TEAM LEBOB #3236

FIRST LEGO LEAGUE UNEARTHED · WESTERN AUSTRALIAN NATIONAL CHAMPIONS

Robot Design

Andre Nijman • Oliver Liu • Sean Chan • Kingsley Wong
Chris Wang • Subesh Sukumaran • Aaron Zhang • Leven Shi

Contents

Identify

Our Mission Strategy

Mission No. and Name		Distance	Programming	Mechanical
1 Surface Brushing	30 (10+10+10)	1	1	3 (for left and right) 2 (for removing the stick)
2 Map Reveal	30 (10+10+10)	3	4 (for all three) 3 (for 2 only) 2 (for only the rotate one)	2
3 Mineshaft Explorer	40 (30+10(technically))	5	4	2
4 Careful Recovery	40 (30+10)	5	5 (with standing) 3 (without)	5 (with standing) 3 (without)
5 Who Lived Here?	30	3	2	1
6 Forge	30 (10+10+10)	3	3	1
7 Heavy Lifting	30	5	3	3
8 Silo	30 (10+10+10)	1	4	2
9 What's on Sale?	30 (20+10)	1.5	2	3
10 Tip the Scales	30 (20+10)	2	4	2.5
11 Angler Artefacts	30 (20+10)	2	4 (time wise)	3
12 Salvage Operation	30 (20+10)	1	3	2
13 Statue Rebuild	30	3	4	2
14 Forum (Might do some only)	35 (5+5+5+5+5+5+5)	10	5	5
15 Site Marking (Won't unless on the way)	30 (10+10+10)	7	5	3

MAX RUN LINK: https://www.youtube.com/watch?v=c62qryVwEQw

Green: Might Do

Red: see again

Blue: Will DO

None: won't do

MECHANISMS for the missions we are doing

Mission Number	Name	Dist	Side	Diff
4	Artefact	4	L	5
7	Heavy	5	R	5
11	Raised	1	R	5
11	Flag	1	R	5
14	Forum items	2	L	5
1	Brush	1	L	4
2	Map	3	L	4
9	Roof	2	R	4
6	Forge	3	R	3
10	Pan	2	R	3
5	Flip	3	R	2
8	Silo	1	R	2
10	Bucket	2	R	2
13	Statue	4	L	2
1	Push	1	L	1
3	Your minecart	4	L	1
9	Market	2	R	1
12	Sand	1	L	1
12	Ship	1	L	1
15	Site flag	5	A	1
3	Other minecart	4	L	0
4	Support	0	L	0

Figure 1: *Nationals mission planning (left) and Internationals planning (right)*

Figure 1 shows our first stage of mission planning where we come together as a team to list all components of the robot game and rate each in **distance** and **difficulty**. (a combination of programming and mechanical difficulty) Each were colour-coded, green as easy and red and hard.

The purpose of the side column is so we could figure out which zone to launch the robot from to complete that mission, saving time. If a mission was difficult it was usually because either there were many actions/programming steps to complete it, or that the mechanism required to complete it needed to be complicated/large. (more than just an arm)

This was important since then we could decide which missions to do that were worth our time during early stages, and in what order. We also took into account how much each mission was worth in points and their proximity. For example, we combined missions near each other into one run.

Making the Plan

1. We looked at the hardest missions first, which included **M04:** Mission Recovery.
2. To save time, we did **M03:** Minecart Explorer at the same time, since it was close by and simple.
3. We repeated these steps over all of our runs, grouping missions into 7 runs, eventually coming up with a mostly optimised plan.
4. After repeated testing, we made small modifications to our plan to make it more efficient, resulting in multiple iterations over time.

Runs

We colour coded our runs based on an array of colours from the **rainbow**. We also coloured the starting positions of the robot in the home bases to ensure clarity, In the internationals game plan we also added a key.

