；CPU を介さずに周辺装置とメモリ間でデータを自動転送する仕組み．以下，DMA）を併用することで CPU 負荷は低減し，取得可能周波数は 130 kHz まで向上したが，遅延は残存し，数 MHz 帯には到達しなかった．以上より，割り込み起動時の遅延は問題であることが確認された．

2.2.2　提案手法：デュアルスイッチ FSM アーキテクチャ

割り込み遅延を排除するため，タイマ・ADC・DMA をハードウェアトリガで直結するアーキテクチャ（Timer→TRGO→ADC→DMA）は必須である．しかし，この構成には C12880MA 特有の「競合状態（Race Condition）」の問題が存在する．

2.2.2.1　C12880MA 駆動における「フライングスタート」問題

C12880MA の TRG 信号は，CLK（クロック）信号のミラであり，CLK が供給されている限り TRG も常時出力され続ける．一方，我々の制御フローは「(1) ST=HIGH で積分 → (2) ST=LOW で ADC 読出開始」である．もし，(1) の積分期間中に ADC がすでに DMA（`HAL_ADC_Start_DMA()`）によって待機状態（Armed）に設定されていた場合，常時入力されている TRG 信号が ADC を即座に誤トリガしてしまう．その結果，ST=HIGH 期間中の無効な暗レベルデータのみが DMA バッファに書き込まれてしまい，正しいスペクトルデータを取得できない．

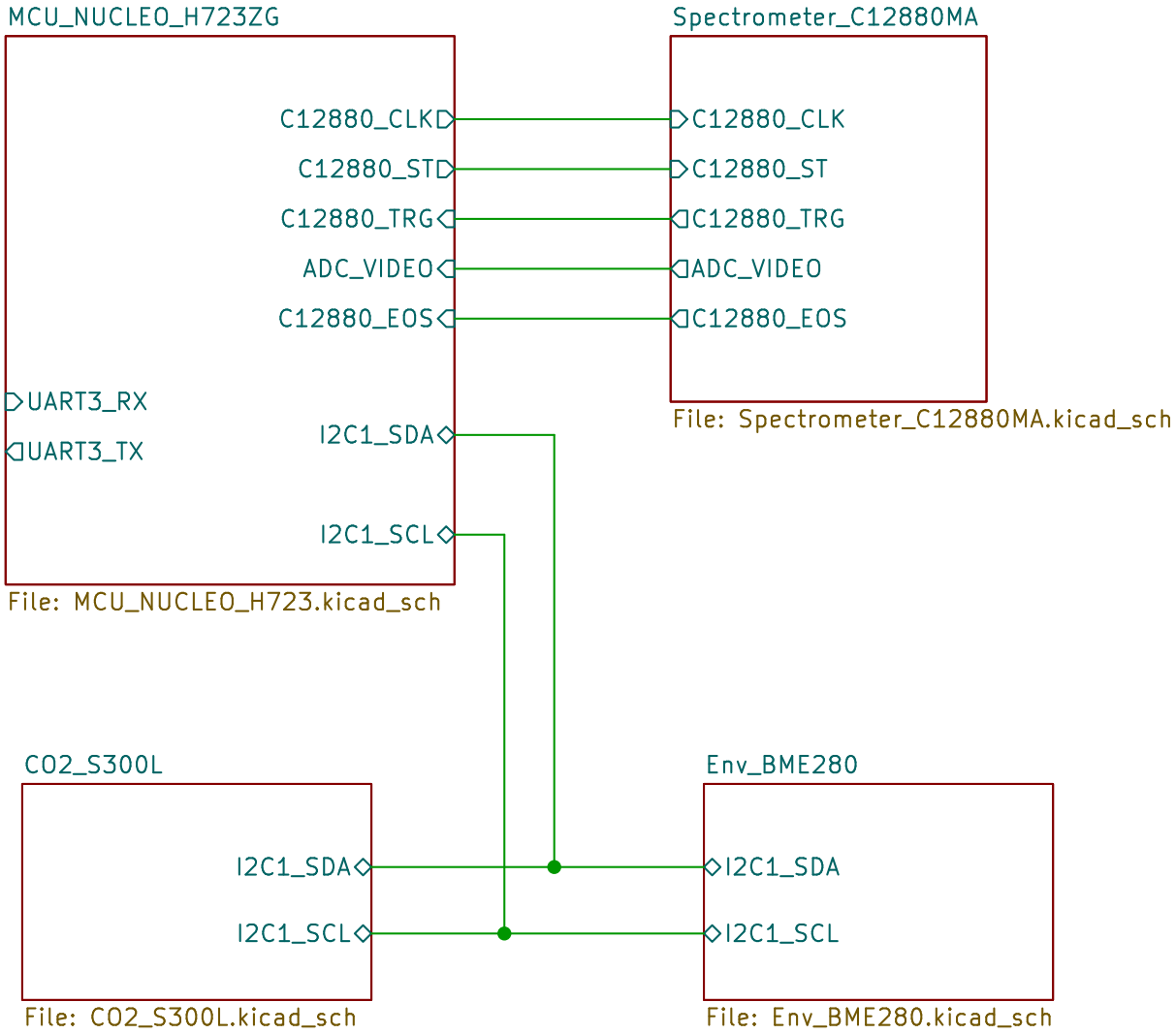


Fig. 2.1　Overall multisensor system architecture

2.2.2.2　FSM（有限状態機械，以下，FSM）の設計

この問題を根本的に解決するため，ソフトウェアのタイミング制御に依存せず，2 つの独立したハードウェア「スイッチ」によってデータフローを厳密に制御する FSM を設計した．

