Manuel Technique du Robot

Table des matières

1 Introduction **3**
- 1.1 Objectif . . . 3
- 1.2 Cahier des charges . . . 3
  - 1.2.1 Cahier des charges officiel *FIRST®* . . . 3
  - 1.2.2 Résumé du cahier des charges, points importants à retenir et contraintes personnelles . . . 3
- 1.3 Résumé . . . 4

2 Conception Mécanique **5**
- 2.1 Châssis et structure . . . 5
  - 2.1.1 Plateforme de base . . . 5
  - 2.1.2 Entrée (« intake ») . . . 6
  - 2.1.3 « Buffer » et lanceur . . . 7
- 2.2 Diagrammes et CAD . . . 8

3 Système Électrique **9**
- 3.1 Architecture électrique . . . 9

4 Programmation **10**
- 4.1 Environnement et outils . . . 10
  - 4.1.1 Outils nécessaires et recommandés . . . 10
  - 4.1.2 Préparation de l'environnement . . . 10
- 4.2 Architecture logicielle . . . 12
  - 4.2.1 Conventions internes . . . 12
  - 4.2.2 Structure générale . . . 13
  - 4.2.3 Systèmes de coordonnées . . . 15
- 4.3 Tests et débogage . . . 16
  - 4.3.1 Modes TeleOp . . . 16
  - 4.3.2 Utilisation du FTC Dashboard . . . 18

5 Stratégie et Opérations **20**
- 5.1 Scénarios de match . . . 20
- 5.2 Séquences autonomes . . . 20
- 5.3 Gestion des erreurs et dépannage . . . 20

Chapitre 1

Introduction

1.1 Objectif

Notre objectif est de fabriquer le robot le plus performant et fiable pour grimper le plus loin possible dans les classements, mais aussi d'encourager l'esprit d'ingénierie et le travail en équipe. Nous voulons renforcer nos compétences en mécanique, électronique et programmation, appliquer une démarche structurée et collaborative, et développer des solutions innovantes tout en respectant les règles de sécurité et l'éthique de la compétition.

1.2 Cahier des charges

Cahier des charges

Un cahier des charges est un document qui décrit l'ensemble des besoins, contraintes et spécifications techniques qu'un produit ou un système doit respecter. Il sert de référence tout au long de la conception et de la réalisation.

1.2.1 Cahier des charges officiel *FIRST®*

Un cahier des charges complet est fourni par *FIRST®*. Celui-ci décrit toutes les contraintes précises du robot, ainsi que les caractéristiques du terrain et les règles à respecter avant, pendant et après les matchs.

Veuillez retrouver celui-ci à l'adresse suivante : `https://ftc-resources.firstinspires.org/ftc/game/manual`.

1.2.2 Résumé du cahier des charges, points importants à retenir et contraintes personnelles

Éligibilité et inspection des robots

Le robot doit tenir dans un volume de 18 pouces x 18 pouces x 18 pouces (soit 45,7 cm x 45,7 cm x 45,7 cm). Pour cette saison, les robots n'ont pas le droit de s'étendre au-delà de ces dimensions, sauf pour la fin du match (*l'« endgame »*, les dernières 20 secondes du match).

Taille maximale du robot

La taille maximale initiale d'un robot pour cette saison est de 45,7 cm x 45,7 cm x 45,7 cm. Il ne peut pas dépasser ces dimensions sauf durant l'*endgame*.

L'équipe des FRITES a décidé d'utiliser principalement des composants *GoBilda®*. Il convient donc d'utiliser des pièces *GoBilda®* autant que possible et d'éviter les autres marques afin de prévenir les problèmes d'incompatibilité. Cependant, certaines exceptions existent, comme par exemple les « *Control Hubs* », qui sont principalement fournis par *REV®*.

En respectant ces critères, l'équipe s'assure que le robot sera conforme aux règles de la compétition et pourra passer l'inspection sans difficultés, tout en minimisant les risques d'incompatibilités entre composants.

Dimensions principales à retenir

Voici une liste (non exhaustive) des contraintes de dimensions à respecter pour les matchs :

- Le robot doit tenir dans un volume de 18 pouces x 18 pouces x 18 pouces (soit 45,7 cm x 45,7 cm x 45,7 cm).
- Terrain : 144 pouces par 144 pouces (soit 365,75 cm by 365,75 cm)
- « Artifact » : 5 pouces de diamètre (soit 12,70 cm)

Caractéristiques spécifiques à la saison

Cette saison du *FIRST® Tech Challenge*, intitulée **DECODE™**, se concentre sur un jeu dynamique où deux alliances de deux robots s'affrontent pour marquer des *artéfacts* (objets violets et verts) dans un but, construire des *motifs* sur une rampe selon un schéma aléatoire affiché sur un obélisque, et revenir à leur base avant la fin du match. Le match se déroule en deux phases : **30 secondes en mode autonome**, où les robots doivent décoder le motif et commencer à marquer, suivies de **2 minutes en mode télécommandé**, où les pilotes optimisent le score.

Les équipes sont évaluées sur leur **performance technique**, leur **documentation** (portfolio, processus de conception), et leur **engagement communautaire**. Les récompenses, comme l'*Inspire Award* ou le *Think Award*, mettent en avant l'excellence globale, la créativité, et l'impact social. Les robots doivent respecter des contraintes strictes, notamment une **taille initiale maximale de 45,7 cm x 45,7 cm x 45,7 cm**, tout en restant stables et sûrs.

Cette saison encourage l'**innovation**, la **stratégie d'alliance**, et la **collaboration** entre équipes, dans un esprit de compétition équilibré par l'entraide et le respect des valeurs *FIRST*.