Version 1 Version 2 Version 3

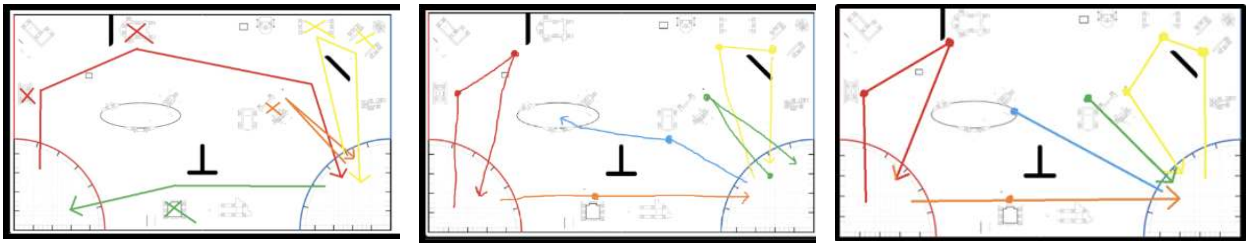


Figure 2: *Regionals Run Planning*

Version 1 Version 2

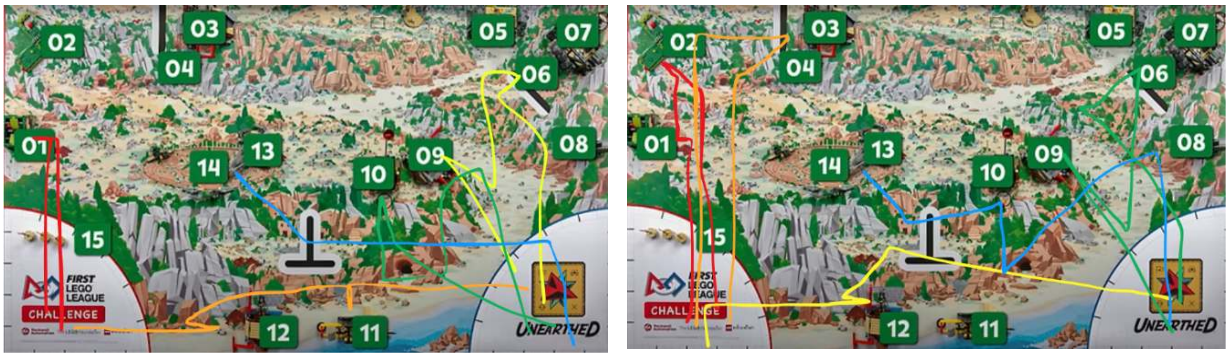


Figure 3: *Nationals Run Planning*

Version 1 Version 2

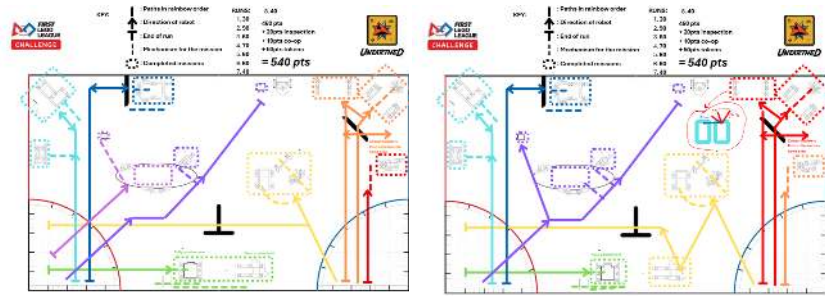


Figure 4: *Internationals Run Planning*

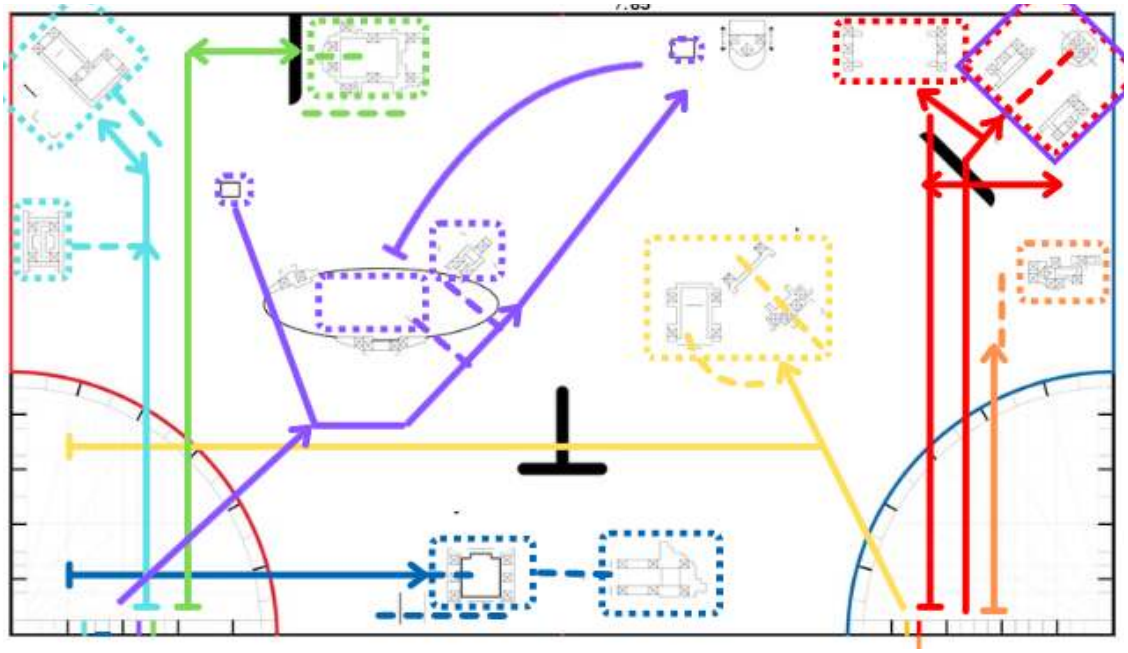


Figure 5: *Final Run Plan, 545 Points*

Figures 2 to 4 show how our plan has changed over different stages of the competition. Figure 5 shows our final plan, showing how we achieved our goal of max (545) points.

To achieve this, we tried many different runs and different mechanisms over time, as we wanted to reduce time spent and achieve max points (545) as our final goal. At the start of the season, we decided on a simpler, achievable plan (below max) for regionals, and progressively increased the standard over the season towards internationals.

For the international plans, we first used version one of the plan which had too many separate launches, which after testing proved to take too much time.

- Later, we decided on version two, which combined our launches to increase speed.
- Although we were quite happy with the final result, after testing we found some missions were too long, so our third version grouped missions differently to increase speed, for example combining both **Angler Artifacts** and **Careful Recovery**.

Coding Resources

These are resources we used for programming our robot. They were either used as a way to learn code or look up python definitions.

Pybricks/Python Documentation (Definitions)

Used to look up function definitions and different methods of controlling the robot through programs. Some examples of us using it include:

- For the heavy mission, the robot could not lift the arm since it was very long and was carrying a heavy object. We referred to the documentation of PyBricks to see if there were any functions which allowed us to move the motor with more force. We found a dc function, (duty cycle) which could send more power (up to 100%), allowing us to complete the mission.
- When first deciding on the programming language we were going to use to program the robot, the PyBricks (Figure 6) and Python (Figure 7) documentation clearly described

what the system could provide for our robot, which allowed us to quickly decide on python programming as a team.

- Runs directly on the hub itself rather than relying on a tablet or computer for more reliability and speed because it doesn't need to rely on external sources like computers.
- Provides advanced models like drivebase and gyro drive base which delivers extremely accurate navigation.
- All our team members knew python from school, meaning it was not a large skill jump to use it with python.

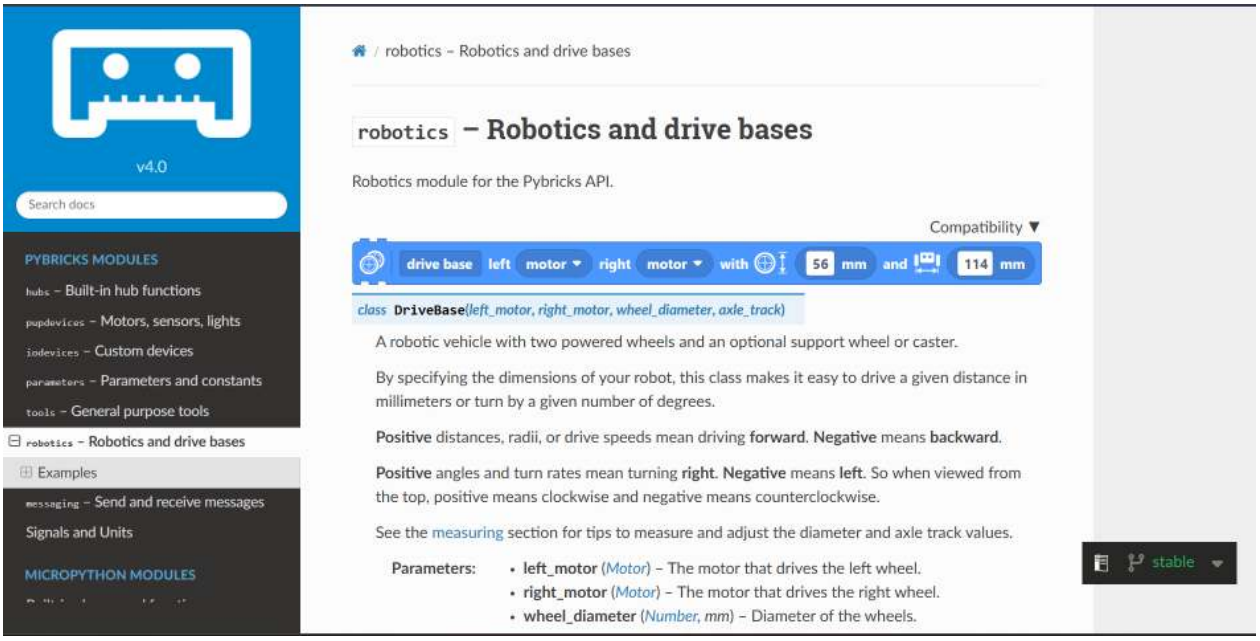


Figure 6: *Example of Pybricks Documentation (https://docs.pybricks.com/en/stable/)*

Figure 7: *Example of Python Documentation (https://docs.python.org/3/)*

As shown in Figure 7, python documentation allowed us to understand, explore, and effectively use python by providing detailed explanations of its syntax, functions, libraries, and modules along with examples and guidelines.

Advanced Resources

Another aspect of our team's coding skills comes from participation in a FRC robotics team which uses C++ and java to program robots. Many of our team members learnt how to program in those languages, which also provided a lot of knowledge in programming that those members could provide others with. An example of this is our knowledge of Object-Oriented Programming (OOP), where we use classes in our robot code because we learnt the base concepts through learning java, which centres around OOP.

```
public class Drive extends SubsystemBase {
  // TunerConstants doesn't include these constants, so they are declared locally
  static final double ODOMETRY_FREQUENCY = TunerConstants.kCANBus.isNetworkFD() ? 250.0 : 100.0;
  public static final double DRIVE_BASE_RADIUS =
      Math.max(
          Math.max(
              Math.hypot(TunerConstants.FrontLeft.LocationX, TunerConstants.FrontLeft.LocationY),
              Math.hypot(TunerConstants.FrontRight.LocationX, TunerConstants.FrontRight.LocationY)),
          Math.max(
              Math.hypot(TunerConstants.BackLeft.LocationX, TunerConstants.BackLeft.LocationY),
              Math.hypot(TunerConstants.BackRight.LocationX, TunerConstants.BackRight.LocationY)));

  static final Lock odometryLock = new ReentrantLock();
  private final GyroIO gyroIO;
  private final GyroIOInputsAutoLogged gyroInputs = new GyroIOInputsAutoLogged();
  private final Module[] modules = new Module[4]; // FL, FR, BL, BR
  private final SysIdRoutine sysId;
```

Figure 8: *Our FRC team's code: https://github.com/CurtinFRC/2026-Offseason/*

Grok Academy (Learning)