2.2.2.2.1　スイッチ 1：CLK 信号の制御　第 1 のスイッチは，TRG 信号の源である CLK 信号自体を制御する．汎用タイマ（TIM4 等）の PWM モードを用いて C12880MA の CLK 信号を生成する．これにより，`HAL_TIM_PWM_Start()` と `HAL_TIM_PWM_Stop()` を呼び出すことで，CLK（ひいては TRG）信号の発生源をソフトウェアレベルで完全にオン・オフ制御することが可能となる．

2.2.2.2.2　スイッチ 2（ゲートの制御）：ST 信号によるブレーキ機能　第 2 のスイッチは，ADC へのトリガ信号を物理的に遮断する「ゲート」である．制御タイマ（TIM1 等）を ADC の

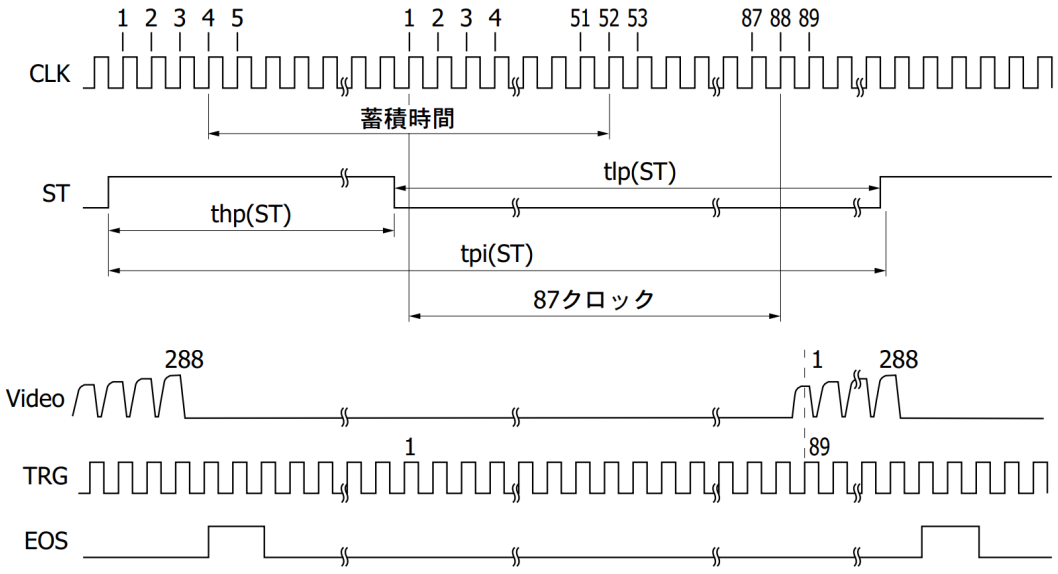


Fig. 2.2　Timing diagram of the C12880MA: excerpted from the Hamamatsu Photonics datasheet[5]

トリガスレーブとして設定し，センサの ST 信号を TIM1 の **BKIN（ブレーキ）ピン**に接続する．ブレーキ極性を「アクティブ・ハイ（High でブレーキ作動）」に設定する．これにより，ハードウェアゲートが実現される：

- **ST = HIGH（積分時）：** ブレーキがハードウェアレベルで有効化される．この状態では，たとえ TRG 信号がタイマ（TIM1）に入力されても，ADC を起動するための TRGO（トリガ出力）は遮断される．
- **ST = LOW（読出時）：** ブレーキがハードウェアレベルで解除される．TRGO 出力が許可され，TRG 信号が ADC に到達可能となる．

2.2.2.3　FSM（有限状態機械）による制御フロー

この「デュアルスイッチ」設計に基づき，極めて堅牢な FSM 制御フローを実装した．主要な状態遷移は以下の通りである．

1. **状態 0: IDLE（待機）**：ST=LOW, CLK(PWM)=**OFF**．ADC は停止中．ブレーキは解除されているが，TRG 信号源が OFF のため安全．
2. **状態 1: Arming（準備）**：`HAL_ADC_Start_DMA()` を呼び出し，ADC を待機状態にする．TRG 信号源は OFF のため，誤トリガは発生しない．
3. **状態 2: Integration（積分）**：`ST=HIGH` に設定．ハードウェアブレーキが即座に作動し，ADC へのゲートが閉じる．その後，`HAL_TIM_PWM_Start()` で CLK(PWM) を **ON** にする．

TRG 信号が出始めるが，ブレーキによって ADC には到達しない．この状態で任意の時間（積分時間）だけ待機する．

4. **状態 3: Readout（読出）**：`ST=LOW` に設定．ハードウェアブレーキが即座に解除され，ADC へのゲートが開く．ADC は待機状態，TRG 信号はすでに入力中，ゲートは開放．次の TRG 信号の有効エッジで，ADC がハードウェア同期され，DMA 転送が自動的に開始される．
5. **状態 4: Complete（完了）→ IDLE へ**：DMA 転送が完了すると，`HAL_ADC_ConvCpltCallback` 割り込みが発生する．ISR 内で `HAL_TIM_PWM_Stop()` を呼び出し，CLK(PWM) を **OFF** にする．ADC は HAL ライブラリによって自動的に停止される．システムは安全に状態 0 (IDLE) に戻る．

この「ブレーキゲート」と「CLK 制御」を組み合わせたデュアルスイッチ FSM は，C12880MA の「フライングスタート」問題をハードウェアレベルで解決し，正確な同期を実現した．このアーキテクチャが次節で述べる高速データ取得の基盤である．