1.3 Résumé

- Construire un robot performant pour le *FTC* et développer les compétences de l'équipe (mécanique, électronique, code).
- Respecter les contraintes officielles : robot limité à **18" x 18" x 18"**, terrain de **144" x 144"**, artéfacts de **Ø 5"**.
- Suivre le cahier des charges *FIRST®* + contraintes internes (usage majoritaire de pièces *GoBilda®*).
- Prendre en compte les phases du match : **30 s** autonome + **120 s** téléopéré.
- Réaliser les objectifs du jeu DECODE™ : dépôt d'artéfacts, construction de motifs, retour à la base.
- Répondre aux critères d'évaluation : performance du robot, conformité, documentation, engagement de l'équipe.

Chapitre 2

Conception Mécanique

La conception mécanique constitue la base du robot et détermine sa robustesse, sa stabilité et sa capacité à accomplir les tâches prévues sur le terrain.

Ce chapitre abordera tous ces aspects afin de présenter une approche structurée et cohérente, garantissant un robot fiable, performant et conforme aux règles de la compétition.

2.1 Châssis et structure

Châssis

Le châssis constitue la structure principale du robot, assurant la rigidité nécessaire pour supporter les systèmes de locomotion, de manipulation et d'alimentation, tout en respectant les contraintes de dimensions et de poids imposées par la compétition.

2.1.1 Plateforme de base

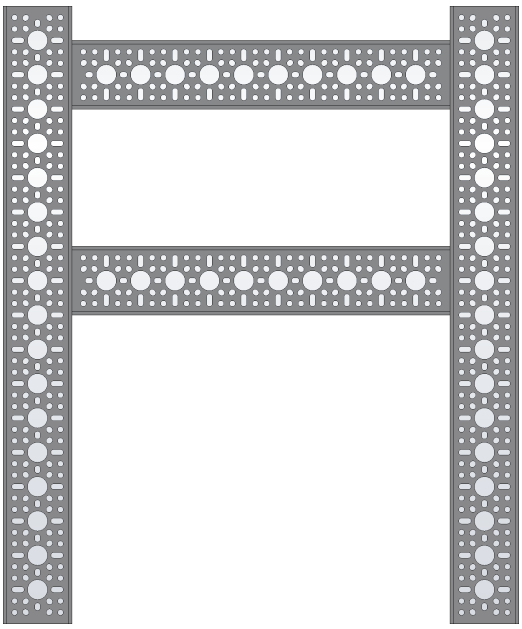


FIGURE 2.1 – Plateforme de base du robot simplifiée

La plateforme de base est un élément essentiel du robot, puisqu'elle supporte l'ensemble des systèmes, y compris les roues et tous les mécanismes supérieurs.

Notre base, ainsi que la majorité des autres composants, est fabriquée à partir de profilés métalliques *GoBilda®*, choisis pour leur rigidité, leur légèreté et leur modularité, ce qui facilite l'assemblage et la maintenance.

Après l'assemblage de ce module essentiel, on peut y rajouter les roues. Nous avons choisi d'utiliser des roues *GripForce™* (SKU 3625-0202-0104) par *GoBilda®*, qui permettent au robot de se déplacer latéralement en faisant tourner les roues de chaque côté en sens opposés.

De plus, nous utlisions des moteurs (SKU 5203-2402-0019) assemblés perpendiculairement pour alimenter les quatres roues en utilisant le moins de place possible. Voici à quoi cela ressemble :

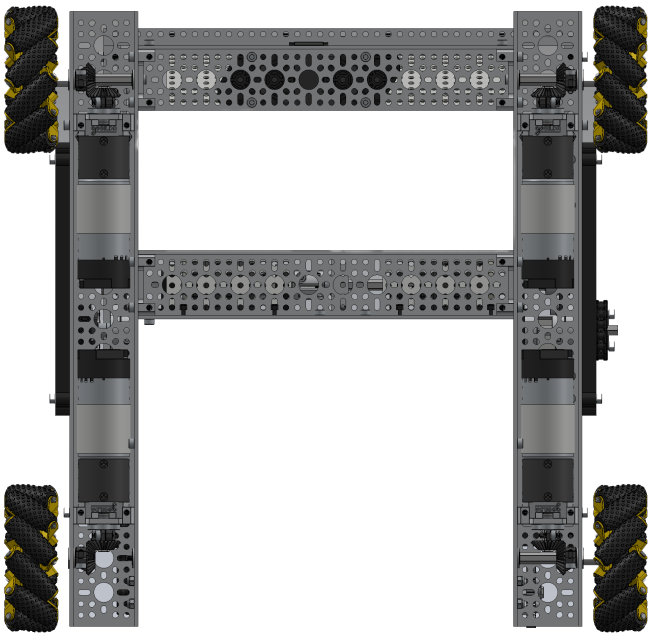


FIGURE 2.2 – Plateforme de base du robot après installation des roues et moteurs

2.1.2 Entrée (« intake »)

L'« intake » permet, comme son nom l'indique, de faire entrer des *artifacts* dans le module de stockage et de triage, appelé le « buffer ». Il a été conçu pour ramasser les *artifacts* directement au sol et les acheminer rapidement dans le buffer.

Il est composé de deux axes horizontaux :

— un axe avec des roues *Gecko™ GoBilda®* de 32 mm (SKU 3613-4008-0032) ;
— un axe avec des roues d'intake de 72 mm (SKU 3615-4008-0072) reliées à un moteur 312 rpm via une chaîne.