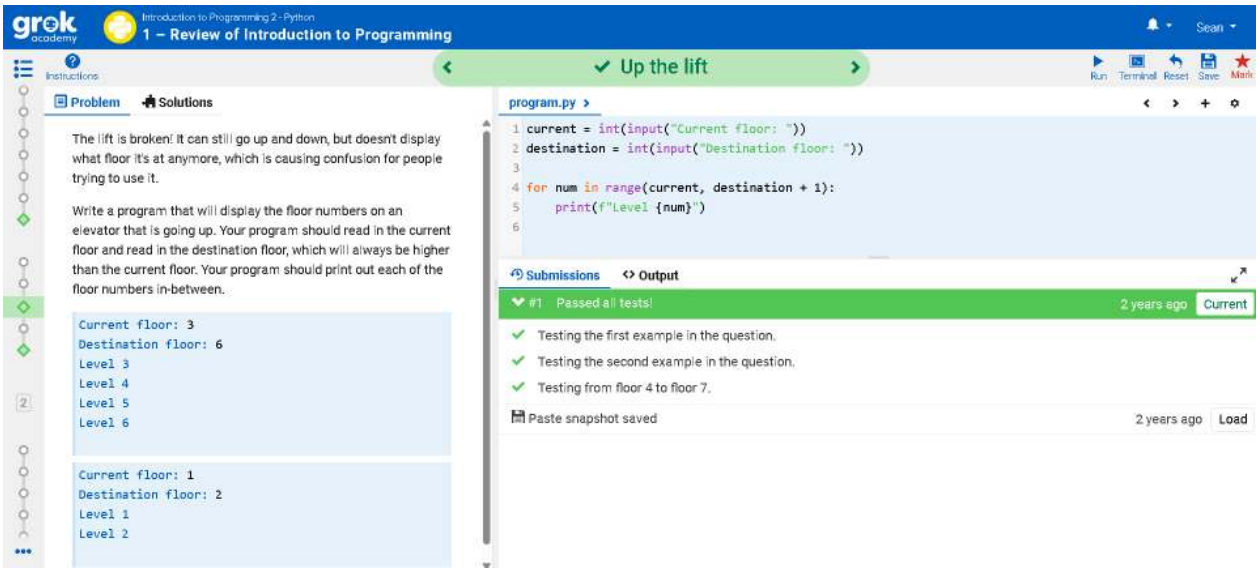


Figure 9: *Example of Grok Academy Usage (https://groklearning.com/)*

For learning how to code in python, many of our team members used Grok Academy in school and in their free time, improving their coding skills and also allowing everyone to know how to control the robot.

Building Resources

Building resources that we used to help us build our robot include watching YouTube videos on the missions so we could construct the routes for our mission plan, and using brick studio and its documentation to CAD our lego mechanisms. We also talked to more experienced teams that achieved high points in robot games from our state, such as BrickToThe Future and TidalTumble Robotics. (an international team, see figure 11 below)
Youtube: Next Level Teacher, RoboticsRulesCompetition
We utilised YouTube FLL coaches to learn more about mechanism design and code.
Brick Studio 3.0 & Documentation: https://studiohelp.bricklink.com/hc/en-us
Used to CAD our mechanisms to create renders or initial designs. We also used the help site for learning how to use it, built into the app itself.

Examples

https://www.youtube.com/watch?v=Lh_dVlIDBKQ **FLL Robot Keeps Drifting? Here's Why (and How to Fix It) - Next Level Teacher**
This video explains code and mechanical failures that cause FLL robots to drift left and right, making it inaccurate and inconsistent. One main change that we implemented after watching the video was to lower the centre of gravity of the robot and switch the wheels to thicker ones with tyres.

https://www.youtube.com/watch?v=V96TLTVYCX4 **AWESOME One-way Door Designs for your FLL Robot - RoboticsRulesCompetition**
This video provides designs of one-way door mechanisms, which we used in the building of the mechanism that completes **Mission 7: Heavy Lifting** and **Mission 10: Scale Pan** making them more reliable because before we were using arms that could not hold the objects properly, sometimes letting them escape. The one-way gate allowed the objects to enter the mechanisms but not escape.

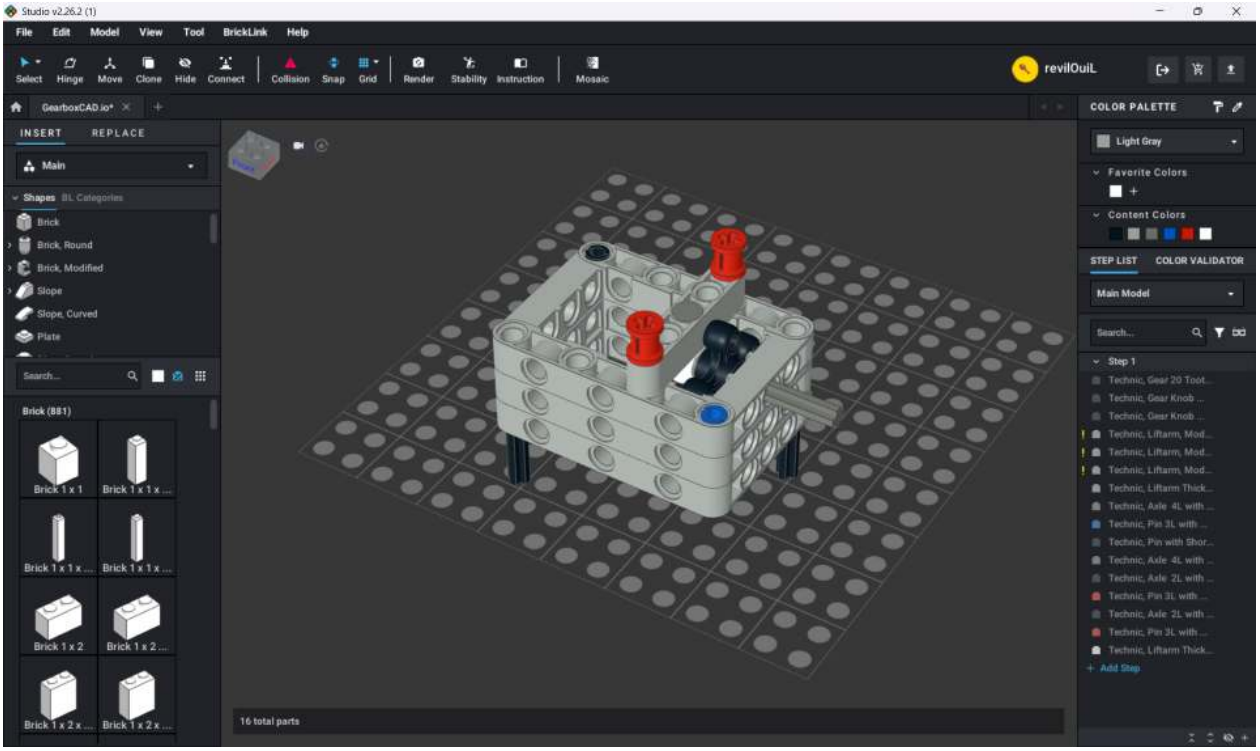


Figure 10: *Example usage of BrickLink Studio 3.0 to CAD a gearbox design*

CAD-ing gearbox design example: Figure 10 shows our workflow in BrickLink Studio 3.0, designing a gearbox for rendering for our documentation/powerpoints. This helped us improve clarity between all team members and create better, clearer images for our presentation.

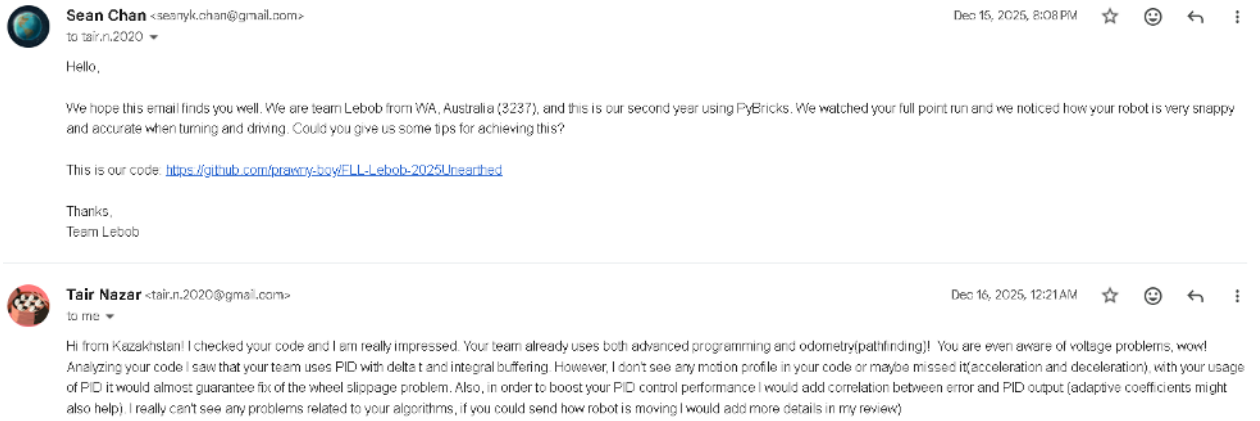


Figure 11: *Emails with Tidal Tumble Robotics*

TidalTumble feedback: Figure 11 shows a part of our communication with Tidal Tumble, a past FLL team who have achieved high standards in robot games many times. We mainly asked them about accurate turning and movement, and requested them to take a look at our code. This helped us improve our own robot by implementing their suggestions of using dynamic acceleration in robot movement.