2.2.2.4　STM32F446RE でのアーキテクチャ検証

従来手法では 130 kHz が限界であった STM32F446RE に新アーキテクチャを実装したところ，0.5 MHz および 1 MHz での安定したデータ取得に成功した．この結果から，従来手法の主要な性能制約が割り込み起動遅延と ISR 処理時間に起因していたこと，および本アーキテクチャがその解消に有効であることが示された．なお，本 MCU に搭載される ADC の性能から，達成可能な最大周波数は理論上およそ 1.5 MHz である．

2.2.2.5　STM32H723ZG での 5 MHz 高速取得の実現

次に，最大 5 MSPS の ADC を搭載する STM32H723ZG に同アーキテクチャを実装した．H7 シリーズ特有のキャッシュ・コヒーレンシ問題に対処するため，DMA の転送先バッファを非キャッシュの DTCM（Data Tightly Coupled Memory）領域に配置し，さらに ADC のハードウェアキャリブレーションとトリガ遅延設定を最適化した．その結果，目標としていた 5 MHz でのスペクトルデータ連続取得に成功した．

2.2.2.6　ADC 性能とサンプリングレートの検証

本システムが目標とする 5 MHz のデータレートを達成可能であることを，MCU の ADC 性能とセンサのタイミング制約から検証する．

センサに供給するクロックが 5 MHz であるため，データ更新周期 $T_{\mathrm{period}} = 1/5\,\mathrm{MHz} = 200\,\mathrm{ns}$ である．しかし，センサのデータシートによれば，アナログ出力（VIDEO）信号が安定している

のは TRG 信号の立ち上がりエッジを中心とした半周期のみである．したがって，ADC が正確な電圧値をサンプリングできるサンプリング可能時間幅 T_{stable} は，わずか 200 ns/2 = 100 ns となる．この厳しい制約を満たすため，ADC の動作を「サンプリング」と「変換」の二段階に分けて評価する必要がある．

1. サンプリング時間：$T_{\text{sampling}} \leq 100\,\text{ns}$.
2. 総変換時間：$T_{\text{total}} < 200\,\text{ns}$.

これらの条件，特に総変換時間 200ns の制約を満たすには，高速な ADC クロックが不可欠である．本研究では，サンプリング時間を 2.5 サイクル，変換時間を 12.5 サイクル（12 ビット分解能）に設定したため，合計 15 サイクルが必要となる．ここから逆算すると，要求される ADC クロック周波数 f_{ADCK} は次式のようになる．

$$f_{\text{ADCK}} > \frac{15}{200\,\text{ns}} = 75\,\text{MHz} \tag{2.1}$$

STM32H723ZG のデータシート [6] によれば，12 ビット ADC の最大クロック周波数 f_{ADC} は 75 MHz と規定されている．この規定周波数で安定動作を検証した結果，C12880MA に入力する CLK 信号を 4 MHz 以下にする必要があった．センサの仕様上限である 5MHz での高速取得を試みるため，ADC のカーネルクロックを 96 MHz に設定した．これはデータシートの仕様を超える値であるが，実験環境下での安定動作を実測により確認した．

この 96 MHz のクロック設定に基づき，実際の動作時間を再計算すると以下のようになる．

$$T_{\text{sampling}} = \frac{2.5}{96\,\text{MHz}} \approx 26.0\,\text{ns} \tag{2.2}$$

$$T_{\text{total}} = \frac{2.5 + 12.5}{96\,\text{MHz}} = \frac{15}{96 \times 10^6} \approx 156.3\,\text{ns} \tag{2.3}$$

計算の結果，サンプリング時間は 26.0ns であり，要求される 100ns の安定時間窓を十分に満たしている．また，総変換時間は 156.3ns であり，これも次のデータ周期である 200ns 未満である．以上の理論評価と測定結果により，本システムが 5 MHz で安定してデータ取得できることを確認した．

2.3 その他のセンサインターフェース

BME280, S300L, GNSS, IMU, Wheel Odometry のデータ取得方法について簡潔に述べる。

Table 2.2 Acquisition frequency (proposed architecture)

Method	MCU	Achieved freq.
Proposed arch.	STM32F446RE	1.5 MHz (theory)
Proposed arch.	STM32H723ZG	5.0 MHz (achieved)

第 3 章　提案手法：GNSS 品質監視に基づくロバスト自己位置推定

Proposed Method: Robust Localization based on GNSS Quality Monitoring 写上大棚和照片

3.1　問題の再定義：農業用ハウスにおける GNSS 信号品質の特性分析

実測したデータ（良好時と不良時の/fix メッセージの協方差、衛星数、FIX/FLOAT 状態の遷移など）を提示し、ハウス環境における GNSS 信号の不安定性と品質指標（協方差）の信頼性（正直さ）を定量的に示す。TF ツリーを示し、特に LiDAR の傾斜搭載について言及する。

3.2　ベースライン：LiDAR 慣性オドメトリ (LIO)

FAST-LIO2 の概要と、本研究における適用方法（IMU キャリブレーション含む）を述べる。
LIO 単独での精度限界（ドリフト）を示すための予備実験結果（あれば）を提示する。

3.3　提案手法：GNSS 品質監視モジュールと適応的 EKF 融合

C12880MA のデータシート要求（数 MHz）に対し、先行研究（割り込み方式）では数十〜百数十 kHz が限界であったことを示す（君の Table 1 を引用）。

3.3.1　全体アーキテクチャ

LIO、Wheel Odometry、GNSS Supervisor Module、EKF (robot localization) からなる融合システムのブロック図を示す。