Problèmes connus

Il est presque impossible de faire entrer plusieurs *artifacts* de manière successive et rapide, en raison des limites de vitesse du servo sélecteur, qui assure le tri des *artifacts*.
Il est donc recommandé d'éviter de ramasser les *artifacts* trop rapidement afin de prévenir leur éjection accidentelle, le blocage de l'intake, ou même la détérioration d'un servo.

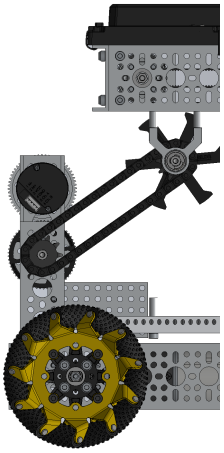


FIGURE 2.3 – Intake du robot, vue de côté

2.1.3 « Buffer » et lanceur

Le buffer permet de temporairement bloquer un ou plusieurs *artifacts* dans le robot. Cela est généralement réalisé dans les cas suivants :

- Attendre que le robot se positionne vers le but ;
- Attendre que la roue libre du lanceur (« flywheel ») accélère ou ralentisse pour atteindre sa vitesse cible ;
- Attendre la confirmation du driver pour tirer ;
- Attendre la confirmation de l'ordinateur pour assurer un tir réussi.

Une fois la décision prise, un de deux servos s'activera, qui poussera l'artifact dans le lanceur.

Problèmes connus

La pièce qui propulse les *artifacts* dans le lanceur peut parfois se bloquer. La cause exacte n'est pas encore déterminée, mais cela pourrait être lié aux trous présents dans les *artifacts*. En cas de blocage, il suffit généralement d'arrêter puis de redémarrer le servo une ou plusieurs fois pour que l'*artifact* soit correctement éjecté.

Le lanceur permet, comme son nom l'indique, de lancer les *artifacts* le plus précisément possible dans le but. Il consiste d'une flywheel accélérée par deux moteurs 6000 rpm (SKU 5203-2402-0001) et d'une rampe.

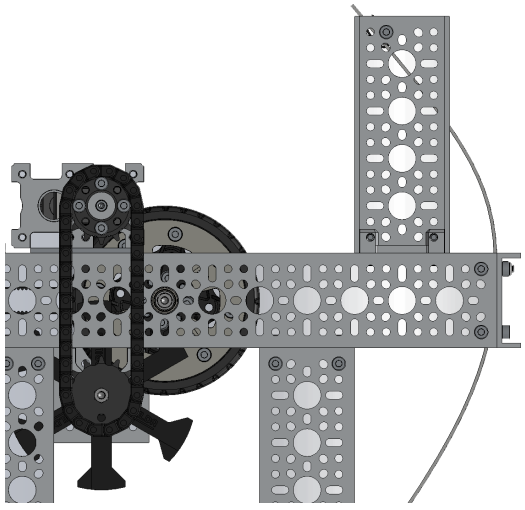


FIGURE 2.4 – Lanceur du robot

2.2 Diagrammes et CAD

Des diagrammes pour le robot sont disponibles. Ceux-ci sont distribués au format .easm, qui peut être lu par SOLIDWORKS ou eDrawings.

Comment obtenir ces logiciels ?

SOLIDWORKS est une application payante qui permet de modifier le fichier, il n'est donc pas recommandé de l'utiliser uniquement pour la lecture de ce fichier.
En parallèle, eDrawings Viewer et un logiciel de la même entreprise qui permet de lire ce fichier et obtenir les références de toutes les pièces. Il est recommandé d'utiliser la version Windows pour avoir plus de performance.
Téléchargez le ici : `https://www.edrawingsviewer.com/download-edrawings`.