Design

Contribution of Team Members

Brainstorming as a Team

Brainstorming allowed every team member to share their opinions on which missions we should attempt and which ones seemed too difficult to pursue. Even a single suggestion often sparked a chain reaction, with others building on the idea, refining it, or proposing alternatives until we identified the most effective approach. Because we aimed to work quickly and efficiently, this collaborative process was especially valuable; it helped us reach a clear decision early on, ensuring that once we selected our preferred route, we could begin coding without worrying about major changes later in the season.

Conflicts as a Team

Even though many of our team members already knew each other well, during discussions many conflicts always came up, especially during high-pressure situations in the last few weeks leading up to the competition. Instead of avoiding conflict, we allowed everyone to express their opinions and used it as an opportunity to create the best solution for everyone.

At the start of the season, we resolved issues informally, but as the season continued, we created a somewhat mental plan on how we could identify and resolve conflicts. Here are our main ideas with examples.

Our Steps	**Example**
Identify the issue.	*Uneven workloads and communication issues*
Let each member explain their view.	*Some members left out, others felt that some members weren't doing any work*
Focus on team goals over personal preferences.	*Win internationals, so everyone had to contribute*
Use past experience/data wherever possible.	*Last time we just let them be, which we decided was a bad idea since less work would be done.*
Agree on what to do in the future.	*Making it more clear to all team members about work assignments and more discussions*
Reflect after what could be improved.	*More follow-up is needed after discussions, members could forget*

Prototyping

Prototyping enabled our team to experiment with a variety of mechanisms and approaches for completing different missions. Through this process, we were able to evaluate multiple concepts, compare their effectiveness, and refine our ideas based on testing results. After exploring several viable solutions, the team collectively discussed the options and ultimately reached an agreement on the most reliable and efficient design.

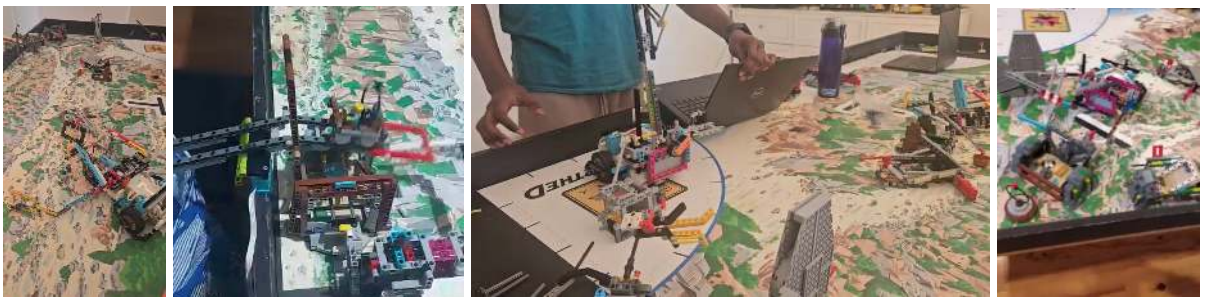


Figure 12: *Examples of us testing our mechanisms and making changes through observation*

GitHub

We used GitHub as our main platform for managing code and tracking changes. It allowed all team members to work on the code at the same time, and know what other team members were doing at any time.

- Because **everyone can contribute**, it allows newer members to learn by doing while still keeping our work organised and version-controlled.
- Experienced team members also created a **guide** so others can understand how to use GitHub effectively, from making commits and pull requests to contributing on the code.
- **All our repositories are public**, which means other FLL teams can explore our robot code, website projects, and pathfinding experiments for inspiration. (Figure 13)
- All of our repositories are in our **Lebob-Robotics GitHub Organisation**.
- **Parallel work**: Allows all members to contribute to coding simultaneously
- **Reverting Code**: Allows code to be reverted back to past versions through commits, showing version history, and improvements.

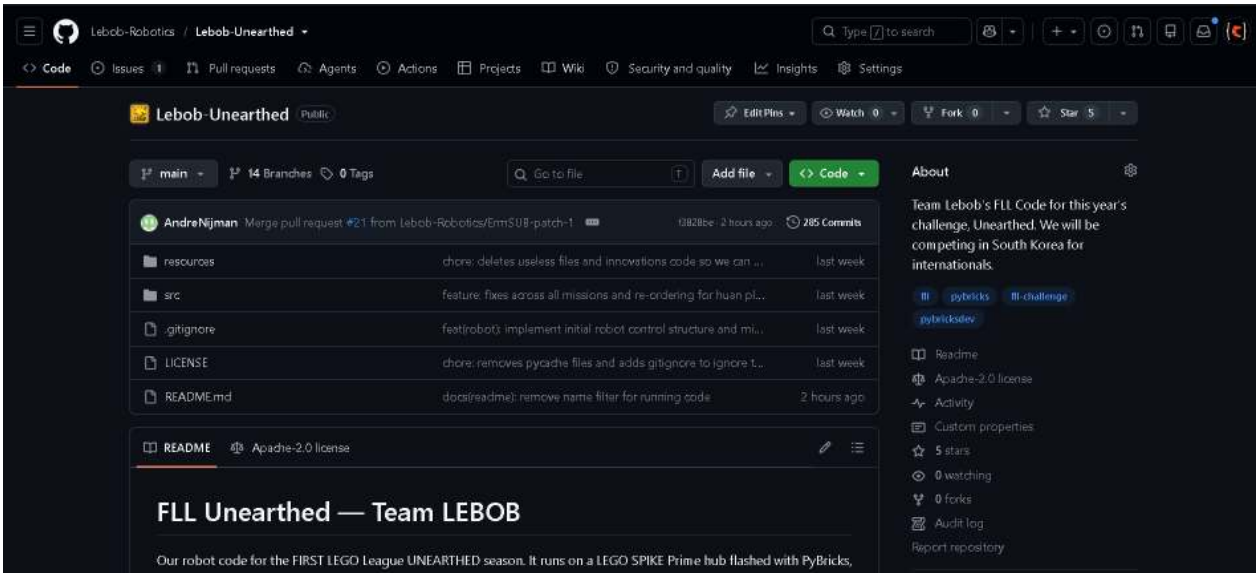


Figure 13: *The GitHub page for our robot code repository (https://github.com/Lebob-Robotics)*

Discord & Trello

We used a combination of discord and trello (see Documentation pages 7-8) to communicate between all of our team members, parents and mentors informed and organised throughout the season. We mainly used it for:

- Sharing session schedules
- Coordinating attendance
- Communication of plans/changes
- Updates on sponsorships, outreach, notes, goals
- Sharing photos and files