3.3.2　GNSS 品質監視モジュール

GNSS データ（/fix）を入力とし、その品質（主に協方差行列、必要なら FIX/FLOAT 状態なども加味）を評価し、「信頼できるデータのみ」を EKF へ送る、あるいは「品質情報を重みとして」EKF へ送るアルゴリズムを提案する（閾値処理や信頼度スケーリングなど）。このモジュールが新規性の中核であることを強調する。

3.3.3　EKF による状態推定

Robot localization を用い、LIO (Wheel Odom) を高頻度の主要な運動推定源とし、品質監視モジュールを経由した GNSS データを低頻度の絶対位置補正源として融合する設定を説明する。

3.4　期待される効果

GNSS 信号が良いときはその精度を活用し、悪いときは LIO によってドリフトを抑制し、全体として精度と頑健性を両立できることを論理的に説明する。

第 4 章　実験による評価 (Experimental Evaluation)

4.1　実験設定

実験フィールド（信州大学農場ハウス）、使用した AGV、データ収集シナリオ（GNSS 良好時、不良時、ハウス内外移動時）を詳細に記述する。

評価に用いた Ground Truth の定義（例：GNSS 良好時の高精度軌跡、あるいは外部計測機器）を明確にする。

4.2　定量的評価：自己位置推定精度の比較

比較手法: (1) LIO (+Wheel Odom) のみ、(2) LIO+Wheel+GNSS のナイーブな融合（品質無視）、(3) 提案手法（品質監視付き融合）の 3 つを用意する。

各シナリオのデータセットに対して 3 手法を適用し、得られた軌跡を Ground Truth と比較する。

結果の提示: (1) 軌跡比較図（Fig. ）、(2) 定量評価指標（RMSE, 最大誤差など）の比較表（Table ）を示す。

4.2.1　考察

実験結果に基づき、提案手法 (3) が比較手法 (1)(2) に対して優位性を持つことを明確に論証する。特に、GNSS 品質が悪化した際に、ナイーブ融合 (2) が破綻するのに対し、提案手法 (3) が安定して精度を維持できることを強調する。

GNSS 品質監視モジュールの閾値設定などのパラメータの妥当性についても議論する

第5章　応用：生態環境マッピング (Application:　Ecological Mapping)

5.1　マッピング手法

第 4 章で最も精度が高いと評価された提案手法による自己位置推定軌跡を用いることを明記する。

第 2 章で開発した高速生態センサユニットからのデータ（スペクトル、CO2、温湿度）を、高精度なタイムスタンプと位置情報に同期させて 3D マップ上に重畳（オーバーレイ）する手法を説明する（必要であれば補間手法なども）。

5.2　生成された生態環境マップ

実際に生成した各種生態環境マップ（スペクトル指標マップ、CO2 濃度分布マップ、温湿度マップなど）を提示する

5.2.1　マッピング結果の考察

生成されたマップから読み取れる空間的な分布パターンや、異なる環境要因間の相関（例：日照条件とスペクトル、CO2 濃度と植物活性など）について考察する。

高精度な自己位置推定が、意味のある生態環境マップ生成にいかに貢献するかを具体的に示す

第 6 章　結論 (Conclusion)

6.1　本研究の成果

本研究で達成したこと（高速センサ駆動、GNSS 品質分析、ロバスト融合手法の提案と実証、生態マップ生成）を簡潔にまとめる。

研究目的が達成されたことを明確に述べる。

6.2　今後の課題と展望

提案手法のさらなる改善点（例：より高度な FDIR、機械学習の導入など）。

完全自律走行の実現。

雑草エリアでの走行（可通行性解析）。

長期運用による時系列マップの作成と農業応用。

参考文献

[1] 農林水産省. スマート農業の展開について. https://www.soumu.go.jp/main_content/000775128.pdf. (Accessed on 16/6/2025).

[2] 農林水産省. 農業労働力に関する統計. https://www.maff.go.jp/j/tokei/sihyo/data/08.html. (Accessed on 16/6/2025).

[3] 農林水産省. 農業データの利活用の推進について. https://www.maff.go.jp/j/kanbo/smart/attach/pdf/index-146.pdf. (Accessed on 16/6/2025).

[4] 小. 拓也. “AGV による作物の生育状況観察システムに関する研究”. 修士学位論文. 信州大学大学院総合理工学研究科, 2021.

[5] Hamamatsu-Photonics. Mini-spectrometers. https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/ssd/c12880ma_c16767ma_kacc1226e.pdf. (Accessed on 10/9/2025).

[6] STMicroelectronics. stm32h723zg datasheet. https://www.st.com/resource/en/datasheet/stm32h723zg.pdf. (Accessed on 10/9/2025).

謝辞

本研究の遂行にあたり，指導教官として終始多大なご指導を賜った河村隆教授に深謝致します．また河村研究室の皆様には，本研究の遂行にあたり多大なご助言，ご協力頂きました，ここに感謝の意を表します