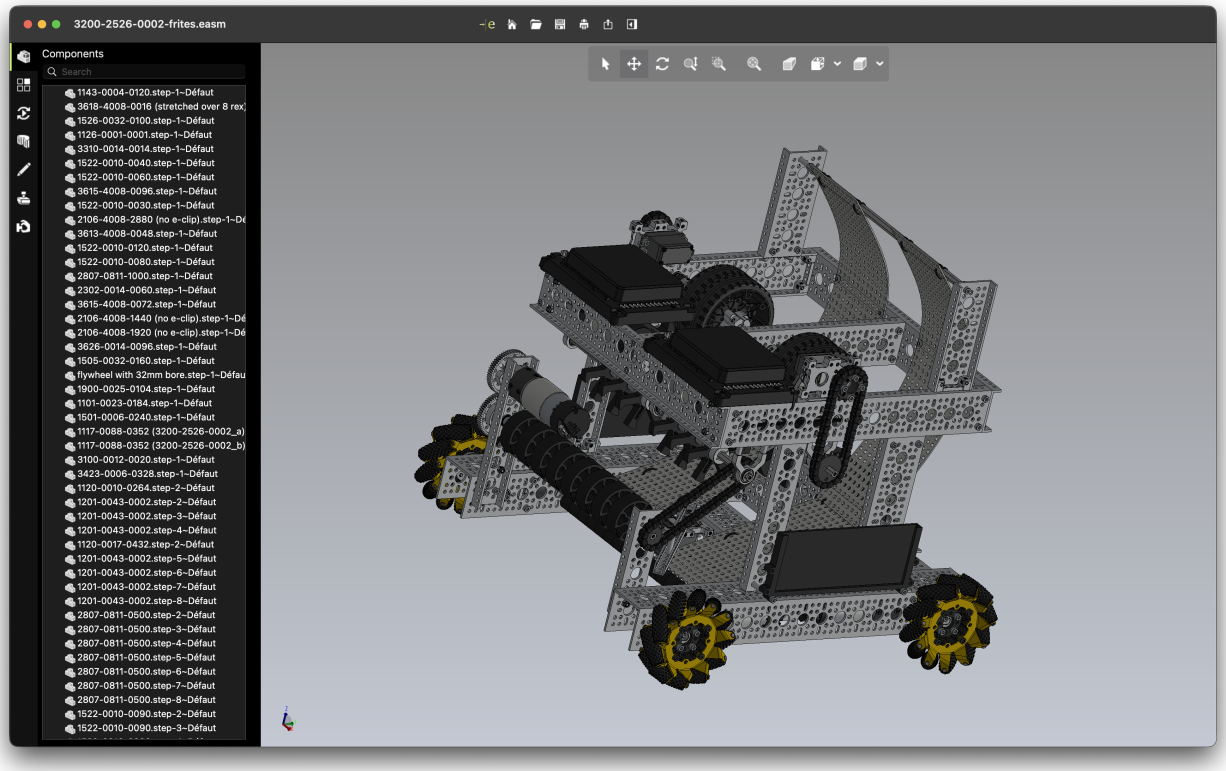
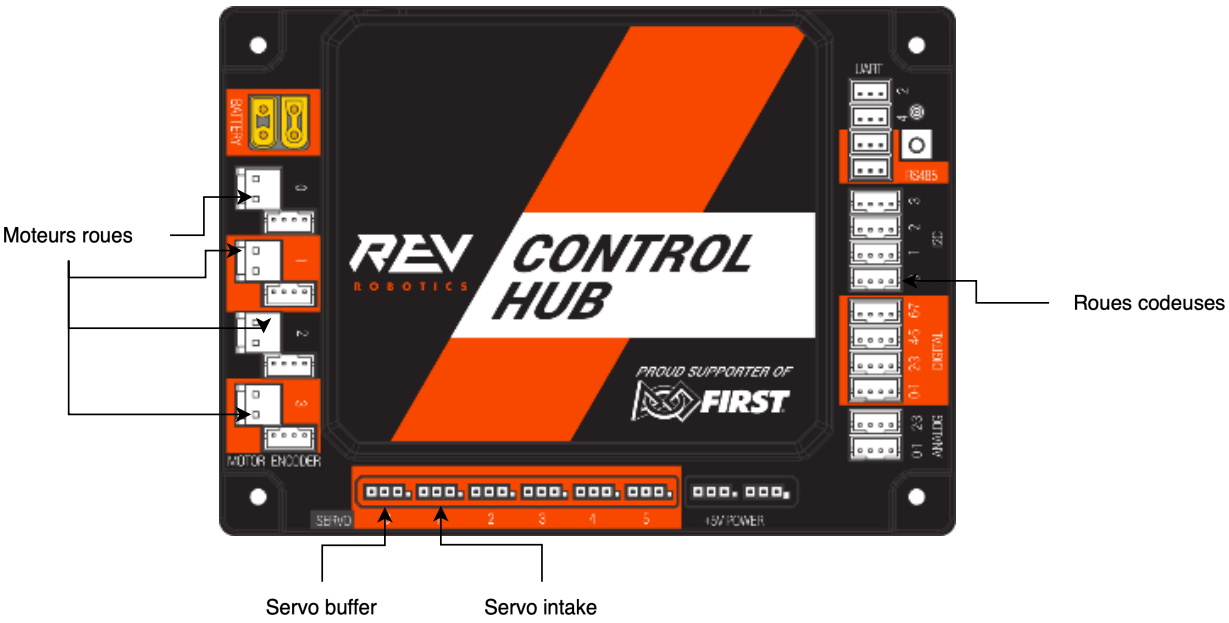


FIGURE 2.5 – Robot prévisualisé dans le lecteur eDrawings

Chapitre 3

Système Électrique

3.1 Architecture électrique



Note sur le schéma électrique

Ce schéma représente l'architecture électrique générale du robot, mais il n'est pas obligatoire de le suivre exactement. L'important est que tous les composants soient correctement connectés et que chaque élément soit correctement déclaré et référencé dans le *Driver Station*.

Chapitre 4

Programmation

4.1 Environnement et outils

4.1.1 Outils nécessaires et recommandés

La base de code du robot est écrite en Java, avec l'utilisation des bibliothèques officelles *First®*, qui permettent une interface facilitée avec les différents composants du robot. Le FTC SDK peut être retrouvé ici : `https://github.com/FIRST-Tech-Challenge/FtcRobotController`.

FTC SDK

Le SDK officel et open source de *First®* est disponible ici : `https://github.com/FIRST-Tech-Challenge/FtcRobotController`.

L'équipe a recyclé le code de l'année dernière pour gagner du temps et éviter de re-tomber sur des bugs des années précédentes. C'est pour cela qu'il est **nécessaire d'utiliser Java 21** pour compiler le code.

De plus, il est fortement recommandé d'utiliser un IDE pour pouvoir compiler et publier le code facilement. Nous recommandons l'utilisation de **Android Studio**, car il contient de nombreuses fonctionnalités avancées permettant d'accélérer le développement, mais permet aussi de compiler et publier son code facilement. Pour cela, **le reste de la documentation assumera l'utilisation d'Android Studio**.

Résumé des outils nécessaires

Voici les outils nécessaires pour le développement et la programmation :

- Java Development Kit (JDK) **21**
- Android Studio

L'utilisation de ces éléments avec les bonnes versions est essentiel. Le reste de la documentation assumera leur bonne utilisation.

4.1.2 Préparation de l'environnement

Comme indiqué précédemment, **il faut obligatoirement avoir JDK 21 et Android Studio**. Il faut aussi avoir un access au dépot git sur Gitlab (Pour vérifier : `https://gitlab.com/ftc-civ/frites/2026`).