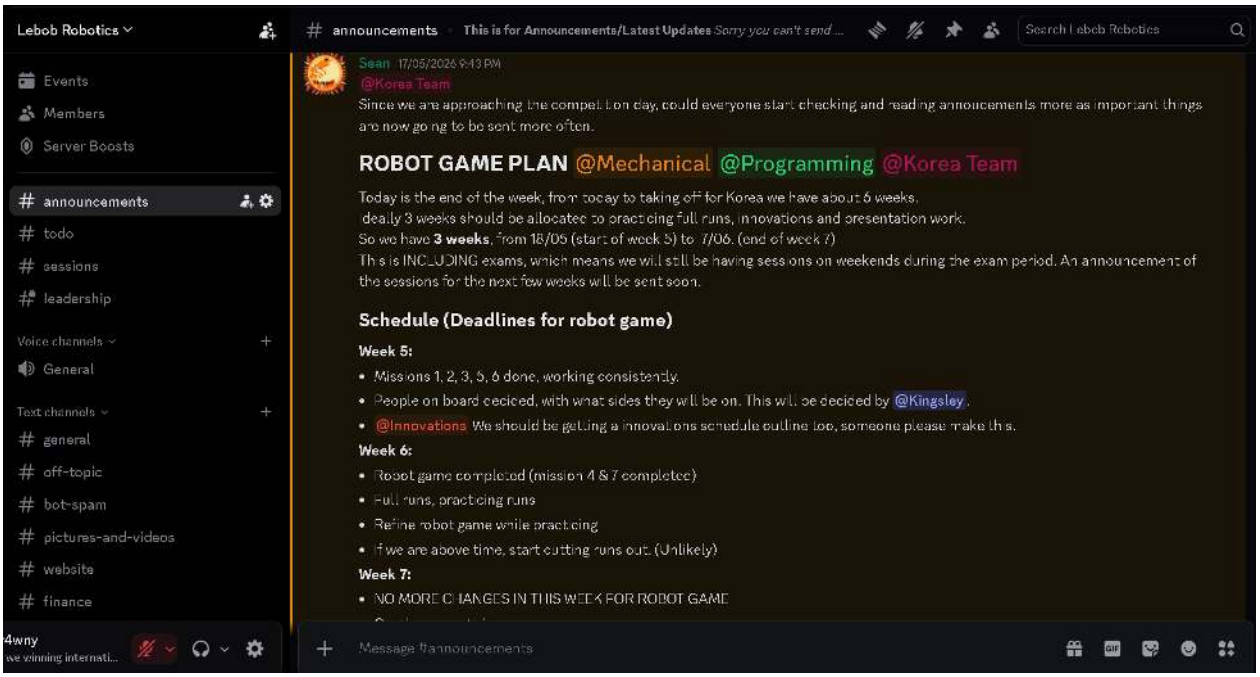


Figure 14: *Our Discord server*

All Lebobians' Building/Coding Skills

Building Skills

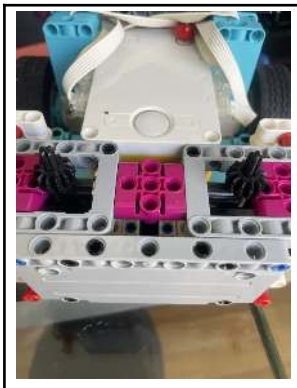
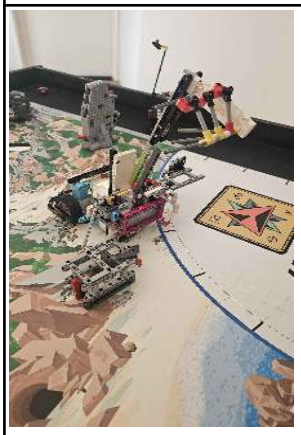
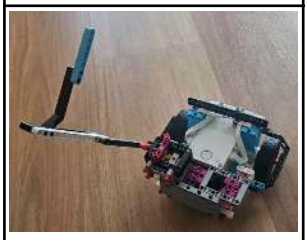
Each person created around one attachment used within the runs, despite not all of them being used, they were considered and tested before deciding to not use it. We all collaborated with each other to construct the attachments so that most of them could both be attached to the robot and be necessary within the run. We also made sure that everyone would get a chance to code the robot showing what they can do.

For each attachment we based each of them on **three engineering principles**:

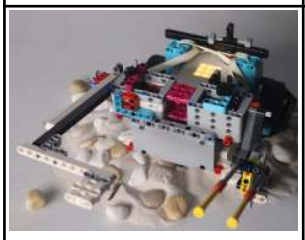
- **Simplicity:** We like to keep our attachments simple but reliable where they are able to complete missions consistently but if they do break, it is easy to maintain. Additionally, we focus on passive mechanisms, which complete missions without the use of a motor, reducing the need for gearboxes and also allowing us to use the limited motors for other missions in the same run.
- **Efficiency:** We focus on keeping our attachments efficient, meaning they complete missions quickly and in combination with others to reduce overall run time. Instead of designing mechanisms that only handle one task slowly, we group compatible missions

together and build attachments that can perform multiple actions in a single trip. This helps us minimise time wasted on unnecessary returns to base. For example, when planning our runs, we looked for ways to combine movements so the robot could complete back-to-back missions without extra adjustments, making our overall strategy faster and more streamlined.

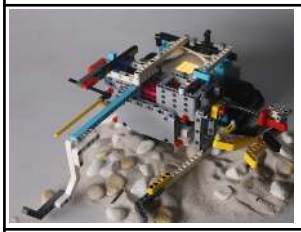
- **Reliability:** We build our attachments with strong safety margins so they can operate reliably without missing game items. To do this, we use hardstops to physically limit movement and set the mechanism to a determined place that we find in trial-and-error and prevent mechanisms from over-rotating, and we rely on sensors built into the hub to ensure the heading of the hub is accurate. By making these changes both in code and in our mechanisms, we increase the reliability of our robot completing missions.

	**Gearbox** Holes around gears for easy attachment and a reduction gear ratio so that we can increase torque. Attachments have pins in the bottom so that we can slip them in and out easily. The attachments are driven by two large motors with small 12-tooth gears.
	**Attachment 1** This attachment is for run 1, completing three missions. It does this by having three moving parts. The longest trident-like arm is used to grab the heavy artifact, the cage is used to collect and bring back the boulders, and the last mechanism is used to flip over the plant surface. It does this all in one motion and then returns, bringing the artifact and the boulders back with it.
	**Attachment 2** Attachment 2 is solely for the silo mission. The large arm is used to smack down on the lever to launch out the planted gears with strength. The gear for this attachment is changed so that there are fewer but bigger teeth allowing a larger torque to be created, meeting the strength requirements to launch the gears.

Attachment 3
This attachment does two missions with two arms. The right arm knocks down the bucket, as well as latching onto the back of the marketplace to lift it up eventually after the robot moves back. The left arm rotates on an axis so that it can catch onto the scalepan to pull it out.



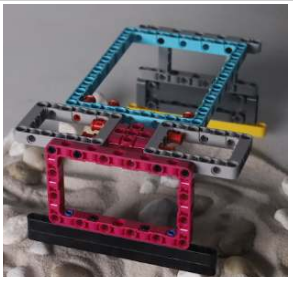
Attachment 4
The two yellow prongs are used to precisely grab the artifact from under the mineshaft rails, whilst the other arm is used to complete the minecart mission by raising the rails.



Attachment 5
This attachment is used for run 5, completing the second mission. The yellow arm is used to pick up the artifact through the grey ring, and the other two arms are used to push/rotate the other blocks of land.

Attachment 6
This attachment is for run 6, completing two missions simultaneously. The longest arm contains gears at the end to reach over the shipwreck, jamming into the gears of angler artifacts and turning them. The arms at the bottom are used to complete the shipwreck mission passively. A passive mechanism is used to release the flag into the mission on return.

Attachment 7
This attachment is for run 7, completing three missions. The right arm does the statue as well as picks up the opposing team's minecart, placing it in the forum. The left arm places one of the site markings down, while a passive second arm is released to place another. The collection at the front holds all of the objects placed in the forum.



Fake Robot
We created a skeleton bot for making the mechanisms as it allowed for us to code on the real robot with the mechanisms we already had while also making improvements or making new mechanisms for other missions at the same time.

Coding Skills

We also used GitHub so that anyone can just clone our code and work on it at home. If they needed help the more experienced coders on our team would help them and teach them. Since we all come from the same school we all also knew how to code from our classes, many of us already knew basic python, but a few team members also did personal projects and helped less-experienced members in using GitHub and programming in python using more advanced features like classes and decorators. An example of a project some of our team had to complete for school was a simulation of the monty hall 3-door problem in python. Other team members have also worked on projects such as games (choose your own adventure, 16 player tag) and technical projects.