付録 A　インタフェース回路

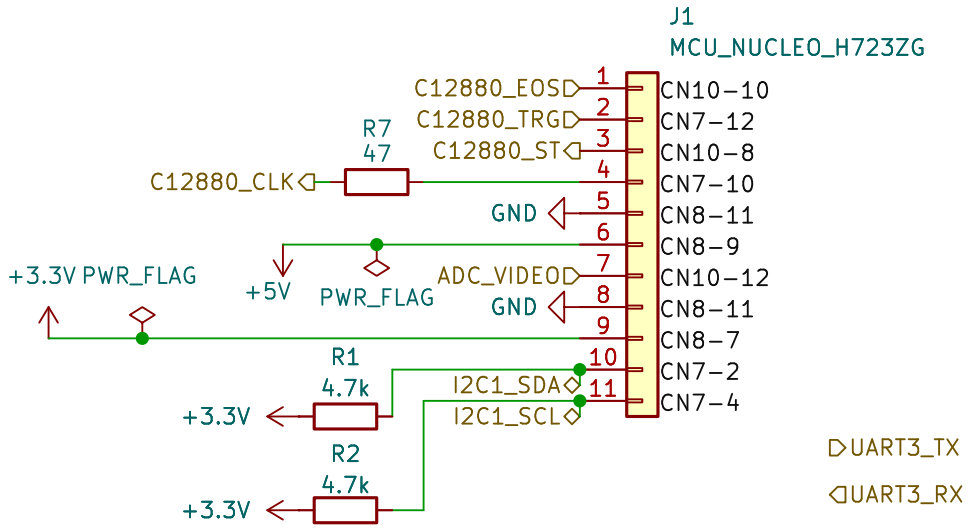


Fig. A.1　MCU (NUCLEO-H723ZG) interface circuit

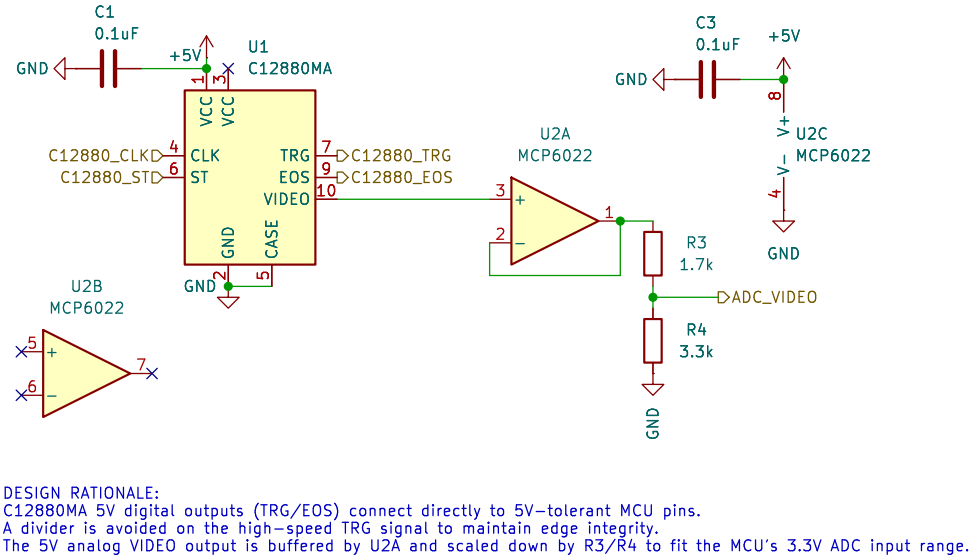
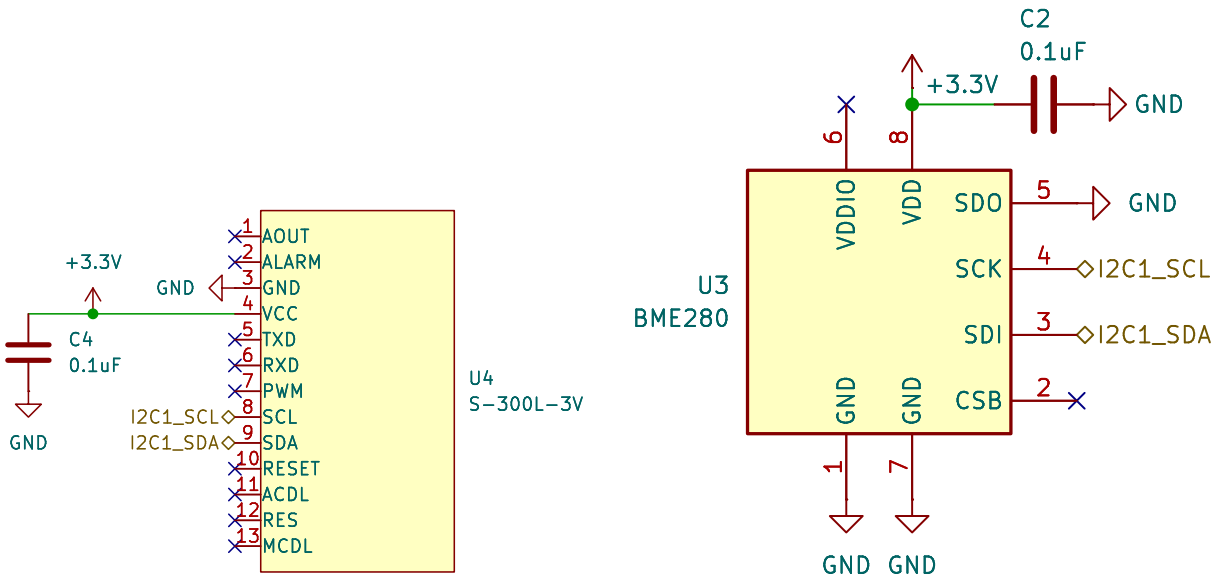


Fig. A.2 Spectrometer (C12880MA) interface circuit



(a) CO_2 sensor (S300L) interface circuit (b) Environmental sensor (BME280) interface circuit

Fig. A.3 Interface circuits for CO_2 and environmental sensors.