Téléchargement du code source

Ceci peut être réalisé de plusieures manières. La plus simple est d'utiliser la fonctionnalité git dans Android Studio, qui téléchargera et préparera automatiquement le projet.

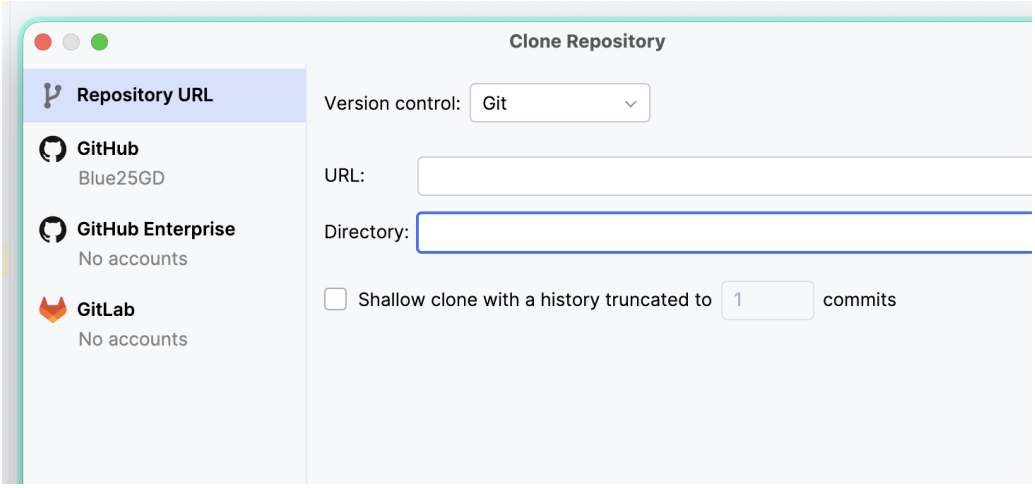


FIGURE 4.1 – Capture d'écran de la fenêtre clone dans Android Studio

Il est aussi possible d'utiliser directement l'outil de CLI `git` :

```
git clone git@gitlab.com:ftc-civ/frites/2026.git
```

Initialization

Dans des conditions normales, Android Studio prépare automatiquement le projet avec Gradle, qui téléchargera toutes les bibliothèques automatiquement. Dans ce cas, cette étape n'est pas nécessaire.

Sinon, il faut cliquer sur le bouton « Download sources » dans l'onglet Gradle.

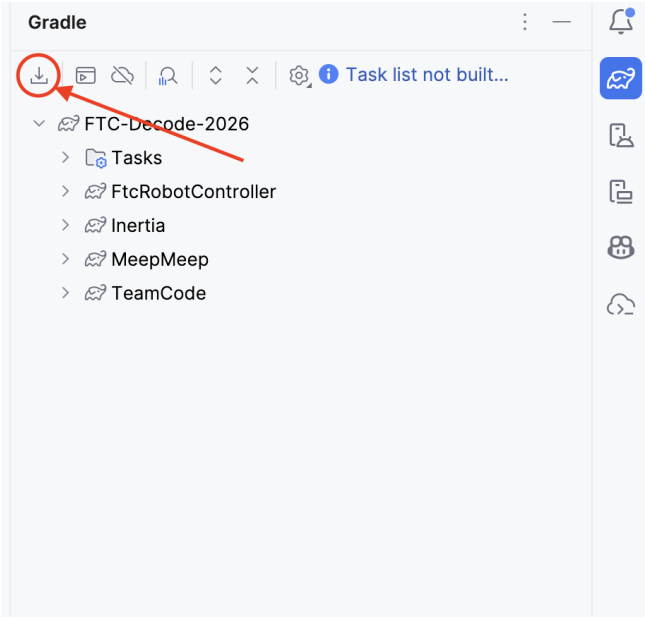


FIGURE 4.2 – Menu Gradle dans Android Studio

Compilation et publication

Une fois des modifications éffectuées, il convient de compiler le code. Pour cela, il faut appuyer sur l'icone « build » (représentée par un marteau) pour compiler le code.

Pour publier le code il faut premièrement connecter le control hub du robot à son ordinateur. Cela peut se faire de deux façons :

- **Par USB** (uniquement sous Windows) en branchant directement le control hub à l'ordinateur.
- **Par Wi-Fi**, méthode généralement plus simple et plus fiable, en se connectant au réseau Wi-Fi du control hub, puis en utilisant la commande `adb connect` sur l'adresse IP du routeur, en général `192.168.43.1`. Une fois la connexion établie, il est alors possible de publier le code vers le robot via Android Studio.

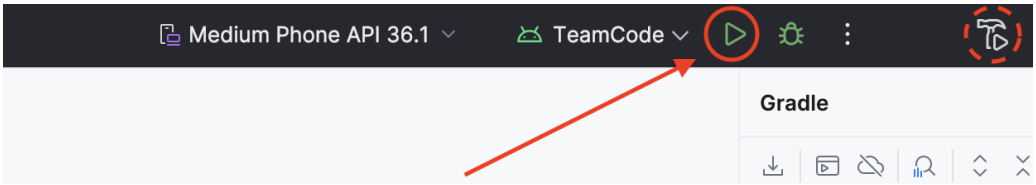


FIGURE 4.3 – Capture d'écran des boutons « Build » et « Start »

Il suffit ensuite de cliquer sur le bouton « Start » dans Android Studio, après s'être assuré que la bonne cible (« target ») est sélectionnée et que le robot est correctement connecté. Le transfert peut parfois prendre une à deux minutes avant d'aboutir.