Create

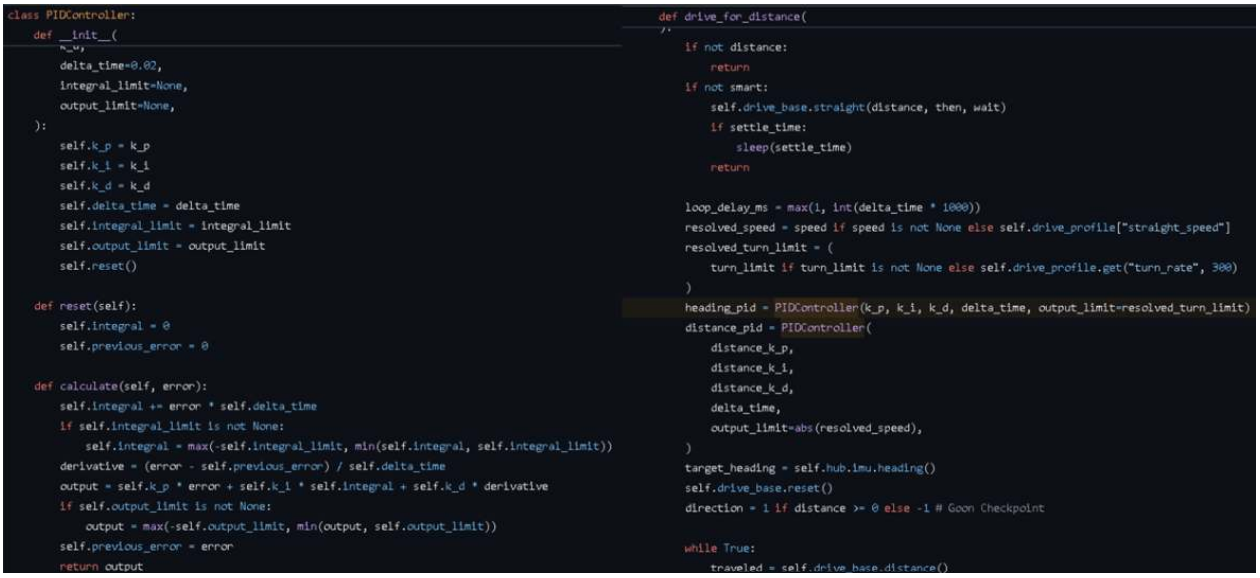
Innovative Code and Sensor Use

We based our code on programming principles to ensure our code is **robust**, **readable**, **maintainable** and **modular**.

- **Robust:** Ensuring the code is impossible to break/crash.
- **Readable:** Making sure all team members can read our code, achieved through using classes to group drivebase functions, making it easier to use in coding missions.
- **Maintainable:** Detailed comments and functions used all over the code to keep it easy to maintain.
- **Modular:** Multiple files used in our codebase and many modules imported to keep the code short and re-use existing code. *"Steal from the best, invent the rest"*

Examples

Using PID: PID stands for Proportional Integral Derivative, used to create smooth and accurate movements of arms, wheels etc. Because we used PyBricks for coding our robot, we can set it so that the drive base uses the built-in gyro in the spike to align our movement for better accuracy, as the module already implements PID in its code. We also created our own PID controller in our nationals code for more accurate turning but found that the in-built one was sufficient.



```
class PIDController:
    def __init__(
        k_p,
        delta_time=0.02,
        integral_limit=None,
        output_limit=None,
    ):
        self.k_p = k_p
        self.k_i = k_i
        self.k_d = k_d
        self.delta_time = delta_time
        self.integral_limit = integral_limit
        self.output_limit = output_limit
        self.reset()

    def reset(self):
        self.integral = 0
        self.previous_error = 0

    def calculate(self, error):
        self.integral += error * self.delta_time
        if self.integral_limit is not None:
            self.integral = max(-self.integral_limit, min(self.integral, self.integral_limit))
        derivative = (error - self.previous_error) / self.delta_time
        output = self.k_p * error + self.k_i * self.integral + self.k_d * derivative
        if self.output_limit is not None:
            output = max(-self.output_limit, min(output, self.output_limit))
        self.previous_error = error
        return output
```

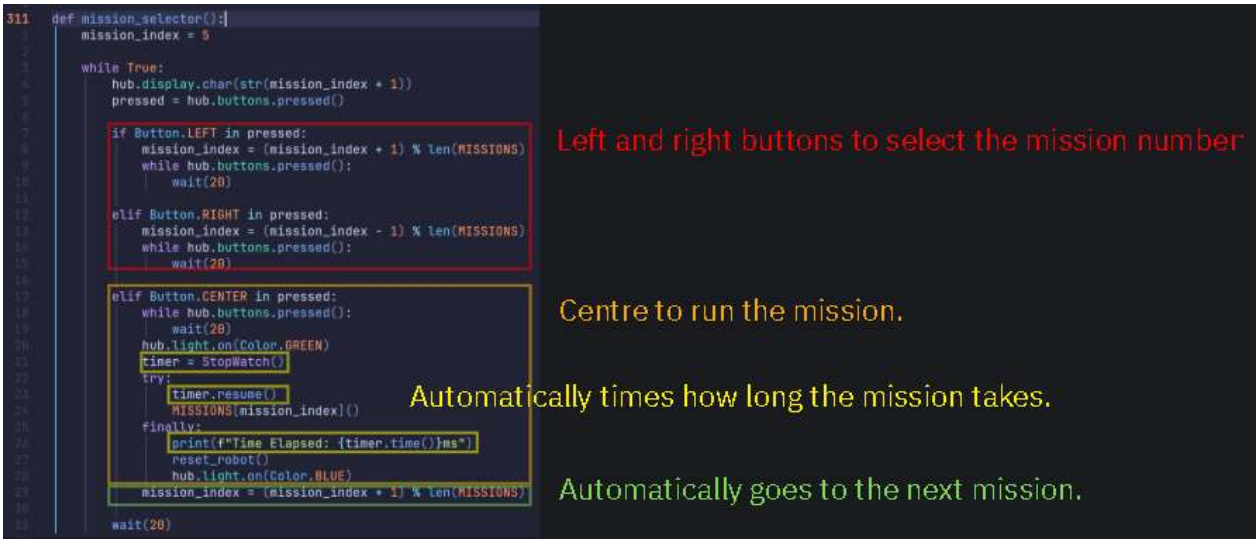
```
def drive_for_distance(
    if not distance:
        return
    if not smart:
        self.drive_base.straight(distance, then, wait)
        if settle_time:
            sleep(settle_time)
        return

    loop_delay_ms = max(1, int(delta_time * 1000))
    resolved_speed = speed if speed is not None else self.drive_profile["straight_speed"]
    resolved_turn_limit = (
        turn_limit if turn_limit is not None else self.drive_profile.get("turn_rate", 300)
    )
    heading_pid = PIDController(k_p, k_i, k_d, delta_time, output_limit=resolved_turn_limit)
    distance_pid = PIDController(
        distance_k_p,
        distance_k_i,
        distance_k_d,
        delta_time,
        output_limit=abs(resolved_speed),
    )
    target_heading = self.hub.imu.heading()
    self.drive_base.reset()
    direction = 1 if distance >= 0 else -1 # Goon Checkpoint

    while True:
        traveled = self.drive_base.distance()
```

Figure 15: *Screenshot of our PID code*

A PID (Proportional, Integral, Derivative) controller is a feedback control loop mechanism used to automatically regulate systems (like temperature, speed, or robotics) by comparing a current measurement to a target value and continuously adjusting the output to eliminate the difference, essentially enabling us to drive to a specific position or move an arm to a specific angle. We used it in our drive and turning functions to ensure our robot was more accurate.

Resetting the angle of the mechanism motors: We made a function to do this, move until stalled, which moves the motor at a certain speed until it is unable to move physically at all. This lets us align the motor to a certain surface so that we can have better accuracy and so that people don't need to set the mechanism to the right position in the pit-stops.