付録 B　sensor_unit_code

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
```

```
21
22  /* Private includes
    ↪  ----------------------------------------------------------*/
23  /* USER CODE BEGIN Includes */
24  #include <stdio.h>
25  #include <string.h>
26  #include <stdbool.h>
27  /* USER CODE END Includes */
28
29  /* Private typedef
    ↪  -----------------------------------------------------------*/
30  /* USER CODE BEGIN PTD */
31
32  /* USER CODE END PTD */
33
34  /* Private define
    ↪  ------------------------------------------------------------*/
35  /* USER CODE BEGIN PD */
36
37  /* USER CODE END PD */
38
39  /* Private macro
    ↪  -------------------------------------------------------------*/
40  /* USER CODE BEGIN PM */
41
42  /* USER CODE END PM */
43
44  /* Private variables
    ↪  ---------------------------------------------------------*/
45  ADC_HandleTypeDef hadc1;
46  DMA_HandleTypeDef hdma_adc1;
```

```
47
48  TIM_HandleTypeDef htim1;
49  TIM_HandleTypeDef htim3;
50
51  UART_HandleTypeDef huart2;
52
53  /* USER CODE BEGIN PV */
54  // スペクトロメータのピクセル数を定義
55  #define NUM_PIXELS 288
56  // DMA 受信バッファ
57  uint16_t adc_buffer[NUM_PIXELS];
58  // データ収集完了フラグ
59  volatile bool data_ready_flag = false;
60  /* USER CODE END PV */
61
62  /* Private function prototypes
    ↪  -----------------------------------------------*/
63  void SystemClock_Config(void);
64  static void MX_GPIO_Init(void);
65  static void MX_DMA_Init(void);
66  static void MX_USART2_UART_Init(void);
67  static void MX_ADC1_Init(void);
68  static void MX_TIM1_Init(void);
69  static void MX_TIM3_Init(void);
70  /* USER CODE BEGIN PFP */
71  void print_spectrum_data(void);
72  /* USER CODE END PFP */
73
74  /* Private user code
    ↪  ---------------------------------------------------------*/
75  /* USER CODE BEGIN 0 */
```

```c
/**
  * @brief  UART経由でスペクトルデータを送信
  * @retval None
  */
void print_spectrum_data(void) {
    static char buf[4096];
    int n = 0;

    // フレームヘッダー
    n += snprintf(buf + n, sizeof(buf) - n, "BEGIN,");

    // 288個のデータ値
    for (int i = 0; i < 288; ++i) {
            // 最後の値の後にもカンマを追加し、末尾でまとめてENDを追加
        n += snprintf(buf + n, sizeof(buf) - n, "%u,", adc_buffer[i]);
    }

    // フレームフッター（Pythonのreadlineのために、必ず\r\nを付ける）
    n += snprintf(buf + n, sizeof(buf) - n, "END\r\n");

    // 一括で送信
    HAL_UART_Transmit(&huart2, (uint8_t*)buf, n, HAL_MAX_DELAY);
}



/**
  * @brief  ADC変換完了コールバック関数
  * @note   DMAモードでは、この関数は指定された数のデータ転送がすべて完了した後に一
    ↪ 度だけ呼び出される。
  * @param  hadc: ADCハンドル
```

```
106   * @retval None
107   */
108 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
109 {
110         // 対象の ADC1 であるかを確認
111   if(hadc->Instance == ADC1)
112   {
113           // ハードウェアが自動的に 288 回のサンプリングと転送を完了したので、メイン
              ↪ ループに通知するためのフラグを設定する
114     data_ready_flag = true;
115   }
116 }
117 /* USER CODE END 0 */
118
119 /**
120   * @brief  The application entry point.
121   * @retval int
122   */
123 int main(void)
124 {
125
126   /* USER CODE BEGIN 1 */
127
128   /* USER CODE END 1 */
129
130   /* MCU
      ↪ Configuration--------------------------------------------------------*/
131
132   /* Reset of all peripherals, Initializes the Flash interface and the
      ↪ Systick. */
133   HAL_Init();
```

```
134
135     /* USER CODE BEGIN Init */
136
137     /* USER CODE END Init */
138
139     /* Configure the system clock */
140     SystemClock_Config();
141
142     /* USER CODE BEGIN SysInit */
143
144     /* USER CODE END SysInit */
145
146     /* Initialize all configured peripherals */
147     MX_GPIO_Init();
148     MX_DMA_Init();
149     MX_USART2_UART_Init();
150     MX_ADC1_Init();
151     MX_TIM1_Init();
152     MX_TIM3_Init();
153     /* USER CODE BEGIN 2 */
154
        ↪  //------------------------------------------------------------------------//
155       // 1. CLKクロック信号を起動（TIM1のPWMで生成）
156
        ↪  //------------------------------------------------------------------------//
157       if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) != HAL_OK) {
158           Error_Handler();
159       }
160
161
        ↪  //------------------------------------------------------------------------//
```

```
162     // 2. TRG信号の監視を開始（TIM3の入力キャプチャで）
163     //    これにより、TIM3が外部TRG信号に応答を開始する
164
        ↪ //-------------------------------------------------------------------------//
165     if (HAL_TIM_IC_Start(&htim3, TIM_CHANNEL_1) != HAL_OK) {
166         Error_Handler();
167     }
168
169     HAL_UART_Transmit(&huart2, (uint8_t*)"Spectrometer Ready
        ↪ (Full-Hardware-Trigger Mode).\r\n", 49, HAL_MAX_DELAY);
170   /* USER CODE END 2 */
171
172   /* Infinite loop */
173   /* USER CODE BEGIN WHILE */
174     while (1)
175     {
176     /* USER CODE END WHILE */
177
178     /* USER CODE BEGIN 3 */
179           // 3. スペクトル収集サイクルを1回トリガーする
180       HAL_GPIO_WritePin(ST_GPIO_Port, ST_Pin, GPIO_PIN_SET);
181       HAL_Delay(10); // 積分時間を10msに設定（必要に応じて調整可能）
182       HAL_GPIO_WritePin(ST_GPIO_Port, ST_Pin, GPIO_PIN_RESET);
183
184       // 4. ADCとDMAを起動。ハードウェアは自動的にTIM3から転送されるTRG信号を
          ↪ 288回待機する
185       if (HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, NUM_PIXELS) !=
          ↪ HAL_OK) {
186           Error_Handler();
187       }
188
```