4.2 Architecture logicielle

4.2.1 Conventions internes

Les conventions internes définissent les règles à suivre pour garantir un code lisible, cohérent et maintenable. Elles sont regroupées par thèmes pour faciliter la lecture.

Nommage des éléments

- **Classes :** `PascalCase` (ex : `DriveSubsystem`, `IntakeMotor`)
- **Méthodes / fonctions :** `camelCase` (ex : `setPower()`, `runIntake()`)
- **Variables :** `camelCase` pour les variables locales, `UPPER_SNAKE_CASE` pour les constantes
- **Packages / dossiers :** minuscules, sans espaces (ex : `subsystems`, `opmodes`)
- **Boutons de gamepad :** noms explicites (ex : `buttonA`, `leftTrigger`)

À retenir

Un bon nommage facilite la lecture et la maintenance du code, surtout lorsque plusieurs développeurs travaillent sur le même projet. Il est toujours mieux d'écrire des noms d'objets très longs plutôt que quelque chose d'ambigu.

Organisation des fichiers

- Une classe par fichier, nom du fichier = nom de la classe
- Ordre recommandé dans le fichier :
  1. Déclarations des constantes
  2. Variables d'instance
  3. Constructeur / initialisation
  4. Méthodes publiques
  5. Méthodes privées / utilitaires

Documentation et commentaires

- Chaque sous-système et méthode publique doit être commenté
- Expliquer le rôle des paramètres et des valeurs retournées
- Utiliser `TODO` et `FIXME` pour signaler le code en développement

Bonnes pratiques

- Initialiser le matériel dans `init()` ou dans le constructeur de la classe Hardware
- Respecter les limites physiques et logicielles des moteurs et servos
- Préférer des méthodes courtes et claires plutôt que longues et complexes
- Tester chaque sous-système indépendamment avant de l'intégrer aux OpModes

Astuce pratique

Avant chaque commit Git, vérifier que le code respecte les conventions de nommage et de formatage.

Formatage et style

- Indentation : 4 espaces
- Accolades : sur la même ligne pour les méthodes et les blocs
- Espaces autour des opérateurs
- Longueur de ligne recommandée : maximum 120 caractères

4.2.2 Structure générale

Le projet est organisé de manière modulaire pour séparer clairement les responsabilités et faciliter la maintenance. Sa structure est basée sur le FTC SDK. Chaque dossier a un rôle précis, décrit ci-dessous.

Racine du projet

- `build.gradle`, `settings.gradle`, `gradle.properties` : fichiers de configuration du projet et de Gradle
- `gradlew`, `gradlew.bat` : scripts pour exécuter Gradle localement
- `libs` : bibliothèques et keystore pour le déploiement sur Android
- `images` : ressources graphiques pour le manuel ou l'interface

FtcRobotController

FtcRobotController

La librairie officielle de *First®* qui contient l'application Android du robot, les OpModes d'exemple et les classes internes nécessaires à l'exécution du robot sur le téléphone. C'est la base sur laquelle tout le code spécifique de l'équipe s'appuie.

Contient l'application principale du robot. Il ne doit pas être modifié.

- `src/main/java/org/firstinspires/ftc/robotcontroller` : code source principal, incluant :
  - `external/samples` : exemples d'OpModes et de gestion du hardware
  - `external/externalhardware` : classes de configuration de hardware personnalisées
  - `internal` : classes internes pour l'application, comme l'enregistrement des OpModes et les activités Android
- `src/main/res` : ressources Android (layouts, drawables, menus, sons)

TeamCode

C'est ici que se trouve le code spécifique à l'équipe.

- `src/main/java/core` : code principal du robot, structuré en sous-dossiers :
  - `localization` : gestion de la position du robot (Odometry, Limelight)
  - `logic` : actions et gestion du terrain
  - `math` : classes utilitaires pour la géométrie et les unités
  - `modules` : sous-systèmes du robot (Intake, Cannon, GamepadController, etc.)
  - `opmodes` : OpModes autonomes et téléop, avec versions par équipe/couleur

  Roadrunner

  Une librairie de trajectoire et de cinématique pour robots FTC. Elle fournit des localisateurs (localizers), des calculs de trajectoire et des abstractions pour différents types de châssis (tank, mecanum, etc.). Elle est utilisée pour la navigation précise et l'exécution des trajectoires autonomes.

  - `roadrunner` : code de trajectoire et localisateurs
  - `debug` : outils de test et d'inspection
  - `messages` : classes pour échanger des données entre modules
- `src/main/assets/trajectory` : fichiers de trajectoire pour les OpModes autonomes
- `src/main/res` : ressources additionnelles pour TeamCode

Modules supplémentaires

- `Inertia` : librairie interne pour la sérialisation, le stockage et le replay des données
- `MeepMeep` : simulation et visualisation des trajectoires (hors robot réel)

Résumé

Cette organisation permet de séparer clairement :

- le code de base du SDK (`FtcRobotController`) ;
- le code spécifique à l'équipe (`TeamCode`) ;
- les bibliothèques et outils de simulation (`Inertia`, `MeepMeep`) ;
- et les ressources (assets, sons, images, configurations Android).

Elle facilite la réutilisation de code et le débogage.

4.2.3 Systèmes de coordonnées

Préface

Veuillez lire attentivement ce document afin d'éviter des bogues subtils liés à des systèmes de coordonnées non correspondants.

La base de code utilise plusieurs systèmes de coordonnées avec différentes directions d'axes, origines et sens de rotation. Ces différences sont imposées par des bibliothèques tierces (par exemple, le SDK FTC, Road Runner, les pilotes matériels) et ne peuvent pas être modifiées.