Automatically Updating Menu using Decorators: We used a decorator to automatically add new functions of runs to our robot menu, allowing us to save time when coding runs.




```
def mission_selector():
    mission_index = 5

    while True:
        hub.display.char(str(mission_index + 1))
        pressed = hub.buttons.pressed()

        if Button.LEFT in pressed:
            mission_index = (mission_index + 1) % len(MISSIONS)
            while hub.buttons.pressed():
                wait(20)

        elif Button.RIGHT in pressed:
            mission_index = (mission_index - 1) % len(MISSIONS)
            while hub.buttons.pressed():
                wait(20)

        elif Button.CENTER in pressed:
            while hub.buttons.pressed():
                wait(20)
            hub.light.on(Color.GREEN)
            timer = StopWatch()
            try:
                timer.resume()
                MISSIONS[mission_index]()
            finally:
                print(f"Time Elapsed: {timer.time()}ms")
                reset_robot()
                hub.light.on(Color.BLUE)
            mission_index = (mission_index + 1) % len(MISSIONS)

        wait(20)
```

Figure 16: *Mission Selector Function with Annotations*

Our Own Mission Selector: We created our own menu using code (Figure 16) to have the features that we wanted. This also allowed us to be able to change it and adapt it in any way we wanted.

Mission Autosave: This uses the hub's non-volatile storage to save the next mission index before a run begins, so the robot can be ready to run the next mission if the code ends forcefully. It translates the mission number to binary and then stores it on the hub. This saves time so that technicians don't need to reselect it from the mission menu.

How our Code Works

This system uses Object-Oriented Programming to keep the robot's code organized and efficient. A LebobDriveBase class manages motors, drive settings, gyrometer, and arm movements. The MissionControl class provides a user interface on the hub, letting users select missions during matches, and it also manages animations, gyro resets, drive profiles, and timing. Missions are easily added with an @mission() decorator, and each mission function leverages Robot helpers for reliable driving, turning, and arm control.

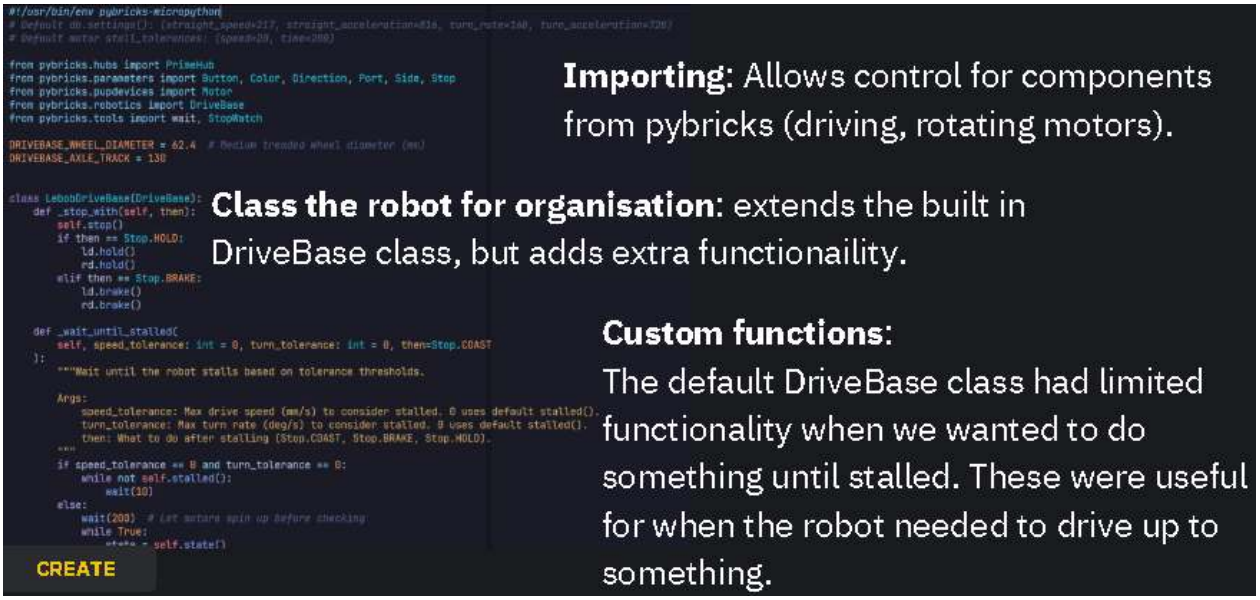
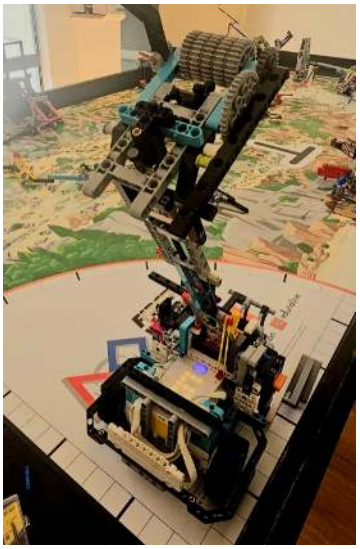


Figure 17: *Class and functions example with Annotations*

Innovative Attachments

We see all of our attachments as innovative, but here are the two best ones we think are worth writing about.

Attachment One



This attachment is innovative because it uses a single gearbox to power three separate arms at the same time. Each arm is designed to move in sync with the others, allowing the robot to complete multiple mission actions in one motion. By combining three mechanisms into one coordinated system increased efficiency on the field. This design not only streamlines how the robot performs missions but also shows creative engineering in making multiple components work together through one unified drive system. This attachment shows an example of efficiency, doing 3 missions at the same time. It also helps our mission strategy we stated at the start, where we save time by doing missions at the same time.

< Figure 18: *Attachment one image*

Attachment Six

Figure 19: *Attachment six image*

This mechanism uses two separate gearboxes to perform a single, coordinated task. One gearbox controls the vertical movement of the main arm, lifting it up and down, while the second gearbox rotates a set of gears through a long axle to trigger additional actions. On the right side, a passive arm automatically pulls the sand back as the robot reverses, requiring no extra motors. At the front, another passive element drops the flag into the ship as the robot drives forward, and the left side of the mechanism extends outward to push the ship into position. Together, these active and passive components create a highly efficient system that completes multiple mission steps in one smooth sequence. This attachment shows how we group missions close together so that we can save time. We used passive mechanisms at the same time to maximise the completion of missions.

Iterate

Repeated Testing of Robot and Code

Logging

See documentation booklet pages 9-78.

We created logs for most runs we tested on our robot, describing what happened and how we fixed it in real time after every test.

GitHub Commits

By using GitHub, we also used its **commit message system** to track changes and allow for collaboration, meaning that everyone could contribute to code. We ensured that all team members followed a **convention in naming** for consistency, and utilised multiple branches for

working on different sections/projects on the code. Over time, we accumulated many commits which allowed all team members to see how our code changed over time and even backtrack to old code if newer code broke.