```
189         // 5. 収集完了を待機。この間、CPUは完全にアイドル状態
190         while (!data_ready_flag) {}
191         data_ready_flag = false; // 次の収集のためにフラグをリセット
192 
193         // 6. 収集完了後、データを処理して送信
194             print_spectrum_data();
195 
196         // 7. 100ms待機し、次の収集を開始
197         HAL_Delay(100);
198       }
199   /* USER CODE END 3 */
200 }
201 
202 /**
203   * @brief System Clock Configuration
204   * @retval None
205   */
206 void SystemClock_Config(void)
207 {
208   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
209   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
210 
211   /** Configure the main internal regulator output voltage
212   */
213   __HAL_RCC_PWR_CLK_ENABLE();
214   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
215 
216   /** Initializes the RCC Oscillators according to the specified parameters
217   * in the RCC_OscInitTypeDef structure.
218   */
219   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
```

```
220   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
221   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
222   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
223   RCC_OscInitStruct.PLL.PLLM = 4;
224   RCC_OscInitStruct.PLL.PLLN = 180;
225   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
226   RCC_OscInitStruct.PLL.PLLQ = 2;
227   RCC_OscInitStruct.PLL.PLLR = 2;
228   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
229   {
230     Error_Handler();
231   }
232
233   /** Activate the Over-Drive mode
234   */
235   if (HAL_PWREx_EnableOverDrive() != HAL_OK)
236   {
237     Error_Handler();
238   }
239
240   /** Initializes the CPU, AHB and APB buses clocks
241   */
242   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
243                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
244   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
245   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
246   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
247   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
248
249   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
250   {
```

```
251       Error_Handler();
252     }
253   }
254
255   /**
256     * @brief ADC1 Initialization Function
257     * @param None
258     * @retval None
259     */
260   static void MX_ADC1_Init(void)
261   {
262
263     /* USER CODE BEGIN ADC1_Init 0 */
264
265     /* USER CODE END ADC1_Init 0 */
266
267     ADC_ChannelConfTypeDef sConfig = {0};
268
269     /* USER CODE BEGIN ADC1_Init 1 */
270
271     /* USER CODE END ADC1_Init 1 */
272
273     /** Configure the global features of the ADC (Clock, Resolution, Data
        ↪  Alignment and number of conversion)
274     */
275     hadc1.Instance = ADC1;
276     hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
277     hadc1.Init.Resolution = ADC_RESOLUTION_12B;
278     hadc1.Init.ScanConvMode = DISABLE;
279     hadc1.Init.ContinuousConvMode = DISABLE;
280     hadc1.Init.DiscontinuousConvMode = DISABLE;
```

```
281   hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
282   hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
283   hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
284   hadc1.Init.NbrOfConversion = 1;
285   hadc1.Init.DMAContinuousRequests = DISABLE;
286   hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
287   if (HAL_ADC_Init(&hadc1) != HAL_OK)
288   {
289     Error_Handler();
290   }
291
292   /** Configure for the selected ADC regular channel its corresponding rank in
      ↪  the sequencer and its sample time.
293   */
294   sConfig.Channel = ADC_CHANNEL_1;
295   sConfig.Rank = 1;
296   sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
297   if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
298   {
299     Error_Handler();
300   }
301   /* USER CODE BEGIN ADC1_Init 2 */
302
303   /* USER CODE END ADC1_Init 2 */
304
305 }
306
307 /**
308   * @brief TIM1 Initialization Function
309   * @param None
310   * @retval None
```

```
311   */
312 static void MX_TIM1_Init(void)
313 {
314
315   /* USER CODE BEGIN TIM1_Init 0 */
316
317   /* USER CODE END TIM1_Init 0 */
318
319   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
320   TIM_MasterConfigTypeDef sMasterConfig = {0};
321   TIM_OC_InitTypeDef sConfigOC = {0};
322   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
323
324   /* USER CODE BEGIN TIM1_Init 1 */
325
326   /* USER CODE END TIM1_Init 1 */
327   htim1.Instance = TIM1;
328   htim1.Init.Prescaler = 0;
329   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
330   htim1.Init.Period = 180-1;
331   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
332   htim1.Init.RepetitionCounter = 0;
333   htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
334   if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
335   {
336     Error_Handler();
337   }
338   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
339   if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
340   {
341     Error_Handler();
```

```
342   }
343   if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
344   {
345     Error_Handler();
346   }
347   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
348   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
349   if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
350   {
351     Error_Handler();
352   }
353   sConfigOC.OCMode = TIM_OCMODE_PWM1;
354   sConfigOC.Pulse = 90;
355   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
356   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
357   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
358   sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
359   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
360   if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
361   {
362     Error_Handler();
363   }
364   sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
365   sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
366   sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
367   sBreakDeadTimeConfig.DeadTime = 0;
368   sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
369   sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
370   sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
371   if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
372   {
```