Mélanger des systèmes de coordonnées sans conversion appropriée entraîne des bogues difficiles à déboguer et à comprendre, compromettant tout le système de navigation. Ce document vise à clarifier les systèmes de coordonnées et à établir des directives strictes pour éviter tout problème.

Directives

Pour éviter toute confusion et bogue, nous adoptons les directives suivantes. Consultez la section 4.2.3 pour les détails sur chaque système de coordonnées.

- Toute la logique interne doit utiliser uniquement le **système de coordonnées standard**.
- Les systèmes de coordonnées spécifiques aux bibliothèques sont autorisés **uniquement aux frontières**, c'est-à-dire lors de l'interface en entrée ou en sortie avec une bibliothèque. Ils doivent être convertis en système de coordonnées standard avant toute utilisation ultérieure. Toute violation de cette règle doit être clairement documentée.
- **Centraliser** toutes les conversions de systèmes de coordonnées dans des fonctions utilitaires définies dans les classes `Pose2D` (ou classes connexes).
- Les systèmes de coordonnées des nouvelles bibliothèques positionnelles doivent être **documentés** dans ce fichier.

Systèmes de coordonnées

Système de coordonnées standard

- Origine : Centre du terrain
- Axe X : X positif pointant vers le côté du public
- Axe Y : Y positif pointant vers le côté de l'alliance bleue
- Rotation zéro : Direction de l'axe +X
- Sens de rotation : Rotation positive dans le sens anti-horaire
- Unités : Toutes unités, utilisées via des classes auto-convertibles (par ex. `Pose2D`, `Distance`, etc.)

Système de coordonnées FTC

- Origine : Centre du terrain
- Axe X : X positif pointant vers le côté du public
- Axe Y : Y positif pointant vers le côté de l'alliance bleue
- Rotation zéro : Direction de l'axe +X
- Sens de rotation : Rotation positive dans le sens anti-horaire
- Unités : Pouces, degrés. Elles sont généralement auto-converties lors de l'accès.

Système de coordonnées Road Runner Relatif au robot

- Origine : Centre du robot
- Axe X : X positif pointant vers la droite du robot

— Axe Y : Y positif pointant vers l'avant du robot
— Rotation zéro : Direction de l'axe +Y
— Sens de rotation : Rotation positive dans le sens anti-horaire
— Unités : Radians. Il n'impose pas d'unité de distance, nous utilisons donc des classes auto-convertibles (par ex. `Pose2D`, `Distance`, etc.)

Système de coordonnées du pilote Gobilda Pinpoint Peut fonctionner en mode absolu ou relatif.

Mode absolu Le mode absolu utilise le système de coordonnées FTC.

Mode relatif Le mode relatif est défini comme suit :
— Origine : Centre du robot
— Axe X : X positif pointant vers l'avant du robot
— Axe Y : Y positif pointant vers la gauche du robot
— Sens de rotation : Rotation positive dans le sens anti-horaire
— Unités : Pouces, degrés. Elles sont généralement auto-converties lors de l'accès.

4.3 Tests et débogage

Avant de lancer un OpMode complet, il est important de tester le robot en conditions réelles pour s'assurer que chaque sous-système fonctionne correctement. Cette section présente les méthodes de test et les outils de débogage recommandés pour le développement et la compétition. Les tests peuvent se faire en mode TeleOp, avec des OpModes autonomes simplifiés, ou à l'aide d'outils spécifiques pour vérifier les moteurs, servos et capteurs.

4.3.1 Modes TeleOp

Les **modes TeleOp** (ou téléopération) sont des programmes permettant de contrôler le robot manuellement via les gamepads. Ces modes sont utilisés pour tester le robot, ajuster les mécaniques en conditions réelles et pour la partie du match où le pilote prend le contrôle.

TeleOp

Un *TeleOp* est un `OpMode` qui lit les entrées des gamepads et agit en conséquence sur les moteurs, servos et autres composants. Contrairement aux OpModes autonomes, il ne suit pas de trajectoire prédéfinie.

Écrire un TeleOp simple

Voici un exemple minimaliste d'OpMode TeleOp pour contrôler un moteur à l'aide du joystick gauche :

```java
package core.opmodes;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;

@com.qualcomm.robotcore.eventloop.opmode.TeleOp(name="SimpleTeleOp")
public class SimpleTeleOp extends LinearOpMode {
    private DcMotor leftMotor;

    @Override
    public void runOpMode() {
        leftMotor = hardwareMap.get(DcMotor.class, "left_motor");
        waitForStart();

        while (opModeIsActive()) {
            double power = -gamepad1.left_stick_y;
            leftMotor.setPower(power);
        }
    }
}
```

À retenir

- Toujours initialiser le hardware avant `waitForStart()`.
- Utiliser `opModeIsActive()` pour que la boucle se termine si l'OpMode est stoppé.
- Lire les valeurs du gamepad et les convertir en puissance ou commandes pour les moteurs/servos.

OpModes TeleOp disponibles

Le robot dispose de plusieurs OpModes, chacun ayant un rôle spécifique selon le type de match ou de test. Ces OpModes sont regroupés dans `core.opmodes` et `debug`, avec des sous-dossiers pour les versions automatiques, manuelles et de calibration.

OpModes principaux

- **AutoOpMode.java** : OpMode autonome générique. Utilise la classe `DriveActions` et le localisateur pour effectuer des déplacements automatiques selon l'état initial et la stratégie choisie.
- **CannonCalibrationOpMode.java** : OpMode utilisé pour calibrer le canon du robot. Il permet de régler la puissance et la position pour les tirs précis.
- **ManualOpMode.java** : Base pour le contrôle manuel du robot en TeleOp, généralement étendu par les versions spécifiques `Blue` et `Red`.