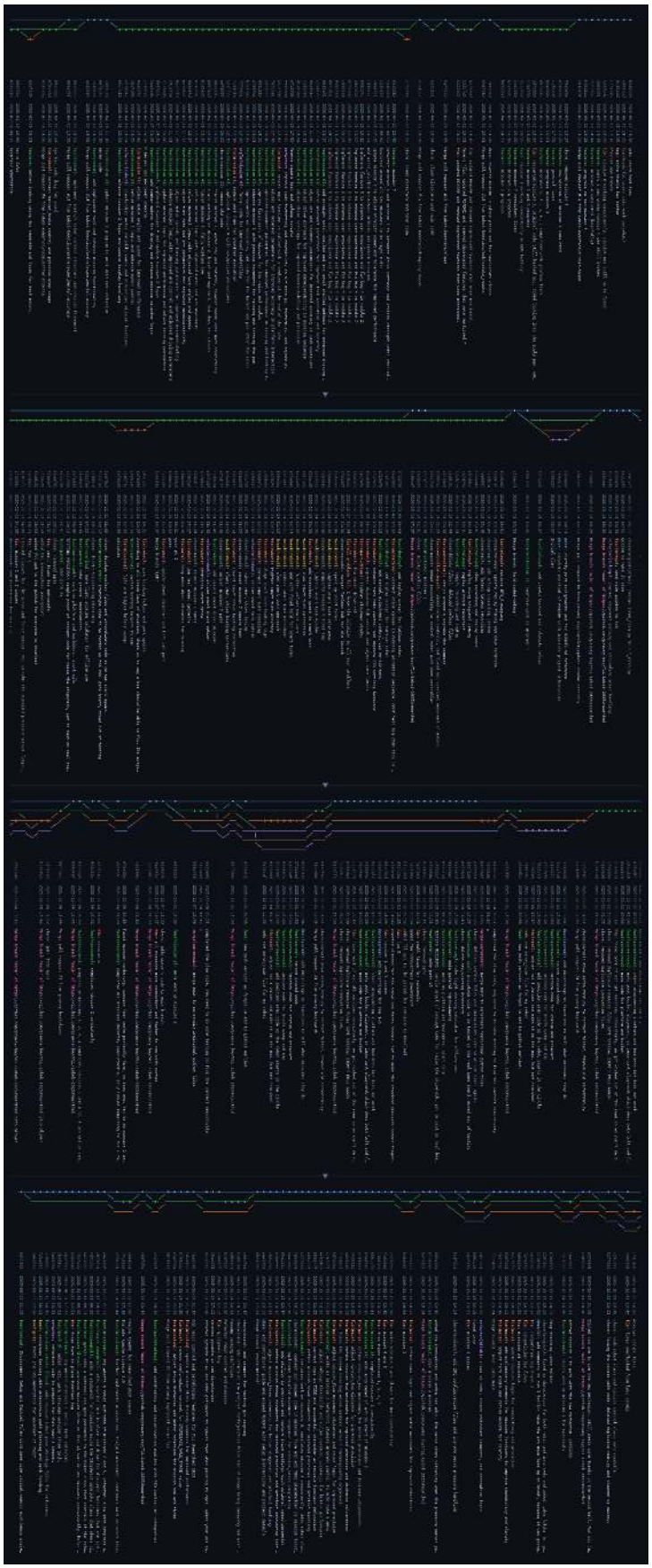


Figure 20: *All of our commits (500+)*

Figure 21: *A specific section of our commits*

Improvements Based on Testing

Robot Improvements

Base robot:

We tested several iterations of our base robot, trying different friction-reducing supports like a plastic bulb and a caster ball to see which gave us the most consistent and stable movement without swerving. Our final design used two medium motors for the wheels and two non-friction wheels for balance, with two large motors and gears powering the attachments. We kept the centre of gravity low and avoided hanging cables to make the robot more reliable during runs.

Gearbox:
Something that we noticed after our regional competition was that changing the attachments on our current gearbox was extremely tedious, and it was limiting the points that we can get. Most of our time went into changing the attachments rather than the robot getting points. We didn't have time between our nationals comp to change the gearbox, as changing it will result in a need to restart the robot and its code.
After the comp, we developed different versions of gearboxes, making sure that they work with key criteria:

- **Perpendicular transmission:** Arms should be able to rotate horizontally
- **Vertical transmission:** The robot should be able to rotate arms vertically
- **Reliable gear meshing:** The gears should reliably mesh and should not skip under large loads.
- **Easy attachment switching:** The pairs of gearboxes should ideally slide on and off, and should not have the need for additional locks.

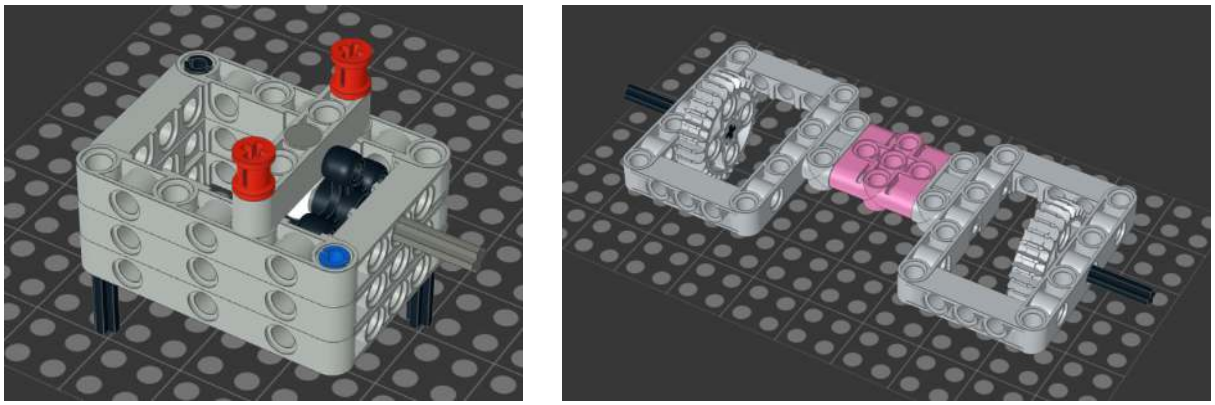


Figure 22: *New (Left) vs Old (Right) Gearbox*

Code Improvements

By using GitHub, we were able to track our changes and improvements in code over time in commit messages. (Figure 21)

To ensure our runs were consistent, we constantly improved our code and mechanisms after running them repeatedly. Although it was very slow, we ended up with consistent runs which can be run multiple times without failing. The commits and logs in the documentation document show the many changes and fixes we made to the code and mechanisms as we tested. **See documentation booklet pages 9-78.**
E.g. 13:16:33 – run 2. We fixed the gearbox and ran it, but it misaligned with the mineshaft

Communicate

Our Process and Lessons Learned

"It's about the friends we made along the way" - Kingsley
This was what we learnt in FLL, to achieve a lot, but also know that we can't be perfect but we just need to do our best.

Working Together: As a team, we learned how to support each other and stay connected, even when things got tough like physical sicknesses or power outages.
Division of Work: We learned that splitting up tasks works way better than having too many people crowd around one job. It kept us organised, efficient, and a lot less stressed.
Gracious Professionalism: Throughout the season, we practiced GP by helping rookie teams in our competitions and in our school club, such as a team called *John's cold team* and some younger first-year teams.
Healthy Work Environment: We focused on making a space where everyone felt comfortable and supported and included. Our teamwork improved and because of the environment around us we got things finished on time
No big changes: Last year we rebuilt the whole robot before nationals and ran out of time to code it properly which is why we lost. This year, we didn't repeat that mistake. Instead of redesigning everything, we just adjusted the path, which meant way less recoding.
Documentation: In the past, we didn't keep proper records of our testing or progress. This season, we kept a real logbook, which gave us clear proof of what we tried, what worked, and how we improved in the documentation document.
Communication: Communication used to be messy, but using both WhatsApp and Discord made things much easier. Sharing updates, photos, and announcements became way more organised.
Organisation: Trello helped us keep track of tasks and see what needed to be done. It kept everyone aligned and made our workflow smoother than last year.

Comments

Our team is extremely proud of our robot in many ways; here are statements from each person in our team.

Sean: I am proud of how good our robot turned out, considering the short amount of time we had to build it. We achieved an amazing score! One thing I learnt throughout this journey was PID, creating my own custom PID controller for robot code. (But we didn't use it)
Andre: I really appreciated our team's effort this semester, everyone put their best work into this. I learned FEA (Finite Element Analysis) and how it can simulate stress on designs to point out possible issues, such as weak points prone to breakage.
Kingsley: I am very proud of making the robot and some of its attachments. Our teammates collaborated and worked very well together as well as listening and interpreting different things. Over the season I improved at CAD, especially while doing it for the innovations project.
Chris: I am very happy to work on the robot this year and having the chance to work on the robot allowed me to expand on my existing knowledge and abilities of making mechanisms.
Leven: I am glad that I was able to gain this experience, as it was only my second year of robotics, being able to contribute ideas to all aspects of our project, designing a mechanism, working on documents, and being on the board operating the robot in nationals. I think it has been a valuable experience for my teamwork and cooperation skills.
Oliver: I think our team collaborated very well, and we all cooperated to complete the mission to the best of our ability. I learned how to express ideas simply and I can also use this skill in presentations outside of school.
Aaron: I think that my team helped me understand the code that they wrote, told me what errors to fix, and collaborated well across all categories in general. The group chats made it easy to be organised and to communicate. I learned out to CAD in OnShape and how to design complex mechanisms to do multiple things at once.

Subesh: I believe that our team worked cohesively throughout this semester, despite setbacks with delays in receiving the mission models. I believe we were organised and that everyone did their part in all parts of FLL. I learnt how to CAD using Onshape so that I can use this skill in engineering and design outside of FLL like school and FRC.

END OF DOCUMENT