```
373       Error_Handler();
374     }
375     /* USER CODE BEGIN TIM1_Init 2 */
376
377     /* USER CODE END TIM1_Init 2 */
378     HAL_TIM_MspPostInit(&htim1);
379
380   }
381
382   /**
383     * @brief TIM3 Initialization Function
384     * @param None
385     * @retval None
386     */
387   static void MX_TIM3_Init(void)
388   {
389
390     /* USER CODE BEGIN TIM3_Init 0 */
391
392     /* USER CODE END TIM3_Init 0 */
393
394     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
395     TIM_MasterConfigTypeDef sMasterConfig = {0};
396     TIM_IC_InitTypeDef sConfigIC = {0};
397
398     /* USER CODE BEGIN TIM3_Init 1 */
399
400     /* USER CODE END TIM3_Init 1 */
401     htim3.Instance = TIM3;
402     htim3.Init.Prescaler = 0;
403     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
```

```
404   htim3.Init.Period = 65535;
405   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
406   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
407   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
408   {
409     Error_Handler();
410   }
411   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
412   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
413   {
414     Error_Handler();
415   }
416   if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
417   {
418     Error_Handler();
419   }
420   sMasterConfig.MasterOutputTrigger = TIM_TRGO_OC1;
421   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
422   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
423   {
424     Error_Handler();
425   }
426   sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
427   sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
428   sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
429   sConfigIC.ICFilter = 0;
430   if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
431   {
432     Error_Handler();
433   }
434   /* USER CODE BEGIN TIM3_Init 2 */
```

```
435
436     /* USER CODE END TIM3_Init 2 */
437
438   }
439
440   /**
441     * @brief USART2 Initialization Function
442     * @param None
443     * @retval None
444     */
445   static void MX_USART2_UART_Init(void)
446   {
447
448     /* USER CODE BEGIN USART2_Init 0 */
449
450     /* USER CODE END USART2_Init 0 */
451
452     /* USER CODE BEGIN USART2_Init 1 */
453
454     /* USER CODE END USART2_Init 1 */
455     huart2.Instance = USART2;
456     huart2.Init.BaudRate = 115200;
457     huart2.Init.WordLength = UART_WORDLENGTH_8B;
458     huart2.Init.StopBits = UART_STOPBITS_1;
459     huart2.Init.Parity = UART_PARITY_NONE;
460     huart2.Init.Mode = UART_MODE_TX_RX;
461     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
462     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
463     if (HAL_UART_Init(&huart2) != HAL_OK)
464     {
465       Error_Handler();
```

```
466   }
467   /* USER CODE BEGIN USART2_Init 2 */
468
469   /* USER CODE END USART2_Init 2 */
470
471 }
472
473 /**
474   * Enable DMA controller clock
475   */
476 static void MX_DMA_Init(void)
477 {
478
479   /* DMA controller clock enable */
480   __HAL_RCC_DMA2_CLK_ENABLE();
481
482   /* DMA interrupt init */
483   /* DMA2_Stream0_IRQn interrupt configuration */
484   HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
485   HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
486
487 }
488
489 /**
490   * @brief GPIO Initialization Function
491   * @param None
492   * @retval None
493   */
494 static void MX_GPIO_Init(void)
495 {
496   GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
497  /* USER CODE BEGIN MX_GPIO_Init_1 */
498  /* USER CODE END MX_GPIO_Init_1 */
499
500    /* GPIO Ports Clock Enable */
501    __HAL_RCC_GPIOC_CLK_ENABLE();
502    __HAL_RCC_GPIOH_CLK_ENABLE();
503    __HAL_RCC_GPIOA_CLK_ENABLE();
504    __HAL_RCC_GPIOB_CLK_ENABLE();
505
506    /*Configure GPIO pin Output Level */
507    HAL_GPIO_WritePin(ST_GPIO_Port, ST_Pin, GPIO_PIN_RESET);
508
509    /*Configure GPIO pin : ST_Pin */
510    GPIO_InitStruct.Pin = ST_Pin;
511    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
512    GPIO_InitStruct.Pull = GPIO_NOPULL;
513    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
514    HAL_GPIO_Init(ST_GPIO_Port, &GPIO_InitStruct);
515
516    /*Configure GPIO pin : EOS_Pin */
517    GPIO_InitStruct.Pin = EOS_Pin;
518    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
519    GPIO_InitStruct.Pull = GPIO_NOPULL;
520    HAL_GPIO_Init(EOS_GPIO_Port, &GPIO_InitStruct);
521
522  /* USER CODE BEGIN MX_GPIO_Init_2 */
523  /* USER CODE END MX_GPIO_Init_2 */
524  }
525
526  /* USER CODE BEGIN 4 */
527
```

```
528 /* USER CODE END 4 */
529
530 /**
531   * @brief  This function is executed in case of error occurrence.
532   * @retval None
533   */
534 void Error_Handler(void)
535 {
536   /* USER CODE BEGIN Error_Handler_Debug */
537   /* User can add his own implementation to report the HAL error return state
      ↪  */
538   __disable_irq();
539   while (1)
540   {
541   }
542   /* USER CODE END Error_Handler_Debug */
543 }
544
545 #ifdef  USE_FULL_ASSERT
546 /**
547   * @brief  Reports the name of the source file and the source line number
548   *         where the assert_param error has occurred.
549   * @param  file: pointer to the source file name
550   * @param  line: assert_param error line source number
551   * @retval None
552   */
553 void assert_failed(uint8_t *file, uint32_t line)
554 {
555   /* USER CODE BEGIN 6 */
556   /* User can add his own implementation to report the file name and line
      ↪  number,
```

```
557         ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
            ↪  */
558     /* USER CODE END 6 */
559   }
560   #endif /* USE_FULL_ASSERT */
```