Versions spécifiques Les sous-dossiers `versions/auto`, `versions/manual` et `versions/calibration/cannon` contiennent des OpModes adaptés aux situations suivantes :

- `versions/auto/Blue.java`, `Red.java` : Stratégies autonomes spécifiques aux couleurs de l'alliance. Exécutent des séquences de déplacement et de manipulation des éléments du terrain.

— `versions/manual/Blue.java`, `Red.java` : Versions TeleOp manuelles avec pose tracking et contrôle complet des modules du robot.
— `versions/manual/Blue_No_Pose_Calculation.java`, `Red_No_Pose_Calculation.java` : Versions TeleOp simplifiées, sans calcul de la pose robotique. Plus légères pour tester les modules indépendamment.
— `versions/calibration/cannon/Blue.java`, `Red.java` : OpModes de calibration du canon, adaptés à chaque alliance.

OpModes de débogage Dans `debug`, plusieurs OpModes permettent de tester ou de calibrer des composants spécifiques :
— **CameraLocalizerOpMode.java** : Test et calibration du système de localisation via caméra.
— **ColorSensorDebugger.java** : Vérification et debug des capteurs de couleur.
— **DcMotorDebugger.java** : Test individuel des moteurs DC pour s'assurer de leur fonctionnement.
— **DistanceSensorDebugger.java** : Vérifie les capteurs de distance.
— **IndependentMotorTest.java** : Test des moteurs séparément.
— **OdometryOpMode.java** : Permet de tester l'odométrie du robot.
— **PinpointLocalizerOpMode.java** : Test du localisateur de précision.
— **ServoTest.java** : Vérifie le fonctionnement des servos du robot.

Résumé Chaque OpMode a un objectif précis :
— Les OpModes génériques (`AutoOpMode`, `ManualOpMode`) servent de base.
— Les versions spécifiques aux alliances ou à la calibration adaptent le robot à la situation réelle.
— Les OpModes de debug sont utilisés pour tester et calibrer les modules et capteurs.

Conseil pratique

Pour tester un nouveau TeleOp, il est recommandé de commencer par un mode simple (un seul moteur ou un sous-système), puis d'ajouter progressivement d'autres composants.

Bonnes pratiques pour les TeleOp

— Toujours prévoir un arrêt d'urgence (bouton ou combinaison) pour stopper tous les moteurs rapidement.
— Logger les données critiques avec `Telemetry` pour détecter rapidement des erreurs ou des comportements inattendus.
— Isoler les tests par module : tester d'abord le châssis, ensuite les bras, servos et capteurs.

4.3.2 Utilisation du FTC Dashboard

Le **FTC Dashboard** est un outil puissant permettant de visualiser en temps réel l'état du robot, de modifier certaines variables et de faciliter le débogage sans avoir besoin de publier du code à chaque changement.

FTC Dashboard

Le FTC Dashboard est une interface web qui se connecte au robot via le réseau Wi-Fi. Il permet de visualiser les capteurs, les moteurs, et les données personnalisées envoyées par le robot en temps réel. Pour plus d'informations et la documentation officielle : `https://acmerobotics.github.io/ftc-dashboard/`.

Installation et configuration

Pour utiliser le FTC Dashboard, il faut l'ajouter au projet comme une dépendance. Cette étape est généralement déjà faite dans la base de code fournie, mais voici comment la vérifier :

1. Ouvrir le fichier `build.gradle` du module `:app`.
2. Vérifier que la ligne suivante est présente dans la section `dependencies` :

```
implementation 'com.acmerobotics:dashboard:3.7.0'
```

3. Synchroniser le projet avec Gradle pour télécharger la bibliothèque.

Remarque

Assurez-vous que le robot et l'ordinateur ou smartphone utilisé pour le Dashboard sont connectés au même réseau Wi-Fi, sinon le Dashboard ne pourra pas communiquer avec le robot.

Télémetrie dans le Dashboard

Le Dashboard peut être utilisé directement depuis le code du robot. Dans la plupart des programmes, il suffit d'ajouter les lignes suivantes dans un opmode :

```
import com.acmerobotics.dashboard.FtcDashboard;

FtcDashboard dashboard = FtcDashboard.getInstance();
telemetry = dashboard.getTelemetry();
```

Ensuite, démarrez l'opmode sur le robot. Le Dashboard sera accessible via un navigateur web, généralement à `http://<IP_DU_ROBOT>:8080/dash`.

Fonctionnalités principales

Le Dashboard offre plusieurs fonctionnalités utiles pour le développement et le débogage :
— Visualisation en temps réel des capteurs et moteurs.
— Modification de variables *live* grâce aux *configurables*.
— Visualisation de graphiques et de courbes des données envoyées par le robot.
— Interface web accessible depuis n'importe quel appareil connecté au même réseau.

Conseil d'utilisation

Lors du développement, il est pratique d'envoyer régulièrement les valeurs importantes du robot via `telemetry.addData()`. Ces valeurs seront automatiquement visibles sur le Dashboard.

Chapitre 5

Stratégie et Opérations

Attention !

Ce chapitre est en cours de fabrication ; il est donc recommandé de le lire avec prudence, car certaines informations peuvent être incomplètes ou sujettes à modification.

5.1 Scénarios de match

Manque de données

Cette section est incomplète à cause d'un manque de données lié au faible nombre de tests et de matchs réalisées.

5.2 Séquences autonomes

5.3 Gestion des erreurs et dépannage